Benchmarking AI Models in Software Engineering: A Review, Search Tool, and Enhancement Protocol

Roham Koohestani, Philippe de Bekker, and Maliheh Izadi

Abstract—Benchmarks are essential for consistent evaluation and reproducibility. The integration of Artificial Intelligence into Software Engineering (AI4SE) has given rise to numerous benchmarks for tasks such as code generation and bug fixing. However, this surge presents challenges: (1) scattered benchmark knowledge across tasks, (2) difficulty in selecting relevant benchmarks, (3) the absence of a uniform standard for benchmark development, and (4) limitations of existing benchmarks. In this paper, we review 173 studies and identify 204 AI4SE benchmarks. We classify these benchmarks, analyze their limitations, and expose gaps in practices. Based on our review, we created BenchScout, a semantic search tool to find relevant benchmarks, using automated clustering of the contexts from associated studies. We conducted a user study with 22 participants to evaluate BenchScout's *usability*, *effectiveness*, and *intuitiveness* which resulted in average scores of 4.5, 4.0, and 4.1 out of 5. To advance benchmarking standards, we propose BenchFrame, a unified method to enhance benchmark quality. As a case study, we applied BenchFrame to the HumanEval benchmark and addressed its main limitations. This led to HumanEvalNext, featuring (1) corrected errors, (2) improved language conversion, (3) expanded test coverage, and (4) increased difficulty. We then evaluated ten state-of-the-art code language models on HumanEval, HumanEvalPlus, and HumanEvalNext. On HumanEvalNext, models showed a pass@1 score reduction of 31.22% and 19.94% compared to HumanEval and HumanEvalPlus, respectively.

Index Terms—Benchmark, Code Generation, AI4SE, Large Language Models, Search

I. 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 [1], a widely-used dataset that has been used for assessing large language models (LLMs)

R. Koohestani, P. de Bekker, and M. Izadi are with the Delft University of Technology, Delft, The Netherlands. (e-mail: rkoohestani@tudelft.nl; research@philippedebekker.com; m.izadi@tudelft.nl).

R. Koohestani's ORCID: 0009-0000-1649-9596, M. Izadi's ORCID: 0000-0001-5093-5523.

such as Codex [1], Gemini [2], and GPT-4 [3]. HumanEval was downloaded $97,444$ times in February 2025 on a single platform (Hugging Face) [1]. This high number demonstrates the strong and sustained interest in this benchmark. Despite its broad adoption, HumanEval contains numerous flaws and inconsistencies [4]. 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 [5], [6], [7], [8] or improve test coverage [9], [10]. However, these extensions often build upon the original dataset without addressing its fundamental deficiencies, allowing inherent issues—such as 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. [2] This necessitates continuous elevation of the complexity of programs to better reflect the capabilities of contemporary models.

With this study 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 tackle these issues, we systematically map the current benchmark landscape to facilitate informed selection and establish unified guidelines for developing robust and adaptable benchmarks. We further demonstrate the effectiveness of our framework through a case study on the highly-popular benchmark, HumanEval.

More specifically, we first conducted a systematic review of benchmarking efforts in AI4SE from 2014 onward and identified **204 benchmarks** from 173 studies. To the best of our knowledge, this is the first comprehensive analysis of AI4SE benchmarks. We then extracted a given benchmark's key metadata, including objectives, category, programming language, natural language, and relevant tasks, to structure

---

[1]https://huggingface.co/datasets/openai/openai_humaneval
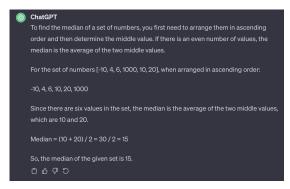[2]https://paperswithcode.com/sota/code-generation-on-humaneval

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

the AI4SE benchmark landscape. Leveraging this data, we developed **BenchScout** [3], 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. To obtain the optimal configuration, we performed a parameter search. 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.

Next, 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 [10], 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 173 studies, identified 204 AI4SE benchmarks, and analyzed their limitations and gaps (section III),
- 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 IV),
- 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 V). We assessed ten recent LLMs on this benchmark and verified significant drops in performance compared to the original HumanEval[1] and the enhanced version, HumanEvalPlus [10],
- We publicly share our data and details of the user study in our replication package. [4]

II. RESEARCH QUESTIONS

We propose a set of research questions (RQ) to guide our study, as presented below.

- **RQ1:** What is the current landscape of AI4SE benchmarks?
  1) What are the existing benchmarks used in the field of AI4SE?
  2) What are the key limitations and shortcomings of commonly used benchmarks?
- **RQ2:** How can we effectively process the growing number of benchmarks and extract meaningful insights?
  1) How can we design a tool to efficiently parse this dense data and make it accessible to SE researchers and practitioners?
  2) How useful is such a tool from users' perspective?
- **RQ3:** How can we develop a general framework to improve AI4SE benchmarks?
  1) What aspects of existing benchmarks does this framework address?
  2) How would the process of applying this framework to an existing benchmark work?
  3) What are the effects of applying this framework to an existing benchmark?

III. EXISTING BENCHMARKS, A REVIEW

A. 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 platform [5].

We selected these keywords based on a preliminary assessment of highly-cited benchmark research. Three authors worked together to revise the search criteria, ensuring they were both relevant and complete. Our selection criteria targeted primary studies published in English from 2014 to 2024. The initial retrieval resulted in 962 papers, which, after removing duplicates, reduced to 240. Two authors reviewed the relevance of the literature. During the quality evaluation, the originality (i.e., status as a primary study), reproducibility,

---

[3]Accessible through https://evalpro.online/

[4]https://github.com/AISE-TUDelft/AI4SE-benchmarks
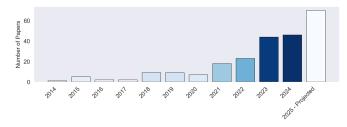[5]https://paperswithcode.com/datasets

Fig. 2: Number of published benchmark papers since 2014

and accessibility of each study were assessed. To capture a thorough set of benchmarks, we also performed forward and backward snowballing. After the manual screening, 155 papers were deemed relevant, with an additional 18 identified through snowballing. Ultimately, 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 [11]). 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.

B. Results of the Review

Our review revealed a significant increase in the number of benchmarks published over time (see Figure 2). We identified 204 benchmarks in total, with 47 linked papers published in 2024 alone. Using an exponential projection, we estimate that this number will reach 70 for the year 2025. This highlights the growing impact of AI4SE benchmarking and the need for a comprehensive overview of the existing literature. In the following, we summarize key insights from our review. Due to space constraints we present the created taxonomy along with the benchmarks in each category. For more detailed information about the benchmarks please refer to the appendix and replication package.

HumanEval Benchmark family: Currently, one of the most popular AI4SE benchmarks is HumanEval [1], used to evaluate the performance of many notable code-aware models (e.g., Codex [1], Gemini [2], and GPT-4 [3]). This benchmark is used mainly for code synthesis, though there also exist some variations for code repair and code explanation [7]. Table XVIII 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

---

TABLE I: Overview of AI4SE benchmarks stemming from HumanEval [1].

Category	Name
Original	HumanEval [1]
Improved Language Support	MultiPL-HumanEval [6] HumanEval-Fix [7] HumanEval-Explain [7] HumanEval-Synthesize [7] HumanEval-X [8] Multi-HumanEval [5] HumanEvalXL [12]
Improved Testing	HumanEval+ [10] HumanEval-MINI [10] HE-Eval [9]
Instruction-based	InstructHumanEval [6]
Extended	EvoEval [13]

TABLE II: Overview of AI4SE benchmarks derived from MBPP [14].

Category	Name
Original	MBPP [14]
Improved Language Support	MultiPL-MBPP [6] MBXP [5]
Improved Testing	MBPP+ [10] MBPP-Eval [9]

proper test coverage, (3) incorrect canonical solutions, and (4) imprecise problem definitions. While there are versions that have improved the language support [5], [6], [7], [8] and test coverage [9], [10], 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 [14]: 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 XIX). Upon a more in-depth analysis of MBPP and its family of

TABLE III: Overview of competitive programming, code complexity, and code efficiency benchmarks.

Category	Name
Competitive Programming	CodeContests [15] APPS [16] LiveCodeBench [17] LeetCode [18] CodeElo [19]
Code Complexity	CoRCoD [20] GeeksForGeeks (GFG) [21] CODAIT [7] CodeComplex [22] PythonSaga [23]
Code Efficiency	EffiBench [24] CODAL [25] PIE [26]

TABLE IV: Overview of data science & domain-specific benchmarks.

Category	Benchmark
Data Science	DS-1000 [27] NumpyEval [28] PandasEval [28] JuICe [29] DSP [30] ExeDS [31] DSEval [32]
Domain-Specific	Bio-Coder [33] Bio-Coder-Rosalind [33] WebApp1k [34]

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 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 [10]). 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 XX features benchmarks with competitive program-

[7]CODAIT-2021 https://ibm.co/4emPBIa

[8]https://huggingface.co/datasets/AI-MO/aimo-validation-amc

[9]https://github.com/Aider-AI/aider/blob/main/benchmark/README.md

TABLE V: Overview of Mathematical Reasoning Benchmarks.

