Position: Code Benchmarks Should Prioritize Rigor, Reliability, and Reproducibility

Jialun Cao [1 2] **Yuk-Kit Chan** [* 3] **Zixuan Ling** [* 3] **Wenxuan Wang** [4 †] **Shuqing Li** [3] **Mingwei Liu** [5] **Ruixi Qiao** [6] **Yuting Han** [7] **Chaozheng Wang** [3] **Boxi Yu** [8] **Pinjia He** [9] **Shuai Wang** [1] **Zibin Zheng** [5] **Michael R. Lyu** [3] **Shing-Chi Cheung** [1 2]

Abstract

Code-related benchmarks play a critical role in evaluating large language models (LLMs), yet their quality fundamentally shapes how the community interprets model capabilities. In the past few years, awareness of benchmark quality has grown. Yet, after a decade-scale (2014 - 2025) survey over 672 code benchmarks, we observed *a lag between growing awareness and actual practice*. For example, in 2025 alone, the number of benchmarks that ignore code coverage when providing test cases nearly matches the total count accumulated across the previous ten years. In response, we take a clear position: **Code benchmarks must prioritize rigor in benchmark construction, reliability in evaluation, and reproducibility in release**. To operationalize this position, we introduce a code benchmark guideline HOW2BENCH with 55 checklists. Finally, our further human study also exposed that the current issues not only stem from the significant effort required, but also from a lack of awareness regarding their importance.

1. Introduction

> ❝ *Awareness is the beginning of action; action is the fulfillment of awareness.* ❞ — Yangming Wang (1472 - 1529)

Large Language Models (LLMs) are increasingly evaluated on code benchmarks such as SWE-Bench (Jimenez et al., 2024) to measure their code generation, code reasoning, and debugging capabilities. These benchmarks play a critical role in shaping the understanding of LLMs' true capabilities and limitations. However, **the validity of conclusions drawn from these code benchmarks depends on the rigor, reliability, and reproducibility** of the benchmarks themselves. When the benchmark construction is flawed, the validity of conclusions may not hold.

Encouragingly, awareness of benchmark quality has grown. Guidelines (Wu et al., 2025) and enhancing approaches (Liu et al., 2023b; Qiu et al., 2024b; Yadav et al., 2024b) were proposed to provide the good practices and enhance the quality in certain aspects (*e.g.*, code coverage (Liu et al., 2023b), data contamination (Cao et al., 2024b)). As a result, *one might expect that benchmark construction practices have already become more rigorous over time.*

However, our findings suggest the opposite. Through a large-scale survey of 672 code benchmarks across the last decade (2014 – 2025), we uncover a concerning trend: **despite rising awareness of benchmark quality, the number of flawed benchmarks has continued to grow**. For example, in 2025 alone, the number of benchmarks that *ignore code coverage* when providing test cases nearly matches the total count accumulated across the previous ten years. More **empirical evidence** includes:

- 46% benchmarks did not go through ***quality assurance check***; 79.8% did not consider data contamination; 67.1% did not deduplicate the data points;
- 85.0% did not ensure a ***reliable judgement***, such as ensuring a high code coverage when test suites are provided; 66.1% benchmark evaluation was one-pass, without repeating the experiment to avoid randomness;
- 38.2% of the benchmarks did not provide the essential information (*e.g.*, prompts) for ***reproducibility***; 80.0% did not provide the log information on the benchmarks; 14.7% are ***not open source***, 2.2% only partially released;

∗ Equal contribution. † Corresponding author. [1]The Hong Kong University of Science and Technology [2]Guangzhou HKUST Fok Ying Tung Research Institute [3]The Chinese University of Hong Kong [4]Renmin University of China [5]Sun Yat-Sen University [6]Chinese Academy of Sciences, Institute of Automation [7]Beijing Language and Culture University [8]Lero the Research Ireland Centre for Software, University of Limerick [9]The Chinese University of Hong Kong, Shenzhen. Correspondence to: Jialun Cao <jialuncao@ust.hk>, Wenxuan Wang <jwxwang@gmail.com>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

In response, we take a clear position: **Code benchmarks must prioritize rigor in benchmark construction, reliability in evaluation, and reproducibility in release as first-class objectives.**

To operationalize this position, we introduce **a comprehensive code benchmark guideline, HOW2BENCH,** comprising 55 rigor-oriented checklists, each annotated with its relative threat level to the benchmark validity. This checklist covers the entire lifecycle of benchmark development, from design and construction to *evaluation*, analysis, and release, as shown in Figure 1.

Upon the 55-item checklist, we revisited 672 code benchmarks, and quantified longitudinal trends in how benchmark practices have evolved. We observe **a lag between growing awareness and actual practice**. Although the yearly proportion of benchmarks that prioritize data quality and evaluation credibility has increased, the absolute number of flawed benchmarks continues to rise, largely due to the rapid overall growth in benchmark production.

In addition to these concerning trends, we also identify several **positive signals**:

• During *Benchmark Design*: recent benchmarks **increasingly focus on practical, real-world problems**; notably, the number of project-level benchmarks in 2025 nearly tripled compared to 2024, and the distribution of code task types has become more diverse;

• During *Benchmark Construction*: creators are showing **stronger awareness of manual quality assurance**: the number of benchmarks incorporating human verification more than doubled from 2024 to 2025;

• During *Benchmark Evaluation*: benchmark creators increasingly include a larger and more diverse set of studied LLMs, improving the generalizability and robustness of reported findings;

• During *Benchmark Release*: growing open-science engagement, with **an upward trend in publicly released artifacts, prompts, and evaluation resources**, reflecting stronger community commitment to transparency and reuse.

Human Study – To better understand whether the current situation stem from lack of awareness, resource constraints, or incentive misalignment, we conducted a human study involving 49 participants through questionnaires. All participants concurred on the necessity of having a checklist for benchmark construction to enhance quality. Interestingly, beyond admitting the its importance, the collected feedbacks also exposed ***gaps in awareness***: 16% of participants were unaware of the necessity for data denoising; over 40% were not aware that the experimental setup and environment could impact the reproducibility and transparency.

The questionnaires results revealed that the current issues in flawed benchmark not only stems from the significant effort required to construct a rigorous and reliable benchmark, but also from a lack of awareness regarding how crucial a rigorous development process is to the quality of benchmarks and the reliability of evaluations. Without being aware of and addressing the quality of code-related benchmarks (as well as other kinds of LLM benchmarks), the reliability and reproducibility of benchmark results remain compromised, misleading research directions, and hindering meaningful progress.

This paper makes contributions in five aspects:

- ***Novelty***. We introduce HOW2BENCH, a comprehensive set of guidelines packaged as a 55-criteria checklist that covers the lifecycle of code-related benchmark development.
- ***Significance***. HOW2BENCH presents the first comprehensive set of actionable guidelines for developing high-quality benchmarks, striving to create a more reliable and transparent environment. The human study also highlighted the demand for such a detailed guideline.
- ***Usefulness***. HOW2BENCH serves as a guideline for practitioners before/during developing code-related benchmarks, and a checklist for evaluating existing benchmarks after their release. For ease of use, we also provide a ***printable version*** of HOW2BENCH on Appendix G.
- ***Generalizability***. Most criteria listed in HOW2BENCH can be adopted or adapted to other benchmarks such as Question-answering, mathematical reasoning, and multimodal benchmarks.
- ***Long-term Impact***. Our statistics alert the community to the severity and prevalence of non-standard practices in benchmark development. It ultimately improves the overall quality of benchmarks through propagation effects among them.

2. Related Work

2.1. Code Benchmarks

Benchmarks for coding tasks like code generation (Chen et al., 2021a; Austin et al., 2021), defect detection (Just et al., 2014; Gao et al., 2023b; Liu et al., 2024d), and program repair (Jimenez et al., 2024; Risse & Böhme, 2024) are increasingly common, reflecting the growing needs for using LLMs for coding tasks. Recent studies have highlighted various issues with these benchmarks, ranging from design inconsistencies to scope and applicability limitations. For example, (Liu et al., 2023b) found that even some widely used benchmarks, such as HumanEval (Chen et al., 2021a)

and MBPP (Austin et al., 2021), contains a non-trivial proportion of bugs in implementation, documentation, and test cases. Our work, in comparison, introduces a detailed guideline that ***guides the benchmark development*** during the entire lifecycle.

2.2. Benchmark Surveys and Studies

Several recent surveys (Koohestani et al., 2025a) and empirical studies have profiled the status quo of LLM development. These studies either explore the overall performance for certain areas, such as software engineering (Hou et al., 2023; Wang et al., 2024a), or investigate the capabilities of LLMs on specific tasks, such as code generation (Dou et al., 2024; Yu et al., 2024) and test generation (Schäfer et al., 2024; Yuan et al., 2024b; 2023b). A survey (Chang et al., 2024) about how to evaluate LLMs was proposed to answer what/where/how to evaluate LLMs. This paper differs from these studies in its purpose and perspectives.

Assessments or guidelines for benchmarks (Qian et al., 2026; Koohestani et al., 2025a; Hu et al., 2025b; Reuel et al., 2024) are also constantly released. For example, BetterBench (Reuel et al., 2024) is a related work assessing the AI benchmarks against 46 criteria. Then, it scored 24 AI benchmarks in various domains and ranked them. BetterBench differs from this paper in several key aspects: scope (general benchmarks vs. code-related benchmarks), lifecycle division (it addresses benchmark retirement, while How2Bench focuses on benchmark evaluation, analysis, and release), and objectives (scoring benchmarks vs. offering comprehensive guidelines for future benchmark development). Additionally, the study in this paper was conducted on a much larger scale (24 vs. 572 benchmarks), statistically highlighting the prevalent issues in existing benchmarks.

Unlike these surveys and guidelines, our work reveals a longitudinal trends before and after them came out.

3. Guideline Design

3.1. The Lifecycle of Benchmark Development

Code-related benchmark development comprises five typical phases (Phase 1 - 5), as shown in Figure 1, explained in detail as follows.

Phase 1. Design. At the beginning of benchmark development, it is vital to identify the motivation, the ***scope*** and the ***capabilities*** required by the ***application*** scenario of interest. To achieve this objective, one needs to carefully consider the application scenarios, making sure these scenarios align with real-world demands (Moriarty, 2011). Also, it is necessary to assess whether other benchmarks already exist that address similar tasks, and to identify any shortcomings they may possess (McIntosh et al., 2025; Malode, 2024). Furthermore, this new benchmark should be designed to evaluate specific LLMs' capabilities; the crafted tasks are expected to reflect these capabilities (Hodak et al., 2023).

Phase 2. Construction. Benchmark Construction phase moves from design to execution. Typically, data is ***collected*** from public coding websites such as GitHub, LeetCode, and StackOverflow. This is followed by preprocessing, which includes filtering, cleaning (e.g., deduplication, denoising), and ***curation*** (e.g., aligning tests with corresponding code). The phase usually ends with a ***validation*** process, which can be manual or automated (McIntosh et al., 2025).

Phase 3. Evaluation. Once the benchmark is available, the next step is to apply it to LLMs, validating if it can effectively measure the intended LLM capabilities. Essential considering factors include ***selecting a representative array of LLMs***, configuring ***settings*** like prompts and hyperparameters for consistency, choosing appropriate experimental ***environments*** to meet LLM requirements, and implementing thorough ***logging*** to ensure dependable and reproducible results.

Phase 4. Analysis. After evaluation, experimental results are analyzed, drawing conclusions on LLMs' capabilities. This phase involves comparing each LLM's ***performance*** to identify standout or underperforming models. Then, proper visual aids such as bar charts and tables can be used to ***display*** the experimental results, presenting clearer observation and deeper ***inspiration***, such as the correlations between models, the correlations with related benchmarks, or performance in upper-/down-stream tasks (Hendee & Wells, 1997). Indeed, a thorough analysis helps pinpoint areas for improvement and guides future LLM enhancements.

Phase 5. Release. The final phase is to make the benchmark open-accessible. This phase involves meticulously preparing all ***materials*** associated with the benchmark, ensuring they are ready for ***open access*** to foster widespread adoption and collaboration. Clear, comprehensive ***documentation*** is provided to guide users on effectively utilizing the benchmark. Additionally, all logged ***experiment details*** are made available, enhancing the reproducibility and transparency of the benchmark.

3.2. Study Design

Our study consists of four steps (Figure 2). All steps are explained as follows.

Step 1. Guideline Construction. To begin with, we ***sketched the initial guidelines*** for each phase in the benchmark development lifecycle (Section 3.1, Figure 1) by reviewing existing literature (Suppes et al., 1962; Zheng et al., 2023b; Schäfer et al., 2024; Reuel et al., 2024) and brainstorming. After that, we ***refined the guidelines*** through a series of interviews with various stakeholders, includ-

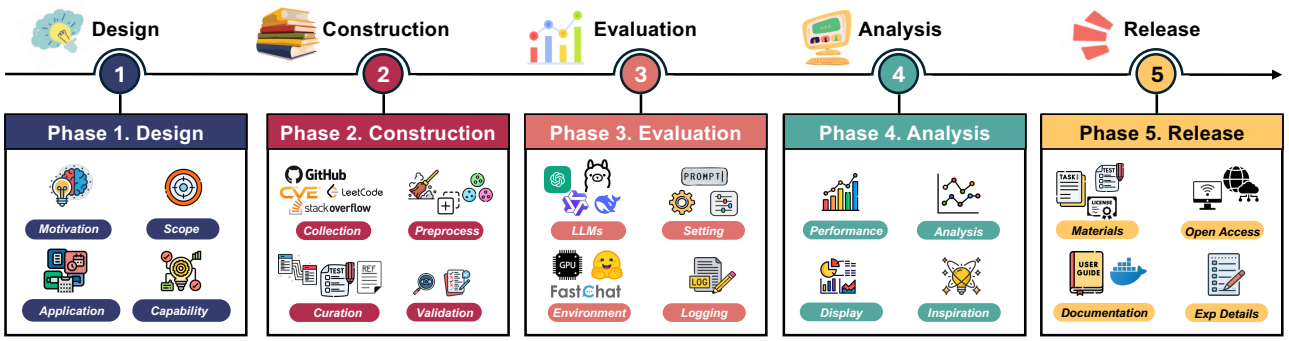

Figure 1. Lifecycle of Benchmark Development

ing model developers and benchmark builders, allowing for the addition, deletion, or modification of criteria based on expert feedback and practical insights. In order to allow more flexibility and increase the practicality of the guideline, we prioritized them with ★★★ (Highly important), ★★ (Important), and ★ (Optional). By doing so, the highly-recommended criteria are essential to comply with when setting up a new benchmark, while other criteria allow compromise, making the guideline simpler for benchmark developers to follow.

Step 2. Literature Profiling. This step begins by ***collecting related benchmarks*** according to their publication time, venue, and coding tasks, then employing techniques like ***snowballing*** to ensure a comprehensive collection. This step leads to 672 code-related benchmarks for study. The detailed statistics are available in Appendix F. This step is followed by ***profiling*** each selected benchmark through a thorough review of *corresponding papers* and examination of *the released artifacts* or homepages associated with these benchmarks. The phase is completed by ***reporting statistics*** that highlight overall trends, pros, and cons identified during the profiling, providing a structured overview of existing benchmarks.

Step 3. Focused Case Study. After obtaining an overall impression of existing benchmarks, we ***selected 30 (= 5 * 6) representative benchmarks*** from the top-5 most frequent coding tasks (see Figure 9), with top-5 highly-cited benchmarks plus the latest 1 benchmark (Appendix E). Each selected benchmark is then ***analyzed against*** How2Bench, examining how well they meet the established criteria, studying their overall statistics, and identifying both exemplary and poor cases. Insights and references from existing literature are also incorporated to enrich the analysis, providing a deeper understanding of the benchmarks' performance and areas for improvement.

Step 4. Human Study. The final step is a human study that evaluates the importance and practicality of How2Bench. This involves ***designing a questionnaire*** by first initiating and iterating to gather diverse, logical insights, which is then ***distributed*** to a targeted audience. After collecting and filtering responses for quality, the data is ***analyzed*** to derive insights. See Appendix D for details.

4. Guideline and Key Statistics

The completed guideline How2Bench with 55 criteria can be found in the Appendix. Each guideline contains: an actionable check criterion with necessary explanations; a priority indicator, divided into three levels in total, marking the importance of the checklist; and a checkbox for convenience.

4.1. Guideline for Benchmark Design

Explanation – For benchmark design, we listed four essential criteria, as shown in Figure 3. In particular, the guideline starts by recommending that benchmarks should initially assess if they are addressing a ***significant gap*** in existing research, ensuring the relevance and necessity of the benchmark. The ***scope*** of the benchmark is expected to be well-defined, clarifying the ***capabilities*** or characteristics being tested, how these relate to practical scenarios such as programming assistance or automated testing, and the relevance of these capabilities in real-world ***applications***.

■ ***Key Statistics*** – According to our statistics among 672 benchmarks, ***apparent research bias*** can be observed in terms of coding tasks, programming languages, and code granularities (Appendix C.1). For example, ***Code Generation*** is most prevalent coding task, with 235 benchmarks focusing on this area, according to 34.97% (235/672) of studied benchmarks, indicating a significant interest in generating code automatically. The second most prevalent is code reasoning 12.64% (85/672), followed by Program Repair and Defect Detection.

When examining the distribution of coding tasks by year (see Figure 10), we observe that benchmark growth accelerated most rapidly in the past two years (2024–2025),

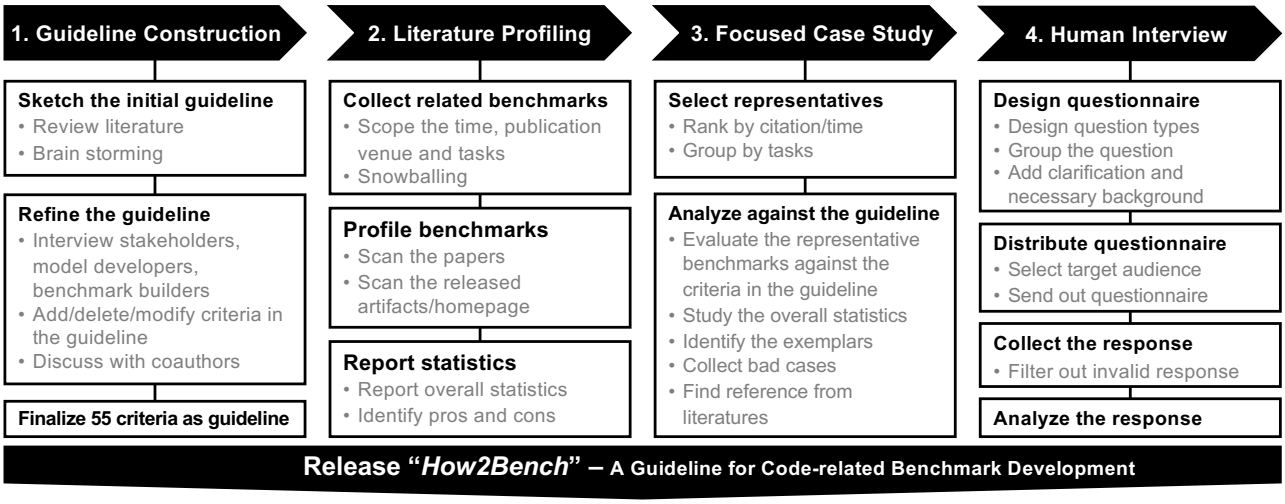

Figure 2. Workflow of study process

with code generation remaining the most frequently benchmarked task. At the same time, **demand for code reasoning has increased substantially**, rising from 19 benchmarks in 2024 to 61 in 2025. Benchmarks for program repair also increased over the same period, from 22 to 33.

Regarding the bias in programming language, Figure 11 shows that 409 (71.50%) benchmarks are in Python, followed by Java and C++, with 229 and 160, respectively. This observation consolidates the observation from previous works (Cao et al., 2024a; Hou et al., 2023) on a larger scale. When analyzed by year (Figure 12), the number of benchmarks for C/C++ and JavaScript has increased noticeably between 2024 and 2025. We also observe a sharp rise in Rust-related benchmarks: although only a few benchmarks existed for Rust since its release in 2015, 57 benchmarks were published in the past two years (27 in 2024 and 29 in 2025). These trends in programming language coverage reflect the evolving demand for LLMs across different languages, highlighting particularly the growing reliance on large models to generate, reason about, and maintain code in these languages.

Regarding the bias in granularity, a dramatic upward trend was observed in project-level benchmarks (Figure 16): between 2024 and 2025, the number of project-level benchmarks surged, increasing from 35 to 97 new benchmarks. Importantly, these numbers refer to benchmarks newly introduced in each year, rather than cumulative totals. This surge indicates a **growing community focus on the real-world applicability and large-scale practical utility of LLMs**.

Regarding the LLMs' capability evaluated, during the focused case study (listed in Appendix E), we identified that 20% benchmarks have not explicitly specified the capabilities (e.g., intention understanding, program synthesis) to be evaluated, and ***23.3% have not specified application scenarios*** the benchmark targets.

Besides, we also identified a case in MBPP (Austin et al., 2021) where a case fell out of the target evaluation capabilities (Appendix C.2). Indeed, clearly defining the application scenarios/scopes/capabilities could help benchmark constructors establish precise goals for the design and development of the benchmark, ensuring accuracy in the evaluation.

▲ **Severity –** Current benchmarks exhibit ***an apparent imbalance*** in coding tasks and programming languages dominated by code generation and Python, leaving research blank to be filled. Also, even highly cited benchmarks may have samples that do not fall into the examined capabilities.

4.2. Guideline for Construction

☞ ***Explanation –*** Figure 4 shows 19 criteria for benchmark construction. Essentially, for **data source**, the key considerations include verifying the traceability and quality of the data source, addressing potential *data contamination* (Sainz et al., 2023), and ensuring that the ***data sampling processes*** are scientifically robust and rigorous. Also, for **data representativeness**, it also guides through specific checks to ensure the benchmark's scope is strictly adhered to, such as making sure every data point falls within the targeted scope and that the data can cover all studied capabilities, domain knowledge, and application scenarios.

For data preprocessing and cleaning, it also stresses handling code-specific aspects, such as compilability and execution, along with cleaning and ***manually reviewing*** data for quality assurance. Output validation methods and evaluation metrics must be carefully designed and reviewed to ensure

they effectively measure the benchmark's goals. Lastly, it suggests considering additional evaluation perspectives, such as safety (Wei et al., 2024; Yuan et al., 2024a) checks, ensuring the code does not contain sensitive information.

■ ***Key Statistics*** – According to our statistics (Appendix C.3), the 572 benchmarks exhibit ***numerous irregularities*** in their implementation, which could significantly threaten the reliability of the benchmarks. Surprisingly, 67.1% of benchmarks did not deduplicate or did not mention. 79.8% benchmarks did not consider or handle data contamination threats. About 46.0% of the benchmarks did not go through any quality assurance checks such as manual checks and code execution. In particular, we summarized the *commonly-used data quality assurance metrics* and their frequency: manual check (45.8%), code execution (1.2%),others (*e.g.*, heuristic rules, 6.4%).

Examining the trends by year (Figure 24), however, **one positive development is the increasing prevalence of manual quality checks**. The number of benchmarks with manual check guarantees doubled from 84 in 2024 to 193 in 2025, indicating that a growing share of code benchmarks now undergo partial or full human verification.

Also, since we focus on code-related benchmarks, which usually accompany test cases, ***code coverage*** also needs to be considered. According to the statistics over oracles (Figure 31), **passing test cases** (257 / 672 = 38.2%) and **the exact match** (193 / 672 = 28.7%) are the most common oracles used in code benchmarks. However, we observed that only 15.9% considered and reported code coverage (*i.e.*, line coverage, branch coverage) explicitly in their papers, while 82.5% did not consider the code coverage when constructing the test suites for benchmark evaluation. The annual distribution (Figure 33) makes this trend even clearer: although many benchmarks in the past three years (2023–2025) did not consider code coverage, the sheer volume of benchmarks in 2025 means that the absolute number of such benchmarks is high, with 24, 77, and 123 benchmarks respectively for 2023, 2024, and 2025. This underscores that, despite growing awareness of evaluation rigor, a substantial number of benchmarks continue to provide incomplete testing, highlighting a persistent threat to the validity and reliability of benchmark-driven assessment. It severely affects the reliability of findings on these benchmarks, potentially misguiding future research and applications based on these flawed assessments.

▲ **Severity –** Most benchmarks display ***severe loopholes*** in data preparation and curation, *i.e.*, only 54% of benchmarks went through a quality assurance check, and only 20.2% of them considered and handled data contamination threats.

4.3. Guideline for Evaluation

☞ ***Explanation –*** Guidelines for benchmark evaluation focus on the rigorousness and reliability of the evaluation. HOW2BENCH provides 12 criteria for benchmark evaluation, as shown in Figure 5. It mainly focuses on the comprehensive evaluation processes for benchmarks involving LLMs. For evaluation design, it stresses the importance of assessing a sufficient and ***representative range of LLMs*** to ensure the benchmark's applicability across various model families and configurations, both open and closed-source. Figure 35 and Figure 37 show the distribution of numbers of LLMs studied and the most exercised LLMs.

Also, ***prompting*** has a direct impact on the quality of the LLMs' output results (Wei et al., 2022; He et al., 2024a; Jin et al., 2024; Ye et al., 2023). As pointed out by a recent study, up to 40% performance gap could be observed in code translation when prompts vary (He et al., 2024b).

Additionally, ***the experiment environment*** is essential for reproducibility and transparency. Indeed, the hardware, software, and platform environments used during experiments might influence the outcomes (Zhang & Huang, 2025). Furthermore, because of the nondeterministic nature of LLMs, experiments should be repeated, and randomization strategies should be used to mitigate the effects of randomness and parameter configuration biases. Lastly, ***meticulously documented logs*** of the experimental process is advised to facilitate transparency and reproducibility, detailing everything from parameter settings to the specific LLM pipelines such as vLLM (Kwon et al., 2023) used.

■ ***Key Statistics –*** Among the 672 benchmarks, 585 of them are evaluated over LLMs. As shown in Figure 35, most benchmarks were evaluated against six LLMs (10.0% = 59 / 585), followed by six LLMs. Encouragingly, increasing LLMs are being studied for code benchmarks. As shown in Figure 36, in 2024, 39.5% of benchmarks were evaluated against fewer than five LLMs, whereas in 2025, 64.2% of benchmarks (36.7% + 27.5%) were evaluated against 5 - 20 models. This shift indicates that benchmark studies are increasingly aiming for more comprehensive and generalizable evaluations. However, it also introduces substantially higher computational and financial costs, highlighting the trade-off between evaluation rigor and resource efficiency.

For reference, we listed the top 10 most studied LLM families in Figure 37. Among them, the GPT series from OpenAI is the most extensively studied, accounting for 76% (446/585), followed by Deepseek and Qwen.

The prompt quality also significantly impacts the LLM evaluation (He et al., 2024b). According to a recent study, up to 40% of performance variation could be observed in the code translation task (He et al., 2024b). So, carefully designing a prompt needs consideration. However, ***76.7%*** represen-

tative benchmarks (Appendix E) do not validate whether the prompts they used are well-designed (Appendix C.4). Similarly, though 89.4% benchmarks were evaluated in a zero-shot manner, only 18.6% benchmarks were evaluated under few-shot, 3.7% under Chain-of-Thought and 1.0% under RAG (Figure 40). However, as shown in Figure 41, 76.7% representative benchmarks (Appendix E) do not validate whether the prompt they used is well-designed.

Regarding the evaluation process, our statistics exposed that **only 33.4% of benchmark evaluations have been repeated** (Appendix C.4). Also, regarding the transparency and matriculated documents, the observation is not optimistic – **Only 6.3% benchmarks provided their experiment environment**. **More than 38% of benchmarks did not provide reproducible instructions** such as prompts, examples for few-shot learning, or content for retrieval (Figure 48). **Only half (50.5%) provide hyperparameters** such as temperature for reproduction.

▲ **Severity –** 66.6% of evaluations have not been repeated to eliminate the impact of randomness, and the trend grows from the year 2023 to 2025 (from 31 to 205).