Category	Name
Mathematical Reasoning	MATH [35] MATH500 [36] MathQA [37] MathQA-Python [14] MathQA-X [5] LīLA [38] MultiArith [39] GSM8K [40] GSM-HARD [40] TheoremQA [41] PECC [42] BRIGHT [43] AMC12[8]

TABLE VI: Overview of Natural Language Benchmarks.

Category	Name
Text2Code	CoNaLa [44] MCoNaLa [45] CoNaLa-SO [46] APPS [16] APPS-Eval [9] AixBench [47] Natural2Code [2] CoSQA [48] WebQueryTest [49] AdvTest [49] CONCODE [50] MTPB [51] CAASD [52] Shellcode_IA32 [53] Odex [53] PSB2 [54] TACO [55] Turbulence [56] Aider [9] NL2ML-lib [57] RMCBench [58] Evil [59]
Text2Text (about code)	InfiCoder-Eval [60] BRIGHT [43]
Code2Text	DeepCom [61] Hybrid-DeepCom [62] BinSum [63] Code Attention [64] Funcom [65] CodeSum [66] CoDesc [67] Parallel [68] CoDocBench [69]

TABLE VII: Overview of SQL-related Benchmarks.

Category	Name
Text2Code (SQL)	BIRD [70] KaggleDBQA [71] StacQc [72] Spider(V2) [73] Spider-Syn [74] Spider-Real [75] Spider-DK [76] Spider-CN [77] SParC [78] Lyra [79] DuSQL [80] CoSQL [81]

TABLE VIII: Overview of Programming Language Translation Benchmarks.

Category	Name
Programming Languages	CodeTrans [49] TransCoder-ST [83] CoST [84] AVATAR [85] Multilingual-Trans [86] NicheTrans [86] LLMTrans [86] G-TransEva [87] CODEDITOR [88]
Libraries	DLTrans [86]
Intermediate Representation	SLTrans [89]
Language Conversion Frameworks	MultiPL-E [6] MultiEval [5]

TABLE IX: Overview of Selected Real-to-Life SE Benchmarks.

Category	Benchmark
Software Development & Agent Benchmarks	DevBench [90] DevEval [91] CoderUJB [92] CODAL [25] ToolQA [93] MIT [94] SAFIM [95] AgentBench [96]
Class Level	ClassEval [97] CONCODE [50] BigCodeBench [98]
Project & Cross-file	SWE-bench [99] CrossCodeEval [100] CoderEval [101] DotPrompts [102] BigCloneBench [103] DI-Bench [104] DyPyBench [105]
Repository Level	RepoBench [106] RepoEval [107] EvoCodeBench [108] SketchEval [109] Stack Repo [109] ML-BENCH [110] CodeGen4Libs [111]

TABLE X: Overview of Selected API and Retrieval Benchmarks by Category.

Category	Benchmark
API Prediction	RestBench [112] APIBench-Q [113] BIKER [114] Gorilla APIBench [115] Gorilla APIZoo [115]
Retrieval & Planning	API-Bank [116] CodeRAG-Bench [117] Search4Code [118] CoIR [119]
Memorization	SATML-ext [120]

TABLE XI: Overview of AI4SE Benchmarks Related to Pseudocode.

Category	Benchmark
Pseudocode to Code	SPoC [121] NAPS [122]
Code to Pseudocode	Django [123]

TABLE XII: Overview of Selected Crowd-sourced Benchmarks.

Category	Benchmark
Crowd-sourced Benchmarks	WikiSQL [124] Spider [125] NL2Bash [126] NAPS [122] SPoC [121] MBPP [14]

ming as their root, i.e., those used for understanding code complexity and efficiency. Table XXI features a set of benchmarks specifically designed to evaluate the performance of 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 XXII. 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 [82], that aim to accomplish question and answering (QA) similar to the widely recognized platform StackOverflow, to summarizing code and generating tags, Table XXIII features 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 XXIV.

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 XXV 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 collective code base. For this, Table XXVI provides 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 XXVII, prominent benchmarks focusing on leveraging the power of APIs are denoted. Furthermore, Table XXVIII lists benchmarks related to pseudocode, followed by an overview of notable crowd-sourced AI4SE resources in Table XXIX.

TABLE XIII: Overview of Automated Program Repair, Fault Localization, and Vulnerability Detection Benchmarks.

Category	Benchmark
Automated Program Repair & Fault Localization	Defects4J [127] GitBug-Java [128] EvalGPTFix [129] TutorCode [130] GHRB [131] IntroClass [132] ManyBugs [132] DebugBench [133] QuixBugs [134] RES-Q [135] StudentEval [136] Re-Factory [137] ConDefects [138] Cerberus [139]
Vulnerability Detection	CVEFixes [140] LLMSecEval [141] SecurityEval [142] Vul4J [143] FormAI [144] VJBbench [145] SmartBugs [146] Devign [147] D2A [148] BigVul [149] SARD [10] Juliet 1.3 [11] NVD [12]
Software Testing	CoverageEval [150] ATLAS [151] HITS [152] MeMo [153] MLAPIs [154]

TABLE XIV: Overview of Selected SE-Workflow Benchmarks.

Category	Benchmark
Code Synthesis & Understanding	Methods2Test [155] CRUXEval [156] CRQBench [157] CriticBench [158] CodeScope [159]
Merge Conflict Repair	ConflictBench [160]
Type Inference	TypeEvalPy [161] TypeEvalPy AutoGen [161]
Automatic Code Quality Review	CodeReview [162] Software Maintainability [163]
Hallucination Detection	HALLUCODE [164]

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. [165] observed Codex [1] being able to get above 80% coverage for HumanEval [1], yet many test smells were discovered and for another dataset, no higher than 2% coverage was attained. This reveals the

---

[10] https://samate.nist.gov/SARD/
[11] https://samate.nist.gov/SARD/test-suites/112
[12] https://nvd.nist.gov/developers/data-sources

TABLE XV: Overview of Multi-Category Benchmarks, Covering Various Tasks.

Category	Name
Multi-Task Benchmark	Big-Bench [166]
Cross-Language Code Tasks	XLCoST [167]
Cross-Language Code Benchmark	CrossCodeBench [168]
Long Code Tasks	Long Code Arena [169]
Code Search & Documentation	CodeXGLUE [169] MicrosoftDocs [13] CodeSearchNet [170]

importance of benchmarking AI4SE models' capabilities in test generation, bug repair, and understanding. In Table XXX, several benchmarks are listed that make an effort to assess the aforementioned. Additionally, Table XXXI 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.).

Finally, we collect a set of generic benchmarks in Table XXXII. 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).

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 guiding light for shaping our methodology proposed in section V.

IV. 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 [1] which has its own flaws.

To address this gap, we developed BENCHSCOUT [14], 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.

A. Context Extraction and Visualization

Approach Overview: Based on our review in section III, we compile a collection of datasets and benchmarks along with their associated papers. To present these data in a clear and understandable way, we must first map the unstructured textual content of the papers to a semistructured domain. We

---

[13] https://github.com/MicrosoftDocs/
[14] https://evalpro.online/search.html

accomplished this using text embedding models pre-trained on relevant data, such as abstracts and academic papers.

We apply dimensionality reduction to the data to create a 2D representation that is easy for users to interpret. To further enhance understanding, we assess the similarity between benchmarks by applying clustering techniques.

Implementation: We used models from the `Sentence Transformers` [15] library [171] to create embeddings, followed by t-SNE and k-means clustering from scikit-learn [172].

A randomized search was conducted across parameters like the embedding model, text templates, and number of clusters. We selected nine top-performing models from the Sentence Transformers leaderboard: *all-MiniLM-L6-v20*, *facebook-dpr-ctx_encoder-multiset-base*, *all-mpnet-base-v2*, *all-distilroberta-v1*, *all-MiniLM-L12-v2*, *paraphrase-multilingual-mpnet-base-v2*, *paraphrase-albert-small-v2*, *paraphrase-multilingual-MiniLM-L12-v2*, and *paraphrase-MiniLM-L3-v2*.

We used several text templates such as "Title Only", "Abstract Only", and combinations of title, abstract, and venue with and without explicit labels. The number of clusters varied from 8 to 15. Each combination of model and template was evaluated 100 times using different random seeds, and the Silhouette Score was used to assess the quality of the cluster.

The top Silhouette Score was obtained by the *Title + Abstract Without Indication* with the *all-mpnet-base-v2* model, initially forming 8 clusters. These were subsequently reviewed and refined manually, resulting in the final 11 clusters. We developed an interactive interface around this clustering to explore and verify the clusters, naming each for easy navigation.

B. Additional Features

To improve the search experience, we've added key features for finding relevant articles. First, using a **text-based search interface**, users can search articles by title and abstract with **fuzzy search** from Fuse.js library[16], enhancing flexibility for imprecise queries. We also added a **Paper Content Tooltip** for exploring a 2-D map of papers. Hovering over a point reveals a brief overview, and double-clicking leads to the paper's DOI page for easy access. Clicking a point with **Related Papers feature** gives four insights: visual **most similar papers** (cosine similarity) in a vector space, a **detailed overview of the paper** with metadata, a list of **papers that cite the selected paper**, and a **Paper Citations Graph** showing connections between citing papers, illustrating research impact and relations. In Figure 3 we provide a screenshot of the first page of the tool and the current clustering of papers.

C. 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

Fig. 3: Screenshot of BenchScout

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 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.

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 XVI is an overview of the posed questions.

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. [17] Below, we present an overview.

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).

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

---

[15]https://sbert.net/

[16]https://www.fusejs.io/

[17]https://github.com/AISE-TUDelft/AI4SE-benchmarks

TABLE XVI: 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

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.

V. 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 [2] and OpenAI GPT4 [3]. Despite the great fame, however, our review in section III 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.

A. Approach

To improve the quality of the given benchmark, we pursue the following approach – also illustrated in Figure 4. First, we initiate a comprehensive code review, leading to standardized observations (subsubsection V-A1). Then, we address these identified issues through a series of modifications (subsubsection V-A2), followed by a peer review (subsubsection V-A3) to ensure accuracy and reliability. Finally, we experiment with the

Fig. 4: Outline of the general approach behind BENCHFRAME through a case-study of HUMANEVAL.

TABLE XVII: Comparison of test statistics between HumanEval[18].

Metric	HumanEval	HUMANEVALNEXT	Δ
Total assertions	1325	2551	$\times 1.92$
Average assertions	8	16	$\times 2$
Median assertions	7	11	$\times 1.57$
Min. assertions	1	4	$+3$
<5 assertions	34	2	-94%

revised benchmark to evaluate and discuss the results. Below, more details are provided for specific steps in this approach.

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 suboptimal 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.

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 [6], 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 [14]. 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 HUMANEVAL-NEXT, consider Table XVII.

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 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

---

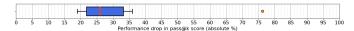[18]based on `human-eval-v2-20210705.json`

Fig. 5: Boxplot depicting the distribution of absolute drops in `pass@1` score between HumanEval and the newly introduced HUMANEVALNEXT benchmark, based on 10 LLMs (Figure 6).

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.

B. 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 [10]. 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.

C. Results

Upon conducting the experiments outlined in subsection V-B, 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 Figure 6, with an overall depiction of the performance decreases displayed in Figure 5. 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 HU-MANEVALNEXT 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.
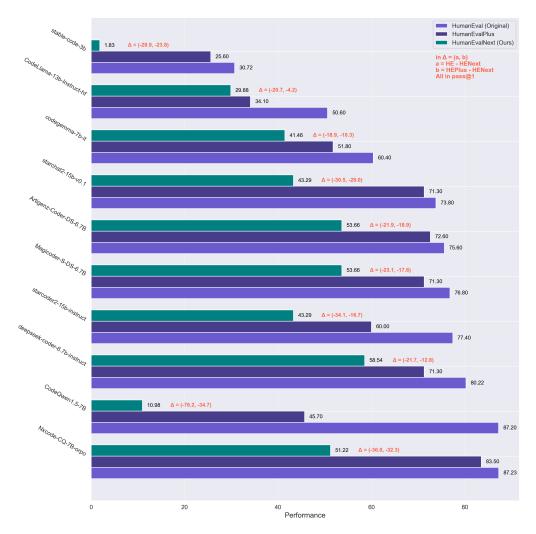
Fig. 6: Comparison of `pass@1` scores between the original HumanEval benchmark and HUMANEVALNEXT.

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.

VI. DISCUSSION

A. Implications

Our examination of the current AI4SE benchmarking environment leads us to a series of standardized insights about the domain, which we outline below.

1) **Limited Programming Language Variety:** There is a strong tendency for new benchmarks to follow the norms of the field which revolve around well-known languages like **Python**, **Java**, or **C++**; With this, more *niche* languages such as **Haskell**, **Visual Basic**, and **Julia**. Although projects like the MultiPL-E [6] series of automatically translated benchmarks help reduce this disparity, a lack of benchmarks focused on native low-resource languages remains.

2) **Limited Natural Language Variety:** Similar to programming languages, most benchmarks focus mainly on English, with Chinese being the next most common. Although resources like MicrosoftDocs [173] include languages like Norwegian, Danish, and Latvian, evaluation datasets still lack sufficient natural language diversity, restricting the insights into how these models perform in different languages.

3) **Limited Task Diversity:** Over the years, tasks like Text2Code have been subjected to numerous evaluation benchmarks, whereas tasks like domain-specific code generation and memorization have been largely overlooked; This, highlights an imbalance in benchmark availability.

4) **Real-World Applicability Issues:** Benchmarks traditionally centered on isolated tasks such as function-level

code generation, which limited the practicality of their results in real-world scenarios. Recently, this has shifted towards more comprehensive software engineering and project-level tasks to enhance the real-world relevance of benchmark results.

With tools such as BenchScout, users can select more relevant benchmarks and gain deeper insights into model performance for their specific needs. Furthermore, 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 highlights the issue of data leakage in current benchmarks, it remains true that HumanEvalNext would also be affected by this issue. Nonetheless, we argue that benchmarks should not be employed indefinitely and ought to evolve to pose increasing challenges in alignment with model improvements.

B. 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. Additionally, given that BENCHFRAME is a generalizable framework, future work would focus on applying rigorous peer review to other common benchmarks such as the MBPP benchmark family. While evaluating ten LLMs provides valuable insights, 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.

C. 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. All participants received the same tool, guidance, and instructions. To reduce bias in the peer review, 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. While we evaluated the effects of the BENCHFRAME framework on the performance of ten models with one of the most used benchmarks, this might not be seen as 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. Nevertheless, considering its demonstrated potential and the discrepancies in outcomes relative to automated approaches, these efforts can be deemed worthwhile.

VII. 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.

A. Data Avalaibilty

We publicly release the results of our literature review, user study, and 50% of the manually refined benchmark to demonstrate the quality of the data. [19] Upon acceptance, the complete benchmark will be made available on both GitHub and HuggingFace. Because of space constraints, more detailed tables from our review are available in the appendix and also to be found in the replication package.

REFERENCES

[1] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," 2021.
[2] G. Gemini Team, "Gemini: A family of highly capable multimodal models," 2023.
[3] OpenAI, "Gpt-4 technical report," 2023.
[4] T. van Dam, F. van der Heijden, P. de Bekker, B. Nieuwschepen, M. Otten, and M. Izadi, "Investigating the performance of language models for completing code in functional programming languages: a haskell case study," 2024.

[5] B. Athiwaratkun, S. K. Gouda, Z. Wang, X. Li, Y. Tian, M. Tan, W. U. Ahmad, S. Wang, Q. Sun, M. Shang, S. K. Gonugondla, H. Ding, V. Kumar, N. Fulton, A. Farahani, S. Jain, R. Giaquinto, H. Qian, M. K. Ramanathan, R. Nallapati, B. Ray, P. Bhatia, S. Sengupta, D. Roth, and B. Xiang, "Multi-lingual evaluation of code generation models," 2022.

[6] F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C. J. Anderson, M. Q. Feldman, A. Guha, M. Greenberg, and A. Jangda, "Multipl-e: a scalable and extensible approach to benchmarking neural code generation," 2022.

[7] N. Muennighoff, Q. Liu, A. Zebaze, Q. Zheng, B. Hui, T. Y. Zhuo, S. Singh, X. Tang, L. von Werra, and S. Longpre, "Octopack: Instruction tuning code large language models," 2023.

[8] Q. Zheng, X. Xia, X. Zou, Y. Dong, S. Wang, Y. Xue, Z. Wang, L. Shen, A. Wang, Y. Li, T. Su, Z. Yang, and J. Tang, "Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x," 2023.

[9] Y. Dong, J. Ding, X. Jiang, G. Li, Z. Li, and Z. Jin, "Codescore: Evaluating code generation by learning code execution," 2023.

[10] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," 2023.

[11] I. CODAIT, "Ai for code: Predict code complexity using ibm's codenet dataset," 2021.

[12] Q. Peng, Y. Chai, and X. Li, "HumanEval-XL: A Multilingual Code Generation Benchmark for Cross-lingual Natural Language Generalization," Mar. 2024.

[13] C. S. Xia, Y. Deng, and L. Zhang, "Top leaderboard ranking = top coding proficiency, always? evoeval: Evolving coding benchmarks via llm," 2024.

[14] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton, "Program synthesis with large language models," 2021.

[15] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. De Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. De Freitas, K. Kavukcuoglu, and O. Vinyals, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.

[16] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, and J. Steinhardt, "Measuring coding challenge competence with apps," 2021.

[17] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica, "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code," June 2024.

[18] H. Tian, W. Lu, T. O. Li, X. Tang, S.-C. Cheung, J. Klein, and T. F. Bissyandé, "Is ChatGPT the Ultimate Programming Assistant – How far is it?," Aug. 2023.

[19] S. Quan, J. Yang, B. Yu, B. Zheng, D. Liu, A. Yang, X. Ren, B. Gao, Y. Miao, Y. Feng, Z. Wang, J. Yang, Z. Cui, Y. Fan, Y. Zhang, B. Hui, and J. Lin, "CodeElo: Benchmarking Competition-level Code Generation of LLMs with Human-comparable Elo Ratings," Jan. 2025.

[20] J. Sikka, K. Satya, Y. Kumar, S. Uppal, R. R. Shah, and R. Zimmermann, "Learning based methods for code runtime complexity prediction," 2019.