4.4. Guideline for Evaluation Analysis

☞ ***Explanation –*** The analysis of the experiment results is expected to be objective and comprehensive, hopefully providing insights or actionable advice. So, we listed 10 criteria for the evaluation analysis phase, as shown in Figure 6. Regarding **the perspectives of analysis**, inspired by classic measurement theory (Suppes et al., 1962), we suggest four essential perspectives, including ***difficulty*** (whether a benchmark is appropriately challenging for LLMs), ***stability*** (whether the results are consistent through repeated trials), ***differentiability*** (whether benchmarks can differentiate the strengths and weaknesses of various LLMs), and ***inspiration*** (e.g., the correlations between the upper-/down-stream coding tasks and LLM scores).

Moreover, effective ***presentation of results*** using clear visual and textual descriptions could ensure the findings are understandable and actionable. The phase concludes with the suggestion to interpret and explain the results comprehensively, providing a basis for future enhancements.

■ ***Key Statistics –*** Because experimental analysis is relatively subjective and cannot be obtained through mechanical scanning, we focus on 30 representative focus benchmarks (Appendix E), covering the highest cited and latest benchmarks in top-5 tasks. Figure 45 shows an example from CruxEval (Gu et al., 2024) where the experimental scores can hardly be read from the figures.

Also, ***explaining experiment results*** is crucial for other practitioners to understand what the outcomes mean in the context of the research questions. According to our statistics (Appendix C.5), 70% of benchmarks have detailed explanations and analyses of their evaluation results, while still ***30% have not***. Indeed, an explanation contributes to the body of knowledge by making it possible to understand and compare results with previous studies, promoting transparency within the community.

▲ **Severity –** The analysis of experimental data and the clarity of data presentation may receive less attention and be worth consideration. Even in papers cited 2k+ times like MBPP (Austin et al., 2021), there are instances of***unclear evaluation analysis and display***.

4.5. Guideline for Benchmark Release

☞ ***Explanation –*** Finally, releasing a benchmark for open access also needs careful consideration. We offered 10 suggestions for this step, as shown in Figure 7, to highlight essential steps for public release preparation, emphasizing accessibility and ethical compliance. This includes setting an appropriate **license** to clarify usage rights, conducting a thorough review to **eliminate sensitive or harmful content** such as the API keys to access LLMs, the personal emails or toxic code comments (Miller et al., 2022) unless they are a part of the benchmark, and ensuring reproducibility by making all related materials openly available. **Detailed prompts** and clear descriptions of the experimental setup are advised to facilitate replication. Additionally, providing user manuals and evaluation interfaces is crucial for effective user engagement with the benchmark, enhancing its reliability and value for the research community.

■ ***Key Statistics –*** The final step involves the release of the benchmark. The fundamental requirement for releasing a benchmark is that it must be open-sourced. However, surprisingly, we observed that 2.2% of the benchmarks are only partially open-sourced (e.g., missing some subjects or tests), and ***14.7% are not open-sourced at all*** (e.g., links/web pages are no longer active). Furthermore, prompts, which are necessary for reproducibility, are not disclosed in 38.2% of the benchmarks (Figure 48), not to mention the lack of public information on experimental settings (Figure 39 and Figure 38) and experimental parameters (Figure 53). What is worse, 19.3% benchmarks do not set up licenses (Figure 54). The absence of licensing may lead to severe legal and ethical issues, potentially resulting in unauthorized use and distribution of proprietary technologies. Additionally, only 16.7% of the benchmarks make their logged experimental results publicly available (Appendix C.6). Note that this conclusion about log availability are based on publicly accessible materials; logs are often missing for a combination of policy, privacy, and practicality reasons.

Fortunately, the recent years show a positive shift toward

openness (Figure 47). From 2024 to 2025, the number of open-sourced benchmarks increased substantially, rising from 169 to 266, indicating growing community commitment to transparency, accessibility, and reproducibility.

▲ **Severity –** The release of existing benchmarks exhibits several issues. For example, 16.8% of the benchmarks are either not open to public access or are only partially open-sourced. Only 47.4% of benchmarks are released with replicable prompts.

5. Human Study

To delve deeper into the integration of knowledge and action, we ***surveyed 49 global researchers*** in AI (42.6%) and Software Engineering (57.14%), as shown in Figure 61. Each participant had published at least one research paper to ensure their research maturity, and ***half had constructed code benchmarks***. See Appendix D for more details about the participants' demographics and questions in questionnaires.

First, **all participants agreed** that having a checklist for benchmark construction would contribute to the quality of the benchmark. 47/55 criteria in HOW2BENCH are deemed important by more 80% participants. Additionally, among the 21 participants who have constructed code-related benchmarks, ***53 out of 55 criteria were deemed important by all benchmark developers***; only two criteria (criteria 3 and 4 in Section 4) were considered unimportant by a few individuals (3 and 2 participants, respectively). Additionally, we received two valuable suggestions that draw importance to recording ***the time/monetary costs*** of constructing the benchmark and conducting the experiments.

However, we also identified some ***notable gaps in awareness***. First, regarding the ***data preparation***, more than 15% of participants were not aware that the selection of data should consider the target scope of the evaluation set (i.e., the data must be representative). 16% of participants were ***unaware of the need for data denoising***, while half (8%) of these have already published at least one paper on benchmarking construction. This oversight can significantly affect the validity and generalizability of experimental results, underscoring the importance of a comprehensive understanding of data handling for reliable research outcomes.

Second, regarding ***evaluation replicability and reliability***. Over 40% of participants believe that recording and publicizing the hardware and software environments, software versions, and libraries used in experiments is not important, with more than 20% still considering it unimportant despite already done so. This reveals ***a significant lack of awareness*** about the impact that experimental environments can have on ***the reliability, reproducibility, and stability of evaluation results***. In fact, various studies have demonstrated that different experimental environments, parameters, and prompts can lead to substantial variations in outcomes (Xiao et al., 2024; Wang et al., 2019; 2023a).

6. Alternative Views

We clarify several seemingly natural but ultimately misleading ways to frame our work. **Alternative View 1: Frame our work as a survey of code benchmarks**. Our decade-scale analysis of 672 benchmarks may suggest that this paper is primarily a survey. While we do provide quantitative and qualitative observations, our main contribution is normative and prescriptive, not purely descriptive. Rather than cataloging existing work, we (i) expose systematic gaps between awareness and practice, and (ii) propose actionable standards and tooling directions for future benchmark design. **Alternative View 2: Frame our work as a criticism of specific benchmarks or communities**. Our intent is not to single out particular benchmarks or communities as bad. The issues we study are widespread and systemic. We emphasize structural incentives and use HOW2BENCH to offer a constructive, forward-looking path for community-wide improvement. **Alternative View 3: Frame our work as a call for perfectionism or over-engineered benchmarks**. Prioritizing rigor, reliability, and reproducibility does not mean that all checklists must be satisfied for a benchmark to be "acceptable". HOW2BENCH is a structured checklist and prioritization tool, not a rigid standard. It makes trade-offs explicit, helps authors identify the most critical gaps, and supports transparent choices. Our goal is to shift the default toward more principled practices and clearer documentation, not to demand maximal rigor in every dimension.

7. Discussion

7.1. Trade-offs Between Benchmark Rigor and Development Efficiency

Despite all the above arguments, we admit that constructing rigorous benchmarks often entails a significant investment of time and human effort, which can lead to reduced efficiency in the development and evaluation process. Indeed, ensuring data quality, implementing thorough validation procedures, and designing comprehensive evaluation protocols requires non-trivial effort and may slow down the pace of research. It also echoed our human study. Through our human study, we found that researchers are often aware of the importance of several criteria (*e.g.*, data denoise and repeating the experiments), but did not implement them due to time constraints or other limitations.

However, this trade-off between rigor and efficiency is necessary to guarantee the reliability, reproducibility, and scientific value of benchmark results. While faster, less rigorous

benchmarks might accelerate short-term experimentation, they risk producing misleading or non-generalizable findings that ultimately hinder long-term progress. Therefore, each checklist in HOW2BENCH is labeled with a **priority**: ★★★ (Highly important), ★★ (Important), and ★ (Optional). We hope the design of this indicator may better help benchmark developers balance efficiency with rigor.

7.2. Awareness and Action

It is also worth recapping the notable gaps in awareness of the importance of data preparation and reproducibility (Section 5). For example, 16% of participants were unaware of the need for data denoising; 40% of participants believe that recording and publicizing the hardware and software environments, software versions, and libraries used in experiments is not important. However, without being aware of and addressing the quality of LLM benchmarks, the reliability and reproducibility of benchmark results remain compromised. This observation should be a call to action for the research community to strengthen education and awareness around best practices in benchmark development.

7.3. Criteria on Literature Labeling

Labeling Process: Because the literature scanning and annotation (*i.e.*, Step 2 in the study workflow) involve manual labeling, we summarize the concrete steps we followed when labeling the 672 benchmarks. First, **Independent annotation by two primary annotators**. For each benchmark paper, two primary annotators independently: (1) Skimmed the full paper and then carefully read sections likely to contain relevant information (e.g., Dataset/Data, Data Collection, Data Curation/Cleaning, Quality Control/Assurance). (2) Assigned each attribute a label from present, not mentioned. (3) Recorded textual evidence (exact sentences or locations) whenever an attribute was labeled as "present", storing all evidence in a shared annotation sheet. Second, **Conflict detection and resolution with additional annotators**. After both primary annotators completed their labels, we automatically identified conflicts (e.g., one annotator labeled "has contamination check" as present while the other labeled it as absent). For each conflicted benchmark–attribute pair: The two primary annotators first re-checked the paper and their evidence. If disagreement remained, two additional annotators joined the discussion to reach a consensus. Finally, **Finalization and quality control**. Only consensus labels were used in our descriptive statistics. We calculated inter-annotator agreement using Cohen's kappa; it ranged from 0.76 to 0.92 across attributes.

Labeling Criteria: The manual annotation (*e.g.*, determining whether a paper considers data denoising) follows the annotation criteria: (1) **Criteria for checking whether data quality assurance is present**: We labeled a benchmark as "considers data quality assurance" only if we identified a dedicated section, subsection, or explicit sentences describing how data quality was ensured. In the absence of such textual evidence, we labeled the benchmark as not considering data quality assurance. Annotators look for explicit discussion of how data quality is ensured. Typical evidence includes sections or sentences tagged as "Data Curation / Cleaning / Preprocessing", "Quality Control / Quality Assurance", etc., or detailed descriptions of construction and filtering procedures. (2) **Criteria for data quality assurance by manual check**: We mark a benchmark as using manual checks if it explicitly mentions human involvement in inspecting or validating data/labels, with keywords such as "manual", "human", "inspection", "verification", "review", "spot check", "random sampling", "audt", "sanity check", "hand-curated", "human-in-the-loop", "expert", "label correction", "error analysis", "case-by-case". (3) **Criteria for data quality assurance by code execution**: We mark a benchmark as using code execution checks if it describes validating items by actually running code or tests, with keywords such as "execute/execution", "run", "compile", "build", "unit tests", "test suite", "pass/fail". (4) **Criteria for data quality assurance by LLM-as-judge**: We mark a benchmark as using LLM-as-judge if it delegates evaluation or filtering to a model as a judge, with phrases like "LLM-as-a-judge", "LLM judge", "model judge", "[model name] as judge". (5) **Criteria for data contamination check**: We mark a benchmark as checking for contamination if it explicitly examines or mitigates training–test overlap. We rely on terms such as: "data contamination", "leakage", "overlap", "Jaccard similarity", "time-based split", "cutoff date", "post-YYYY". (6) **Criteria for checking Data deduplication**: We mark deduplication as present if the benchmark explicitly states with cues like: "deduplication", "de-dup", "remove duplicates", "duplicate removal", "redundant", "repetitive". (7) **Criteria for checking coverage considerations**: For code coverage, we look for mentions of: "coverage", "test adequacy", "test strength", "test suite quality", 'corner case(s)" or related phrasing.

8. Conclusion

Awareness of benchmark quality has grown. Yet, we observed a lag between growing awareness and actual practice after surveying 672 code benchmarks. This position paper argues that code benchmarks should prioritize rigor in benchmark construction, reliability in evaluation, and reproducibility in release as first-class objectives; users should actively scrutinize the quality and credibility of the evidence the benchmark designer provides for each claimed category. To operationalize this position, we introduce a code benchmark guideline HOW2BENCH with 55 checklists. We hope this work serves as both a warning and a catalyst, encouraging the community to adopt stronger shared standards.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant No. 92582201, No. 62402113), the Hong Kong SAR Research Grant Council/General Research Fund (Ref No. 16210725), the Hong Kong SAR Research Grant Council/Theme-based Research Scheme (Ref No. T41-517/25-N), Guangdong Basic and Applied Basic Research Foundation (Grant No. 2024A1515010145), Taighde Éireann – Research Ireland under Grant Number. 13/RC/2094_2, and Research Grants Council of the Hong Kong Special Administrative Region, China (RGC Ref. No. SRFS2425-4S03 of the Senior Research Fellow Scheme).

Impact Statement

This position paper argues that code benchmarks should prioritize **rigor** in construction, **reliability** in evaluation, and **reproducibility** in release.

Reframing Benchmark Quality By analyzing a decade of code benchmark development (2014–2025), we present evidence that *awareness of benchmark quality has not yet fully translated into a more rigorous practice*. We argue that benchmark quality should not be treated as an informal best effort, but as a first-class obligation. Our proposed lifecycle-aware framework and 55-item checklist articulate a concrete, actionable standard for what constitutes a methodologically rigorous, reliable, and reproducible benchmark, providing a shared reference point for researchers, reviewers, and benchmark creators.

Shaping Community Norm This work advocates for a shift in community norms *away from speed-driven benchmark releases and leaderboard-centric evaluation toward long-term reliability*. By making evaluation rigor more explicit and measurable with the checklist (HOW2BENCH), we aim to influence how benchmark papers are reviewed and how research contributions are rewarded.

Broader Influence on Machine Learning Evaluation. Although our empirical analysis focuses on code-related benchmarks, the core position advanced in this paper, *i.e.*, benchmark progress should be constrained by methodological rigor, reliability, and reproducibility, *extends to evaluation practices across machine learning*, including vision, natural language processing, robotics, and multimodal AI. We hope this work advances a broader shift and serves as a warning toward rigorous evaluation, strengthening the transparency, comparability, and trustworthiness of benchmarks.

References

Agarwal, Y., Batra, D., and Bagler, G. Building hierarchically disentangled language models for text generation with named entities. In Scott, D., Bel, N., and Zong, C. (eds.), *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pp. 26–38. International Committee on Computational Linguistics, 2020. doi: 10.18653/V1/2020.COLING-MAIN.3. URL https://doi.org/10.18653/v1/2020.coling-main.3.

Agashe, R., Iyer, S., and Zettlemoyer, L. Juice: A large scale distantly supervised dataset for open domain context-based code generation. In Inui, K., Jiang, J., Ng, V., and Wan, X. (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 5435–5445. Association for Computational Linguistics, 2019. doi: 10.18653/V1/D19-1546. URL https://doi.org/10.18653/v1/D19-1546.

Agrawal, L. A., Kanade, A., Goyal, N., Lahiri, S. K., and Rajamani, S. K. Monitor-guided decoding of code lms with static analysis of repository context. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Ahmad, W. U., Tushar, M. G. R., Chakraborty, S., and Chang, K. AVATAR: A parallel corpus for java-python program translation. In Rogers, A., Boyd-Graber, J. L., and Okazaki, N. (eds.), *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pp. 2268–2281. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-ACL.143. URL https://doi.org/10.18653/v1/2023.findings-acl.143.

Ahmed, M. B. U., Harzevili, N. S., Shin, J., Pham, H. V., and Wang, S. Secvuleval: Benchmarking llms for real-world c/c++ vulnerability detection, 2025. URL https://arxiv.org/abs/2505.19828.

Ahmed, T., Hirzel, M., Pan, R., Shinnar, A., and Sinha, S. Tdd-bench verified: Can llms generate tests for issues before they get resolved?, 2024. URL https://arxiv.org/abs/2412.02883.

Aleithan, R., Xue, H., Mohajer, M. M., Nnorom, E., Uddin, G., and Wang, S. Swe-bench+: Enhanced coding benchmark for llms, 2024. URL https://arxiv.org/abs/2410.06992.

Allamanis, M., Jackson-Flux, H., and Brockschmidt, M. Self-supervised bug detection and repair. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan,

J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 27865–27876, 2021.

Alon, U., Brody, S., Levy, O., and Yahav, E. code2seq: Generating sequences from structured representations of code. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=H1gKYo09tX.

Amini, A., Gabriel, S., Lin, S., Koncel-Kedziorski, R., Choi, Y., and Hajishirzi, H. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2357–2367, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1245. URL https://aclanthology.org/N19-1245.

Athiwaratkun, B., Gouda, S. K., Wang, Z., Li, X., Tian, Y., Tan, M., Ahmad, W. U., Wang, S., Sun, Q., Shang, M., Gonugondla, S. K., Ding, H., Kumar, V., Fulton, N., Farahani, A., Jain, S., Giaquinto, R., Qian, H., Ramanathan, M. K., Nallapati, R., Ray, B., Bhatia, P., Sengupta, S., Roth, D., and Xiang, B. Multi-lingual evaluation of code generation models. 2022. doi: 10.48550/ARXIV.2210.14868. URL https://arxiv.org/abs/2210.14868.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Awal, R., Massoud, M., Feizi, A., Li, Z., Wang, S., Pal, C., Agrawal, A., Vazquez, D., Reddy, S., Rodriguez, J. A., Taslakian, P., Gella, S., and Rajeswar, S. Webmmu: A benchmark for multimodal multilingual website understanding and code generation, 2025. URL https://arxiv.org/abs/2508.16763.

Babe, H. M., Nguyen, S., Zi, Y., Guha, A., Feldman, M. Q., and Anderson, C. J. Studenteval: A benchmark of student-written prompts for large language models of code. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 8452–8474. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.501. URL https://doi.org/10.18653/v1/2024.findings-acl.501.

Badertdinov, I., Golubev, A., Nekrashevich, M., Shevtsov, A., Karasik, S., Andriushchenko, A., Trofimova, M., Litvintseva, D., and Yangel, B. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=nMpJoVmRy1.

Bairi, R., Sonwane, A., Kanade, A., C., V. D., Iyer, A., Parthasarathy, S., Rajamani, S. K., Ashok, B., and Shet, S. Codeplan: Repository-level coding using llms and planning. *Proc. ACM Softw. Eng.*, 1(FSE):675–698, 2024. doi: 10.1145/3643757. URL https://doi.org/10.1145/3643757.

Barone, A. V. M. and Sennrich, R. A parallel corpus of python functions and documentation strings for automated code documentation and code generation. In Kondrak, G. and Watanabe, T. (eds.), *Proceedings of the Eighth International Joint Conference on Natural Language Processing, IJCNLP 2017, Taipei, Taiwan, November 27 - December 1, 2017, Volume 2: Short Papers*, pp. 314–319. Asian Federation of Natural Language Processing, 2017. URL https://aclanthology.org/I17-2053/.

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., and Yoo, S. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2014.

Berabi, B., He, J., Raychev, V., and Vechev, M. T. Tfix: Learning to fix coding errors with a text-to-text transformer. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 780–791. PMLR, 2021. URL http://proceedings.mlr.press/v139/berabi21a.html.

Bhargava, V., Ghosh, R., and Dutta, D. Cpp-ut-bench: Can llms write complex unit tests in c++?, 2024. URL https://arxiv.org/abs/2412.02735.

Bogomolov, E., Eliseeva, A., Galimzyanov, T., Glukhov, E., Shapkin, A., Tigina, M., Golubev, Y., Kovrigin, A., van Deursen, A., Izadi, M., and Bryksin, T. Long code arena: a set of benchmarks for long-context code models. *CoRR*, abs/2406.11612, 2024. doi: 10.48550/ARXIV.2406.11612. URL https://doi.org/10.48550/arXiv.2406.11612.

Bradbury, J. and More, R. Addressing data leakage in humaneval using combinatorial test design. pp. 587–591, 03 2025. doi: 10.1109/ICST62969.2025.10989022.

Bui, T., Tun, Y. N., Nguyen, T. P., Su, Y., Thung, F., Li, Y., Ang, H. W., Yin, Y., Liauw, F., Shar, L. K., Ouh, E. L., Zhang, T., and Lo, D. Vulcoco: A simple yet effective method for detecting vulnerable code clones, 2025. URL https://arxiv.org/abs/2507.16661.

Bytedance-Seed-Foundation-Code-Team, :, Cheng, Y., Chen, J., Chen, J., Chen, L., Chen, L., Chen, W., Chen, Z., Geng, S., Li, A., Li, B., Li, B., Li, L., Liu, B., Liu, J., Liu, K., Liu, Q., Liu, S., Liu, S., Liu, T., Liu, T., Liu, Y., Long, R., Mai, J., Ning, G., Peng, Z. Y., Shen, K., Su, J., Su, J., Sun, T., Sun, Y., Tao, Y., Wang, G., Wang, S., Wang, X., Wang, Y., Wang, Z., Xia, J., Xiang, L., Xiao, X., Xiao, Y., Xi, C., Xin, S., Xu, J., Xu, S., Yang, H., Yang, J., Yang, Y., Yuan, J., Zhang, J., Zhang, Y., Zhang, Y., Zheng, S., Zhu, H., and Zhu, M. Fullstack bench: Evaluating llms as full stack coders, 2025. URL https://arxiv.org/abs/2412.00535.

Cao, J., Chen, Z., Wu, J., Cheung, S., and Xu, C. Can AI beat undergraduates in entry-level java assignments? benchmarking large language models on javabench. *CoRR*, abs/2406.12902, 2024a. doi: 10.48550/ARXIV.2406.12902. URL https://doi.org/10.48550/arXiv.2406.12902.

Cao, J., Zhang, W., and Cheung, S.-C. Concerned with data contamination? assessing countermeasures in code language model. *arXiv preprint arXiv:2403.16898*, 2024b.

Cao, R., Lei, F., Wu, H., Chen, J., Fu, Y., Gao, H., Xiong, X., Zhang, H., Mao, Y., Hu, W., Xie, T., Xu, H., Zhang, D., Wang, S., Sun, R., Yin, P., Xiong, C., Ni, A., Liu, Q., Zhong, V., Chen, L., Yu, K., and Yu, T. Spider2-v: How far are multimodal agents from automating data science and engineering workflows? *CoRR*, abs/2407.10956, 2024c. doi: 10.48550/ARXIV.2407.10956. URL https://doi.org/10.48550/arXiv.2407.10956.

Cassano, F., Gouwar, J., Nguyen, D., Nguyen, S., Phipps-Costin, L., Pinckney, D., Yee, M.-H., Zi, Y., Anderson, C. J., Feldman, M. Q., et al. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.

Chai, L., Liu, S., Yang, J., Yin, Y., Jin, K., Liu, J., Sun, T., Zhang, G., Ren, C., Guo, H., Wang, Z., Wang, B., Wu, X., Wang, B., Li, T., Yang, L., Duan, S., and Li, Z. Mceval: Massively multilingual code evaluation, 2024. URL https://arxiv.org/abs/2406.07436.

Chakraborty, S., Krishna, R., Ding, Y., and Ray, B. Deep learning based vulnerability detection: Are we there yet? *IEEE Trans. Software Eng.*, 48(9):3280–3296, 2022. doi: 10.1109/TSE.2021.3087402. URL https://doi.org/10.1109/TSE.2021.3087402.

Chambon, P., Roziere, B., Sagot, B., and Synnaeve, G. Bigo(bench) – can llms generate code with controlled time and space complexity?, 2025. URL https://arxiv.org/abs/2503.15242.

Chandel, S., Clement, C. B., Serrato, G., and Sundaresan, N. Training and evaluating a jupyter notebook data science assistant. *arXiv preprint arXiv:2201.12901*, 2022.

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45, 2024.

Chauvin, T. eyeballvul: a future-proof benchmark for vulnerability detection in the wild, 2024. URL https://arxiv.org/abs/2407.08708.

Chen, J., Huang, H., Lyu, Y., An, J., Shi, J., Yang, C., Zhang, T., Tian, H., Li, Y., Li, Z., Zhou, X., Hu, X., and Lo, D. Secureagentbench: Benchmarking secure code generation under realistic vulnerability scenarios, 2025a. URL https://arxiv.org/abs/2509.22097.

Chen, J., Zhao, K., Liu, J., Peng, C., Liu, J., Zhu, H., Gao, P., Yang, P., and Deng, S. Coreqa: Uncovering potentials of language models in code repository question answering, 2025b. URL https://arxiv.org/abs/2501.03447.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. 2021a.

Chen, P. B., Wenz, F., Zhang, Y., Yang, D., Choi, J., Tatbul, N., Cafarella, M., Çağatay Demiralp, and Stonebraker, M. Beaver: An enterprise benchmark for text-to-sql, 2025c. URL https://arxiv.org/abs/2409.02038.

Chen, X., Gong, L., Cheung, A., and Song, D. Plotcoder: Hierarchical decoding for synthesizing visualization code in programmatic context. In Zong, C., Xia, F., Li, W., and Navigli, R. (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint*

Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021, pp. 2169–2181. Association for Computational Linguistics, 2021b. doi: 10.18653/V1/2021.ACL-LONG.169. URL https://doi.org/10.18653/v1/2021.acl-long.169.

Chen, Y., Ding, Z., Alowain, L., Chen, X., and Wagner, D. A. Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2023, Hong Kong, China, October 16-18, 2023*, pp. 654–668. ACM, 2023. doi: 10.1145/3607199.3607242. URL https://doi.org/10.1145/3607199.3607242.

Chen, Z., Qin, H., Chen, N., Zhao, X., Xue, L., Luo, X., and Wu, X.-M. Solbench: A dataset and benchmark for evaluating functional correctness in solidity code completion and repair, 2025d. URL https://arxiv.org/abs/2503.01098.

Chi, W., Chen, V., Shar, R., Mittal, A., Liang, J., Chiang, W.-L., Angelopoulos, A. N., Stoica, I., Neubig, G., Talwalkar, A., and Donahue, C. Edit-bench: Evaluating llm abilities to perform real-world instructed code edits, 2025. URL https://arxiv.org/abs/2511.04486.

Chou, J., Liu, A., Deng, Y., Zeng, Z., Zhang, T., Zhu, H., Cai, J., Mao, Y., Zhang, C., Tan, L., Xu, Z., Zhai, B., Liu, H., Zhu, S., Zhou, W., and Lian, F. Autocodebench: Large language models are automatic code benchmark generators, 2025. URL https://arxiv.org/abs/2508.09101.

Chua, G. Running in circle? a simple benchmark for llm code interpreter security, 2025. URL https://arxiv.org/abs/2507.19399.

Cui, B., Ramesh, T., Hernandez, O. R., and Zhou, K. Do large language models understand performance optimization? *CoRR*, abs/2503.13772, 2025. doi: 10.48550/ARXIV.2503.13772. URL https://doi.org/10.48550/arXiv.2503.13772.

Daghighfarsoodeh, A., Wang, C.-Y., Taherkhani, H., Sepidband, M., Abdollahi, M., Hemmati, H., and Pham, H. V. Deep-bench: Deep learning benchmark dataset for code generation, 2025. URL https://arxiv.org/abs/2502.18726.

Dai, J., Lu, J., Feng, Y., Ruan, R., Cheng, M., Tan, H., and Guo, Z. MHPP: exploring the capabilities and limitations of language models beyond basic code generation. *CoRR*, abs/2405.11430, 2024. doi: 10.48550/ARXIV.2405.11430. URL https://doi.org/10.48550/arXiv.2405.11430.

Deng, K., Liu, J., Zhu, H., Liu, C., Li, J., Wang, J., Zhao, P., Zhang, C., Wu, Y., Yin, X., Zhang, Y., Zhan, Z., Su, W., Xiang, B., Ge, T., and Zheng, B. R2c2-coder: Enhancing and benchmarking real-world repository-level code completion abilities of code large language models, 2025a. URL https://arxiv.org/abs/2406.01359.