[21] K. Moudgalya, A. Ramakrishnan, V. Chemudupati, and X. H. Lu, "Tasty: A transformer based approach to space and time complexity," 2023.

[22] S.-Y. Baik, M. Jeon, J. Hahn, J. Kim, Y.-S. Han, and S.-K. Ko, "Codecomplex: A time-complexity dataset for bilingual source codes," 2024.

[23] A. Yadav and M. Singh, "Pythonsaga: Redefining the benchmark to evaluate code generating llm," 2024.

[24] D. Huang, J. M. Zhang, Y. Qing, and H. Cui, "Effibench: Benchmarking the efficiency of automatically generated code," 2024.

[25] M. Weyssow, A. Kamanda, and H. Sahraoui, "Codeultrafeedback: An llm-as-a-judge dataset for aligning large language models to coding preferences," *arXiv.org*, 2024.

[26] A. Shypula, A. Madaan, Y. Zeng, U. Alon, J. Gardner, M. Hashemi, G. Neubig, P. Ranganathan, O. Bastani, and A. Yazdanbakhsh, "Learning Performance-Improving Code Edits," Apr. 2024.

[27] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, S. W.-T. Yih, D. Fried, S. Wang, and Y. Chen, "Ds-1000: A natural and reliable benchmark for data science code generation," *arXiv (Cornell University)*, 2022.

[28] D. Zan, B. Chen, D. Yang, Z. Lin, M. Kim, B. Guan, Y. Wang, W. Chen, and J.-G. Lou, "Cert: Continual pre-training on sketches for library-oriented code generation," 2022.

[29] R. Agashe, S. Iyer, and L. Zettlemoyer, "Juice: 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)*, (China), pp. 5436–5446, Association for Computational Linguistics, 2019.

[30] S. Chandel, C. B. Clement, G. Serrato, and N. Sundaresan, "Training and evaluating a jupyter notebook data science assistant," 2022.

[31] J. Huang, J. Huang, C. Wang, C. Wang, J. Zhang, J. Zhang, C. Yan, C. Ye, H. Cui, H. Cui, J. P. Inala, J. P. Inala, C. Clement, C. I. Clement, N. Duan, N. Duan, J. Gao, and J. Gao, "Execution-based evaluation for data science code generation models," *Cornell University - arXiv*, 2022.

[32] Y. Zhang, Q. Jiang, X. Han, N. Chen, Y. Yang, and K. Ren, "Benchmarking Data Science Agents," Feb. 2024.

[33] X. Tang, B. Qian, R. Gao, J. Chen, X. Chen, and M. B. Gerstein, "Biocoder: a benchmark for bioinformatics code generation with large language models," *Bioinformatics*, 2024.

[34] Y. Cui, "Webapp1k: A practical code-generation benchmark for web app development," 2024.

[35] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, and J. Steinhardt, "Measuring coding challenge competence with apps," *NeurIPS Datasets and Benchmarks*, 2021.

[36] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, "Let's Verify Step by Step," May 2023.

[37] A. Amini, S. Gabriel, P. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi, "Mathqa: Towards interpretable math word problem solving with operation-based formalisms," 2019.

[38] S. Mishra, M. Finlayson, P. Lu, L. Tang, S. Welleck, C. Baral, T. Rajpurohit, O. Tafjord, A. Sabharwal, P. Clark, and A. Kalyan, "Lila: A unified benchmark for mathematical reasoning," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, (Abu Dhabi, United Arab Emirates), pp. 5807–5832, Association for Computational Linguistics, 2022.

[39] S. Roy and D. Roth, "Solving general arithmetic word problems," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (Lisbon, Portugal), pp. 1743–1752, Association for Computational Linguistics, 2015.

[40] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "Pal: Program-aided language models," 2022.

[41] W. Chen, M. Yin, M. Ku, P. Lu, Y. Wan, X. Ma, J. Xu, X. Wang, and T. Xia, "Theoremqa: A theorem-driven question answering dataset," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, (Singapore), pp. 7889–7901, Association for Computational Linguistics, 2023.

[42] P. Haller, J. Golde, and A. Akbik, "Pecc: Problem extraction and coding challenges," *International Conference on Language Resources and Evaluation*, 2024.

[43] H. Su, H. Yen, M. Xia, W. Shi, N. Muennighoff, H. yu Wang, H. Liu, Q. Shi, Z. S. Siegel, M. Tang, R. Sun, J. Yoon, S. o. Arik, D. Chen, and T. Yu, "Bright: A realistic and challenging benchmark for reasoning-intensive retrieval," 2024.

[44] P. Yin, B. Deng, E. Chen, B. Vasilescu, and G. Neubig, "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*, (NY, USA), pp. 476–486, Association for Computing Machinery, 2018.

[45] Z. Wang, G. Cuenca, S. Zhou, F. F. Xu, and G. Neubig, "Mconala: A benchmark for code generation from multiple natural languages," 2022.

[46] G. Orlanski and A. Gittens, "Reading StackOverflow Encourages Cheating: Adding Question Text Improves Extractive Code Generation," June 2021.

[47] Y. Hao, G. Li, Y. Liu, X. Miao, H. Zong, S. Jiang, Y. Liu, and H. Wei, "Aixbench: a code generation benchmark dataset," 2022.

[48] J. Huang, D. Tang, L. Shou, M. Gong, K. Xu, D. Jiang, M. Zhou, and N. Duan, "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*, (Online), pp. 5690–5700, ACL, 2021.

[49] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou,

L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "Codexglue: a machine learning benchmark dataset for code understanding and generation," 2021.

[50] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Mapping language to code in programmatic context," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2018.

[51] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," 2023.

[52] S. Zhang, J. Wang, G. Dong, J. Sun, Y. Zhang, and G. Pu, "Experimenting a new programming practice with llms," *arXiv.org*, 2024.

[53] P. Liguori, P. Liguori, E. Al-Hossami, E. Al-Hossami, D. Cotroneo, D. Cotroneo, R. Natella, R. Natella, B. Cukic, B. Čukić, S. Shaikh, and S. Shaikh, "Can we generate shellcodes via natural language? an empirical study," *Automated software engineering*, vol. 29, no. 1, 2022.

[54] T. Helmuth, T. Helmuth, P. Kelly, and P. Kelly, "Psb2: the second program synthesis benchmark suite," *Annual Conference on Genetic and Evolutionary Computation*, pp. 785–794, 2021.

[55] R. Li, J. Fu, B.-W. Zhang, T. Huang, Z. Sun, C. Lyu, G. Liu, Z. Jin, G. L. B. A. O. A. Intelligence, S. O. M. Science, Engineering, S. University, China, K. L. O. Hcst, Moe, Scs, and P. University, "Taco: Topics in algorithmic code generation dataset," *arXiv.org*, 2023.

[56] S. Honarvar, M. V. D. Wilk, and A. Donaldson, "Turbulence: Systematically and automatically testing instruction-tuned large language models for code," *arXiv.org*, 2023.

[57] J. Shin, M. Wei, J. Wang, L. Shi, and S. Wang, "The Good, the Bad, and the Missing: Neural Code Generation for Machine Learning Tasks," *ACM Transactions on Software Engineering and Methodology*, vol. 33, pp. 1–24, Feb. 2024.

[58] J. Chen, Q. Zhong, Y. Wang, K. Ning, Y. Liu, Z. Xu, Z. Zhao, T. Chen, and Z. Zheng, "RMCBench: Benchmarking Large Language Models' Resistance to Malicious Code," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 995–1006, Oct. 2024.

[59] P. Liguori, E. Al-Hossami, V. Orbinato, R. Natella, S. Shaikh, D. Cotroneo, and B. Cukic, "EVIL: Exploiting Software via Natural Language," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pp. 321–332, Oct. 2021.

[60] InfiCoder, "Inficoder-eval: Systematically evaluating question-answering for code large language models," 2023.

[61] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, (NY, USA), pp. 200–210, Association for Computing Machinery, 2018.

[62] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation with hybrid lexical and syntactical information," *Empirical Software Engineering*, vol. 25, pp. 2179–2217, May 2020.

[63] X. Jin, J. Larson, W. Yang, and Z. Lin, "Binary code summarization: Benchmarking chatgpt/gpt-4 and other large language models," 2023.

[64] M. Allamanis, M. Allamanis, H. Peng, H. Peng, C. Sutton, and C. Sutton, "A convolutional attention network for extreme summarization of source code," *arXiv: Learning*, 2016.

[65] A. LeClair, A. LeClair, S. Jiang, S. Jiang, C. McMillan, and C. McMillan, "A neural model for generating natural language summaries of program subroutines," *International Conference on Software Engineering*, pp. 795–806, 2019.

[66] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, and Z. Jin, "Summarizing Source Code with Transferred API Knowledge,"

[67] M. Hasan, T. Muttaqueen, A. A. Ishtiaq, K. S. Mehrab, M. M. A. Haque, T. Hasan, W. U. Ahmad, A. Iqbal, and R. Shahriyar, "CoDesc: A Large Code-Description Parallel Dataset," May 2021.

[68] A. V. M. Barone and R. Sennrich, "A Parallel Corpus of Python Functions and Documentation Strings for Automated Code Documentation and Code Generation,"

[69] K. Pai, P. Devanbu, and T. Ahmed, "Codocbench: A dataset for code-documentation alignment in software maintenance," 2025.