Deng, L., Jiang, Z., Cao, J., Pradel, M., and Liu, Z. Nocodebench: A benchmark for evaluating natural language-driven feature addition, 2025b. URL https://arxiv.org/abs/2507.18130.

Deng, X., Awadallah, A. H., Meek, C., Polozov, O., Sun, H., and Richardson, M. Structure-grounded pretraining for text-to-sql. In Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tür, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., and Zhou, Y. (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pp. 1337–1350. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.NAACL-MAIN.105. URL https://doi.org/10.18653/v1/2021.naacl-main.105.

Deng, X., Da, J., Pan, E., He, Y. Y., Ide, C., Garg, K., Lauffer, N., Park, A., Pasari, N., Rane, C., Sampath, K., Krishnan, M., Kundurthy, S., Hendryx, S., Wang, Z., Bharadwaj, V., Holm, J., Aluri, R., Zhang, C. B. C., Jacobson, N., Liu, B., and Kenstler, B. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks?, 2025c. URL https://arxiv.org/abs/2509.16941.

Dinella, E., Chandra, S., and Maniatis, P. Crqbench: A benchmark of code reasoning questions, 2024. URL https://arxiv.org/abs/2408.08453.

Ding, J., Long, S., Pu, C., Zhou, H., Gao, H., Gao, X., He, C., Hou, Y., Hu, F., Li, Z., Shi, W., Wang, Z., Zan, D., Zhang, C., Zhang, X., Chen, Q., Cheng, X., Deng, B., Gu, Q., Hua, K., Lin, J., Liu, P., Li, M., Pan, X., Peng, Z., Qin, Y., Shan, Y., Tan, Z., Xie, W., Wang, Z., Yuan, Y., Zhang, J., Zhao, E., Zhao, Y., Zhu, H., Zhu, L., Zou, C., Ding, M., Jiao, J., Liu, J., Liu, M., Liu, Q., Tao, C., Yang, J., Yang, T., Zhang, Z., Chen, X., Huang, W., and Zhang, G. Nl2repo-bench: Towards long-horizon repository generation evaluation of coding agents, 2026. URL https://arxiv.org/abs/2512.12730.

Ding, Y., Wang, Z., Ahmad, W. U., Ding, H., Tan, M., Jain, N., Ramanathan, M. K., Nallapati, R., Bhatia, P., Roth, D., and Xiang, B. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural*

Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.

Ding, Y., Fu, Y., Ibrahim, O., Sitawarin, C., Chen, X., Alomair, B., Wagner, D. A., Ray, B., and Chen, Y. Vulnerability detection with code language models: How far are we? *CoRR*, abs/2403.18624, 2024a. doi: 10.48550/ARXIV.2403.18624. URL https://doi.org/10.48550/arXiv.2403.18624.

Ding, Y., Wang, Z., Ahmad, W. U., Ramanathan, M. K., Nallapati, R., Bhatia, P., Roth, D., and Xiang, B. Cocomic: Code completion by jointly modeling in-file and cross-file context. In Calzolari, N., Kan, M., Hoste, V., Lenci, A., Sakti, S., and Xue, N. (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pp. 3433–3445. ELRA and ICCL, 2024b. URL https://aclanthology.org/2024.lrec-main.305.

Dong, H., Yang, J., Deng, X., Jiang, Y., Pekhimenko, G., Long, F., and Si, X. Typybench: Evaluating llm type inference for untyped python repositories, 2025. URL https://arxiv.org/abs/2507.22086.

Dou, S., Jia, H., Wu, S., Zheng, H., Zhou, W., Wu, M., Chai, M., Fan, J., Huang, C., Tao, Y., et al. What's wrong with your code generated by large language models? an extensive study. *arXiv preprint arXiv:2407.06153*, 2024.

Dreyfuss, I., Nassar, A. A., Ackerman, S., David, A. B., Farchi, E., Katan, R., Raz, O., and Zalmanovici, M. Pacific: a framework for generating benchmarks to check precise automatically checked instruction following in code, 2025. URL https://arxiv.org/abs/2512.10713.

Du, J., Liu, Y., Guo, H., Wang, J., Huang, H., Ni, Y., and Li, Z. DependEval: Benchmarking LLMs for repository dependency understanding. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 7150–7179, Vienna, Austria, July 2025a. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.373. URL https://aclanthology.org/2025.findings-acl.373/.

Du, M., Luu, A. T., Ji, B., Liu, Q., and Ng, S.-K. Mercury: A code efficiency benchmark for code large language models. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Du, M., Luu, A. T., Ji, B., Wu, X., Huang, D., Zhuo, T. Y., Liu, Q., and Ng, S.-K. Codearena: A collective evaluation platform for llm code generation, 2025b. URL https://arxiv.org/abs/2503.01295.

Du, X., Liu, M., Wang, K., Wang, H., Liu, J., Chen, Y., Feng, J., Sha, C., Peng, X., and Lou, Y. Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation. *CoRR*, abs/2308.01861, 2023. doi: 10.48550/ARXIV.2308.01861. URL https://doi.org/10.48550/arXiv.2308.01861.

Duan, G., Liu, M., Wang, Y., Wang, C., Peng, X., and Zheng, Z. A hierarchical and evolvable benchmark for fine-grained code instruction following with multi-turn feedback, 2025. URL https://arxiv.org/abs/2507.00699.

Dubniczky, R. A., Horvát, K. Z., Bisztray, T., Ferrag, M. A., Cordeiro, L. C., and Tihanyi, N. Castle: Benchmarking dataset for static code analyzers and llms towards cwe detection, 2025. URL https://arxiv.org/abs/2503.09433.

Eliseeva, A., Sokolov, Y., Bogomolov, E., Golubev, Y., Dig, D., and Bryksin, T. From commit message generation to history-aware commit message completion. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*, pp. 723–735. IEEE, 2023. doi: 10.1109/ASE56229.2023.00078. URL https://doi.org/10.1109/ASE56229.2023.00078.

Feng, J., Liu, J., Gao, C., Chong, C. Y., Wang, C., Gao, S., and Xia, X. Complexcodeeval: A benchmark for evaluating large code models on more complex code. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, pp. 1895–1906. ACM, October 2024. doi: 10.1145/3691620.3695552. URL http://dx.doi.org/10.1145/3691620.3695552.

Finegan-Dollak, C., Kummerfeld, J. K., Zhang, L., Ramanathan, K., Sadasivam, S., Zhang, R., and Radev, D. R. Improving text-to-sql evaluation methodology. In Gurevych, I. and Miyao, Y. (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pp. 351–360. Association for Computational Linguistics, 2018. doi: 10.18653/V1/P18-1033. URL https://aclanthology.org/P18-1033/.

Fu, K., Liu, T., Shang, Z., Ma, Y., Yang, J., Liu, J., and Bian, K. Multi-docker-eval: A 'shovel of the gold rush' benchmark on automatic environment building for software engineering, 2025a. URL https://arxiv.org/abs/2512.06915.

Fu, L., Chai, H., Luo, S., Du, K., Zhang, W., Fan, L., Lei, J., Rui, R., Lin, J., Fang, Y., Liu, Y., Wang, J., Qi, S., Zhang, K., Zhang, W., and Yu, Y. Codeapex: A bilingual programming evaluation benchmark for large language models. *CoRR*, abs/2309.01940, 2023. doi: 10.48550/ARXIV.2309.01940. URL https://doi.org/10.48550/arXiv.2309.01940.

Fu, L., Zhang, B., Guan, H., Zhu, Y., Qiu, L., Liu, W., Cao, X., Cai, X., Zhang, W., and Yu, Y. Automatically benchmarking llm code agents through agent-driven annotation and evaluation, 2025b. URL https://arxiv.org/abs/2510.24358.

Fu, L., Guan, H., Zhang, B., Yuan, H., Zhu, Y., Xu, J., Wang, Z., Qiu, L., Cai, X., Cao, X., Liu, W., Zhang, W., and Yu, Y. Corecodebench: Decoupling code intelligence via fine-grained repository-level tasks, 2026. URL https://arxiv.org/abs/2507.05281.

Fu, Y., Baker, E., and Chen, Y. Constrained decoding for secure code generation. *CoRR*, abs/2405.00218, 2024. doi: 10.48550/ARXIV.2405.00218. URL https://doi.org/10.48550/arXiv.2405.00218.

Gan, Y., Chen, X., Huang, Q., Purver, M., Woodward, J. R., Xie, J., and Huang, P. Towards robustness of text-to-sql models against synonym substitution. In Zong, C., Xia, F., Li, W., and Navigli, R. (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pp. 2505–2515. Association for Computational Linguistics, 2021a. doi: 10.18653/V1/2021.ACL-LONG.195. URL https://doi.org/10.18653/v1/2021.acl-long.195.

Gan, Y., Chen, X., and Purver, M. Exploring underexplored limitations of cross-domain text-to-sql generalization. In Moens, M., Huang, X., Specia, L., and Yih, S. W. (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 8926–8931. Association for Computational Linguistics, 2021b. doi: 10.18653/V1/2021.EMNLP-MAIN.702. URL https://doi.org/10.18653/v1/2021.emnlp-main.702.

Gan, Y., Chen, X., Huang, Q., and Purver, M. Measuring and improving compositional generalization in text-to-sql via component alignment. In Carpuat, M., de Marneffe, M., and Ruíz, I. V. M. (eds.), *Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pp. 831–843. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.FINDINGS-NAACL.62. URL https://doi.org/10.18653/v1/2022.findings-naacl.62.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. PAL: program-aided language models. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023a. URL https://proceedings.mlr.press/v202/gao23f.html.

Gao, Z., Wang, H., Zhou, Y., Zhu, W., and Zhang, C. How far have we gone in vulnerability detection using large language models. *CoRR*, abs/2311.12420, 2023b. doi: 10.48550/ARXIV.2311.12420. URL https://doi.org/10.48550/arXiv.2311.12420.

Garg, S., Moghaddam, R. Z., Clement, C. B., Sundaresan, N., and Wu, C. Deepperf: A deep learning-based approach for improving software performance. *CoRR*, abs/2206.13619, 2022. doi: 10.48550/ARXIV.2206.13619. URL https://doi.org/10.48550/arXiv.2206.13619.

Geng, J., Cai, F., Cui, S., Li, Q., Chen, L., Lyu, C., Li, H., Zhu, D., Pretschner, W., Koeppl, H., and Karray, F. Coquir: A comprehensive benchmark for code quality-aware information retrieval, 2025. URL https://arxiv.org/abs/2506.11066.

Gnieciak, D. and Szandala, T. Large language models versus static code analysis tools: A systematic benchmark for vulnerability detection, 2025. URL https://arxiv.org/abs/2508.04448.

Golchin, S. and Surdeanu, M. Time travel in llms: Tracing data contamination in large language models. *CoRR*, abs/2308.08493, 2023. doi: 10.48550/ARXIV.2308.08493. URL https://doi.org/10.48550/arXiv.2308.08493.

Gong, J., Wu, Y., Liang, L., Zheng, Z., and Wang, Y. Cosqa+: Enhancing code search dataset with matching code. *CoRR*, abs/2406.11589, 2024a. doi: 10.48550/ARXIV.2406.11589. URL https://doi.org/10.48550/arXiv.2406.11589.

Gong, L., Wang, S., Elhoushi, M., and Cheung, A. Evaluation of llms on syntax-aware code fill-in-the-middle tasks, 2024b. URL https://arxiv.org/abs/2403.04814.

Gong, Z., Sun, Z., Huang, D., Liang, Q., Zhang, J. M., and Hao, D. Tracy: Benchmarking execution efficiency of llm-based code translation, 2025. URL https://arxiv.org/abs/2508.11468.

Gu, A., Rozière, B., Leather, H., Solar-Lezama, A., Synnaeve, G., and Wang, S. I. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.

Gu, X., Zhang, H., and Kim, S. Deep code search. In Chaudron, M., Crnkovic, I., Chechik, M., and Harman, M. (eds.), *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pp. 933–944. ACM, 2018. doi: 10.1145/3180155.3180167. URL https://doi.org/10.1145/3180155.3180167.

Gui, Y., Li, Z., Wan, Y., Shi, Y., Zhang, H., Su, Y., Chen, B., Chen, D., Wu, S., Zhou, X., Jiang, W., Jin, H., and Zhang, X. Webcode2m: A real-world dataset for code generation from webpage designs. In *THE WEB CONFERENCE 2025*, 2025. URL https://openreview.net/forum?id=aeP5nmlw5B.

Gunnu, S., Guttula, S., and Patel, H. Cife: Code instruction-following evaluation, 2025. URL https://arxiv.org/abs/2512.17387.

Guo, H., Zheng, X., Liao, Z., Yu, H., DI, P., Zhang, Z., and Dai, H.-N. Codefuse-cr-bench: A comprehensiveness-aware benchmark for end-to-end code review evaluation in python projects, 2025a. URL https://arxiv.org/abs/2509.14856.

Guo, J., Li, Z., Liu, X., Ma, K., Zheng, T., Yu, Z., Pan, D., Li, Y., Liu, R., Wang, Y., Guo, S., Qu, X., Yue, X., Zhang, G., Chen, W., and Fu, J. Codeeditorbench: Evaluating code editing capability of large language models. *CoRR*, abs/2404.03543, 2024. doi: 10.48550/ARXIV.2404.03543. URL https://doi.org/10.48550/arXiv.2404.03543.

Guo, L., Wang, Y., Li, C., Tao, W., Yang, P., Chen, J., Song, H., Tang, D., and Zheng, Z. Swe-factory: Your automated factory for issue resolution training data and evaluation benchmarks, 2026. URL https://arxiv.org/abs/2506.10954.

Guo, X., Wang, M., and Zhao, J. Quanbench: Benchmarking quantum code generation with large language models, 2025b. URL https://arxiv.org/abs/2510.16779.

Gupta, R., Pal, S., Kanade, A., and Shevade, S. K. Deepfix: Fixing common C language errors by deep learning. In Singh, S. and Markovitch, S. (eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 1345–1351. AAAI Press, 2017. doi: 10.1609/AAAI.V31I1.10742. URL https://doi.org/10.1609/aaai.v31i1.10742.

Hai, N. L., Nguyen, D. M., and Bui, N. D. Q. On the impacts of contexts on repository-level code generation, 2024. URL https://arxiv.org/abs/2406.11927.

Haller, P., Golde, J., and Akbik, A. PECC: problem extraction and coding challenges. In Calzolari, N., Kan, M., Hoste, V., Lenci, A., Sakti, S., and Xue, N. (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pp. 12690–12699. ELRA and ICCL, 2024. URL https://aclanthology.org/2024.lrec-main.1111.

Hao, Y., Li, G., Liu, Y., Miao, X., Zong, H., Jiang, S., Liu, Y., and He, W. Aixbench: A code generation benchmark dataset. *CoRR*, abs/2206.13179, 2022. doi: 10.48550/ARXIV.2206.13179. URL https://doi.org/10.48550/arXiv.2206.13179.

Haque, M. M. A., Ahmad, W. U., Lourentzou, I., and Brown, C. Fixeval: Execution-based evaluation of program fixes for programming problems. In *IEEE/ACM International Workshop on Automated Program Repair, APR@ICSE 2023, Melbourne, Australia, May 16, 2023*, pp. 11–18. IEEE, 2023. doi: 10.1109/APR59189.2023.00009. URL https://doi.org/10.1109/APR59189.2023.00009.

Hasan, M., Muttaqueen, T., Ishtiaq, A. A., Mehrab, K. S., Haque, M. M. A., Hasan, T., Ahmad, W. U., Iqbal, A., and Shahriyar, R. Codesc: A large code-description parallel dataset. In Zong, C., Xia, F., Li, W., and Navigli, R. (eds.), *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pp. 210–218. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.FINDINGS-ACL.18. URL https://doi.org/10.18653/v1/2021.findings-acl.18.

Havare, J., Chaudhary, S., Ramakrishnan, G., Maharajan, K., and Tamilselvam, S. A code comprehension benchmark for large language models for code, 2025. URL https://arxiv.org/abs/2507.10641.

Hazoom, M., Malik, V., and Bogin, B. Text-to-sql in the wild: A naturally-occurring dataset based on stack exchange data. *CoRR*, abs/2106.05006, 2021. URL https://arxiv.org/abs/2106.05006.

He, J., Rungta, M., Koleczek, D., Sekhon, A., Wang, F. X., and Hasan, S. Does prompt formatting have any impact on llm performance?, 2024a. URL https://arxiv.org/abs/2411.10541.

He, J., Rungta, M., Koleczek, D., Sekhon, A., Wang, F. X., and Hasan, S. Does prompt formatting have any impact on llm performance? *arXiv preprint arXiv:2411.10541*, 2024b.

He, X., Liu, Q., Du, M., Yan, L., Fan, Z., Huang, Y., Yuan, Z., and Ma, Z. Swe-perf: Can language models optimize code performance on real-world repositories?, 2025a. URL https://arxiv.org/abs/2507.12415.

He, Y., She, H., Qian, X., Zheng, X., Chen, Z., Qin, Z., and Cavallaro, L. On benchmarking code llms for android malware analysis. ISSTA Companion '25, pp. 153–160, New York, NY, USA, 2025b. Association for Computing Machinery. ISBN 9798400714740. doi: 10.1145/3713081.3731745. URL https://doi.org/10.1145/3713081.3731745.

Hellendoorn, V. J., Sutton, C., Singh, R., Maniatis, P., and Bieber, D. Global relational models of source code. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=B1lnbRNtwr.

Hendee, W. R. and Wells, P. N. *The perception of visual information*. Springer Science & Business Media, 1997.

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

Heyman, G. and Cutsem, T. V. Neural code search revisited: Enhancing code snippet retrieval through natural language intent. *CoRR*, abs/2008.12193, 2020. URL https://arxiv.org/abs/2008.12193.

Hodak, M., Ellison, D., Van Buren, C., Jiang, X., and Dholakia, A. Benchmarking large language models: opportunities and challenges. In *Technology Conference on Performance Evaluation and Benchmarking*, pp. 77–89. Springer, 2023.

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J. C., and Wang, H. Large language models for software engineering: A systematic literature review. *CoRR*, abs/2308.10620, 2023.

Hu, R., Wang, X., Wen, X.-C., Zhang, Z., Jiang, B., Gao, P., Peng, C., and Gao, C. Benchmarking llms for fine-grained code review with enriched context in practice, 2025a. URL https://arxiv.org/abs/2511.07017.

Hu, X., Li, G., Xia, X., Lo, D., and Jin, Z. Deep code comment generation. In Khomh, F., Roy, C. K., and Siegmund, J. (eds.), *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018*, pp. 200–210. ACM, 2018a. doi: 10.1145/3196321.3196334. URL https://doi.org/10.1145/3196321.3196334.

Hu, X., Li, G., Xia, X., Lo, D., Lu, S., and Jin, Z. Summarizing source code with transferred API knowledge. In Lang, J. (ed.), *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pp. 2269–2275. ijcai.org, 2018b. doi: 10.24963/IJCAI.2018/314. URL https://doi.org/10.24963/ijcai.2018/314.

Hu, X., Zhao, Z., Wei, S., Chai, Z., Ma, Q., Wang, G., Wang, X., Su, J., Xu, J., Zhu, M., Cheng, Y., Yuan, J., Li, J., Kuang, K., Yang, Y., Yang, H., and Wu, F. Infiagent-dabench: Evaluating agents on data analysis tasks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=d5LURMSfTx.

Hu, X., Niu, F., Chen, J., Zhou, X., Zhang, J., He, J., Xia, X., and Lo, D. Assessing and advancing benchmarks for evaluating large language models in software engineering tasks. *ACM Transactions on Software Engineering and Methodology*, 2025b.

Hu, Y., Ahmed, U. Z., Mechtaev, S., Leong, B., and Roychoudhury, A. 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. doi: 10.1109/ASE.2019.00044.

Hua, T., Hua, H., Xiang, V., Klieger, B., Truong, S. T., Liang, W., Sun, F.-Y., and Haber, N. Researchcodebench: Benchmarking LLMs on implementing novel machine learning research code. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=3k70Vt0YFS.

HUANG, D., QING, Y., Shang, W., Cui, H., and Zhang, J. Effibench: Benchmarking the efficiency of automatically generated code. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=30XanJanJP.

Huang, D., Zhang, J. M., Harman, M., Zhang, Q., Du, M., and Ng, S.-K. Benchmarking llms for unit test generation from real-world functions, 2025. URL https://arxiv.org/abs/2508.00408.

Huang, J., Tang, D., Shou, L., Gong, M., Xu, K., Jiang, D., Zhou, M., and Duan, N. CoSQA: 20,000+ web

queries for code search and question answering. In Zong, C., Xia, F., Li, W., and Navigli, R. (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 5690–5700, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.442. URL https://aclanthology.org/2021.acl-long.442/.

Huang, J., Wang, C., Zhang, J., Yan, C., Cui, H., Inala, J. P., Clement, C. B., Duan, N., and Gao, J. Execution-based evaluation for data science code generation models. *CoRR*, abs/2211.09374, 2022. doi: 10.48550/ARXIV.2211.09374. URL https://doi.org/10.48550/arXiv.2211.09374.

Huang, Y., Lin, Z., Liu, X., Gong, Y., Lu, S., Lei, F., Liang, Y., Shen, Y., Lin, C., Duan, N., and Chen, W. Competition-level problems are effective LLM evaluators. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 13526–13544. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.803. URL https://doi.org/10.18653/v1/2024.findings-acl.803.

Huo, N., Xu, X., Li, J., Jacobsson, P., Lin, S., Qin, B., Hui, B., Li, X., Qu, G., Si, S., Han, L., Alexander, E., Zhu, X., Qin, R., Yu, R., Jin, Y., Zhou, F., Zhong, W., Chen, Y., Liu, H., Ma, C., Ozcan, F., Papakonstantinou, Y., and Cheng, R. Bird-interact: Re-imagining text-to-sql evaluation for large language models via lens of dynamic interactions, 2025. URL https://arxiv.org/abs/2510.05318.

Husain, H., Wu, H., Gazit, T., Allamanis, M., and Brockschmidt, M. Codesearchnet challenge: Evaluating the state of semantic code search. *CoRR*, abs/1909.09436, 2019. URL http://arxiv.org/abs/1909.09436.

Ivanković, M., Petrović, G., Just, R., and Fraser, G. Code coverage at google. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pp. 955–963, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450355728. doi: 10.1145/3338906.3340459. URL https://doi.org/10.1145/3338906.3340459.

Iyer, S., Konstas, I., Cheung, A., and Zettlemoyer, L. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016. doi: 10.18653/V1/P16-1195. URL https://doi.org/10.18653/v1/p16-1195.

Iyer, S., Konstas, I., Cheung, A., and Zettlemoyer, L. Mapping language to code in programmatic context. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J. (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1643–1652, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1192. URL https://aclanthology.org/D18-1192.

Jain, K., Synnaeve, G., and Rozière, B. Testgeneval: A real world unit test generation and test completion benchmark, 2025. URL https://arxiv.org/abs/2410.00752.

Jain, N., Han, K., Gu, A., Li, W., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. *CoRR*, abs/2403.07974, 2024. doi: 10.48550/ARXIV.2403.07974. URL https://doi.org/10.48550/arXiv.2403.07974.

Jiang, H., Chen, Y., Cao, Y., yi Lee, H., and Tan, R. T. Codejudgebench: Benchmarking llm-as-a-judge for coding tasks, 2025a. URL https://arxiv.org/abs/2507.10535.

Jiang, N., Liu, K., Lutellier, T., and Tan, L. Impact of code language models on automated program repair. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pp. 1430–1442. IEEE, 2023. doi: 10.1109/ICSE48619.2023.00125. URL https://doi.org/10.1109/ICSE48619.2023.00125.

Jiang, Y., Yap, R., and Liang, Z. Oss-bench: Benchmark generator for coding llms, 2025b. URL https://arxiv.org/abs/2505.12331.

Jiao, M., Yu, T., Li, X., Qiu, G., Gu, X., and Shen, B. On the evaluation of neural code translation: Taxonomy and benchmark. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*, pp. 1529–1541. IEEE, 2023. doi: 10.1109/ASE56229.2023.00114. URL https://doi.org/10.1109/ASE56229.2023.00114.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Jin, M., Shahriar, S., Tufano, M., Shi, X., Lu, S., Sundaresan, N., and Svyatkovskiy, A. Inferfix: End-to-end program repair with llms. In Chandra, S., Blincoe, K., and Tonella, P. (eds.), *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pp. 1646–1656. ACM, 2023. doi: 10.1145/3611643.3613892. URL https://doi.org/10.1145/3611643.3613892.

Jin, M., Yu, Q., Shu, D., Zhao, H., Hua, W., Meng, Y., Zhang, Y., and Du, M. The impact of reasoning step length on large language models. *arXiv preprint arXiv:2401.04925*, 2024.

Just, R., Jalali, D., and Ernst, M. D. Defects4j: a database of existing faults to enable controlled testing studies for java programs. In Pasareanu, C. S. and Marinov, D. (eds.), *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*, pp. 437–440. ACM, 2014. doi: 10.1145/2610384.2628055. URL https://doi.org/10.1145/2610384.2628055.

Khan, M. A. M., Bari, M. S., Long, X. D., Wang, W., Parvez, M. R., and Joty, S. Xcodeeval: An execution-based large scale multilingual multitask benchmark for code understanding, generation, translation and retrieval. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 6766–6805. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.367. URL https://doi.org/10.18653/v1/2024.acl-long.367.

Khatry, A., Zhang, R., Pan, J., Wang, Z., Chen, Q., Durrett, G., and Dillig, I. CRUST-bench: A comprehensive benchmark for c-to-safe-rust transpilation. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=8xofWL61S9.

Khojah, R., de Oliveira Neto, F. G., Mohamad, M., and Leitner, P. The impact of prompt programming on function-level code generation, 2025. URL https://arxiv.org/abs/2412.20545.

Kim, M., Garg, S., Ray, B., Kumar, V., and Deoras, A. Codeassistbench (CAB): Dataset & benchmarking for multi-turn chat-based code assistance. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=2R6y4Ku9kG.

Koohestani, R., de Bekker, P., Koç, B., and Izadi, M. Benchmarking ai models in software engineering: A review, search tool, and unified approach for elevating benchmark quality. *IEEE Transactions on Software Engineering*, 2025a.

Koohestani, R., de Bekker, P., Koç, B., and Izadi, M. Benchmarking ai models in software engineering: A review, search tool, and unified approach for elevating benchmark quality, 2025b. URL https://arxiv.org/abs/2503.05860.

Kumar, R., Dibbu, A. R., Harsola, S., Subrahmaniam, V., and Modi, A. Booksql: A large scale text-to-sql dataset for accounting domain. In Duh, K., Gómez-Adorno, H., and Bethard, S. (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pp. 497–516. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.NAACL-LONG.28. URL https://doi.org/10.18653/v1/2024.naacl-long.28.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

LaBash, B., Rosedale, A., Reents, A., Negritto, L., and Wiel, C. RES-Q: evaluating code-editing large language model systems at the repository scale. *CoRR*, abs/2406.16801, 2024. doi: 10.48550/ARXIV.2406.16801. URL https://doi.org/10.48550/arXiv.2406.16801.

Lai, Y., Li, C., Wang, Y., Zhang, T., Zhong, R., Zettlemoyer, L., Yih, W., Fried, D., Wang, S. I., and Yu, T. DS-1000: A natural and reliable benchmark for data science code generation. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 18319–18345. PMLR, 2023. URL https://proceedings.mlr.press/v202/lai23b.html.

Le Goues, C., Holtschulte, N. J., Smith, E. K., Brun, Y., Devanbu, P. T., Forrest, S., and Weimer, W. The manybugs and introclass benchmarks for automated repair of C programs. *IEEE Trans. Software Eng.*, 41(12):1236–1256, 2015. doi: 10.1109/TSE.2015.2454513. URL https://doi.org/10.1109/TSE.2015.2454513.

LeClair, A., Jiang, S., and McMillan, C. A neural model for generating natural language summaries of program subroutines. In Atlee, J. M., Bultan, T., and Whittle,

J. (eds.), *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pp. 795–806. IEEE / ACM, 2019. doi: 10.1109/ICSE.2019.00087. URL https://doi.org/10.1109/ICSE.2019.00087.