[70] J. Li, B. Hui, G. Qu, B. Li, J. Yang, B. Li, B. Wang, B. Qin, R. Cao, R. Geng, H. Nan, C. Ma, K. C.-C. Chang, F. Huang, R. Cheng, and Y. Li, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," *Neural Information Processing Systems*, 2023.

[71] C.-H. Lee, O. Polozov, and M. Richardson, "Kaggledbqa: Realistic evaluation of text-to-sql parsers," *Annual Meeting of the Association for Computational Linguistics*, 2021.

[72] Z. Yao, Z. Yao, D. S. Weld, D. Weld, W.-P. Chen, W.-P. Chen, H. Sun, and H. Sun, "Staqc: A systematically mined question-code dataset from stack overflow," *The Web Conference*, 2018.

[73] F. Lei, J. Chen, Y. Ye, R. Cao, D. Shin, H. Su, Z. Suo, H. Gao, W. Hu, P. Yin, V. Zhong, C. Xiong, R. Sun, Q. Liu, S. Wang, and T. Yu, "Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows," 2024.

[74] Y. Gan, X. Chen, Q. Huang, M. Purver, J. R. Woodward, J. Xie, and P. Huang, "Towards Robustness of Text-to-SQL Models against Synonym Substitution," June 2021.

[75] X. Deng, A. H. Awadallah, C. Meek, O. Polozov, H. Sun, and M. Richardson, "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*, pp. 1337–1350, 2021.

[76] Y. Gan, X. Chen, and M. Purver, "Exploring Underexplored Limitations of Cross-Domain Text-to-SQL Generalization," Sept. 2021.

[77] Q. Min, Y. Shi, and Y. Zhang, "A Pilot Study for Chinese SQL Semantic Parsing," Oct. 2019.

[78] T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. V. Lin, S. Li, H. Er, I. Li, B. Pang, T. Chen, E. Ji, S. Dixit, D. Proctor, S. Shim, J. Kraft, V. Zhang, C. Xiong, R. Socher, and D. Radev, "SParC: Cross-Domain Semantic Parsing in Context," June 2019.

[79] Q. Liang, Z. Sun, Q. Zhu, W. Zhang, L. Yu, Y. Xiong, and L. Zhang, "Lyra: A Benchmark for Turducken-Style Code Generation," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 4238–4244, July 2022.

[80] L. Wang, A. Zhang, K. Wu, K. Sun, Z. Li, H. Wu, M. Zhang, and H. Wang, "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)*, (Online), pp. 6923–6935, Association for Computational Linguistics, 2020.

[81] T. Yu, R. Zhang, H. Y. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li, Y. Jiang, M. Yasunaga, S. Shim, T. Chen, A. Fabbri, Z. Li, L. Chen, Y. Zhang, S. Dixit, V. Zhang, C. Xiong, R. Socher, W. S. Lasecki, and D. Radev, "CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases," Sept. 2019.

[82] K. L. Aw, S. Montariol, B. AlKhamissi, M. Schrimpf, and A. Bosselut, "Instruction-tuning aligns llms to the human brain," 2023.

[83] B. Roziere, J. M. Zhang, F. Charton, M. Harman, G. Synnaeve, and G. Lample, "Leveraging automated unit tests for unsupervised code translation," 2022.

[84] M. Zhu, K. Suresh, and C. K. Reddy, "Multilingual code snippets training for program translation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, pp. 11783–11790, 2022.

[85] W. U. Ahmad, M. G. R. Tushar, S. Chakraborty, and K.-W. Chang, "Avatar: A parallel corpus for java-python program translation," in *Findings of the Association for Computational Linguistics*, (Toronto, Canada), pp. 2268–2281, Association for Computational Linguistics, 2023.

[86] W. Yan, Y. Tian, Y. Li, Q. Chen, and W. Wang, "Codetransocean: A comprehensive multilingual benchmark for code translation," 2023.

[87] M. Jiao, T. Yu, X. Li, G. Qiu, X. Gu, and B. Shen, "On the Evaluation of Neural Code Translation: Taxonomy and Benchmark," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, (Luxembourg, Luxembourg), pp. 1529–1541, IEEE, Sept. 2023.

[88] J. Zhang, P. Nie, J. J. Li, and M. Gligoric, "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*, (San Francisco CA USA), pp. 695–707, ACM, Nov. 2023.

[89] I. Paul, G. Glavaš, and I. Gurevych, "IRCoder: Intermediate Representations Make Language Models Robust Multilingual Code Generators," Apr. 2024.

[90] B. Li, W. Wu, Z. Tang, L. Shi, J. Yang, J. Li, S. Yao, C. Qian, B. Hui, Q. Zhang, Z. Yu, H. Du, P. Yang, D. Lin, C. Peng, and K. Chen, "Devbench: A comprehensive benchmark for software development," 2024.

[91] J. Li, G. Li, Y. Zhao, Y. Li, H. Liu, H. Zhu, L. Wang, K. Liu, Z. Fang, L. Wang, J. Ding, X. Zhang, Y. Zhu, Y. Dong, Z. Jin, B. Li, F. Huang, and Y. Li, "Deveval: A manually-annotated code generation benchmark aligned with real-world code repositories," 2024.

[92] Z. Zeng, Y. Wang, R. Xie, W. Ye, and S. Zhang, "Coderujb: An executable and unified java benchmark for practical programming scenarios," 2024.

[93] Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang, "ToolQA: A Dataset for LLM Question Answering with External Tools," June 2023.

[94] X. Wang, Z. Wang, J. Liu, Y. Chen, L. Yuan, H. Peng, and H. Ji, "MINT: Evaluating LLMs in Multi-turn Interaction with Tools and Language Feedback," Mar. 2024.

[95] L. Gong, S. Wang, M. Elhoushi, and A. Cheung, "Evaluation of LLMs on Syntax-Aware Code Fill-in-the-Middle Tasks," June 2024.

[96] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, S. Zhang, X. Deng, A. Zeng, Z. Du, C. Zhang, S. Shen, T. Zhang, Y. Su, H. Sun, M. Huang, Y. Dong, and J. Tang, "AgentBench: Evaluating LLMs as Agents," Oct. 2023.

[97] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, and Y. Lou, "Classeval: a manually-crafted benchmark for evaluating llms on class-level code generation," 2023.

[98] T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul, S. Brunner, C. Gong, T. Hoang, A. R. Zebaze, X. Hong, W.-D. Li, J. Kaddour, M. Xu, Z. Zhang, P. Yadav, N. Jain, A. Gu, Z. Cheng, J. Liu, Q. Liu, Z. Wang, D. Lo, B. Hui, N. Muennighoff, D. Fried, X. Du, H. de Vries, and L. V. Werra, "Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions," 2024.

[99] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "Swe-bench: Can language models resolve real-world github issues?," *arXiv (Cornell University)*, 2023.

[100] Y. Ding, Z. Wang, W. U. Ahmad, H. Ding, M. Tan, N. Jain, M. K. Ramanathan, R. Nallapati, P. Bhatia, D. Roth, and B. Xiang, "Cross-codeeval: a diverse and multilingual benchmark for cross-file code completion," 2023.

[101] H. Yu, B. Shen, D. Ran, J. Zhang, Q. Zhang, Y. Ma, G. Y. Liang, Y. Liu, T. Xie, and Q. Wang, "Codereval: A benchmark of pragmatic code generation with generative pre-trained models," *arXiv (Cornell University)*, 2023.

[102] L. A. Agrawal, A. Kanade, N. Goyal, S. K. Lahiri, and S. K. Rajamani, "Guiding Language Models of Code with Global Context using Monitors," Nov. 2023.

[103] J. Svajlenko and C. K. Roy, "Evaluating clone detection tools with BigCloneBench," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, (Bremen, Germany), pp. 131–140, IEEE, Sept. 2015.

[104] L. Zhang, J. Wang, S. He, C. Zhang, Y. Kang, B. Li, J. Wen, C. Xie, M. Wang, Y. Huang, E. Nallipogu, Q. Lin, Y. Dang, S. Rajmohan, D. Zhang, and Q. Zhang, "Di-bench: Benchmarking large language models on dependency inference with testable repositories at scale," 2025.

[105] I. Bouzenia, B. P. Krishan, and M. Pradel, "Dypybench: A benchmark of executable python software," *Proceedings of the ACM on Software Engineering*, vol. 1, p. 338–358, July 2024.

[106] T. Liu, C. Xu, and J. McAuley, "Repobench: Benchmarking repository-level code auto-completion systems," 2023.

[107] F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J.-G. Lou, and W. Chen, "Repocoder: Repository-level code completion through iterative retrieval and generation," 2023.

[108] J. Li, G. Li, X. Zhang, Y. Dong, and Z. Jin, "Evocodebench: An evolving code generation benchmark aligned with real-world code repositories," 2024.

[109] D. Zan, A. Yu, W. Liu, D. Chen, B. Shen, W. Li, Y. Yao, Y. Gong, X. Chen, B. Guan, Z. Yang, Y. Wang, Q. Wang, and L. Cui, "Codes: Natural language to code repository via multi-layer sketch," *arXiv.org*, 2024.