Lee, C., Polozov, O., and Richardson, M. Kaggledbqa: Realistic evaluation of text-to-sql parsers. In Zong, C., Xia, F., Li, W., and Navigli, R. (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pp. 2261–2273. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.ACL-LONG.176. URL https://doi.org/10.18653/v1/2021.acl-long.176.

Lee, C., Seonwoo, Y., and Oh, A. CS1QA: A dataset for assisting code-based question answering in an introductory programming course. In Carpuat, M., de Marneffe, M., and Ruíz, I. V. M. (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pp. 2026–2040. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.NAACL-MAIN.148. URL https://doi.org/10.18653/v1/2022.naacl-main.148.

Lee, G., Hwang, H., Bae, S., Kwon, Y., Shin, W., Yang, S., Seo, M., Kim, J., and Choi, E. EHRSQL: A practical text-to-sql benchmark for electronic health records. *CoRR*, abs/2301.07695, 2023. doi: 10.48550/ARXIV.2301.07695. URL https://doi.org/10.48550/arXiv.2301.07695.

Lei, F., Chen, J., Ye, Y., Cao, R., Shin, D., Su, H., Suo, Z., Gao, H., Hu, W., Yin, P., Zhong, V., Xiong, C., Sun, R., Liu, Q., Wang, S., and Yu, T. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows, 2025. URL https://arxiv.org/abs/2411.07763.

Li, F., Jiang, J., Sun, J., and Zhang, H. Evaluating the generalizability of llms in automated program repair, 2025a. URL https://arxiv.org/abs/2503.09217.

Li, H. Mrg-bench: Evaluating and exploring the requirements of context for repository-level code generation, 2025. URL https://arxiv.org/abs/2508.02998.

Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K. C., Huang, F., Cheng, R., and Li, Y. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023a. URL http://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.

Li, J., Li, G., Zhao, Y., Li, Y., Liu, H., Zhu, H., Wang, L., Liu, K., Fang, Z., Wang, L., Ding, J., Zhang, X., Zhu, Y., Dong, Y., Jin, Z., Li, B., Huang, F., and Li, Y. DevEval: A Manually-Annotated Code Generation Benchmark Aligned with Real-World Code Repositories, May 2024a. URL http://arxiv.org/abs/2405.19856. arXiv:2405.19856 [cs].

Li, K., Hu, Q., Zhao, J. X., Chen, H., Xie, Y., Liu, T., Shieh, M., and He, J. Instructcoder: Instruction tuning large language models for code editing. In Fu, X. and Fleisig, E. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, ACL 2024 - Student Research Workshop, Bangkok, Thailand, August 11-16, 2024*, pp. 50–70. Association for Computational Linguistics, 2024b. URL https://aclanthology.org/2024.acl-srw.6.

Li, K., Tian, Y., Hu, Q., Luo, Z., and Ma, J. Mmcode: Evaluating multi-modal code large language models with visually rich programming problems. *CoRR*, abs/2404.09486, 2024c. doi: 10.48550/ARXIV.2404.09486. URL https://doi.org/10.48550/arXiv.2404.09486.

Li, L., Geng, S., Li, Z., He, Y., Yu, H., Hua, Z., Ning, G., Wang, S., Xie, T., and Yang, H. Infibench: Evaluating the question-answering capabilities of code large language models, 2024d. URL https://arxiv.org/abs/2404.07940.

Li, R., Fu, J., Zhang, B., Huang, T., Sun, Z., Lyu, C., Liu, G., Jin, Z., and Li, G. TACO: topics in algorithmic code generation dataset. *CoRR*, abs/2312.14852, 2023b. doi: 10.48550/ARXIV.2312.14852. URL https://doi.org/10.48550/arXiv.2312.14852.

Li, S., Jiang, J., Zhao, T., and Shen, J. Osvbench: Benchmarking llms on specification generation tasks for operating system verification, 2025b. URL https://arxiv.org/abs/2504.20964.

Li, W., Zhang, X., Guo, Z., Mao, S., Luo, W., Peng, G., Huang, Y., Wang, H., and Li, S. Fea-bench: A benchmark for evaluating repository-level code generation for feature implementation, 2025c. URL https://arxiv.org/abs/2503.06680.

Li, X., Ding, J., Peng, C., Zhao, B., Gao, X., Gao, H., and Gu, X. Safegenbench: A benchmark framework for security vulnerability detection in llm-generated code, 2025d. URL https://arxiv.org/abs/2506.05692.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A. D., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J., Robson, E. S., Kohli, P., de Freitas, N., Kavukcuoglu, K., and Vinyals, O. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

Li, Y., Tao, R., Hommel, D., Dönder, Y. D., Chang, S., Mimno, D., and Jo, U. E. S. Agent bain vs. agent mckinsey: A new text-to-sql benchmark for the business domain, 2026. URL https://arxiv.org/abs/2510.07309.

Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., Chen, Z., Wang, S., and Wang, J. Sysevr: A framework for using deep learning to detect software vulnerabilities. *CoRR*, abs/1807.06756, 2018a. URL http://arxiv.org/abs/1807.06756.

Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., and Zhong, Y. Vuldeepecker: A deep learning-based system for vulnerability detection. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018b. URL https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_03A-2_Li_paper.pdf.

Li, Z., Zhang, J., Yin, C., Ouyang, Y., and Rong, W. Procqa: A large-scale community-based programming question answering dataset for code search. In Calzolari, N., Kan, M., Hoste, V., Lenci, A., Sakti, S., and Xue, N. (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pp. 13057–13067. ELRA and ICCL, 2024e. URL https://aclanthology.org/2024.lrec-main.1143.

Liang, L., Gong, J., Liu, M., Wang, C., Ou, G., Wang, Y., Peng, X., and Zheng, Z. Rustevo$\hat{2}$: An evolving benchmark for api evolution in llm-based rust code generation, 2025. URL https://arxiv.org/abs/2503.16922.

Liang, Q., Sun, Z., Zhu, Q., Zhang, W., Yu, L., Xiong, Y., and Zhang, L. Lyra: A benchmark for turducken-style code generation. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, IJCAI-2022, pp. 4238–4244. International Joint Conferences on Artificial Intelligence Organization, July 2022. doi: 10.24963/ijcai.2022/588. URL http://dx.doi.org/10.24963/ijcai.2022/588.

Liao, D., Pan, S., Sun, X., Ren, X., Huang, Q., Xing, Z., Jin, H., and Li, Q. A 3-codgen: A repository-level code generation framework for code reuse with local-aware, global-aware, and third-party-library-aware. *IEEE Transactions on Software Engineering*, 2024.

Lin, D., Koppel, J., Chen, A., and Solar-Lezama, A. Quixbugs: a multi-lingual program repair benchmark set based on the quixey challenge. In Murphy, G. C. (ed.), *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity, SPLASH 2017, Vancouver, BC, Canada, October 23 - 27, 2017*, pp. 55–56. ACM, 2017. doi: 10.1145/3135932.3135941. URL https://doi.org/10.1145/3135932.3135941.

Lin, G., Zhang, J., Luo, W., Pan, L., Xiang, Y., de Vel, O. Y., and Montague, P. Cross-project transfer representation learning for vulnerable function discovery. *IEEE Trans. Ind. Informatics*, 14(7):3289–3297, 2018. doi: 10.1109/TII.2018.2821768. URL https://doi.org/10.1109/TII.2018.2821768.

Lin, G., Xiao, W., Zhang, J., and Xiang, Y. Deep learning-based vulnerable function detection: A benchmark. In Zhou, J., Luo, X., Shen, Q., and Xu, Z. (eds.), *Information and Communications Security - 21st International Conference, ICICS 2019, Beijing, China, December 15-17, 2019, Revised Selected Papers*, volume 11999 of *Lecture Notes in Computer Science*, pp. 219–232. Springer, 2019. doi: 10.1007/978-3-030-41579-2_13. URL https://doi.org/10.1007/978-3-030-41579-2_13.

Lin, G., Zhang, J., Luo, W., Pan, L., de Vel, O. Y., Montague, P., and Xiang, Y. Software vulnerability discovery via learning multi-domain knowledge bases. *IEEE Trans. Dependable Secur. Comput.*, 18(5):2469–2485, 2021. doi: 10.1109/TDSC.2019.2954088. URL https://doi.org/10.1109/TDSC.2019.2954088.

Lin, H. Y., Liu, C., Gao, H., Thongtanunam, P., and Treude, C. CodeReviewQA: The code review comprehension assessment for large language models. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 9138–9166, Vienna, Austria, July

2025a. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.476. URL https://aclanthology.org/2025.findings-acl.476/.

Lin, Z., Zhou, Z., Zhao, Z., Wan, T., Ma, Y., Gao, J., and Li, X. WebUIBench: A comprehensive benchmark for evaluating multimodal large language models in WebUI-to-code. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 15780–15797, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.815. URL https://aclanthology.org/2025.findings-acl.815/.

Ling, X., Wu, L., Wang, S., Pan, G., Ma, T., Xu, F., Liu, A. X., Wu, C., and Ji, S. Deep graph matching and searching for semantic code retrieval. *ACM Trans. Knowl. Discov. Data*, 15(5):88:1–88:21, 2021. doi: 10.1145/3447571. URL https://doi.org/10.1145/3447571.

Liu, C. and Wan, X. Codeqa: A question answering dataset for source code comprehension. In Moens, M., Huang, X., Specia, L., and Yih, S. W. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pp. 2618–2632. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.FINDINGS-EMNLP.223. URL https://doi.org/10.18653/v1/2021.findings-emnlp.223.

Liu, C., Ghazanfari, A., Chen, Y., and Jabbarvand, R. Evaluating code reasoning abilities of large language models under real-world settings, 2025a. URL https://arxiv.org/abs/2512.14917.

Liu, E. T., Wang, A., Mateega, S., Georgescu, C., and Tang, D. Vader: A human-evaluated benchmark for vulnerability assessment, detection, explanation, and remediation, 2025b. URL https://arxiv.org/abs/2505.19395.

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023a.

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL https://openreview.net/forum?id=1qvx610Cu7.

Liu, J., Deng, K., Liu, C., Yang, J., Liu, S., Zhu, H., Zhao, P., Chai, L., Wu, Y., Jin, K., Zhang, G., Wang, Z., Zhang, G., Xiang, B., Su, W., and Zheng, B. M2rc-eval: Massively multilingual repository-level code completion evaluation, 2024a. URL https://arxiv.org/abs/2410.21157.

Liu, J., Tian, J. L., Daita, V., Wei, Y., Ding, Y., Wang, Y. K., Yang, J., and Zhang, L. Repoqa: Evaluating long context code understanding. *CoRR*, abs/2406.06025, 2024b. doi: 10.48550/ARXIV.2406.06025. URL https://doi.org/10.48550/arXiv.2406.06025.

Liu, J., Huang, C., Guan, Z., Lei, W., and Deng, Y. E2edev: Benchmarking large language models in end-to-end software development task, 2025c. URL https://arxiv.org/abs/2510.14509.

Liu, K., Pan, Y., Xiang, Y., He, D., Li, J., Du, Y., and Gao, T. Projecteval: A benchmark for programming agents automated evaluation on project-level code generation, 2025d. URL https://arxiv.org/abs/2503.07010.

Liu, S., Chen, Y., Xie, X., Siow, J. K., and Liu, Y. Retrieval-augmented generation for code summarization via hybrid GNN. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=zv-typ1gPxA.

Liu, S., Gao, C., Chen, S., Nie, L. Y., and Liu, Y. ATOM: commit message generation based on abstract syntax tree and hybrid ranking. *IEEE Trans. Software Eng.*, 48(5):1800–1817, 2022. doi: 10.1109/TSE.2020.3038681. URL https://doi.org/10.1109/TSE.2020.3038681.

Liu, S., Chai, L., Yang, J., Shi, J., Zhu, H., Wang, L., Jin, K., Zhang, W., Zhu, H., Guo, S., Sun, T., Liu, J., Duan, Y., Hao, Y., Yang, L., Niu, G., Zhang, G., and Li, Z. Mdeval: Massively multilingual code debugging, 2025e. URL https://arxiv.org/abs/2411.02310.

Liu, T., Xu, C., and McAuley, J. J. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024c. URL https://openreview.net/forum?id=pPjZIOuQuF.

Liu, T., Mao, X., Zan, H., Zhang, D., Li, Y., Liu, H., Kong, L., Hou, J., Li, R., Li, Y., aoze zheng, Zhang, Z., Zhewei,

L., Zhang, K., and Peng, M. Logiccat: A chain-of-thought text-to-sql benchmark for complex reasoning, 2025f. URL https://arxiv.org/abs/2505.18744.

Liu, Y., Tang, X., Cai, Z., Lu, J., Zhang, Y., Shao, Y., Deng, Z., Hu, H., Yang, Z., An, K., Huang, R., Si, S., Chen, S., Zhao, H., Li, Z., Chen, L., Zong, Y., Wang, Y., Liu, T., Jiang, Z., Chang, B., Qin, Y., Zhou, W., Zhao, Y., Cohan, A., and Gerstein, M. Ml-bench: Large language models leverage open-source libraries for machine learning tasks. *CoRR*, abs/2311.09835, 2023c. doi: 10.48550/ARXIV.2311.09835. URL https://doi.org/10.48550/arXiv.2311.09835.

Liu, Y., Gao, L., Yang, M., Xie, Y., Chen, P., Zhang, X., and Chen, W. Vuldetectbench: Evaluating the deep capability of vulnerability detection with large language models. *CoRR*, abs/2406.07595, 2024d. doi: 10.48550/ARXIV.2406.07595. URL https://doi.org/10.48550/arXiv.2406.07595.

Liu, Z., Xia, X., Hassan, A. E., Lo, D., Xing, Z., and Wang, X. Neural-machine-translation-based commit message generation: how far are we? In Huchard, M., Kästner, C., and Fraser, G. (eds.), *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pp. 373–384. ACM, 2018. doi: 10.1145/3238147.3238190. URL https://doi.org/10.1145/3238147.3238190.

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C. B., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S. K., Fu, S., and Liu, S. Codexglue: A machine learning benchmark dataset for code understanding and generation. In Vanschoren, J. and Yeung, S. (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.

Lu, Z., Yang, Y., Ren, H., Hou, H., Xiao, H., Wang, K., Shi, W., Zhou, A., Zhan, M., and Li, H. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch, 2025. URL https://arxiv.org/abs/2505.03733.

Luo, W., Guan, W., Yao, Y., Pan, Y., Wang, F., Yu, Z., Wen, Z., Chen, L., and Zhuang, Y. Falcon: A comprehensive chinese text-to-sql benchmark for enterprise-grade evaluation, 2025a. URL https://arxiv.org/abs/2510.24762.

Luo, X., Huang, J., Zheng, W., Zhu, Q., Xu, M., Xu, Y., Fan, Y., Qin, L., and Che, W. How many code and test cases are enough? evaluating test cases generation from a binary-matrix perspective, 2025b. URL https://arxiv.org/abs/2510.08720.

Ma, Z., An, S., Xie, B., and Lin, Z. Compositional api recommendation for library-oriented code generation. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, ICPC '24, pp. 87–98. ACM, April 2024. doi: 10.1145/3643916.3644403. URL http://dx.doi.org/10.1145/3643916.3644403.

Malik, R. S., Patra, J., and Pradel, M. Nl2type: inferring javascript function types from natural language information. In Atlee, J. M., Bultan, T., and Whittle, J. (eds.), *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pp. 304–315. IEEE / ACM, 2019. doi: 10.1109/ICSE.2019.00045. URL https://doi.org/10.1109/ICSE.2019.00045.

Malode, V. M. *Benchmarking public large language model*. PhD thesis, Technische Hochschule Ingolstadt, 2024.

May, V., Misra, D., Luo, Y., Sridhar, A., Gehring, J., and Junior, S. S. R. Freshbrew: A benchmark for evaluating ai agents on java code migration, 2025. URL https://arxiv.org/abs/2510.04852.

McIntosh, T. R., Susnjak, T., Arachchilage, N., Liu, T., Xu, D., Watters, P., and Halgamuge, M. N. Inadequacies of large language model benchmarks in the era of generative artificial intelligence. *IEEE Transactions on Artificial Intelligence*, 2025.

Mehralian, F., Shar, R., Rae, J. R., and Hashemi, A. Codealignbench: Assessing code generation models on developer-preferred code adjustments, 2025. URL https://arxiv.org/abs/2510.27565.

Miao, C., Zou, H. P., Li, Y., Chen, Y., Wang, Y., Wang, F., Li, Y., Yang, W., He, B., Zhang, X., Yu, D., Yang, H., Nguyen, H. H., Zhou, Y., Yang, J., Guo, J., Fan, W., Yeh, C.-Y., Meng, P., Fang, L., Qi, J., Huang, W.-C., Gu, Z., Han, Y., He, L., Yang, Y., Li, Y., Zheng, H.-T., Liu, X., King, I., and Yu, P. S. Recode-h: A benchmark for research code development with interactive human feedback, 2025. URL https://arxiv.org/abs/2510.06186.

Miller, C., Cohen, S., Klug, D., Vasilescu, B., and KaUstner, C. "did you miss my comment or what?": understanding toxicity in open source discussions. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, pp. 710–722, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392211. doi: 10.1145/3510003.3510111. URL https://doi.org/10.1145/3510003.3510111.

Mir, A. M., Latoskinas, E., Proksch, S., and Gousios, G. Type4py: Practical deep similarity learning-based type inference for python. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pp. 2241–2252. ACM, 2022. doi: 10.1145/3510003.3510124. URL https://doi.org/10.1145/3510003.3510124.

Moriarty, J. P. A theory of benchmarking. *Benchmarking: An International Journal*, 18(4):588–611, 2011.

Mou, L., Li, G., Zhang, L., Wang, T., and Jin, Z. Convolutional neural networks over tree structures for programming language processing. In Schuurmans, D. and Wellman, M. P. (eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pp. 1287–1293. AAAI Press, 2016. doi: 10.1609/AAAI.V30I1.10139. URL https://doi.org/10.1609/aaai.v30i1.10139.

Mozannar, H., Chen, V., Alsobay, M., Das, S., Zhao, S., Wei, D., Nagireddy, M., Sattigeri, P., Talwalkar, A., and Sontag, D. A. The realhumaneval: Evaluating large language models' abilities to support programmers. *CoRR*, abs/2404.02806, 2024. doi: 10.48550/ARXIV.2404.02806. URL https://doi.org/10.48550/arXiv.2404.02806.

Muennighoff, N., Liu, Q., Zebaze, A. R., Zheng, Q., Hui, B., Zhuo, T. Y., Singh, S., Tang, X., von Werra, L., and Longpre, S. Octopack: Instruction tuning code large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=mw1PWNSWZP.

Mündler, N., Müller, M. N., He, J., and Vechev, M. Swt-bench: Testing and validating real-world bug-fixes with code agents, 2025. URL https://arxiv.org/abs/2406.12952.

Nguyen, H. A., Nguyen, T. N., Dig, D., Nguyen, S., Tran, H., and Hilton, M. Graph-based mining of in-the-wild, fine-grained, semantic code change patterns. In Atlee, J. M., Bultan, T., and Whittle, J. (eds.), *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pp. 819–830. IEEE / ACM, 2019. doi: 10.1109/ICSE.2019.00089. URL https://doi.org/10.1109/ICSE.2019.00089.

Nichols, D., Davis, J. H., Xie, Z., Rajaram, A., and Bhatele, A. Can large language models write parallel code? In Dazzi, P., Mencagli, G., Lowenthal, D. K., and Badia, R. M. (eds.), *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2024, Pisa, Italy, June 3-7, 2024*, pp. 281–294. ACM, 2024. doi: 10.1145/3625549.3658689. URL https://doi.org/10.1145/3625549.3658689.

Nie, P., Banerjee, R., Li, J. J., Mooney, R. J., and Gligoric, M. Learning deep semantics for test completion. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pp. 2111–2123. IEEE, 2023. doi: 10.1109/ICSE48619.2023.00178. URL https://doi.org/10.1109/ICSE48619.2023.00178.

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., and Xiong, C. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

Nikiema, S. L., Samhi, J., Kaboré, A. K., Klein, J., and Bissyandé, T. F. The code barrier: What llms actually understand?, 2025. URL https://arxiv.org/abs/2504.10557.

Nikitopoulos, G., Dritsa, K., Louridas, P., and Mitropoulos, D. Crossvul: a cross-language vulnerability dataset with commit data. In Spinellis, D., Gousios, G., Chechik, M., and Penta, M. D. (eds.), *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, pp. 1565–1569. ACM, 2021. doi: 10.1145/3468264.3473122. URL https://doi.org/10.1145/3468264.3473122.

Oh, W. and Oh, H. Pyter: effective program repair for python type errors. In Roychoudhury, A., Cadar, C., and Kim, M. (eds.), *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, pp. 922–934. ACM, 2022. doi: 10.1145/3540250.3549130. URL https://doi.org/10.1145/3540250.3549130.

Ou, G., Liu, M., Chen, Y., Wang, Y., Peng, X., and Zheng, Z. Rustrepotrans: Repository-level code translation benchmark targeting rust, 2025. URL https://arxiv.org/abs/2411.13990.

Ouyang, S., Huang, D., Guo, J., Sun, Z., Zhu, Q., and Zhang, J. M. Dscodebench: A realistic benchmark for data science code generation, 2025. URL https://arxiv.org/abs/2505.15621.

Oza, N., Govil, I., Gupta, P., Khandelwal, D., Garg, D., and Singla, P. Cetbench: A novel dataset constructed via transformations over programs for benchmarking llms for code-equivalence checking, 2025. URL https://arxiv.org/abs/2506.04019.

Pan, R., Ibrahimzada, A. R., Krishna, R., Sankar, D., Wassi, L. P., Merler, M., Sobolev, B., Pavuluri, R., Sinha, S., and Jabbarvand, R. Lost in translation: A study of bugs introduced by large language models while translating code. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3639226. URL https://doi.org/10.1145/3597503.3639226.

Pan, Z., Cao, R., Cao, Y., Ma, Y., Li, B., Huang, F., Liu, H., and Li, Y. Codev-bench: How do llms understand developer-centric code completion?, 2024b. URL https://arxiv.org/abs/2410.01353.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. Gorilla: Large language model connected with massive apis. *CoRR*, abs/2305.15334, 2023. doi: 10.48550/ARXIV.2305.15334. URL https://doi.org/10.48550/arXiv.2305.15334.

Paul, D. G., Zhu, H., and Bayley, I. Sceneval: A benchmark for scenario-based evaluation of code generation. In *IEEE International Conference on Artificial Intelligence Testing, AITest 2024, Shanghai, China, July 15-18, 2024*, pp. 55–63. IEEE, 2024. doi: 10.1109/AITEST62860.2024.00015. URL https://doi.org/10.1109/AITest62860.2024.00015.

Pavel, A., Mikhail, I., Valeev, A., Levichev, R., Zadorozhny, P., Lopatin, I., Babaev, D., Fenogenova, A., and Malykh, V. SWE-MERA: A dynamic benchmark for agenticly evaluating large language models on software engineering tasks. In Habernal, I., Schulam, P., and Tiedemann, J. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 440–452, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-334-0. doi: 10.18653/v1/2025.emnlp-demos.30. URL https://aclanthology.org/2025.emnlp-demos.30/.

Peng, J., Cui, L., Huang, K., Yang, J., and Ray, B. Cweval: Outcome-driven evaluation on functionality and security of llm code generation, 2025a. URL https://arxiv.org/abs/2501.08200.

Peng, Q., Chai, Y., and Li, X. HumanEval-XL: A multilingual code generation benchmark for cross-lingual natural language generalization. In Calzolari, N., Kan, M.-Y., Hoste, V., Lenci, A., Sakti, S., and Xue, N. (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 8383–8394, Torino, Italia, May 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.735/.

Peng, W., Shi, Y., Wang, Y., Zhang, X., Shen, B., and Gu, X. Swe-qa: Can language models answer repository-level code questions?, 2025b. URL https://arxiv.org/abs/2509.14635.

Peng, Y., Wan, J., Li, Y., and Ren, X. Coffe: A code efficiency benchmark for code generation. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025c. doi: 10.1145/3715727. URL https://doi.org/10.1145/3715727.

Peng, Z., Yin, X., Qian, R., Lin, P., Liu, Y., Zhang, H., Ying, C., and Luo, Y. Soleval: Benchmarking large language models for repository-level solidity code generation, 2025d. URL https://arxiv.org/abs/2502.18793.

Petrukha, I., Kurliak, Y., and Stulova, N. Swifteval: Developing a language-specific benchmark for llm-generated code evaluation, 2025. URL https://arxiv.org/abs/2505.24324.

Pham, K. T., Nguyen, T. H., Jo, J., Nguyen, Q. V. H., and Nguyen, T. T. Multilingual text-to-sql: Benchmarking the limits of language models with collaborative language agents, 2025. URL https://arxiv.org/abs/2509.24405.

Pinckney, N. R., Batten, C., Liu, M., Ren, H., and Khailany, B. Revisiting verilogeval: Newer llms, in-context learning, and specification-to-rtl tasks. *CoRR*, abs/2408.11053, 2024. doi: 10.48550/ARXIV.2408.11053. URL https://doi.org/10.48550/arXiv.2408.11053.

Pipalani, Y., Raj, H., Ghosh, R., Bhargava, V., and Dutta, D. Go-ut-bench: A fine-tuning dataset for llm-based unit test generation in go, 2025. URL https://arxiv.org/abs/2511.10868.

Prenner, J. A. and Robbes, R. Runbugrun - an executable dataset for automated program repair. *CoRR*, abs/2304.01102, 2023. doi: 10.48550/ARXIV.2304.01102. URL https://doi.org/10.48550/arXiv.2304.01102.

Prenner, J. A. and Robbes, R. Throwbench: Benchmarking llms by predicting runtime exceptions, 2025. URL https://arxiv.org/abs/2503.04241.

Prenner, J. A., Babii, H., and Robbes, R. Can openai's codex fix bugs?: An evaluation on quixbugs. In *3rd IEEE/ACM International Workshop on Automated Program Repair, APR@ICSE 2022, Pittsburgh, PA, USA, May 19, 2022*, pp. 69–75. IEEE, 2022. doi: 10.1145/3524459.3527351. URL https://doi.org/10.1145/3524459.3527351.

Pushkar, C., Kabra, S., Kumar, D., and Challa, J. S. Beyond single bugs: Benchmarking large language models for multi-vulnerability detection, 2025a. URL https://arxiv.org/abs/2512.22306.

Pushkar, C., Kabra, S., Kumar, D., and Challa, J. S. Beyond single bugs: Benchmarking large language models for multi-vulnerability detection, 2025b. URL https://arxiv.org/abs/2512.22306.

Qian, Q., Huang, C., Xu, J., Lv, C., Wu, M., Liu, W., Wang, X., Wang, Z., Huang, Z., Tian, M., et al. Benchmarkˆ 2: Systematic evaluation of llm benchmarks. *arXiv preprint arXiv:2601.03986*, 2026.

Qing, Y., Zhu, B., Du, M., Guo, Z., Zhuo, T. Y., Zhang, Q., Zhang, J. M., Cui, H., Yiu, S.-M., Huang, D., Ng, S.-K., and Tuan, L. A. Effibench-x: A multi-language benchmark for measuring efficiency of llm-generated code, 2025. URL https://arxiv.org/abs/2505.13004.

Qiu, J., Liu, Z., Liu, Z., Murthy, R., Zhang, J., Chen, H., Wang, S., Zhu, M., Yang, L., Tan, J., Cen, Z., Qian, C., Heinecke, S., Yao, W., Savarese, S., Xiong, C., and Wang, H. Locobench: A benchmark for long-context large language models in complex software engineering, 2025. URL https://arxiv.org/abs/2509.09614.

Qiu, R., Zeng, W. W., Tong, H., Ezick, J., and Lott, C. How efficient is llm-generated code? A rigorous & high-standard benchmark. *CoRR*, abs/2406.06647, 2024a. doi: 10.48550/ARXIV.2406.06647. URL https://doi.org/10.48550/arXiv.2406.06647.

Qiu, R., Zeng, W. W., Tong, H., Ezick, J., and Lott, C. How efficient is llm-generated code? a rigorous & high-standard benchmark. *arXiv preprint arXiv:2406.06647*, 2024b.

Quan, S., Yang, J., Yu, B., Zheng, B., Liu, D., Yang, A., Ren, X., Gao, B., Miao, Y., Feng, Y., Wang, Z., Yang, J., Cui, Z., Fan, Y., Zhang, Y., Hui, B., and Lin, J. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings, 2025. URL https://arxiv.org/abs/2501.01257.

Rahman, M., Khatoonabadi, S., and Shihab, E. Beyond synthetic benchmarks: Evaluating llm performance on real-world class-level code generation, 2025a. URL https://arxiv.org/abs/2510.26130.

Rahman, S., Hameed, A., Srivastava, G., and Danish, S. M. Refactorcoderqa: Benchmarking llms for multi-domain coding question solutions in cloud and edge deployment, 2025b. URL https://arxiv.org/abs/2509.10436.

Raihan, N., Anastasopoulos, A., and Zampieri, M. mHumanEval - a multilingual benchmark to evaluate large language models for code generation. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11432–11461, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.570. URL https://aclanthology.org/2025.naacl-long.570/.

Ran, H., Zhang, H., and Tang, X. Gdpr-bench-android: A benchmark for evaluating automated gdpr compliance detection in android, 2025. URL https://arxiv.org/abs/2511.00619.

Rando, S., Romani, L., Sampieri, A., Franco, L., Yang, J., Kyuragi, Y., Galasso, F., and Hashimoto, T. Longcodebench: Evaluating coding llms at 1m context windows, 2025. URL https://arxiv.org/abs/2505.07897.

Rashid, M. S., Bock, C., Zhuang, Y., Buchholz, A., Esler, T., Valentin, S., Franceschi, L., Wistuba, M., Sivaprasad, P. T., Kim, W. J., Deoras, A., Zappella, G., and Callot, L. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents, 2025. URL https://arxiv.org/abs/2504.08703.

Reuel, A., Hardy, A., Smith, C., Lamparth, M., Hardy, M., and Kochenderfer, M. J. Betterbench: Assessing ai benchmarks, uncovering issues, and establishing best practices, 2024. URL https://arxiv.org/abs/2411.12990.

Risse, N. and Böhme, M. Uncovering the limits of machine learning for automatic vulnerability detection. In Balzarotti, D. and Xu, W. (eds.), *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024. URL https://www.usenix.org/conference/usenixsecurity24/presentation/risse.

Rondon, P., Wei, R., Cambronero, J., Cito, J., Sun, A., Sanyam, S., Tufano, M., and Chandra, S. Evaluating agent-based program repair at google, 2025. URL https://arxiv.org/abs/2501.07531.

Roy, M. K., Chen, S., Steenhoek, B., Peng, J., Kaiser, G., Ray, B., and Le, W. Codesense: a real-world benchmark and dataset for code semantic reasoning, 2025. URL https://arxiv.org/abs/2506.00750.

Rozière, B., Lachaux, M., Chanussot, L., and Lample, G. Unsupervised translation of programming languages. In

Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Rozière, B., Zhang, J., Charton, F., Harman, M., Synnaeve, G., and Lample, G. Leveraging automated unit tests for unsupervised code translation. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=cmt-6KtR4c4.

Russell, R. L., Kim, L. Y., Hamilton, L. H., Lazovich, T., Harer, J., Ozdemir, O., Ellingwood, P. M., and McConley, M. W. Automated vulnerability detection in source code using deep representation learning. In Wani, M. A., Kantardzic, M. M., Mouchaweh, M. S., Gama, J., and Lughofer, E. (eds.), *17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA, December 17-20, 2018*, pp. 757–762. IEEE, 2018. doi: 10.1109/ICMLA.2018.00120. URL https://doi.org/10.1109/ICMLA.2018.00120.

Ryu, J., Cho, S., Lee, G., and Choi, E. Ehr-seqsql : A sequential text-to-sql dataset for interactively exploring electronic health records. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 16388–16407. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.971. URL https://doi.org/10.18653/v1/2024.findings-acl.971.

Safdar, R., Mateen, D., Ali, S. T., Ashfaq, M. U., and Hussain, W. Data and context matter: Towards generalizing ai-based software vulnerability detection, 2025. URL https://arxiv.org/abs/2508.16625.

Sainz, O., Campos, J., García-Ferrero, I., Etxaniz, J., de Lacalle, O. L., and Agirre, E. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10776–10787, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.722. URL https://aclanthology.org/2023.findings-emnlp.722.

Saparina, I. and Lapata, M. AMBROSIA: A benchmark for parsing ambiguous questions into database queries. *CoRR*, abs/2406.19073, 2024. doi: 10.48550/ARXIV.2406.19073. URL https://doi.org/10.48550/arXiv.2406.19073.

Schäfer, M., Nadi, S., Eghbali, A., and Tip, F. An empirical evaluation of using large language models for automated unit test generation. *IEEE Trans. Software Eng.*, 50(1):85–105, 2024. doi: 10.1109/TSE.2023.3334955. URL https://doi.org/10.1109/TSE.2023.3334955.

Schall, M., Czinczoll, T., and de Melo, G. Commitbench: A benchmark for commit message generation. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2024, Rovaniemi, Finland, March 12-15, 2024*, pp. 728–739. IEEE, 2024. doi: 10.1109/SANER60148.2024.00080. URL https://doi.org/10.1109/SANER60148.2024.00080.

Schäfer, M., Nadi, S., Eghbali, A., and Tip, F. An empirical evaluation of using large language models for automated unit test generation. *IEEE Transactions on Software Engineering*, 50(1):85–105, 2024. doi: 10.1109/TSE.2023.3334955.

Shang, X., Fu, Z., Cheng, S., Chen, G., Li, G., Hu, L., Zhang, W., and Yu, N. An empirical study on the effectiveness of large language models for binary code understanding, 2025. URL https://arxiv.org/abs/2504.21803.

Shehada, K., Wu, Y., Feng, W. D., Iyer, A., Kumfert, G., Ding, Y., and Qian, Z. Rethinking kernel program repair: Benchmarking and enhancing LLMs with RGym. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*, 2025. URL https://openreview.net/forum?id=NY4wv5C39G.

Shen, C., Dilgren, C., Chiniya, P., Griffith, L., Ding, Y., and Chen, Y. Secrepobench: Benchmarking code agents for secure code completion in real-world repositories, 2025. URL https://arxiv.org/abs/2504.21205.

Sheokand, M. and Sawant, P. Codemixbench: Evaluating large language models on code generation with code-mixed prompts, 2025. URL https://arxiv.org/abs/2505.05063.

Shi, C., Yang, C., Liu, Y., Shui, B., Wang, J., Jing, M., Xu, L., Zhu, X., Li, S., Zhang, Y., Liu, G., Nie, X., Cai, D., and Yang, Y. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. *CoRR*, abs/2406.09961, 2024a. doi: 10.48550/ARXIV.2406.09961. URL https://doi.org/10.48550/arXiv.2406.09961.

Shi, Q., Tang, M., Narasimhan, K., and Yao, S. Can language models solve olympiad programming? *CoRR*, abs/2404.10952, 2024b. doi: 10.48550/ARXIV.2404.10952. URL https://doi.org/10.48550/arXiv.2404.10952.

Shi, T., Zhao, C., Boyd-Graber, J. L., III, H. D., and Lee, L. On the potential of lexico-logical alignments for semantic parsing to SQL queries. In Cohn, T., He, Y., and Liu, Y. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pp. 1849–1864. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.FINDINGS-EMNLP.167. URL https://doi.org/10.18653/v1/2020.findings-emnlp.167.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: language agents with verbal reinforcement learning. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Shrivastava, D., Kocetkov, D., de Vries, H., Bahdanau, D., and Scholak, T. Repofusion: Training code models to understand your repository. *CoRR*, abs/2306.10998, 2023a. doi: 10.48550/ARXIV.2306.10998. URL https://doi.org/10.48550/arXiv.2306.10998.

Shrivastava, D., Larochelle, H., and Tarlow, D. Repository-level prompt generation for large language models of code. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31693–31715. PMLR, 2023b. URL https://proceedings.mlr.press/v202/shrivastava23a.html.

Shypula, A., Madaan, A., Zeng, Y., Alon, U., Gardner, J. R., Yang, Y., Hashemi, M., Neubig, G., Ranganathan, P., Bastani, O., and Yazdanbakhsh, A. Learning performance-improving code edits. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=ix7rLVHXyY.

Si, C., Zhang, Y., Yang, Z., Liu, R., and Yang, D. Design2code: How far are we from automating front-end engineering? *CoRR*, abs/2403.03163, 2024. doi: 10.48550/ARXIV.2403.03163. URL https://doi.org/10.48550/arXiv.2403.03163.

Singhal, M., Aggarwal, T., Awasthi, A., Natarajan, N., and Kanade, A. Nofuneval: Funny how code lms falter on requirements beyond functional correctness. *CoRR*, abs/2401.15963, 2024. doi: 10.48550/ARXIV.2401.15963. URL https://doi.org/10.48550/arXiv.2401.15963.

Sirlanci, M., Yagemann, C., and Lin, Z. C2rust-bench: A minimized, representative dataset for c-to-rust transpilation evaluation, 2025. URL https://arxiv.org/abs/2504.15144.

Sun, S., Zhang, S., Yan, J., Yan, J., and Zhang, J. Co-evolution of types and dependencies: Towards repository-level type inference for python code, 2025a. URL https://arxiv.org/abs/2512.21591.

Sun, Y., Wu, D., Xue, Y., Liu, H., Ma, W., Zhang, L., Liu, Y., and Li, Y. Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning, 2025b. URL https://arxiv.org/abs/2401.16185.

Suppes, P., Zinnes, J. L., et al. *Basic measurement theory*. 1962.

Svajlenko, J., Islam, J. F., Keivanloo, I., Roy, C. K., and Mia, M. M. Towards a big data curated benchmark of inter-project code clones. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, pp. 476–480. IEEE Computer Society, 2014. doi: 10.1109/ICSME.2014.77. URL https://doi.org/10.1109/ICSME.2014.77.

Syromiatnikov, M. V. and Ruvinskaya, V. M. Ua-codebench: A competitive programming benchmark for evaluating large language models code generation in ukrainian. *Informatics Culture Technology*, 2:308–314, November 2025. ISSN 2522-1523. doi: 10.15276/ict.02.2025.47. URL http://dx.doi.org/10.15276/ict.02.2025.47.

Tang, J., Zhao, H. H., Wu, L., Tao, Y., Mao, D., Wan, Y., Tan, J., Zeng, M., Li, M., and Wang, A. J. From charts to code: A hierarchical benchmark for multimodal models, 2026. URL https://arxiv.org/abs/2510.17932.

Tang, X., Qian, B., Gao, R., Chen, J., Chen, X., and Gerstein, M. B. Biocoder: a benchmark for bioinformatics code generation with large language models. *Bioinformatics*, 40(Supplement_1):i266–i276, 2024.

Tao, Q., Yu, T., Gu, X., and Shen, B. Unraveling the potential of large language models in code translation: How far are we?, 2024. URL https://arxiv.org/abs/2410.09812.

Tao, W., Wang, Y., Shi, E., Du, L., Han, S., Zhang, H., Zhang, D., and Zhang, W. A large-scale empirical study of commit message generation: models, datasets and evaluation. *Empir. Softw. Eng.*, 27(7):198, 2022. doi: 10.1007/S10664-022-10219-1. URL https://doi.org/10.1007/s10664-022-10219-1.

Thai, M. V. T., Le, T., Manh, D. N., Nhat, H. P., and Bui, N. D. Q. Swe-evo: Benchmarking coding agents in long-horizon software evolution scenarios, 2026. URL https://arxiv.org/abs/2512.18470.

Tian, R., Ye, Y., Qin, Y., Cong, X., Lin, Y., Pan, Y., Wu, Y., Hui, H., Liu, W., Liu, Z., and Sun, M. Debugbench: Evaluating debugging capability of large language models. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 4173–4198. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.247. URL https://doi.org/10.18653/v1/2024.findings-acl.247.

Tufano, M., Watson, C., Bavota, G., Penta, M. D., White, M., and Poshyvanyk, D. An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Trans. Softw. Eng. Methodol.*, 28(4):19:1–19:29, 2019. doi: 10.1145/3340544. URL https://doi.org/10.1145/3340544.

Tufano, M., Deng, S. K., Sundaresan, N., and Svyatkovskiy, A. METHODS2TEST: A dataset of focal methods mapped to test cases. In *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*, pp. 299–303. ACM, 2022. doi: 10.1145/3524842.3528009. URL https://doi.org/10.1145/3524842.3528009.

Venkatesh, A. P. S., Sunil, R., Sabu, S., Mir, A. M., Reis, S., and Bodden, E. An empirical study of large language models for type and call graph analysis in python and javascript, 2025. URL https://arxiv.org/abs/2410.00603.

Vergopoulos, K., Mueller, M. N., and Vechev, M. Automated benchmark generation for repository-level coding tasks. In *ICLR 2025 Workshop on Foundation Models in the Wild*, 2025. URL https://openreview.net/forum?id=KzKuI0s2Op.

Vijayaraghavan, P., Shi, L., Ambrogio, S., Mackin, C., Nitsure, A., Beymer, D., and Degan, E. Vhdl-eval: A framework for evaluating large language models in VHDL code generation. *CoRR*, abs/2406.04379, 2024. doi: 10.48550/ARXIV.2406.04379. URL https://doi.org/10.48550/arXiv.2406.04379.

Wan, Y., Zhao, Z., Yang, M., Xu, G., Ying, H., Wu, J., and Yu, P. S. Improving automatic source code summarization via deep reinforcement learning. In Huchard, M., Kästner, C., and Fraser, G. (eds.), *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pp. 397–407. ACM, 2018. doi: 10.1145/3238147.3238206. URL https://doi.org/10.1145/3238147.3238206.

Wang, C., Qiu, G., Gu, X., and Shen, B. Apirat: Integrating multi-source api knowledge for enhanced code translation with llms, 2025a. URL https://arxiv.org/abs/2504.14852.

Wang, H., Song, Y., Yin, X., and Chen, X. Beyond select: A comprehensive taxonomy-guided benchmark for real-world text-to-sql translation, 2025b. URL https://arxiv.org/abs/2511.13590.

Wang, H., Zhou, X., Xu, Z., Cheng, K., Zuo, Y., Tian, K., Song, J., Lu, J., Hu, W., and Liu, X. Code-vision: Evaluating multimodal llms logic understanding and code generation capabilities, 2025c. URL https://arxiv.org/abs/2502.11829.

Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., and Wang, Q. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering*, 2024a.

Wang, J., Luo, X., Cao, L., He, H., Huang, H., Xie, J., Jatowt, A., and Cai, Y. Is your ai-generated code really safe? evaluating large language models on secure code generation with codeseceval, 2024b. URL https://arxiv.org/abs/2407.02395.

Wang, J., Xie, X., Hu, Q., Liu, S., Yu, J., Kong, J., and Li, Y. Defects4C: Benchmarking large language model repair capability with C/C++ bugs. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, November 2025d.

Wang, J., Zan, D., Xin, S., Liu, S., Wu, Y., and Shen, K. Swe-mirror: Scaling issue-resolving datasets by mirroring issues across repositories, 2025e. URL https://arxiv.org/abs/2509.08724.

Wang, L., Ramalho, L., Celestino, A., Pham, P. A., Liu, Y., Sinha, U. K., Portillo, A., Osunwa, O., and Maduekwe, G. Swe-bench++: A framework for the scalable generation of software engineering benchmarks from open-source repositories, 2025f. URL https://arxiv.org/abs/2512.17419.

Wang, P., Shi, T., and Reddy, C. K. Text-to-sql generation for question answering on electronic medical records. In

Huang, Y., King, I., Liu, T., and van Steen, M. (eds.), *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pp. 350–361. ACM / IW3C2, 2020. doi: 10.1145/3366423.3380120. URL https://doi.org/10.1145/3366423.3380120.

Wang, P., Zhang, L., Liu, F., Shi, L., Li, M., Shen, B., and Fu, A. Codeif-bench: Evaluating instruction-following capabilities of large language models in interactive code generation, 2025g. URL https://arxiv.org/abs/2503.22688.

Wang, S., Ding, L., Shen, L., Luo, Y., Du, B., and Tao, D. OOP: object-oriented programming evaluation benchmark for large language models. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 13619–13639. Association for Computational Linguistics, 2024c. doi: 10.18653/V1/2024.FINDINGS-ACL.808. URL https://doi.org/10.18653/v1/2024.findings-acl.808.

Wang, S., Wang, Z., Ma, D., Yu, Y., Ling, R., Li, Z., Xiong, F., and Zhang, W. Codeflowbench: A multi-turn, iterative benchmark for complex code generation, 2026. URL https://arxiv.org/abs/2504.21751.

Wang, W., Yang, C., Wang, Z., Huang, Y., Chu, Z., Song, D., Zhang, L., Chen, A. R., and Ma, L. TESTEVAL: benchmarking large language models for test case generation. *CoRR*, abs/2406.04531, 2024d. doi: 10.48550/ARXIV.2406.04531. URL https://doi.org/10.48550/arXiv.2406.04531.

Wang, W., Ma, W., Hu, Q., Zhang, Y., Sun, J., Wu, B., Liu, Y., Xu, G., and Jiang, L. Vulnrepaireval: An exploit-based evaluation framework for assessing large language model vulnerability repair capabilities, 2025h. URL https://arxiv.org/abs/2509.03331.

Wang, X., Cui, Q., Tao, Y., Wang, Y., Chai, Z., Han, X., Liu, B., Yuan, J., Su, J., Wang, G., Liu, T., Chen, L., Liu, T., Sun, T., Zhang, Y., Zheng, S., You, Q., Yang, Y., and Yang, H. Babelbench: An omni benchmark for code-driven analysis of multimodal and multistructured data, 2024e. URL https://arxiv.org/abs/2410.00773.

Wang, X., Li, Z., and Ding, Z. Defects4log: Benchmarking llms for logging code defect detection and reasoning, 2025i. URL https://arxiv.org/abs/2508.11305.

Wang, Y., Wang, Y., Zhang, T., Yu, Y., Cheung, S.-C., Yu, H., and Zhu, Z. Can machine learning pipelines be better configured? In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, pp. 463–475, New York, NY, USA, 2023a. Association for Computing Machinery. ISBN 9798400703270. doi: 10.1145/3611643.3616352. URL https://doi.org/10.1145/3611643.3616352.

Wang, Y., Wang, Y., Wang, S., Guo, D., Chen, J., Grundy, J., Liu, X., Ma, Y., Mao, M., Zhang, H., and Zheng, Z. Repotransbench: A real-world multilingual benchmark for repository-level code translation, 2025j. URL https://arxiv.org/abs/2412.17744.

Wang, Y. E., Wei, G.-Y., and Brooks, D. Benchmarking tpu, gpu, and cpu platforms for deep learning, 2019. URL https://arxiv.org/abs/1907.10701.

Wang, Z., Zhou, S., Fried, D., and Neubig, G. Execution-based evaluation for open-domain code generation. *arXiv preprint arXiv:2212.10481*, 2022.

Wang, Z., Cuenca, G., Zhou, S., Xu, F. F., and Neubig, G. Mconala: A benchmark for code generation from multiple natural languages. In Vlachos, A. and Augenstein, I. (eds.), *Findings of the Association for Computational Linguistics: EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pp. 265–273. Association for Computational Linguistics, 2023b. doi: 10.18653/V1/2023.FINDINGS-EACL.20. URL https://doi.org/10.18653/v1/2023.findings-eacl.20.

Wang, Z. Z., Asai, A., Yu, X. V., Xu, F. F., Xie, Y., Neubig, G., and Fried, D. Coderag-bench: Can retrieval augment code generation? *CoRR*, abs/2406.14497, 2024f. doi: 10.48550/ARXIV.2406.14497. URL https://doi.org/10.48550/arXiv.2406.14497.

Watson, C., Tufano, M., Moran, K., Bavota, G., and Poshyvanyk, D. On learning meaningful assert statements for unit test cases. In Rothermel, G. and Bae, D. (eds.), *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pp. 1398–1409. ACM, 2020. doi: 10.1145/3377811.3380429. URL https://doi.org/10.1145/3377811.3380429.

Wei, A., Haghtalab, N., and Steinhardt, J. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems*, 36, 2024.

Wei, A., Suresh, T., Cao, J., Kannan, N., Wu, Y., Yan, K., Teixeira, T. S. F. X., Wang, K., and Aiken, A. Codearc: Benchmarking reasoning capabilities of llm agents for inductive program synthesis, 2025a. URL https://arxiv.org/abs/2503.23145.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Wei, J., Durrett, G., and Dillig, I. Typet5: Seq2seq type inference using static analysis. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=4TyNEhI2GdN.

Wei, Z., Zeng, J., Wen, M., Yu, Z., Cheng, K., Zhu, Y., Guo, J., Zhou, S., Yin, L., Su, X., and Ma, Z. Patcheval: A new benchmark for evaluating llms on patching real-world vulnerabilities, 2025b. URL https://arxiv.org/abs/2511.11019.

Wu, C., Ge, Y., Guo, Q., Wang, J., Liang, Z., Lu, Z., Shan, Y., and Luo, P. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. *CoRR*, abs/2405.07990, 2024a. doi: 10.48550/ARXIV.2405.07990. URL https://doi.org/10.48550/arXiv.2405.07990.

Wu, M., Wang, W., Liu, S., Yin, H., Wang, X., Zhao, Y., Lyu, C., Wang, L., Luo, W., and Zhang, K. The bitter lesson learned from 2,000+ multilingual benchmarks, 2025. URL https://arxiv.org/abs/2504.15521.

Wu, T., Wu, W., Wang, X., Xu, K., Ma, S., Jiang, B., Yang, P., Xing, Z., Li, Y., and Haffari, G. Versicode: Towards version-controllable code generation. *CoRR*, abs/2406.07411, 2024b. doi: 10.48550/ARXIV.2406.07411. URL https://doi.org/10.48550/arXiv.2406.07411.

Wu, Y., He, P., Wang, Z., Wang, S., Tian, Y., and Chen, T.-H. A comprehensive framework for evaluating api-oriented code generation in large language models, 2024c. URL https://arxiv.org/abs/2409.15228.

Xia, C. S., Deng, Y., and Zhang, L. Top leaderboard ranking = top coding proficiency, always? evoeval: Evolving coding benchmarks via LLM. *CoRR*, abs/2403.19114, 2024a. doi: 10.48550/ARXIV.2403.19114. URL https://doi.org/10.48550/arXiv.2403.19114.

Xia, Y., Chen, Y., Shi, T., Wang, J., and Yang, J. Aicodereval: Improving AI domain code generation of large language models. *CoRR*, abs/2406.04712, 2024b. doi: 10.48550/ARXIV.2406.04712. URL https://doi.org/10.48550/arXiv.2406.04712.

Xia, Y., Shen, W., Wang, Y., Liu, J. K., Sun, H., Wu, S., Hu, J., and Xu, X. Leetcodedataset: A temporal dataset for robust evaluation and efficient training of code llms, 2025. URL https://arxiv.org/abs/2504.14655.

Xiang, Y., Yan, H., Ouyang, S., Gui, L., and He, Y. Scireplicate-bench: Benchmarking llms in agent-driven algorithmic reproduction from research papers, 2025. URL https://arxiv.org/abs/2504.00255.

Xiao, J., Huang, Q., Chen, X., and Tian, C. Large language model performance benchmarking on mobile platforms: A thorough evaluation, 2024. URL https://arxiv.org/abs/2410.03613.

Xiao, J., Wang, M., Lam, M. H., Wan, Y., Liu, J., Huo, Y., and Lyu, M. R. Designbench: A comprehensive benchmark for mllm-based front-end code generation, 2025. URL https://arxiv.org/abs/2506.06251.

Xie, D., Zheng, M., Liu, X., Wang, J., Wang, C., Tan, L., and Zhang, X. Core: Benchmarking llms code reasoning capabilities through static analysis tasks, 2026. URL https://arxiv.org/abs/2507.05269.

Xu, J., Pang, B., Qu, J., Hayashi, H., Xiong, C., and Zhou, Y. Clover: A test case generation benchmark with coverage, long-context, and verification, 2025a. URL https://arxiv.org/abs/2502.08806.

Xu, K., Mao, Y., Guan, X., and Feng, Z. Web-bench: A llm code benchmark based on web standards and frameworks, 2025b. URL https://arxiv.org/abs/2505.07473.

Xu, R., Cao, J., Lu, Y., Lin, H., Han, X., He, B., Cheung, S.-C., and Sun, L. Cruxeval-x: A benchmark for multilingual code reasoning, understanding and execution. *CoRR*, abs/2408.13001, 2024. URL https://doi.org/10.48550/arXiv.2408.13001.

Xue, P., Wu, L., Yang, Z., Wang, C., Li, X., Zhang, Y., Li, J., Jin, R., Pei, Y., Shen, Z., Lyu, X., and Keung, J. W. Classeval-t: Evaluating large language models in class-level code translation, 2025. URL https://arxiv.org/abs/2411.06145.

Xue, P., Zheng, K., Yang, Z., Pei, Y., Wu, L., Dong, J., Luo, X., Xiao, Y., Liu, F., Zhang, Y., Lyu, X., Li, X., Zhu, X., and Wang, C. Translibeval: Demystify large language models' capability in third-party library-targeted code translation, 2026. URL https://arxiv.org/abs/2509.12087.

Yadav, A., Beniwal, H., and Singh, M. Pythonsaga: Redefining the benchmark to evaluate code generating llms. In Al-Onaizan, Y., Bansal, M., and Chen, Y. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pp. 17113–17126. Association for Computational Linguistics, 2024a. URL https://aclanthology.org/2024.findings-emnlp.996.

Yadav, A., Beniwal, H., and Singh, M. Pythonsaga: Redefining the benchmark to evaluate code generating llms. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 17113–17126, 2024b.

Yamate, B. Y., Neubauer, T. R., Fantinato, M., and Peres, S. M. Text-to-sql oriented to the process mining domain: A pt-en dataset for query translation, 2025. URL https://arxiv.org/abs/2509.09684.

Yan, K., Chang, Y., Guo, Z., Mou, Y., Ming, J., and Sun, J. Stepwise-codex-bench: Evaluating complex multi-function comprehension and fine-grained execution reasoning, 2025a. URL https://arxiv.org/abs/2508.05193.

Yan, K., Guo, H., Shi, X., Cao, S., Di, D., and Li, Z. Codeif: Benchmarking the instruction-following capabilities of large language models for code generation, 2025b. URL https://arxiv.org/abs/2502.19166.

Yan, S., Yu, H., Chen, Y., Shen, B., and Jiang, L. Are the code snippets what we are searching for? A benchmark and an empirical study on code search with natural-language queries. In Kontogiannis, K., Khomh, F., Chatzigeorgiou, A., Fokaefs, M., and Zhou, M. (eds.), *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, pp. 344–354. IEEE, 2020. doi: 10.1109/SANER48275.2020.9054840. URL https://doi.org/10.1109/SANER48275.2020.9054840.

Yan, W., Tian, Y., Li, Y., Chen, Q., and Wang, W. Codetransocean: A comprehensive multilingual benchmark for code translation. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pp. 5067–5089. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.337. URL https://doi.org/10.18653/v1/2023.findings-emnlp.337.

Yang, G., Zhou, Y., Chen, X., Zheng, W., Hu, X., Zhou, X., Lo, D., and Chen, T. Code-diting: A reasoning-based metric for functional alignment in code evaluation, 2025a. URL https://arxiv.org/abs/2505.19502.

Yang, H., He, L., Hou, M., Shen, S., Li, R., Hou, J., Ma, J., and Zhao, J. Aligning llms through multi-perspective user preference ranking-based feedback for programming question answering. *CoRR*, abs/2406.00037, 2024a. doi: 10.48550/ARXIV.2406.00037. URL https://doi.org/10.48550/arXiv.2406.00037.