[110] X. Tang, Y. Liu, Z. Cai, Y. Shao, J. Lu, Y. Zhang, Z. Deng, H. Hu, K. An, R. Huang, S. Si, S. Chen, H. Zhao, L. Chen, Y. Wang, T. Liu, Z. Jiang, B. Chang, Y. Fang, Y. Qin, W. Zhou, Y. Zhao, A. Cohan, and M. Gerstein, "ML-Bench: Evaluating Large Language Models and Agents for Machine Learning Tasks on Repository-Level Code," Aug. 2024.

[111] M. Liu, T. Yang, Y. Lou, X. Du, Y. Wang, and X. Peng, "Code-Gen4Libs: A Two-Stage Approach for Library-Oriented Code Generation," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, (Luxembourg, Luxembourg), pp. 434–445, IEEE, Sept. 2023.

[112] Y. Song, W. Xiong, D. Zhu, W. Wu, H. Qian, M. Song, H. Huang, C. Li, K. Wang, R. Yao, Y. Tian, and S. Li, "Restgpt: Connecting large language models with real-world restful apis," 2023.

[113] Y. Peng, S. Li, W. Gu, Y. Li, W. Wang, C. Gao, and M. Lyu, "Revisiting, benchmarking and exploring api recommendation: How far are we?," 2021.

[114] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "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, (NY, USA), pp. 293–304, Association for Computing Machinery, 2018.

[115] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive apis," 2023.

[116] M. Li, Y. Zhao, B. Yu, F. Song, H. Li, H. Yu, Z. Li, F. Huang, and Y. Li, "Api-bank: A comprehensive benchmark for tool-augmented llms," 2023.

[117] Z. Z. Wang, A. Asai, X. V. Yu, F. F. Xu, Y. Xie, G. Neubig, and D. Fried, "Coderag-bench: Can retrieval augment code generation?," 2024.

[118] N. Rao, C. Bansal, and J. Guan, "Search4Code: Code Search Intent Classification Using Weak Supervision," Mar. 2021.

[119] X. Li, K. Dong, Y. Q. Lee, W. Xia, Y. Yin, H. Zhang, Y. Liu, Y. Wang, and R. Tang, "CoIR: A Comprehensive Benchmark for Code Information Retrieval Models," July 2024.

[120] A. Al-Kaswan, M. Izadi, and A. Van Deursen, "Traces of Memorisation in Large Language Models for Code," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, (Lisbon Portugal), pp. 1–12, ACM, Apr. 2024.

[121] S. Kulal, P. Pasupat, K. Chandra, M. Lee, O. Padon, A. Aiken, and P. S. Liang, "Spoc: Search-based pseudocode to code," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[122] M. Zavershynskyi, A. Skidanov, and I. Polosukhin, "Naps: Natural program synthesis dataset," 2018.

[123] Y. Oda, H. Fudaba, G. Neubig, H. Hata, S. Sakti, T. Toda, and S. Naka-mura, "Learning to generate pseudo-code from source code using statistical machine translation," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 574–584, 2015.

[124] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," 2017.

[125] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," 2018.

[126] X. V. Lin, X. V. Lin, C. Wang, C. Wang, L. Zettlemoyer, L. Zettle-moyer, M. Ernst, and M. D. Ernst, "Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system," *arXiv: Computation and Language*, 2018.

[127] R. Just, D. Jalali, and M. D. Ernst, "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, (NY, USA), pp. 437–440, Association for Computing Machinery, 2014.

[128] A. Silva, N. Saavedra, and M. Monperrus, "Gitbug-java: A reproducible benchmark of recent java bugs," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, pp. 118–122, 2024.

[129] Q. Zhang, T. Zhang, J. Zhai, C. Fang, B. Yu, W. Sun, and Z. Chen, "A critical review of large language model on software engineering: An example from chatgpt and automated program repair," *arXiv.org*, 2023.

[130] B. Yang, H. Tian, W. Pian, H. Yu, H. Wang, J. Klein, T. F. Bissyand'e, and S. Jin, "Cref: An llm-based conversational software repair framework for programming tutors," *arXiv.org*, 2024.

[131] J. Y. Lee, S. Kang, J. Yoon, and S. Yoo, "The github recent bugs dataset for evaluating llm-based debugging applications," *arXiv.org*, 2023.

[132] C. L. Goues, C. Le Goues, N. J. Holtschulte, N. J. Holtschulte, E. K. M. Smith, E. K. Smith, Y. Brun, Y. Brun, P. Devanbu, P. Devanbu, S. Forrest, S. Forrest, W. Weimer, and W. Weimer, "The manybugs and introclass benchmarks for automated repair of c programs," *IEEE Transactions on Software Engineering*, vol. 41, no. 12, pp. 1236–1256, 2015.

[133] R. Tian, Y. Ye, Y. Qin, X. Cong, Y. Lin, Z. Liu, and M. Sun, "Debugbench: Evaluating debugging capability of large language models," *arXiv.org*, 2024.

[134] D. Lin, D. Lin, J. Koppel, J. Koppel, A. Chen, A. Chen, A. Solar-Lezama, and A. Solar-Lezama, "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*, pp. 55–56, 2017.

[135] B. Labash, A. Rosedale, A. Reents, L. Negritto, and C. Wiel, "Res-q: Evaluating code-editing large language model systems at the repository scale," *arXiv.org*, 2024.

[136] H. M. Babe, S. Nguyen, Y. Zi, A. Guha, M. Q. Feldman, and C. J. Anderson, "Studenteval: A benchmark of student-written prompts for large language models of code," *arXiv.org*, 2023.

[137] Y. Hu, U. Z. Ahmed, S. Mechtaev, B. Leong, and A. Roychoudhury, "Re-factoring based program repair applied to programming assignments," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 388–398, 2019.

[138] Y. Wu, Z. Li, J. M. Zhang, and Y. Liu, "ConDefects: A New Dataset to Address the Data Leakage Concern for LLM-based Fault Localization and Program Repair," Oct. 2023.

[139] R. Shariffdeen, M. Mirchev, Y. Noller, and A. Roychoudhury, "Cerberus: a Program Repair Framework," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, (Melbourne, Australia), pp. 73–77, IEEE, May 2023.

[140] G. P. Bhandari, P. Bhandari, A. Naseer, A. Naseer, L. Moonen, and L. Moonen, "Cvefixes: Automated collection of vulnerabilities and their fixes from open-source software.," *arXiv: Software Engineering*, 2021.

[141] C. Tony, M. Mutas, N. E. D. Ferreyra, and R. Scandariato, "Llmseceval: A dataset of natural language prompts for security evaluations," *IEEE Working Conference on Mining Software Repositories*, 2023.

[142] M. L. Siddiq, M. L. Siddiq, J. C. S. Santos, and J. C. S. Santos, "Securityeval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques," 2022.

[143] Q.-C. Bui, R. Scandariato, and N. E. D. Ferreyra, "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*, (Pittsburgh Pennsylvania), pp. 464–468, ACM, May 2022.

[144] N. Tihanyi, T. Bisztray, R. Jain, M. A. Ferrag, L. C. Cordeiro, and V. Mavroeidis, "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*, (San Francisco CA USA), pp. 33–43, ACM, Dec. 2023.

[145] Y. Wu, N. Jiang, H. V. Pham, T. Lutellier, J. Davis, L. Tan, P. Babkin, and S. Shah, "How Effective Are Neural Networks for Fixing Security Vulnerabilities," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1282–1294, July 2023.

[146] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 530–541, June 2020.

[147] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks," Sept. 2019.

[148] Y. Zheng, S. Pujar, B. Lewis, L. Buratti, E. Epstein, B. Yang, J. Laredo, A. Morari, and Z. Su, "D2A: A Dataset Built for AI-Based Vulnerability Detection Methods Using Differential Analysis," Feb. 2021.

[149] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "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, (New York, NY, USA), p. 508–512, Association for Computing Machinery, 2020.

[150] M. Tufano, S. Chandel, A. Agarwal, N. Sundaresan, and C. B. Clement, "Predicting code coverage without execution," 2023.

[151] C. Watson, M. Tufano, K. Moran, G. Bavota, and D. Poshyvanyk, "On Learning Meaningful Assert Statements for Unit Test Cases," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 1398–1409, June 2020.

[152] Z. Wang, K. Liu, G. Li, and Z. Jin, "HITS: High-coverage LLM-based Unit Test Generation via Method Slicing," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, (Sacramento CA USA), pp. 1258–1268, ACM, Oct. 2024.

[153] P. Bareiß, B. Souza, M. d'Amorim, and M. Pradel, "Code Generation Tools (Almost) for Free? A Study of Few-Shot, Pre-Trained Language Models on Code," June 2022.

[154] C. Wan, S. Liu, S. Xie, Y. Liu, H. Hoffmann, M. Maire, and S. Lu, "Automated testing of software that uses machine learning APIs," in *Proceedings of the 44th International Conference on Software Engineering*, (Pittsburgh Pennsylvania), pp. 212–224, ACM, May 2022.

[155] M. Tufano, D. Drain, A. Svyatkovskiy, S. K. Deng, and N. Sundaresan, "Unit test case generation with transformers and focal context," 2020.