Yang, J., Zhang, J., Yang, J., Jin, K., Zhang, L., Peng, Q., Deng, K., Miao, Y., Liu, T., Cui, Z., Hui, B., and Lin, J. Execrepobench: Multi-level executable code completion evaluation, 2024b. URL https://arxiv.org/abs/2412.11990.

Yang, J., Jimenez, C. E., Zhang, A. L., Lieret, K., Yang, J., Wu, X., Press, O., Muennighoff, N., Synnaeve, G., Narasimhan, K. R., Yang, D., Wang, S., and Press, O. SWE-bench multimodal: Do AI systems generalize to visual software domains? In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=riTiq3i21b.

Yang, J., Lieret, K., Jimenez, C. E., Wettig, A., Khandpur, K., Zhang, Y., Hui, B., Press, O., Schmidt, L., and Yang, D. Swe-smith: Scaling data for software engineering agents. In *Proceedings of the 39th Annual Conference on Neural Information Processing Systems (NeurIPS 2025 D&B Spotlight)*, 2025c. URL https://arxiv.org/abs/2504.21798. arXiv:2504.21798, accepted at NeurIPS 2025 (Spotlight).

Yang, J., Zhang, W., Liu, S., Chai, L., Tan, Y., Liu, J., Zhang, G., Zhou, W., Niu, G., Li, Z., Hui, B., and Lin, J. Ifevalcode: Controlled code generation, 2025d. URL https://arxiv.org/abs/2507.22462.

Yang, Z., Zhou, Z., Wang, S., Cong, X., Han, X., Yan, Y., Liu, Z., Tan, Z., Liu, P., Yu, D., Liu, Z., Shi, X., and Sun, M. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 11789–11804. Association for Computational Linguistics, 2024c. doi: 10.18653/V1/2024.FINDINGS-ACL.701. URL https://doi.org/10.18653/v1/2024.findings-acl.701.

Yao, Z., Weld, D. S., Chen, W., and Sun, H. Staqc: A systematically mined question-code dataset from stack overflow. In Champin, P., Gandon, F., Lalmas, M., and Ipeirotis, P. G. (eds.), *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pp. 1693–1703. ACM, 2018. doi: 10.1145/3178876.3186081. URL https://doi.org/10.1145/3178876.3186081.

Ye, J., Chen, X., Xu, N., Zu, C., Shao, Z., Liu, S., Cui, Y., Zhou, Z., Gong, C., Shen, Y., Zhou, J., Chen, S., Gui, T., Zhang, Q., and Huang, X. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models, 2023. URL https://arxiv.org/abs/2303.10420.

Ye, Z., Yan, Z., He, J., Kasriel, T., Yang, K., and Song, D. Verina: Benchmarking verifiable code generation, 2025. URL https://arxiv.org/abs/2505.23135.

Yildiz, A., Teo, S. G., Lou, Y., Feng, Y., Wang, C., and Divakaran, D. M. Benchmarking llms and llm-based agents in practical vulnerability detection for code repositories, 2025. URL https://arxiv.org/abs/2503.03586.

Yin, P., Deng, B., Chen, E., Vasilescu, B., and Neubig, G. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories*, MSR, pp. 476–486. ACM, 2018. doi: https://doi.org/10.1145/3196398.3196408.

Yin, P., Li, W., Xiao, K., Rao, A., Wen, Y., Shi, K., Howland, J., Bailey, P., Catasta, M., Michalewski, H., Polozov, O., and Sutton, C. Natural language to code generation in interactive data science notebooks. In Rogers, A., Boyd-Graber, J. L., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pp. 126–173. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.ACL-LONG.9. URL https://doi.org/10.18653/v1/2023.acl-long.9.

Yin, W., Sun, T., Yu, Y., Fang, J., Su, G., Wang, J., Wang, Z., Wang, W., Chen, R., Dai, Z., Yuan, S., Dong, M., Luo, P., Cao, D., Lei, D., Zhang, Y., Chen, H., Ma, X., Liu, Y., Liu, W., Xu, Y., and Pei, J. Cocobench: A comprehensive code benchmark for multi-task large language model evaluation, 2025. URL https://arxiv.org/abs/2504.20673.

Yu, H., Shen, B., Ran, D., Zhang, J., Zhang, Q., Ma, Y., Liang, G., Li, Y., Xie, T., and Wang, Q. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. *arXiv preprint arXiv:2302.00288*, 2023.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. R. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J. (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 3911–3921. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1425. URL https://doi.org/10.18653/v1/d18-1425.

Yu, T., Zhang, R., Er, H., Li, S., Xue, E., Pang, B., Lin, X. V., Tan, Y. C., Shi, T., Li, Z., Jiang, Y., Yasunaga, M., Shim, S., Chen, T., Fabbri, A. R., Li, Z., Chen, L., Zhang, Y., Dixit, S., Zhang, V., Xiong, C., Socher, R., Lasecki, W. S., and Radev, D. R. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In Inui, K., Jiang, J., Ng, V., and Wan, X. (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 1962–1979. Association for Computational Linguistics, 2019a. doi: 10.18653/V1/D19-1204. URL https://doi.org/10.18653/v1/D19-1204.

Yu, T., Zhang, R., Yasunaga, M., Tan, Y. C., Lin, X. V., Li, S., Er, H., Li, I., Pang, B., Chen, T., Ji, E., Dixit, S., Proctor, D., Shim, S., Kraft, J., Zhang, V., Xiong, C., Socher, R., and Radev, D. R. Sparc: Cross-domain semantic parsing in context. In Korhonen, A., Traum, D. R., and Màrquez, L. (eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 4511–4523. Association for Computational Linguistics, 2019b. doi: 10.18653/V1/P19-1443. URL https://doi.org/10.18653/v1/p19-1443.

Yu, X., Chen, T., Yu, Z., Li, H., Yang, Y., Jiang, X., and Jiang, A. Dataset and enhanced model for eligibility criteria-to-sql semantic parsing. In Calzolari, N., Béchet, F., Blache, P., Choukri, K., Cieri, C., Declerck, T., Goggi, S., Isahara, H., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., and Piperidis, S. (eds.), *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020*, pp. 5829–5837. European Language Resources Association, 2020. URL https://aclanthology.org/2020.lrec-1.714/.

Yu, X., Liu, L., Hu, X., Keung, J. W., Liu, J., and Xia, X. Fight fire with fire: How much can we trust chatgpt on source code-related tasks? *arXiv preprint arXiv:2405.12641*, 2024.

Yuan, Y., Jiao, W., Wang, W., tse Huang, J., He, P., Shi, S., and Tu, Z. Gpt-4 is too smart to be safe: Stealthy chat with llms via cipher, 2024a. URL https://arxiv.org/abs/2308.06463.

Yuan, Z., Lou, Y., Liu, M., Ding, S., Wang, K., Chen, Y., and Peng, X. No more manual tests? evaluating and improving chatgpt for unit test generation. *CoRR*, abs/2305.04207, 2023a. doi: 10.48550/ARXIV.2305.04207. URL https://doi.org/10.48550/arXiv.2305.04207.

Yuan, Z., Lou, Y., Liu, M., Ding, S., Wang, K., Chen, Y., and Peng, X. No more manual tests? evaluating and improving chatgpt for unit test generation. *arXiv preprint arXiv:2305.04207*, 2023b.

Yuan, Z., Liu, M., Ding, S., Wang, K., Chen, Y., Peng, X., and Lou, Y. Evaluating and improving chatgpt for unit

test generation. *Proceedings of the ACM on Software Engineering*, 1(FSE):1703–1726, 2024b.

Yun, S., Lin, H., Thushara, R., Bhat, M. Q., Wang, Y., Jiang, Z., Deng, M., Wang, J., Tao, T., Li, J., Li, H., Nakov, P., Baldwin, T., Liu, Z., Xing, E. P., Liang, X., and Shen, Z. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *CoRR*, abs/2406.20098, 2024. doi: 10.48550/ARXIV.2406.20098. URL https://doi.org/10.48550/arXiv.2406.20098.

Zan, D., Chen, B., Lin, Z., Guan, B., Wang, Y., and Lou, J.-G. When language model meets private library. *arXiv preprint arXiv:2210.17236*, 2022a.

Zan, D., Chen, B., Yang, D., Lin, Z., Kim, M., Guan, B., Wang, Y., Chen, W., and Lou, J.-G. Cert: continual pre-training on sketches for library-oriented code generation. *arXiv preprint arXiv:2206.06888*, 2022b.

Zan, D., Huang, Z., Yu, A., Lin, S., Shi, Y., Liu, W., Chen, D., Qi, Z., Yu, H., Yu, L., Ran, D., Zeng, M., Shen, B., Bian, P., Liang, G., Guan, B., Huang, P., Xie, T., Wang, Y., and Wang, Q. Swe-bench-java: A github issue resolving benchmark for java, 2024. URL https://arxiv.org/abs/2408.14354.

Zan, D., Huang, Z., Liu, W., Chen, H., Zhang, L., Xin, S., Chen, L., Liu, Q., Zhong, X., Li, A., Liu, S., Xiao, Y., Chen, L., Zhang, Y., Su, J., Liu, T., Long, R., Shen, K., and Xiang, L. Multi-swe-bench: A multilingual benchmark for issue resolving, 2025. URL https://arxiv.org/abs/2504.02605.

Zeng, Q., Zhang, Y., Ma, Z., Jiang, B., Sun, N., Stol, K.-J., Mou, X., and Liu, H. Evaluating generated commit messages with large language models, 2025a. URL https://arxiv.org/abs/2507.10906.

Zeng, Z., Wang, Y., Xie, R., Ye, W., and Zhang, S. Coderujb: An executable and unified java benchmark for practical programming scenarios. In Christakis, M. and Pradel, M. (eds.), *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, pp. 124–136. ACM, 2024. doi: 10.1145/3650212.3652115. URL https://doi.org/10.1145/3650212.3652115.

Zeng, Z., Shi, R., Han, K., Li, Y., Sun, K., Wang, Y., Yu, Z., Xie, R., Ye, W., and Zhang, S. Benchmarking and studying the llm-based code review, 2025b. URL https://arxiv.org/abs/2509.01494.

Zhang, A., Dong, M., Liu, J., Zhang, W., Wang, Y., Yang, J., Zhang, G., Liu, T., Peng, Z., Tan, Y., Zhang, Y., Wang, Z., Wang, W., He, Y., Deng, K., Zhou, W., Huang, W., and Zhang, Z. Codecriticbench: A holistic code critique benchmark for large language models, 2025a. URL https://arxiv.org/abs/2502.16614.

Zhang, B., Ye, Y., Du, G., Hu, X., Li, Z., Yang, S., Liu, C. H., Zhao, R., Li, Z., and Mao, H. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation, 2024a. URL https://arxiv.org/abs/2403.02951.

Zhang, F., Wu, L., Bai, H., Lin, G., Li, X., Yu, X., Wang, Y., Chen, B., and Keung, J. Humaneval-v: Benchmarking high-level visual reasoning with complex diagrams in coding tasks, 2025b. URL https://arxiv.org/abs/2410.12381.

Zhang, H. and Huang, J. Challenging gpu dominance: When cpus outperform for on-device llm inference. *arXiv preprint arXiv:2505.06461*, 2025.

Zhang, K., Li, J., Li, G., Shi, X., and Jin, Z. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 13643–13658. Association for Computational Linguistics, 2024b. doi: 10.18653/V1/2024.ACL-LONG.737. URL https://doi.org/10.18653/v1/2024.acl-long.737.

Zhang, L., He, S., Zhang, C., Kang, Y., Li, B., Xie, C., Wang, J., Wang, M., Huang, Y., Fu, S., Nallipogu, E., Lin, Q., Dang, Y., Rajmohan, S., and Zhang, D. Swe-bench goes live!, 2025c. URL https://arxiv.org/abs/2505.23419.

Zhang, L., Wang, B., Wang, J., Zhao, X., Zhang, M., Yang, H., Zhang, M., LI, Y., Li, J., Yu, J., and Zhang, M. Function-to-style guidance of LLMs for code translation. In *Forty-second International Conference on Machine Learning*, 2025d. URL https://openreview.net/forum?id=bLng6Z10Vx.

Zhang, L., Wang, J., He, S., Zhang, C., Kang, Y., Li, B., Wen, J., Xie, C., Wang, M., Huang, Y., Nallipogu, E., Lin, Q., Dang, Y., Rajmohan, S., Zhang, D., and Zhang, Q. Dibench: Benchmarking large language models on dependency inference with testable repositories at scale, 2025e. URL https://arxiv.org/abs/2501.13699.

Zhang, L., Zan, D., Yang, Q., Huang, Z., Chen, D., Shen, B., Liu, T., Gong, Y., Pengjie, H., Lu, X., Liang, G., Cui, L., and Wang, Q. CodeV: Issue resolving with visual data. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T.

(eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 7350–7361, Vienna, Austria, July 2025f. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.384. URL https://aclanthology.org/2025.findings-acl.384/.

Zhang, Q., Zhang, T., Zhai, J., Fang, C., Yu, B., Sun, W., and Chen, Z. A critical review of large language model on software engineering: An example from chatgpt and automated program repair. *CoRR*, abs/2310.08879, 2023a. doi: 10.48550/ARXIV.2310.08879. URL https://doi.org/10.48550/arXiv.2310.08879.

Zhang, Q., Shang, Y., Fang, C., Gu, S., Zhou, J., and Chen, Z. Testbench: Evaluating class-level test case generation capability of large language models, 2024c. URL https://arxiv.org/abs/2409.17561.

Zhang, Q., Liu, P., Di, P., and Qian, C. Codefuse-commiteval: Towards benchmarking llm's power on commit message and code change inconsistency detection, 2025g. URL https://arxiv.org/abs/2511.19875.

Zhang, S., Zhao, H., Liu, X., Zheng, Q., Qi, Z., Gu, X., Zhang, X., Dong, Y., and Tang, J. Naturalcodebench: Examining coding performance mismatch on humaneval and natural user prompts. *CoRR*, abs/2405.04520, 2024d. doi: 10.48550/ARXIV.2405.04520. URL https://doi.org/10.48550/arXiv.2405.04520.

Zhang, Y., Wang, J., Wang, Z., and Zhang, R. Xsemplr: Cross-lingual semantic parsing in multiple natural languages and meaning representations. In Rogers, A., Boyd-Graber, J. L., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pp. 15918–15947. Association for Computational Linguistics, 2023b. doi: 10.18653/V1/2023.ACL-LONG.887. URL https://doi.org/10.18653/v1/2023.acl-long.887.

Zhang, Y., Xie, Y., Li, S., Liu, K., Wang, C., Jia, Z., Huang, X., Song, J., Luo, C., Zheng, Z., Xu, R., Liu, Y., Zheng, S., and Liao, X. Unseen horizons: Unveiling the real capability of llm code generation beyond the familiar, 2025h. URL https://arxiv.org/abs/2412.08109.

Zhang, Z., Xu, L., Jiang, Z., Hao, H., and Wang, R. Multiple-choice questions are efficient and robust LLM evaluators. *CoRR*, abs/2405.11966, 2024e. doi: 10.48550/ARXIV.2405.11966. URL https://doi.org/10.48550/arXiv.2405.11966.

Zhang, Z., Wang, J., Yang, Q., Pan, Y., Tang, Y., Li, Y., Xing, Z., Zhang, T., Li, X., and Zhang, G. A benchmark for localizing code and non-code issues in software projects, 2025i. URL https://arxiv.org/abs/2509.25242.

Zhao, S., Wang, D., Zhang, K., Luo, J., Li, Z., and Li, L. Is vibe coding safe? benchmarking vulnerability of agent-generated code in real-world tasks, 2025a. URL https://arxiv.org/abs/2512.03262.

Zhao, Y., Luo, Z., Tian, Y., Lin, H., Yan, W., Li, A., and Ma, J. CodeJudge-eval: Can large language models be good judges in code understanding? In Rambow, O., Wanner, L., Apidianaki, M., Al-Khalifa, H., Eugenio, B. D., and Schockaert, S. (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 73–95, Abu Dhabi, UAE, January 2025b. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.7/.

Zheng, D., Wang, Y., Shi, E., Zhang, R., Ma, Y., Zhang, H., and Zheng, Z. Towards more realistic evaluation of llm-based code generation: an experimental study and beyond. *CoRR*, abs/2406.06918, 2024. doi: 10.48550/ARXIV.2406.06918. URL https://doi.org/10.48550/arXiv.2406.06918.

Zheng, D., Wang, Y., Shi, E., Liu, X., Ma, Y., Zhang, H., and Zheng, Z. Top general performance = top domain performance? domaincodebench: A multi-domain code generation benchmark, 2025a. URL https://arxiv.org/abs/2412.18573.

Zheng, Q., Xia, X., Zou, X., Dong, Y., Wang, S., Xue, Y., Shen, L., Wang, Z., Wang, A., Li, Y., Su, T., Yang, Z., and Tang, J. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, pp. 5673–5684, New York, NY, USA, 2023a. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599790. URL https://doi.org/10.1145/3580305.3599790.

Zheng, X., Lin, H., Cai, S., Zheng, Z., and Liang, Y. Unicode: A framework for generating high quality competitive coding problems, 2025b. URL https://arxiv.org/abs/2510.17868.

Zheng, Y., Pujar, S., Lewis, B. L., Buratti, L., Epstein, E. A., Yang, B., Laredo, J., Morari, A., and Su, Z. D2A: A dataset built for ai-based vulnerability detection methods using differential analysis. In *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021*, pp. 111–120. IEEE, 2021. doi: 10.1109/ICSE-SEIP52600.2021.00020. URL https://doi.org/10.1109/ICSE-SEIP52600.2021.00020.

Zheng, Z., Ning, K., Wang, Y., Zhang, J., Zheng, D., Ye, M., and Chen, J. A survey of large language models for code: Evolution, benchmarking, and future trends. *arXiv preprint arXiv:2311.10372*, 2023b.

Zhong, V., Xiong, C., and Socher, R. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017. URL http://arxiv.org/abs/1709.00103.

Zhong, Z., Huang, J., and He, P. Backportbench: A multilingual benchmark for automated backporting of patches, 2025. URL https://arxiv.org/abs/2512.01396.

Zhou, Y., Liu, S., Siow, J. K., Du, X., and Liu, Y. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 10197–10207, 2019.

Zhou, Z., Huang, Z., He, Y., Wang, C., Wang, J., Wu, Y., Peng, X., and Lou, Y. Benchmarking and enhancing llm agents in localizing linux kernel bugs, 2025. URL https://arxiv.org/abs/2505.19489.

Zhu, H., Zhang, Y., Zhao, B., Ding, J., Liu, S., Liu, T., Wang, D., Liu, Y., and Li, Z. Frontendbench: A benchmark for evaluating llms on front-end development via automatic evaluation, 2025a. URL https://arxiv.org/abs/2506.13832.

Zhu, M., Jain, A., Suresh, K., Ravindran, R., Tipirneni, S., and Reddy, C. K. Xlcost: A benchmark dataset for cross-lingual code intelligence. *CoRR*, abs/2206.08474, 2022. doi: 10.48550/ARXIV.2206.08474. URL https://doi.org/10.48550/arXiv.2206.08474.

Zhu, Q., Cao, J., Lu, Y., Lin, H., Han, X., Sun, L., and Cheung, S.-C. Domaineval: An auto-constructed benchmark for multi-domain code generation. *CoRR*, abs/2408.13204, 2024. URL https://doi.org/10.48550/arXiv.2408.13204.

Zhu, Y., Kellermann, A., Bowman, D., Li, P., Gupta, A., Danda, A., Fang, R., Jensen, C., Ihli, E., Benn, J., Geronimo, J., Dhir, A., Rao, S., Yu, K., Stone, T., and Kang, D. CVE-bench: A benchmark for AI agents' ability to exploit real-world web application vulnerabilities. In *Forty-second International Conference on Machine Learning*, 2025b. URL https://openreview.net/forum?id=3pk0p4NGmQ.

Zhuo, T. Y., Vu, M. C., Chim, J., Hu, H., Yu, W., Widyasari, R., Yusuf, I. N. B., Zhan, H., He, J., Paul, I., Brunner, S., Gong, C., Hoang, T., Zebaze, A. R., Hong, X., Li, W., Kaddour, J., Xu, M., Zhang, Z., Yadav, P., Jain, N., Gu, A., Cheng, Z., Liu, J., Liu, Q., Wang, Z., Lo, D., Hui, B., Muennighoff, N., Fried, D., Du, X., de Vries, H., and von Werra, L. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *CoRR*, abs/2406.15877, 2024. doi: 10.48550/ARXIV.2406.15877. URL https://doi.org/10.48550/arXiv.2406.15877.

Zibaeirad, A. and Vieira, M. Vulnllmeval: A framework for evaluating large language models in software vulnerability detection and patching, 2024. URL https://arxiv.org/abs/2409.10756.

Zou, D., Wang, S., Xu, S., Li, Z., and Jin, H. μvuldeepecker: A deep learning-based system for multiclass vulnerability detection. *CoRR*, abs/2001.02334, 2020. URL http://arxiv.org/abs/2001.02334.

A. Appendix

The appendix is organized as follows:

- **Appendix B** lists the checklist items in each benchmark development phase.
- **Appendix C** lists all the statistics in the survey.
- **Appendix D** explains the details of human study.
- **Appendix E** lists all the benchmarks in focused study.
- **Appendix F** lists all the surveyed benchmarks in a chronological order.
- **Appendix G** shows the complete HOW2BENCH with 55 checklists.

B. Detailed Guidance in HOW2BENCH

We present the detailed guidance in each phase.

	Phase 1. Benchmark Design	Priority	☑
1	**Consider whether the benchmark can fill the gap in related research.**	★★★	☐
2	**Consider what is the expected scope of the benchmark set** (e.g., what natural languages, programming languages, task granularity).	★★★	☐
3	**Consider the expected application scenario of this benchmark** (e.g., programming assistant, automated tester).	★★★	☐
4	**Consider the LLMs' capabilities** (e.g., understanding, reasoning, calculation) **and domain knowledge** (e.g., OOP, memory management, fault localization, process scheduling) **that the benchmark hopes to evaluate.**	★★★	☐

Figure 3. **Guideline for Benchmark Design**

	Phase 2. Benchmark Construction (1/2)	Priority	☑
5	**Consider whether the data source of the benchmark is traceable.**	★	☐
6	**Consider whether the data source of the benchmark is of high quality** (e.g., stars, downloads, last update times, number of forks).	★★	☐
7	**Consider whether the benchmark's data source is representative** (e.g., choose an open-source community or code hosting platform that matches the task, capability, and scope under study)	★	☐
8	**Consider data contamination issues during the benchmark collection** (e.g., considering the upload time of the source code or checking whether the data source is included in the training data of LLMs).	★	☐
9	**If data sampling is needed, consider whether the choice of sample size is scientific** (e.g., considering the confidence level/margin of error/sampling proportion, etc.).	★	☐
10	**If data sampling is needed, consider whether the sampling process is rigorous** (e.g., random sampling, stratified sampling, etc.).	★	☐
11	**Ensure each data point in the benchmark falls into the targeted scope** (e.g., checking each data point's evaluated capabilities or domain knowledge).	★	☐
12	**Consider whether the data in the benchmark can cover the studied capabilities/domain knowledge/application scenarios.**	★★★	☐
13	**Consider whether there is a standard answer for each sample in the benchmark** (such as reference code, etc.).	★★	☐
14	**For code, consider whether the code is compilable/executable.**	★	☐
15	**Consider the possibility of noise in the data and perform denoise.**	★★	☐
16	**Consider the possibility of duplication in the data and deduplicate them.**	★★	☐
17	**Clean the sensitive information** (such as data desensitization and anonymization) **unless the benchmark is deliberately designed so.**	★★	☐
18	**Manually review some or all of the data in the benchmark to ensure its quality.**	★★	☐
19	**Use LLMs to review some or all of the data in the benchmark to ensure its quality.**	★	☐
20	**Design appropriate output validation methods for the benchmark** (e.g., using exact matching or designing test cases).	★★★	☐
21	**Design appropriate evaluation metrics for the evaluation set** (e.g., precision, accuracy, pass@K, recall).	★★★	☐
22	**Consider the adequacy of the evaluation metrics** (e.g., is the code coverage high enough).	★★★	☐
23	**Consider if there are any other evaluation perspectives** (e.g., readability, efficiency, safety, security).	★	☐

Figure 4. Guideline for Benchmark Construction

	Phase 3. Benchmark Evaluation	Priority	☑
24	**Consider whether sufficient LLMs are evaluated.**	★	☐
25	**Consider whether representative LLMs** (e.g., covering latest/classical LLM families, small/large LLMs, and open-/closed-source LLMs) **are evaluated.**	★	☐
26	**Consider whether the prompt is of high quality** (e.g., the instruction and intent are clear).	★★★	☐
27	**The prompts have been validated by humans or LLMs** (e.g., evaluated or discussed by participants or preliminarily tried out on several LLMs).	★	☐
28	**Try different paraphrases of the prompt.**	★	☐
29	**Try different prompting strategies to observe the impact on the evaluation results** (e.g., in-context learning, chain-of-thought).	★★	☐
30	**Pay attention to the hardware environment** (such as GPU card, storage size, etc.) of the experiment.	★★★	☐
31	**Pay attention to the operating system and software environment** (e.g. operating system, version, etc.) **used for the experiment.**	★★★	☐
32	**Pay attention to the off-the-shelf platforms, frameworks, or libraries for LLM evaluation** (e.g., fast chat, vllm, huggingface) **that are used.**	★★	☐
33	**Repeat the experiment multiple times to reduce the impact of randomness on the evaluation.**	★	☐
34	**Consider various randomization strategies** (e.g., trying various temperature parameters) **to reduce the impact of parameter configuration on the evaluation.**	★★	☐
35	**Record the experimental process in detail** (e.g., parameter settings, running time, input/output pairs, etc.).	★★★	☐

Figure 5. Guideline for Benchmark Evaluation

	Phase 4. Benchmark Analysis	Priority	☑
36	**Observe the difficulty of the benchmark, checking if the benchmark is too hard or too easy for LLMs** (i.e., most LLMs score too high/low).	★★	☐
37	**Consider whether the benchmark can distinguish the pros and cons of different LLMs.**	★	☐
38	**If the experiment is repeated several times, consider the stability of the benchmark** (i.e., whether the experimental results vary too much in the repeated experiments).	★	☐
39	**Analyze the correlation between the data and their score.** For example, if there is a correlation between the data (such as similar difficulty and knowledge required), then the scores should also be correlated.	★★	☐
40	Compare the performance of LLMs on this benchmark with their performance on **other related benchmarks**.	★	☐
41	**Consider presenting the experiment results in an appropriate way** (e.g., table, line graph, pie chart, etc.).	★★★	☐
42	**Consider presenting the experiment results clearly** (e.g., distinguishable colors/labels/shapes, etc.).	★★★	☐
43	**Explain the experiment results.**	★★★	☐
44	**Observe correlations via multiple perspectives from the experimental results** (e.g., performance is correlated with model size or amount of context).	★	☐

Figure 6. **Guideline for Evaluation Analysis**

	Phase 5. Benchmark Release	Priority	☑
45	**The analysis of the evaluation results will be inspiring** (e.g., shed light on future direction, make actionable advice, etc.).	★	☐
46	**Set the appropriate license for the benchmark.**	★★★	☐
47	**Review the released benchmark or other artifacts to ensure they do NOT contain sensitive information** (e.g., API keys, usernames, passwords, etc.).	★★★	☐
48	**review the released benchmark or other artifacts to ensure they do NOT contain toxicity information** (e.g., abusive comments/identifiers).	★★★	☐
49	**Make sure the benchmark is open-accessible.**	★★★	☐
50	**Make sure the test cases or reference data are open and accessible.**	★★★	☐
51	**Provide prompts used in the experiment to ensure the experiments are reproducible.**	★★★	☐
52	**Disclose the experimental environment** (e.g., hardware, operating system, software version, framework platform) **to ensure the reproducibility of the experiment.**	★★★	☐
53	**Make the detailed experimental results public for verification.**	★★★	☐
54	**Ensure the quality of the user manual** such as README (e.g., it contains necessary benchmark introduction, executable scripts, etc.).	★★	☐
55	**Provide convenient evaluation interfaces for the released benchmark** (e.g., providing a command line interface, docker, etc.).	★★	☐

Figure 7. **Guideline for Benchmark Release**

C. Statistics of studied benchmarks

In this section, we conducted a comprehensive and detailed statistical analysis of the 572 benchmarks collected.