[156] A. Gu, B. Roziere, H. Leather, A. Solar-Lezama, G. Synnaeve, and S. I. Wang, "Cruxeval: A benchmark for code reasoning, understanding and execution," 2024.

[157] E. Dinella, S. Chandra, and P. Maniatis, "Crqbench: A benchmark of code reasoning questions," 2024.

[158] Z. Lin, Z. Gou, T. Liang, R. Luo, H. Liu, and Y. Yang, "CriticBench: Benchmarking LLMs for Critique-Correct Reasoning," June 2024.

[159] W. Yan, H. Liu, Y. Wang, Y. Li, Q. Chen, W. Wang, T. Lin, W. Zhao, L. Zhu, H. Sundaram, and S. Deng, "CodeScope: An Execution-based Multilingual Multitask Multidimensional Benchmark for Evaluating LLMs on Code Understanding and Generation," June 2024.

[160] B. Shen and N. Meng, "Conflictbench: A benchmark to evaluate software merge tools," *Journal of Systems and Software*, vol. 214, p. 112084, 2024.

[161] A. P. S. Venkatesh, S. Sabu, J. Wang, A. M. Mir, L. Li, and E. Bodden, "Typeevalpy: A micro-benchmarking framework for python type inference tools," *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2023.

[162] Z. Li, Z. Li, S. Lu, S. Lu, D. Guo, D. Guo, N. Duan, N. Duan, S. Jannu, S. Jannu, G. Jenks, G. Jenks, D. Majumder, D. Majumder, J. Green, J. Green, A. Svyatkovskiy, A. Svyatkovskiy, S. Fu, S.-Y. Fu, N. Sundaresan, and N. Sundaresan, "Automating code review activities by large-scale pre-training," *ESEC/SIGSOFT FSE*, 2022.

[163] M. Schnappinger, M. Schnappinger, A. Fietzke, A. Fietzke, A. Pretschner, and A. Pretschner, "Defining a software maintainability dataset: Collecting, aggregating and analysing expert evaluations of software maintainability," *IEEE International Conference on Software Maintenance and Evolution*, pp. 278–289, 2020.

[164] F. Liu, Y. Liu, L. Shi, H. Huang, R. Wang, Z. Yang, and L. Zhang, "Exploring and evaluating hallucinations in llm-powered code generation," *arXiv.org*, 2024.

[165] M. L. Siddiq, J. C. S. Santos, R. H. Tanvir, N. Ulfat, F. A. Rifat, and V. C. Lopes, "Using large language models to generate junit tests: An empirical study," 2024.

[166] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, and et al., "Beyond the imitation game: Quantifying and extrapolating the capabilities of language models," 2022.

[167] M. Zhu, A. Jain, K. Suresh, R. Ravindran, S. Tipirneni, and C. K. Reddy, "Xlcost: A benchmark dataset for cross-lingual code intelligence," 2022.

[168] C. Niu, C. Li, V. Ng, and B. Luo, "Crosscodebench: Benchmarking cross-task generalization of source code models," *International Conference on Software Engineering*, 2023.

[169] E. Bogomolov, A. Eliseeva, T. Galimzyanov, E. Glukhov, A. Shapkin, M. Tigina, Y. Golubev, A. Kovrigin, A. van Deursen, M. Izadi, and T. Bryksin, "Long code arena: a set of benchmarks for long-context code models," 2024.

[170] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," 2019.

[171] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2019.

[172] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[173] Microsoft, "Microsoftdocs," 2024.

APPENDIX

TABLE XVIII: Overview of AI4SE benchmarks stemming from HumanEval [1].

Category	Name	Language(s)	# Tests
Original	HumanEval [1]	Python	Avg. 7.7
Improved Language Support	MultiPL-HumanEval [6]	18 programming languages	Avg. 7.7
	HumanEval-Fix [7]	6 programming languages	Avg. 7.7
	HumanEval-Explain [7]	6 programming languages	Avg. 7.7
	HumanEval-Synthesize [7]	6 programming languages	Avg. 7.7
	HumanEval-X [8]	5 programming languages	Avg. 7.7
	Multi-HumanEval [5]	12 programming languages	Avg. 7.7
	HumanEvalXL [12]	12PLs, 23NLs	Avg. 8.33
Improved Testing	HumanEval+ [10]	Python	Scaled ×80
	HumanEval-MINI [10]	Python	Scaled ×47
	HE-Eval [9]	Python	Scaled ×14
Instruction-based	InstructHumanEval [20]	Python	Avg. 7.7
Extended	EvoEval [13]	Python	Multiple categories, scaled with EVALPLUS

TABLE XIX: Overview of AI4SE benchmarks derived from MBPP [14].

Category	Name	Language(s)	# Problems
Original	MBPP [14]	Python	974
Improved Language Support	MultiPL-MBPP [6]	18 programming languages	354-397 per language
	MBXP [5]	13 programming languages	848-974 per language
Improved Testing	MBPP+ [10]	Python	427
	MBPP-Eval [9]	Python	974

TABLE XX: Overview of competitive programming, code complexity, and code efficiency benchmarks.

Category	Name	Language(s)	# Tests
Competitive Programming	CodeContests [15]	12 programming languages	Avg. 203.7
	APPS [16]	Python	Avg. 13.2
	LiveCodeBench [17]	Python	Avg. 17.23
	LeetCode [18]	Python	Avg. 135
	CodeElo [19]	N/A	408 problems
Code Complexity	CoRCoD [20]	Java	932
	GeeksForGeeks (GFG) [21]	C++, Python	±1,400 per lang.&categ
	CODAIT [21]	Python	4,000,000
	CodeComplex [22]	Java, Python	4,900 per language
	PythonSaga [23]	Python	185
Code Efficiency	EffiBench [24]	Python	Self-defined, avg. 100
	CODAL [25]	Python	3 ref. / problem
	PIE [26]	C++	82.5(median, train)

TABLE XXI: Overview of data science & domain-specific benchmarks. (JN refers to Jupyter Notebooks.)

Name	Language(s)	# Tests	Comment
DS-1000 [27]	Python	Avg. 1.6	7 DS/ML libraries
NumpyEval [28]	Python	Avg. 20 functions	NumPy (101 problems) (Avg. 1 variable)
PandasEval [28]	Python	Avg. 20 functions	Pandas (101 problems) (Avg. 1 variable)
JuICe [29]	Python, JN	N/A	Cell completion (1.5M/3.7K train test)
DSP [30]	Python, JN	Available	Cell completion (1,119 problems)
ExeDS [31]	Python, JN	Execution Based	Cell generation (ground truth), 534 tasks
DSEval [32]	Python	custom appraoch	Models Evaluated via the DSEval Approach from the Paper
Bio-Coder [33]	Python	1,026	Identify and import necessary classes for given task
	Java	1,243	
Bio-Coder-Rosalind [33]	Python	253 golden solution	Generate code for question
WebApp1k [34]	React	Available	evaluates whether a model can generate React web-app

TABLE XXII: Overview of Mathematical Reasoning Benchmarks.

Name	Language(s)	# Problems
MATH [35]	English	12,500
MATH500 [36]	English	500
MathQA [37]	English	37,297
MathQA-Python [14]	Python	23,914
MathQA-X [5]	Python, Java, JS	1,883 per language
LĪLA[38]	Python	133,815 questions, 358,769 programs
MultiArith [39]	English	600
GSM8K [40]	English	1,320
GSM-HARD [40]	English	1,320
TheoremQA [41]	English	800
PECC [42]	Python	1,006
BRIGHT [43]	English	395
AMC12[22]	English	82

TABLE XXIII: Overview of Natural Language Benchmarks.

Category	Name	Language(s)	No. of Problems
	CoNaLa [44]	English → Python	2,879
	MCoNaLa [45]	{Spanish, Japanese, Russian} → Python	896
	CoNaLa-SO [46]	Englihs → Python	10,000
	APPS [16]	English → Python	10,000
	APPS-Eval [9]	English → Python	10,000
	AixBench [47]	English, Chinese → Java	175
	Natural2Code [2]	English → Python	Unknown
	CoSQA [48]	English → Python	20,604
	WebQueryTest [49]	English → Python	1,046
	AdvTest [49]	English → Python	280,634
	CONCODE [50]	English → Java	104,000
	MTPB [51]	English → Python	115
	CAASD [52]	English → Python	72
	Shellcode_IA32 [53]	English → IA32/Shell	3200
	Odex [53]	{Spanish, Japanese, Russian, English} → Python	945 {90, 164, 252, 439} 1707 test total
Text2Code	PSB2 [54]	English → {Clojure, Python}	25 question-answer pairs
	TACO [55]	English → Python	1,539,152 on 26,433 distinct tasks
	Turbulence [56]	English → Python	60 (with 420 total test cases)
	Aider [23]	English → {C++, GO, Java, JS, Python, Rust}	225 problems
	NL2ML-lib [57]	English → Python (ML libraries)	11,000
	RMCBench [58]	English → 9 Languages	473 malicious prompts
	Evil [59]	English → {Python, IA_32}	19255
Text2Text (about code)	InfiCoder-Eval [60]	English → English	270
	BRIGHT [43]	English → English	1,398
	DeepCom [61]	Java → English	588K
	Hybrid-DeepCom [62]	Java → English	466k
	BinSum [63]	Binary functions → English	557K
	Code Attention [64]	Java → English	11 projects
Code2Text	Funcom [65]	Java → English	2.1M problems
	CodeSum [66]	Java → English	410,630
	CoDesc [67]	Java → English	4.21M datapoints
	Parallel [68]	Python → English	150k function/doc pais
	CoDocBench [69]	Python → English	4573 code/doc pairs

TABLE XXIV: Overview of SQL-related Benchmarks.