C.1. Profile of Studied Benchmarks

We first show the trend in the development of benchmarks from 2014 to 2025. As shown in Figure 8, the data shows a modest beginning, with only a handful of benchmarks created annually until 2017. From 2018 onwards, there has been a noticeable uptrend in benchmark creation, culminating in a significant jump to 210 benchmarks in 2024, and 316 benchmarks in 2025. This sharp increase indicates a recent heightened interest and demand for comprehensive code-related benchmarks for LLMs, reflecting the evolving complexities and expanding requirements of automated software engineering.

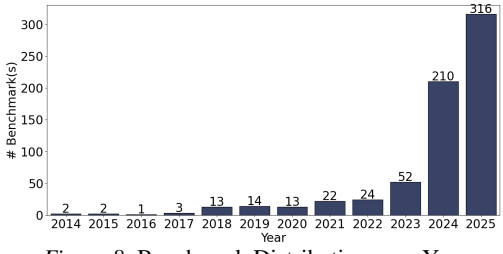

Figure 8. Benchmark Distribution over Years

Hierarchy of Benchmarks. Figure 59 visualizes the inheritance relationships among benchmarks, indicating that the benchmarks on the ***left serve as sources*** for those on the right. It highlights that ***18% of benchmarks act as data sources***, continuously benefiting the construction of subsequent benchmarks.

Figure 59 reveals that ***HumanEval*** (Chen et al., 2021a), as the ***most significant source*** benchmark, benefits at least ***15 downstream benchmarks***, followed by MBPP (Austin et al., 2021) and CodeSearchNet (Husain et al., 2019). From the right side of the figure, some benchmarks, like VulBench (Gao et al., 2023b), incorporate methodologies or data from 4 previous benchmarks, and codeRag-Bench (Wang et al., 2024f) integrates elements from 8 prior benchmarks.

This hierarchical structure among benchmarks also alerts us that the ***data quality of a benchmark not only affects its own credibility but can continue to impact others*** if it serves as a source. This underscores the importance of adhering to stringent guidelines during benchmark development and highlights the crucial role of ***establishing standards*** to ensure the integrity and utility of benchmark data across research and development efforts.

Coding Task. Regarding the ***coding tasks***, Figure 9 illustrates the distribution of various coding tasks across benchmarks. It is clear that the task of ***Code Generation*** is most prevalent, with 235 benchmarks focusing on this area, according to 34.97% (235/672) of studied benchmarks, indicating a significant interest in generating code automatically. The second most prevalent is code reasoning 12.64% (85/672), followed by Program Repair and Defect Detection.

When examining the distribution of coding tasks by year (see Figure 10), we observe that benchmark growth accelerated most rapidly in the past two years (2024–2025), with code generation remaining the most frequently benchmarked task. At the same time, demand for code reasoning has increased substantially, rising from 19 benchmarks in 2024 to 61 in 2025. Benchmarks for program repair have also nearly doubled over the same period, increasing from 22 to 33.

These trends suggest a growing emphasis on LLMs' reasoning capabilities as well as their role in code development and software maintenance, reflecting an evolving expectation of model competence beyond surface-level code synthesis.

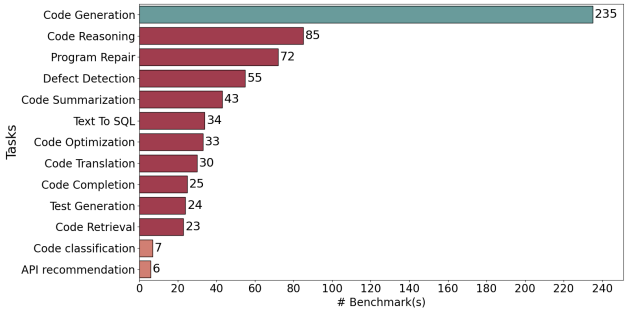

Figure 9. Benchmark Distribution over Tasks

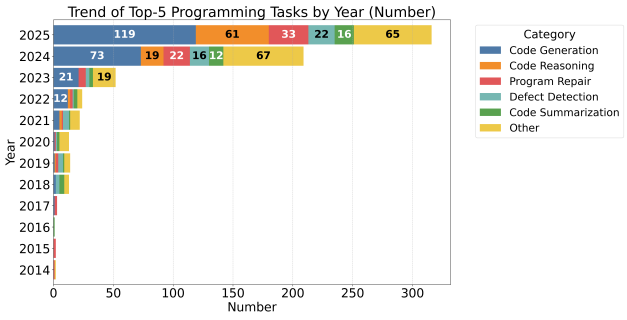

Figure 10. Benchmark Distribution over Tasks per Year

Programming Languages. Figure 11 shows the distribution of benchmarks across various programming languages. The overall trend indicates a strong preference for benchmarking ***Python***, which leads with 409 (71.50%) benchmarks, followed by Java and C++, with 229 and 160, respectively. The graph also reveals a diverse range of languages being used. In total, 67 programming languages are studied by

these 672 benchmarks. Though some programming languages, such as Kotlin, Swift, and Scala, are less frequently exercised, the benchmarks involving them are tailored to different application needs and technology environments. This distribution shows the existing benchmarks are dominated by three mainstream programming languages, leaving other programming languages less studied and benchmarked.

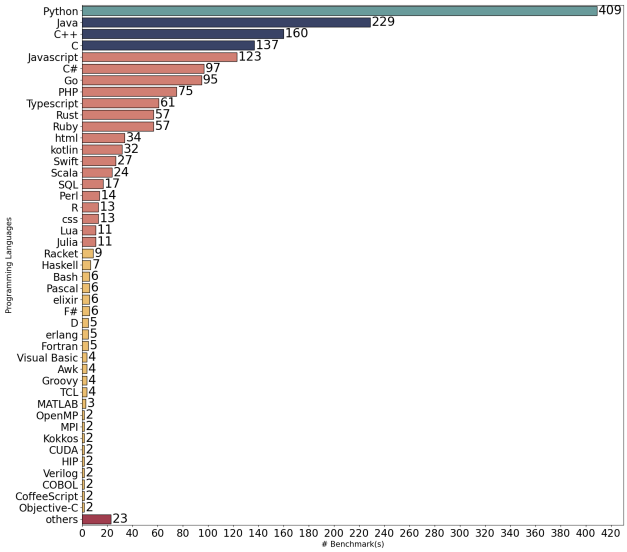

Figure 11. Benchmark Distribution over Programming Language

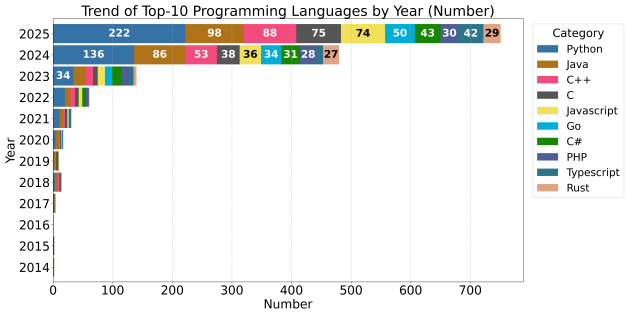

Figure 12. Benchmark Distribution over Programming Language per Year

When analyzed by year (Figure 12), the number of benchmarks for C/C++ and JavaScript has increased noticeably between 2024 and 2025. We also observe a sharp rise in Rust-related benchmarks: although only a few benchmarks existed for Rust since its release in 2015, 57 benchmarks were published in the past two years (27 in 2024 and 29 in 2025). These trends in programming language coverage reflect the evolving demand for LLMs across different languages, highlighting particularly the growing reliance on large models to generate, reason about, and maintain code in these languages.

Natural Language. Figure 13 illustrates the distribution of benchmarks for different natural languages. The bar chart overwhelmingly shows that English is the dominant language, with 577 (85.9%) benchmarks highlighting its ubiquity in research and development. Other languages have significantly fewer benchmarks, with 16 in Chinese and 15 in Russian. The category labeled "Other" includes 24 benchmarks spread across other natural languages, indicating some diversity but limited attention to non-English benchmarks. This distribution highlights the prominence of English in the global research community and also demonstrates the ***uneven representation*** of natural languages in the studied benchmarks.

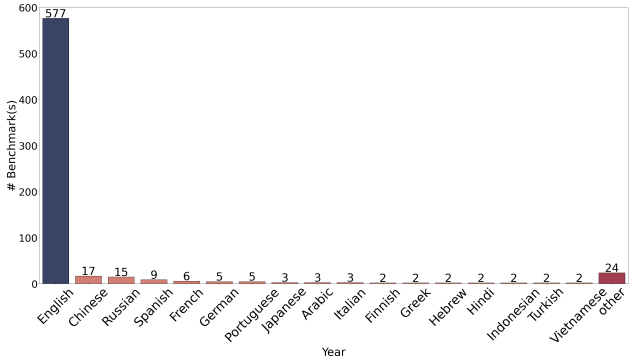

Figure 13. Benchmark Distribution over Natural Language

Modals in the benchmarks. Figure 14 presents the distribution of benchmarks according to the type of language used in their prompts. The chart shows that the majority, at 66.2%, of the benchmarks use a combination of natural language and programming Language, followed by PL only (13.7%) and NL only (18.2%).

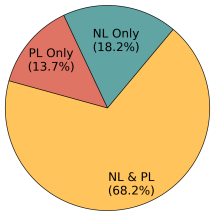

Figure 14. Benchmark Distribution over Modal in Prompt

Granularity. The code snippet in a code-related benchmark varies from statement-level (i.e., one line of code. For example, CoNaLa (Yin et al., 2018) and Math-QA (Amini et al., 2019)), function-level (i.e., a function unit of code. For example, HumanEval (Chen et al., 2021a) and MBPP (Austin et al., 2021)), class-level (i.e., a class with multiple function units of code. For example, ClassEval (Du et al., 2023)) and project-level (i.e., a project with multiple classes or modules. For example, DevEval (Li et al., 2024a) and JavaBench (Cao et al., 2024a)).

Figure 15 illustrates the granularity levels at which bench-

marks are typically conducted. The chart shows that the majority of benchmarks, comprising ***66.9%, focus on the function level***. Projects constitute 21.5% of the benchmarks. Class-level granularity is the least represented at only 3.4%.

When analyzed by year, an interesting phenomenon emerges (Figure 16): between 2024 and 2025, the number of project-level benchmarks surged, increasing from 35 to 95 new benchmarks. Importantly, these numbers refer to benchmarks newly introduced in each year, rather than cumulative totals. This surge indicates a growing community focus on the real-world applicability and large-scale practical utility of LLMs.

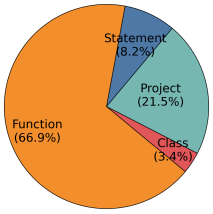

Figure 15. Benchmark Distribution over Granularity

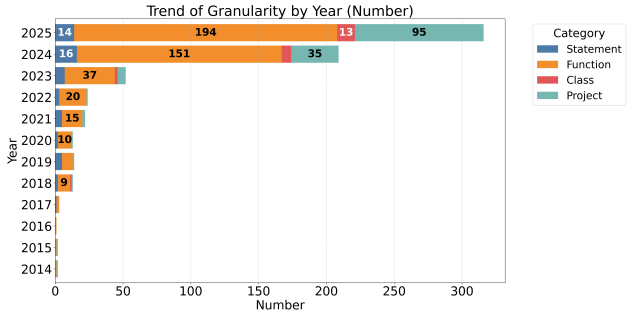

Figure 16. Benchmark Distribution over Granularity per Year

C.2. Statistics about Benchmark Design

Design of Studied Capabilities. To understand whether benchmark developers recognize the capabilities of LLMs they aim to evaluate, we carefully analyzed 30 representative benchmarks (Appendix E) to see if they clearly specify the capabilities being assessed by their benchmarks. As shown in Figure 17, 80% of benchmarks explicitly specify the capabilities (e.g., intention understanding, problem solving, testing, debugging capabilities)to be evaluated, while 20% do not. The statistics show that the most highly cited benchmarks clearly define the assessment capabilities.

Furthermore, we investigated the 30 focused benchmarks and identified a case (Figure 18) from MBPP (Austin et al., 2021) where the case is likely to fall outside of the targeted capability of the benchmark. In particular, MBPP (Austin et al., 2021) aims to "*measure the ability of these models to synthesize short Python programs from natural language*

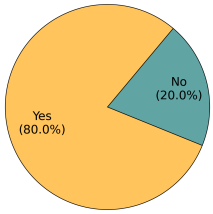

Figure 17. Benchmark Distribution Over Capabilities Consideration

descriptions" for "*entry-level programmers*". As we can see from Figure 18, the prompt requires LLMs to "*Write a function to calculate the dogs' years.*" Simply from this description, an entry-level programmer is unlikely to write a correct program without knowing the conversion equation from dogs' year to dogs' age. In other words, this case is more about assessing whether LLMs have acquired this specific knowledge rather than evaluating the most fundamental programming skills.

Out of Targeted Capabilities

```
{
  'source_file': 'Benchmark Questions
                  Verification V2.ipynb',
  'task_id': 264,
  'prompt': 'Write a function to calculate
             a dog's age in dog's years.'
  'test_list': [ "assert dog_age(12)==61",
                 "assert dog_age(15)==73",
                 "assert dog_age(24)==109" ]
}
```

Figure 18. An Example of Out-of-capability Case from MBPP (Austin et al., 2021).

Design of Studied Application Scenarios. Similarly, to understand whether benchmark developers scoped the application scenarios of LLMs they aim to evaluate, we carefully analyzed 30 representative benchmarks (Appendix E) to see whether they explicitly specify the application scenarios their benchmarks target. As shown in Figure 19, 76.7% representative benchmarks have clearly specified application scenarios (e.g., programming assistant), while the rest do not. Indeed, clearly defining the application scenarios could help benchmark constructors establish precise goals for the design and development of the benchmark, ensuring accuracy in the evaluation.

C.3. Statistics about Data Preparation

C.3.1. Data Preprocessing

Data Deduplication. During benchmark preparation, data cleaning and preprocessing are necessary. However, as shown in Figure 20, only ***32.9% benchmarks have dedu-***

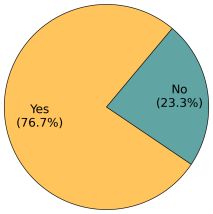

Figure 19. Benchmark Distribution Over Expected Application Scenario Consideration

plicated the collected data. More than half of them didn't mention this process.

The yearly trend (Figure 21) shows that, although the number of benchmarks that applied deduplication increased slightly from 71 in 2024 to 93 in 2025, the total number of benchmarks in 2025 more than doubled compared to 2024. As a result, the absolute number of benchmarks that ignored or did not apply deduplication nearly doubled, highlighting a growing risk of duplicated data despite increased awareness.

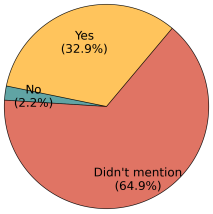

Figure 20. Benchmark Distribution over Deduplication

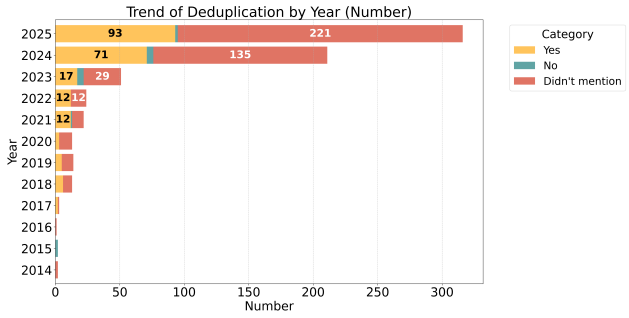

Figure 21. Benchmark Distribution over Deduplication per Year

To investigate the situation, we went through the 30 representative benchmarks (Listed in Appendix E) and found two duplicated subjects in MBPP (Austin et al., 2021). Tasks with id `71` and `141` examined the same functionality, i.e., "*Write a function to sort a list of elements*.", collected from the same source.

Data Quality Assurance. Ensuring data quality for the benchmark is essential. However, our statistics (Figure 23) show disappointing results. ***46.0% of benchmarks do not take any measures for data quality assurance***. Among

Duplicated Data

```
{
  'source_file': 'Mike's Copy of Benchmark
                 Questions Verification V2.ipynb',
  'task_id': 71,
  'prompt': 'Write a function to sort a list of elements. '
}

{
  'source_file': 'Mike's Copy of Benchmark
                 Questions Verification V2.ipynb',
  'task_id': 141,
  'prompt': 'Write a function to sort a list of elements. '
}
```

Figure 22. A Counterexample of Rule 16 from MBPP (Austin et al., 2021).

those benchmarks that do incorporate data quality measures, the majority rely on manual checks, which account for 45.8%. Other countermeasurements, such as code execution, constitute only 1.2%. Additional methods, such as using download counts as a basis, are also employed (6.4%) .

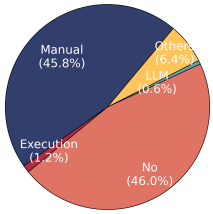

Figure 23. Benchmark Distribution over Quality Assurance Method

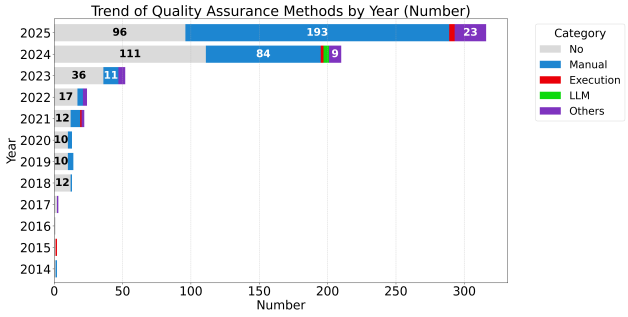

Figure 24. Benchmark Distribution over Quality Assurance Method per Year

Examining the trends by year (Figure 24), **one positive development is the increasing prevalence of manual quality checks**. The number of benchmarks with manual check guarantees doubled from 84 in 2024 to 193 in 2025, indicating that a growing share of code benchmarks now undergo partial or full human verification.

Additionally, we dived into the 30 representative benchmarks (Listed in Appendix E) and identified an example

where the code cannot be executed successfully. As shown in Figure 25, the function `swap()` in line 7 has not been defined, so the execution of the code would fail if the code has been executed. This highlights a significant gap in ensuring the reliability and validity of benchmark data, underscoring the need for more rigorous and automated data quality assurance practices.

Problem in Execution

```
1 Import math
2 def min_Operations (A, B):
3     """ Write a python function to find
4     the minimum operations required
5     to make two numbers equal. """
6     if (A > B):
7         swap(A,B)
8     B = B // math.gcd(A,B);
9     return B - 1
```

Figure 25. An Example from MBPP (Austin et al., 2021) that failed to be executed.

Data Contamination Resolution. Data contamination (Golchin & Surdeanu, 2023; Cao et al., 2024b) threat has been widely discussed. A benchmark with contaminated data may yield overclaimed results, misleading the understanding of the LLMs' capabilities. According to our statistics (Figure 26), most (79.8%) benchmarks were not aware of and have not taken any measures to alleviate data contamination, being vulnerable to data contamination threats. The yearly trend (Figure 27) further highlights the persistent neglect of data contamination in benchmark construction. From 2024 to 2025, the number of benchmarks that did not account for data contamination nearly doubled, increasing from 165 to 236, indicating that this risk persists or has become more widespread.

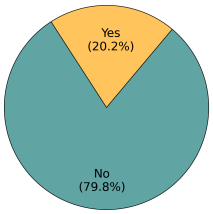

Figure 26. Benchmark Distribution over Quality Assurance on Data Contamination

C.3.2. Statistics about Data Curation

Ground truth solutions. Figure 28 shows that although the majority (89.1%) of benchmarks provide reference code as ground truth, there are 9.8% of benchmarks without reference code. Although it is not compulsory as long as object measurements (e.g., test cases) are provided, ***a reference code is still recommended.*** Indeed, if a benchmark provides reference code, its reliability tends to be better because it ensures that there are feasible solutions for the tasks involved. This guarantees that the tasks are theoretically and practically solvable, enhancing the benchmark's usefulness and credibility in real-world applications.

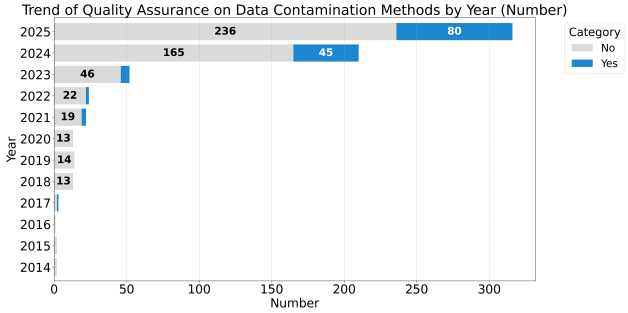

Figure 27. Benchmark Distribution over Quality Assurance on Data Contamination per Year

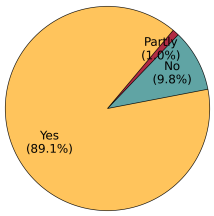

Figure 28. Benchmark Distribution over Solution

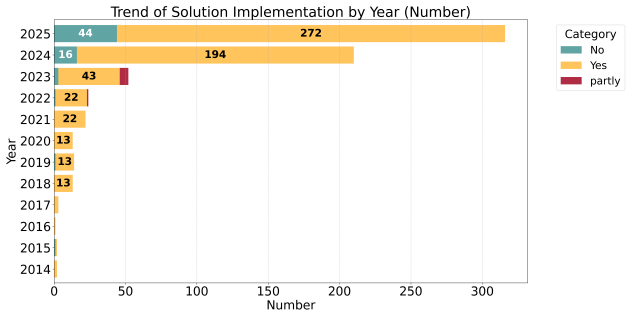

Figure 29. Benchmark Distribution over Solution per Year

From the year-by-year trend (Figure 29), the number of benchmarks without ground truth tripled in 2025, reaching 44 benchmarks. This may reflect **a shift toward more open-ended and challenging benchmarks**, where standard solutions do not yet exist or remain to be explored, highlighting the evolving nature of tasks designed for LLM evaluation.

Additionally, ***the correctness of the ground truth solution*** should also be noted. Figure 30 shows an ***incorrect*** code solution provided in HumanEval (Chen et al., 2021a). This should draw benchmark constructors' attention to the correctness of the benchmark reference code.

Incorrect Ground-truth

```python
def check_dict_case(dict):
    """ Given a dictionary, return True if all keys are strings
    in lower case or all keys are strings in upper case, else
    return False. The function should return False is the
    given dictionary is empty. """
    if len(dict.keys()) == 0:
        return False
    else:
        state = "start"
        for key in dict.keys():
            if isinstance(key, str) == False:
                state = "mixed"
                break
            if state == "start":
                if key.isupper():
                    state = "upper"
                elif key.islower():
                    state = "lower"
                else:
                    break
            elif (state == "upper" and not key.isupper())
                    or (state == "lower" and not
                        key.islower()):
                state = "mixed"
                break
            else:
                break
    return state == "upper"  or state == "lower"
```

Figure 30. An Example from HumanEval (Chen et al., 2021a) which shows ***an incorrect solution*** provided in the benchmark.

Oracle. An oracle (Barr et al., 2014) is a way to determine whether the output is correct or not. For example, assume the output of LLMs is in the form of code, then an oracle could be running tests against the code and see whether the code can pass all the tests. Figure 31 shows the distribution of types of oracle that are used in these benchmarks. Clear that passing test cases (257 / 672 = 38.2%) and the exact match (193 / 672 = 28.7%) are the most common oracles used in code benchmarks, followed by thresholds (i.e., similarities smaller than a specific threshold).

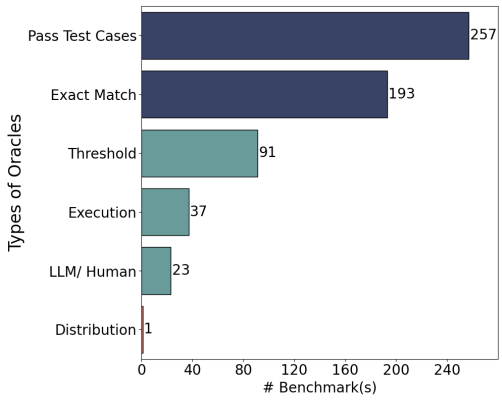

Figure 31. Benchmark Distribution over Test Oracle

Code coverage (Ivanković et al., 2019), as a common oracle for code-related benchmarks, measuring the ratio of covered code by tests, has been widely adopted to determine the output correctness. It should be considered whether a benchmark uses test case passing as a criterion for the correctness of the generated code. Otherwise, a test could be too weak to detect the existence of a defect in the generated code. For example, as pointed out by prior work (Liu et al., 2023b), existing benchmarks such as HumanEval (Chen et al., 2021a) and MBPP (Austin et al., 2021) still suffer from "insufficient tests", allowing incorrect code to pass all the tests without capturing the bugs.

Despite its importance, as shown in Figure 32, among the benchmarks that use test cases as the oracle, only 13.6% considered and reported "test coverage" explicitly in their papers, while 85.0% ignored the test coverage.

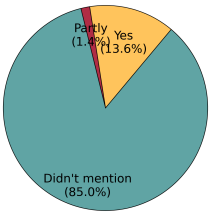

Figure 32. Benchmark Distribution over Test Coverage

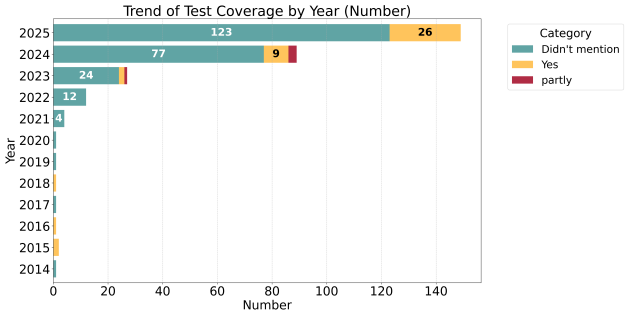

Figure 33. Benchmark Distribution over Test Coverage per Year

The annual distribution (Figure 33) makes this trend even clearer: although many benchmarks in the past three years (2023–2025) did not consider code coverage, the sheer volume of benchmarks in 2025 means that the absolute number of such benchmarks is high, with 24, 77, and 123 benchmarks respectively for 2023, 2024, and 2025. This underscores that, despite growing awareness of evaluation rigor, a substantial number of benchmarks continue to provide incomplete testing, highlighting a persistent threat to the validity and reliability of benchmark-driven assessment.

Furthermore, we dived into 30 representative benchmarks (Listed in Appendix E) and identified an example (Figure 34) from MBPP (Austin et al., 2021) where the test is incorrect. It alerts us that both the quality of the test and the test adequacy (e.g., code coverage) should be considered.

C.4. Statistics about Evaluation

Studied LLMs. We summarize the number of LLMs that have been evaluated in each benchmark evaluation. Among

Wrong Example Tests

```
{
  'source_file': 'charlessutton@: Benchmark
                  Questions Verification V2.ipynb',
  'task_id': 461,
  'prompt': 'Write a python function to count the
             upper case characters in a given string.'
  'test_list': [ "assert upper_ctr('PYthon') == 1",
                 "assert upper_ctr('BigData') == 1",
                 "assert upper_ctr('program') == 0" ]
}
```

Figure 34. An Example of Incorrect Tests from MBPP (Austin et al., 2021).

the 672 benchmarks, 585 of them are evaluated over LLMs, so we show the statistics over them. As shown in Figure 35, most benchmarks were evaluated against six LLMs (10.0% = 59 / 585), followed by three LLMs.