Category	Name	Language(s)	No. of Problems
	BIRD [70]	English → SQL	12,751
	KaggleDBQA [71]	English → SQL	272, paired with golden solutions
	StacQc [72]	English → {Python/SQL}	{147,546 / 119,519} question-answer pairs
	Spider(V2[24]) [73]	English → SQL	632 queries
Text2Code	Spider-Syn [74]	English → SQL	(7000 / 1034)
	Spider-Real [75]	English → SQL	508
	Spider-DK [76]	English → SQL	535 pairs
	Spider-CN [77]	Chinese → SQL	9691 queries
	SParC [78]	English → SQL	4,298 question sequences
	Lyra [79]	{English, Chinese} → {python, SQL}	2000
	DuSQL [80]	Chinese → SQL	23,797 question/SQL pairs
	CoSQL [81]	English → SQL	3,007 Question Sequencess

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

Category	Name	Language(s)	No. of Samples
Programming Languages	CodeTrans [49]	C#, Java	11,800
	TransCoder-ST [83]	C++, Java, Python	437,030
	CoST [84]	7 programming languages	16,738
	AVATAR [85]	Java, Python	7,133 / 476 / 1,906
	Multilingual-Trans [86]	8 programming languages	30,419 total
	NicheTrans [86]	Various niche languages	236,468 total
	LLMTrans [86]	8 programming languages	350
	G-TransEva [87]	5 programming languages	400 total
	CODEDITOR [88]	C# & Java	6613
Libraries	DLTrans [86]	PyTorch, TensorFlow, MXNet, Paddle	408 total
Intermediate Representation	SLTrans [89]	14 Languages $\rightarrow$ LLVM-IR	4M
Language Conversion Frameworks	MultiPL-E [6]	19 programming languages	-
	MultiEval [5]	13 programming languages	-

TABLE XXVI: 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 [90]	Python, C/C++, Java, JavaScript	22 repositories
	DevEval [91]	Python	1,874
	CoderUJB [92]	Java	2,239
	CODAL [25]	Python	500
	ToolQA [93]	Python, Math, English	800(Easy)/730(Hard)
	MIT [94]	Python, English	586 Problems
	SAFIM [95]	Python, Java, C++, C#	17720
	AgentBench [96]	N/A	1360 prompts
Class Level	ClassEval [97]	Python	100
	CONCODE [50]	English, Java	104,000
	BigCodeBench [98]	Python	1,140
Project & Cross-file	SWE-bench [99]	Python	19,008 (Train), 225 (Dev), 2,294 (Test) 144 (Small)
	CrossCodeEval [100]	C#, TypeScript, Java, Python	2,665 (Python), 2,139 (Java), 3,356 (TypeScript), 1,768 (C#)
	CoderEval [101]	Java, Python	230
	DotPrompts [102]	Java	105538 problems (1420 methods)
	BigCloneBench [103]	Java	25,000 Java Systms
	DI-Bench [104]	Python, C#, Rust, JS	581 repositories (w/ dependencies)
	DyPyBench [105]	Python	50 repositories
Repository Level	RepoBench [106]	Python, Java	Cross-file: 8,033, In-file: 7,910
	RepoEval [107]	Python	1,600 (line), 1,600 (API), 373 (function)
	EvoCodeBench [108]	Python	275
	SketchEval [109]	Python	19 repositories (5 easy, 8 medium, 6 hard)
	Stack Repo [109]	Python	(435,890 / 220,615 / 159,822) answer pairs
	ML-BENCH [110]	Python & Bash	9641 problems
	CodeGen4Libs [111]	Java	403,780 prompts

TABLE XXVII: Overview of Selected API and Retrieval Benchmarks by Category.

Category	Benchmark	Sources/API(s)	No. of Problems
API Prediction	RestBench [112]	Spotify, TMDB	57, 100
	APIBench-Q [113]	StackOverflow, Tutorial Websites	6,563 (Java), 4,309 (Python)
	BIKER [114]	StackOverflow	33,000
	Gorilla APIBench [115]	HuggingFace, TensorHub, TorchHub	925, 696, 94
	Gorilla APIZoo [115]	Open submissions (Google, YouTube, Zoom, etc.)	–
Retrieval & Planning	API-Bank [116]	73 commonly used APIs	753
	CodeRAG-Bench [117]	Competition solutions, tutorials, documentation, StackOverflow, GitHub	25,859
	Search4Code [118]	Bing	6596(java)/4974(c#)
	CoIR [119]	GitHub, StackOverflow, and Various Benchmarks	2.38M (corpus) 3.37(queries)
Memorization	SATML-ext [120]	GitHub	1,000 samples

TABLE XXVIII: Overview of AI4SE Benchmarks Related to Pseudocode.

Category	Benchmark	Language(s)	No. of Problems	Crowdsourced
Pseudocode to Code	SPoC [121]	C++	18,356	Yes
	NAPS [122]	Java/UAST	17,477	No
Code to Pseudocode	Django [123]	Python, English & Japanese	18,805 (Train), 1,000 (Dev), 1,805 (Test)	No

TABLE XXIX: Overview of Selected Crowd-sourced Benchmarks.

Benchmark	Language(s)	No. of Problems	Source
WikiSQL [124]	Natural language → SQL query	80,654	Amazon MTurk (deprecated - 2017)
Spider [125]	Natural language → SQL query	10,181	11 Yale students (2018)
NL2Bash [126]	Natural language → Bash	9,305	Upwork (2018)
NAPS [122]	Java/UAST → Pseudocode	17,477	Self-hosted crowdsourcing, competitive programming community (2018)
SPoC [121]	C++	18,356	Competitive programming websites (2019)
MBPP [14]	Python	974	Google Research, internal crowdworkers (2021)

TABLE XXX: Overview of Automated Program Repair, Fault Localization, and Vulenrability Detection Benchmarks.

Category	Benchmark	Language(s)	No. of Samples
Automated Program Repair & Fault Localization	Defects4J [127]	Java	835
	GitBug-Java [128]	Java	199
	EvalGPTFix [129]	Java	4530
	TutorCode [130]	C++	1239
	GHRB [131]	Java	107
	IntroClass [132]	C	998
	ManyBugs [132]	7 Languages	185
	DebugBench [133]	C++, Java, Python	1,438 & 1,401 & 1,414
	QuixBugs [134]	Java	40 (locations of bugs)
	RES-Q [135]	Python, JS	100 hand-crafted questions + test scripts
	StudentEval [136]	Python	1,749 buggy programs on 48 distinct tasks (3 test cases per problem)
	Re-Factory [137]	Python	1783(buggy)/2442(correct)
	ConDefects [138]	Python, Java	526(Python), Java(477)
	Cerberus [139]	C, C++, Java	2242 (across 4 task types)
Vulnerability Detection	CVEFixes [140]	Various Languages	5,365
	LLMSecEval [141]	C	150 (on 25 distinct vulnerabilities)
	SecurityEval [142]	6 languages	130 (on 75 common vulnerabilities)
	Vul4J [143]	Java	79 vulnerabilities
	FormAI [144]	C	112,000 labeled instances
	VJBbench [145]	Java	42 vulnerabilities
	SmartBugs [146]	Solidity	69 Vulnerable Smart Contracts
	Devign [147]	C	4 large-scale software repositories
	D2A [148]	C/C++	6 OSS Programs
	BigVul [149]	C/C++	348 Projects
	SARD [25]	Java, C, C++, C#, PHP	32k[26]
	Juliet 1.3 [27]	C/C++	64k[28]
	NVD [29]	Various Languages	265k[30]
Software Testing	CoverageEval [150]	Python	1160
	ATLAS [151]	Java	9,275 projects
	HITS [152]	Java	10 projects
	MeMo [153]	Java	9 projects
	MLAPIs [154]	Python	63 applications

TABLE XXXI: Overview of Selected SE-Workflow Benchmarks.

Category	Benchmark	Language(s)	No. of Samples
Code Synthesis & Understanding	Methods2Test [155]	Java	780,944
	CRUXEval [156]	Python	800
	CRQBench [157]	C++	100
	CriticBench [158]	Python	3,825(across 5 tasks)
	CodeScope [159]	8 Programming Langauges	13,390 (across 8 tasks)
Merge Conflict Repair	ConflictBench [160]	Java	180
Type Inference	TypeEvalPy [161]	Python	845 (annotated labels)
	TypeEvalPy AutoGen [161]	Python	78373 (annotated labels)
Automatic Code Quality Review	CodeReview [162]	8 languages	7.9M pull requests
	Software Maintainability [163]	Java	519 projects (evaluations of quality)
Hallucination Detection	HALLUCODE [164]	Python	5,663

TABLE XXXII: 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 [167]	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 [168]	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 [169]	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 [169] MicrosoftDocs[31] CodeSearchNet [170]	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