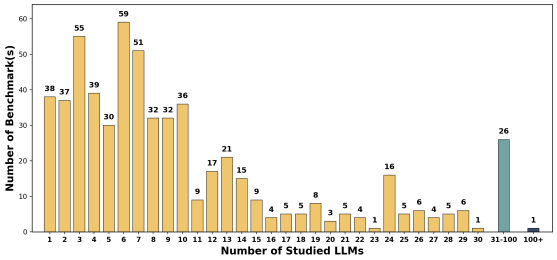

Figure 35. Benchmark Distribution over LLM Experimented

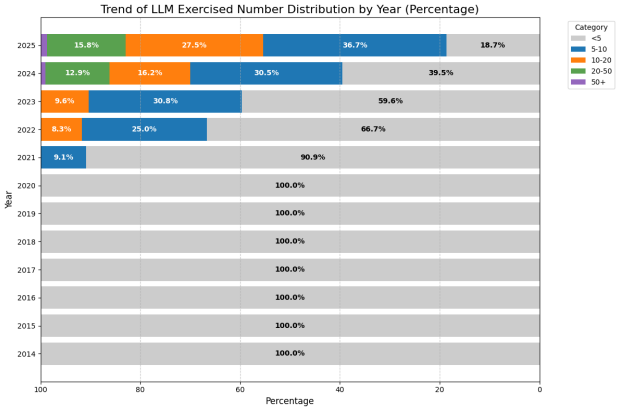

Figure 36. Benchmark Distribution over LLM Experimented Per Year

A positive trend also emerges when examining the year-by-year distribution (Figure 36). In 2024, 39.5% of benchmarks were evaluated against fewer than five LLMs, whereas in 2025, 64.2% of benchmarks (36.7% + 27.5%) were evaluated against 5–20 models. This shift indicates that benchmark studies are increasingly aiming for more comprehensive and generalizable evaluations. However, it also introduces substantially higher computational and financial costs, highlighting the trade-off between evaluation rigor and resource efficiency.

Additionally, we listed the top-10 LLMs by the number of code-related benchmarks they have been evaluated, as shown in Figure 37. GPT series leads significantly with 446 benchmarks, suggesting its widespread adoption and possibly its versatility or superior performance in handling code-related tasks. The rest, including DeepSeek, Qwen, and others, show varying degrees of involvement, with numbers ranging from 269 down to 64 benchmarks for Claude. This figure may provide a reference for choosing which model to evaluate. In addition, it is worth mentioning that different LLMs should be considered for different coding tasks.

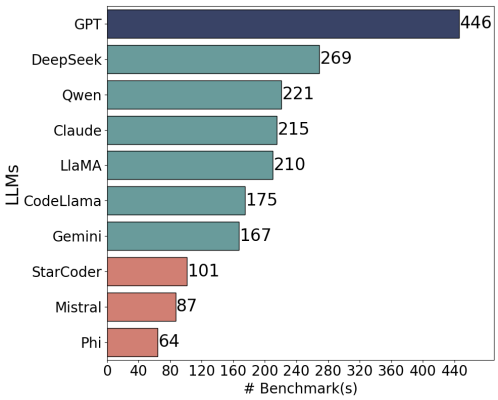

Figure 37. Top-10 Studied LLMs for Code-related Benchmarks

Experiment Environments. The experimental environment (such as the operating system and hardware) is important for the reproduction of the experiment. However, Figure 39 and Figure 38 highlight a significant gap. Only 24.6% of benchmarks document the devices used in their experiments, leaving a substantial 75.4% that do not. The situation appears even more dire when considering os, with only 6.3% of benchmarks documenting the OS used, while a staggering 93.7% neglect to record this information.

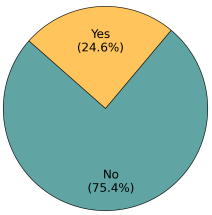

Figure 38. Benchmark Distribution over Recording Experiment Devices

Prompting and Prompting Strategies Prompting has a direct impact on the quality of the LLMs' output results (Wei et al., 2022; He et al., 2024a; Jin et al., 2024). So, we summarized whether different prompting strategies have been

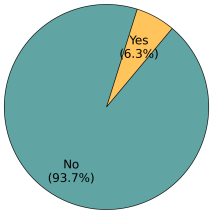

Figure 39. Benchmark Distribution over Recording Experiment OS

evaluated and statistics the distribution. Figure 40 shows the usage of four kinds of prompts: zero-shot, few-shot, chain-of-thought, and retrievals (RAG). From Figure 40, we can see that a vast majority (89.4%) of benchmarks were evaluated in a zero-shot context setting, while only 18.6% of benchmarks were evaluated in a few-shot manner. Even fewer benchmarks were evaluated under the Chain-Of-Thought (CoT) and RAG settings, utilized by only 3.7% and 1.0%.

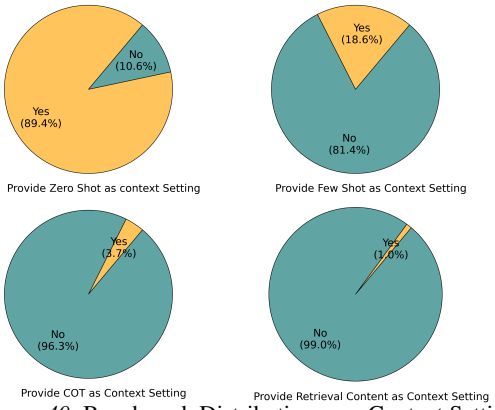

Figure 40. Benchmark Distribution over Context Setting

Prompt Quality The prompt quality also greatly impacts the LLM evaluation (He et al., 2024b). So, carefully designing a prompt needs consideration. However, as shown in Figure 41, 76.7% representative benchmarks (Appendix E) do not validate whether the prompt they used is well-designed.

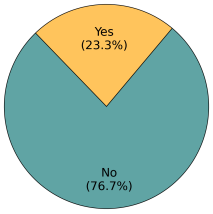

Figure 41. Benchmark Distribution Over Validation of Prompts

Repeated Experiment Given the random nature of LLMs, the experiments are expected to repeat, ensuring the stability and reliability of the results. However, as shown in Figure 42, ***only 33.4% benchmarks went through a repeated experiment***, while a majority of 66.6% opted against repeating the experiment. This reflects a lack of awareness regarding the stability and reproducibility of evaluations.

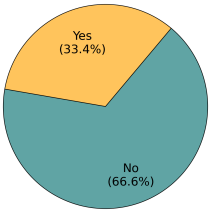

Figure 42. Benchmark Distribution over Repeating the Experiment

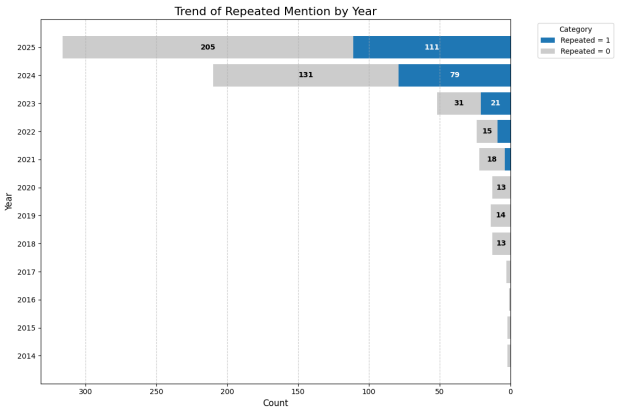

Figure 43. Benchmark Distribution over Repeating Count the Experiment Per Year

Examining trends year by year (Figure 43), the number of benchmark developers who repeated experiments increased in 2025 compared to 2023 and 2024 (31 in 2023, 79 in 2024, and 111 in 2025). However, because the total number of benchmarks grew substantially in 2024 and 2025, the absolute number of benchmarks without repeated experiments remains the highest, totaling 205 benchmarks, which accounts for 30.5% (205/672) of all benchmarks. This indicates that, despite increasing awareness of reproducibility, a significant fraction of benchmarks still lack repeated verification, posing a persistent risk to evaluation reliability.

C.5. Statistics about Analysis

Experiment Explanation. Explaining experiment results is crucial for other practitioners to understand what the outcomes mean in the context of the research questions. So, we investigate whether the representative benchmarks (Appendix E) have explained the experiment results. As shown in Figure 44, 63.3% benchmarks have detailed explanations and analyses of their evaluation results, while still 36.7% have not. Indeed, an explanation contributes to the body of knowledge by making it possible to understand and compare results with previous studies, promoting transparency within the community.

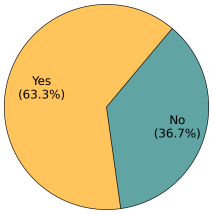

Figure 44. Benchmark Distribution Over Explaining the Experiment

A clear and precise presentation of experimental results is important for enabling robust interpretation and comparison across benchmarks. However, further examination of the 30 representative benchmarks (listed in Appendix E) revealed notable deficiencies in result visualization. As shown in Figure 45, CruxEval (Gu et al., 2024) exhibits unclear experimental result presentation. Specifically, the scatter plot suffers from ambiguous labeling, poor readability of axis values, and inconsistent marker encoding, making it difficult for researchers to extract meaningful insights. Such presentation shortcomings obscure the performance relationships between methods and compromise the benchmark's usability for fair evaluation. To address these issues, benchmarks should adopt standardized and well-documented visualization practices, ensuring results are interpretable, accessible, and reproducible.

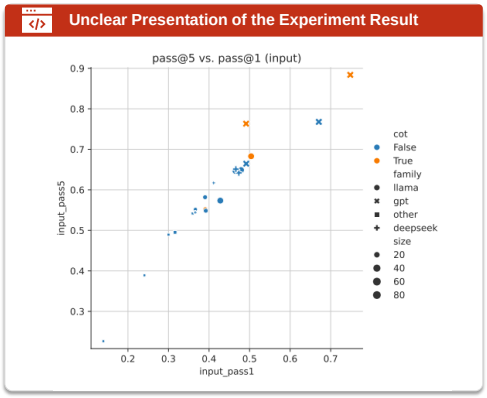

Figure 45. An Example of Unclear Experiment Analysis and Display from CruxEval (Gu et al., 2024)

C.6. Statistics about Release

Data Accessibility. The fundamental requirement for releasing a benchmark is that it must be open-sourced. However, surprisingly, as shown in Figure 46, we observed that 2.2% of the benchmarks are only partially open-sourced (e.g., missing some subjects or tests), and ***14.7% are not open-sourced at all*** (e.g., links/web pages are no longer active).

Fortunately, the year-by-year trend (Figure 47) shows a positive shift toward openness. From 2024 to 2025, the number of open-sourced benchmarks increased substantially, rising

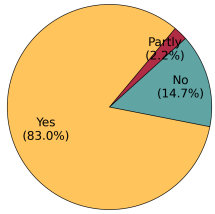

Figure 46. Benchmark Data Availability

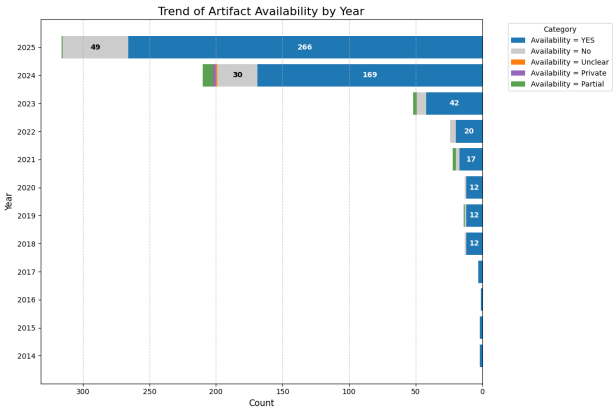

Figure 47. Benchmark Data Availability Count Per Year

from 169 to 266, indicating growing community commitment to transparency, accessibility, and reproducibility.

Prompt Accessibility. Detailed prompts are essential for ensuring the reproducibility and transparency of code-related benchmarks. Yet, Figure 48 indicates that ***38.2% of benchmarks do not provide detailed prompts***, limiting the ability to accurately replicate and evaluate the performance of LLMs. Lack of prompt disclosure highlights a gap in benchmark design practices, as prompts are often indispensable for understanding model performance under specific conditions, raising concerns about the consistency and reproducibility of reported results.

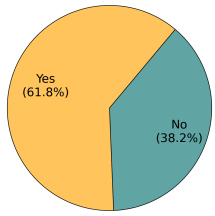

Figure 48. Availability of Prompts

When examined on a year-by-year basis Figure 49, from 2024 to 2025, although the number of benchmarks that release prompts increased from 140 to 233, the number of benchmarks that do not release prompts also grew, from 64 to 82. This indicates that transparency has improved in absolute terms, but has not kept pace with the rapid growth

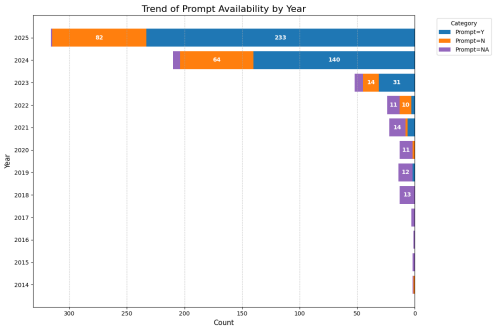

Figure 49. Availability of Prompts Per Year

in benchmark releases.

Logging Info Accessibility. Providing detailed logging information, including comprehensive experimental results, is essential for ensuring transparency, verifiability, and reproducibility in benchmarking research. However, as shown in Figure 50, only ***16.7% of the benchmarks make their experimental results publicly available***, while 80.0% fail to disclose this critical information. Alarmingly, an additional 3.3% provide only partial logging details, further complicating result verification. The absence of complete logging information creates significant barriers for researchers attempting to reproduce experiments or validate reported findings, thereby undermining the reliability of benchmarks. To address this, we emphasize the necessity of making detailed logging information, including intermediate results and metrics, publicly accessible to uphold rigorous scientific standards and foster trustworthy comparisons across models.

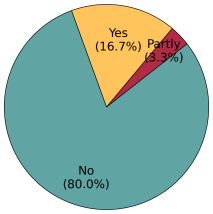

Figure 50. Availability of Logging Information

User Manual Accessibility. A high-quality user manual, such as a well-documented README file, is crucial for enhancing benchmark usability, enabling users to understand the dataset, execute provided scripts, and reproduce results efficiently. However, our analysis revealed that a significant number of benchmarks lack comprehensive user manuals, hindering accessibility and adoption. As depicted in Figure 51, poorly structured or incomplete manuals often omit essential components such as benchmark introductions, usage instructions, and evaluation scripts. This creates unnecessary barriers for researchers who rely on these manuals for setup and experimentation. To address this, we advocate for benchmarks to include clear, standardized user manuals that provide an overview of the benchmark, step-by-step execution guides, and troubleshooting instructions, ensuring a seamless and reproducible user experience.

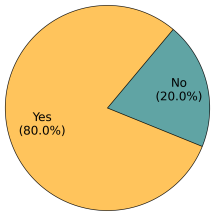

Figure 51. Availability of User Manual

Convenient Evaluation Interface Availability.

Providing convenient evaluation interfaces is essential for enhancing the usability and accessibility of benchmarks, enabling researchers to easily reproduce results and compare models. As shown in Figure 52, ***20% of benchmarks fail to offer any evaluation interfaces***, imposing significant barriers to usability. While a majority of benchmarks (80%) provide such interfaces, including command-line tools, Docker images, or scripts, the absence of standardized and user-friendly evaluation tools in a notable minority of cases highlights an area for improvement. Benchmarks without convenient evaluation interfaces require users to spend additional effort in setup and result verification, which can discourage adoption and hinder reproducibility. To address this, we emphasize the importance of releasing benchmarks with well-documented, ready-to-use evaluation pipelines to promote efficient, reliable, and fair benchmarking practices.

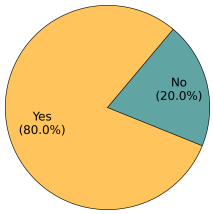

Figure 52. Availability of Convenient Evaluation Interfaces

Temperature Records. One critical parameter for benchmarking is the temperature setting, which influences stochasticity in LLMs. As shown in Figure 53, we observed that ***49.5% of benchmarks fail to record the temperature setting***, hindering reproducibility and fair evaluation. While 50.5% of benchmarks do document this parameter, the majority omission highlights an overlooked yet essential aspect of benchmark transparency.

License Provision. Releasing benchmarks under a clear and accessible license is fundamental for legal compliance and ensuring open collaboration. Figure 54 reveals that ***19.3% of benchmarks do not provide a license***, limiting their usability and distribution. Encouragingly, 80.7% of

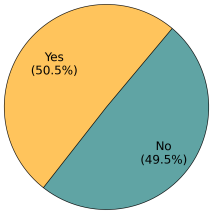

Figure 53. Benchmark Distribution over Recording Temperature

benchmarks do include a license, but the lack of licensing in nearly one-fifth of the benchmarks raises concerns about widespread adoption and usage. These findings emphasize the need for standardized practices in benchmark releases to promote legal clarity and accessibility.

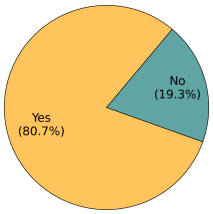

Figure 54. Provision of License

Data Security. Ensuring data security is a critical yet often overlooked aspect of benchmark development. Sensitive information, such as API keys, credentials, or private tokens, should never be included in benchmark releases. However, further investigation into 30 representative benchmarks (listed in Appendix E) revealed instances of sensitive data leakage. As shown in Figure 55, XSemPLR (Zhang et al., 2023b) inadvertently included an ***API key*** in its release, a critical oversight that can expose resources to external exploitation. Similarly, Figure 56 highlights an example from

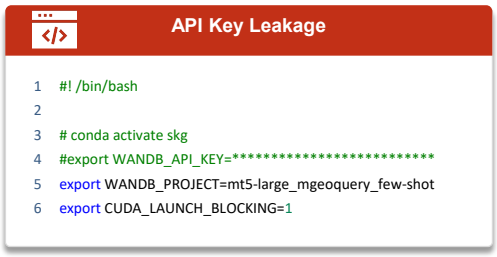

Figure 55. An Example of API Key Leakage in Benchmark Release from XSemPLR (Zhang et al., 2023b).

CrossVul (Nikitopoulos et al., 2021), where ***personal names and email addresses*** were unintentionally disclosed. Such leakage poses risks of unauthorized access and resource misuse, potentially compromising systems and research integrity.

Usability. Clear and comprehensive documentation is crucial for ensuring the usability of benchmarks, as poorly

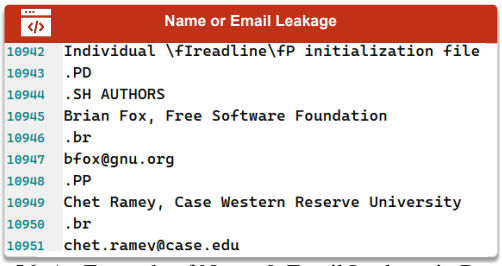

Figure 56. An Example of Name & Email Leakage in Benchmark Release from CrossVul (Nikitopoulos et al., 2021).

written instructions can significantly hinder adoption and reproducibility. We dived into the 30 representative benchmarks (listed in Appendix E) and identified an example where the README file provided insufficient and unclear information. As shown in Figure 57, VulDeePecker (Li et al., 2018b) includes a less-than-ideal ReadMe file, which lacks essential usage instructions and evaluation guidelines, making the benchmark difficult to understand and deploy.

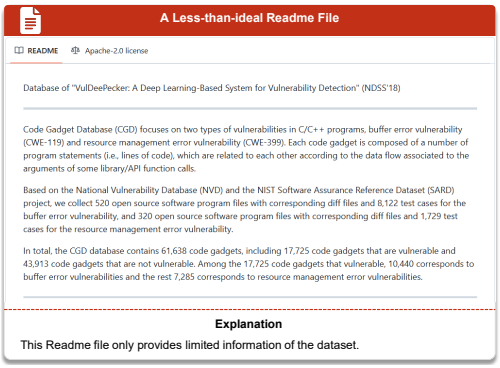

Figure 57. An Example of Unreadable and Hard-to-Use README in Benchmark Release from VulDeePecker (Li et al., 2018b).

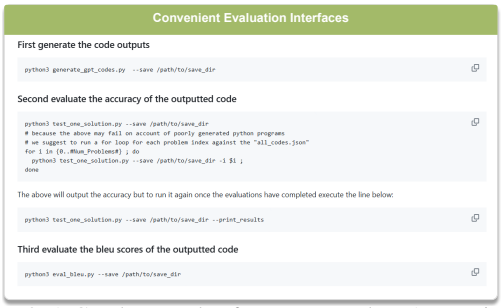

Figure 58. A Good Example of Easy-to-Read README in Benchmark Release from APPS (Hendrycks et al., 2021).

In contrast, Figure 58 highlights APPS (Hendrycks et al., 2021), which provides well-structured and easy-to-follow documentation. The APPS benchmark includes step-by-step instructions for generating, evaluating, and analyzing results, enabling users to efficiently reproduce experiments.

These observations emphasize the importance of high-

quality documentation for benchmarks to enhance accessibility, reduce friction in usage, and foster reproducible research.

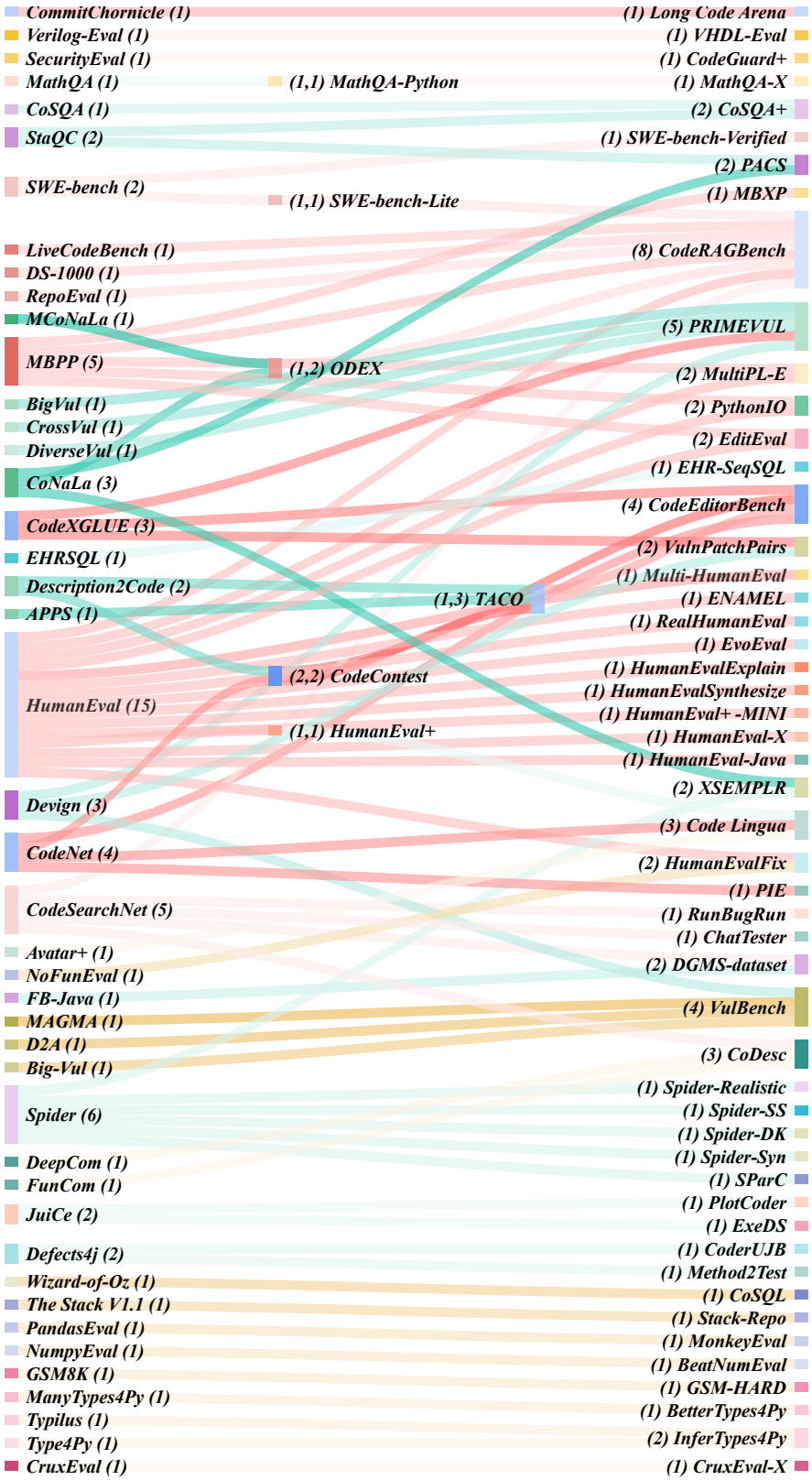

Figure 59. Relationships between Benchmarks

D. Details of Human Study

D.1. Interviewee Selection

The selection of interviewees is pivotal to ensuring the representativeness and relevance of the data collected. This involves identifying individuals with the knowledge or experience pertinent to the research theme.

To this end, we chose graduate students from SE or AI fields who have published at least one paper. This criterion ensures that participants have research experience and judgment capabilities. The focus on SE and AI fields is due to their likely interest in code benchmarks. Particularly, we aimed to recruit individuals who have published papers on code benchmarks to obtain firsthand feedback from experienced benchmark developers.

D.2. Survey Question Design

Questions. The body of the survey was divided into five stages of benchmark development (following Figure 1), with necessary background information provided for each stage. Each criterion in HOW2BENCH was slightly modified to be in the first-person perspective, making it easier for interviewees to empathize and answer the questions from their own viewpoint. Finally, to facilitate comprehension, questions and instructions were translated into both English and Chinese.

Question Setting. To minimize the effort required from respondents, we designed ***single-choice questions*** with four options:

❒ I found it **important**, and I **have done** it.

❒ I found it **important**, although I **haven't done** it.

❒ I found it **not important**, but I **have done** it.

❒ I found it **not important**, and I **wouldn't** do it.

This format is intended to orthogonally explore the correlation between ***awareness*** and ***behavior***.

D.3. Interview Process

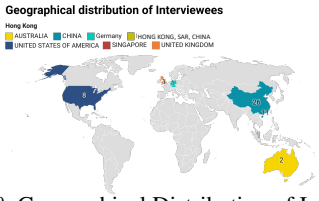

Figure 60. Geographical Distribution of Interviewees

Questionnaire Distribution The questionnaire was distributed via online platforms, targeting academic and professional networks related to SE and AI. *The distribution started on October 27, 2024, and ended on November 27th, 2024*, lasting one month.

Results Collection The responses were automatically collected through the online platform used for distribution.

Survey Screening Since the requirement was for participants who have published papers, responses from those selecting "No" to having published a paper were excluded. Also, incomplete surveys where not all questions were answered were also considered invalid and excluded from the analysis.

D.4. Interview Result Analysis

In total, we collected 50 responses. The respondents were from seven regions, including the United States, the United Kingdom, Germany, Australia, China, and others, as shown in Figure 60. Only one survey was invalid due to the respondent selecting "have not published a paper", leaving **49 valid surveys** for analysis. A breakdown of the respondents' demographics is shown in Figure 61. The detailed responses for all 55 criteria in HOW2BENCH are shown in Figure 62 and Figure 63.

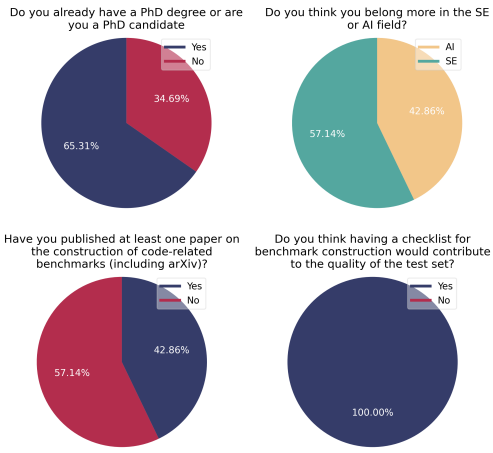

Figure 61. Demography of Interviewees

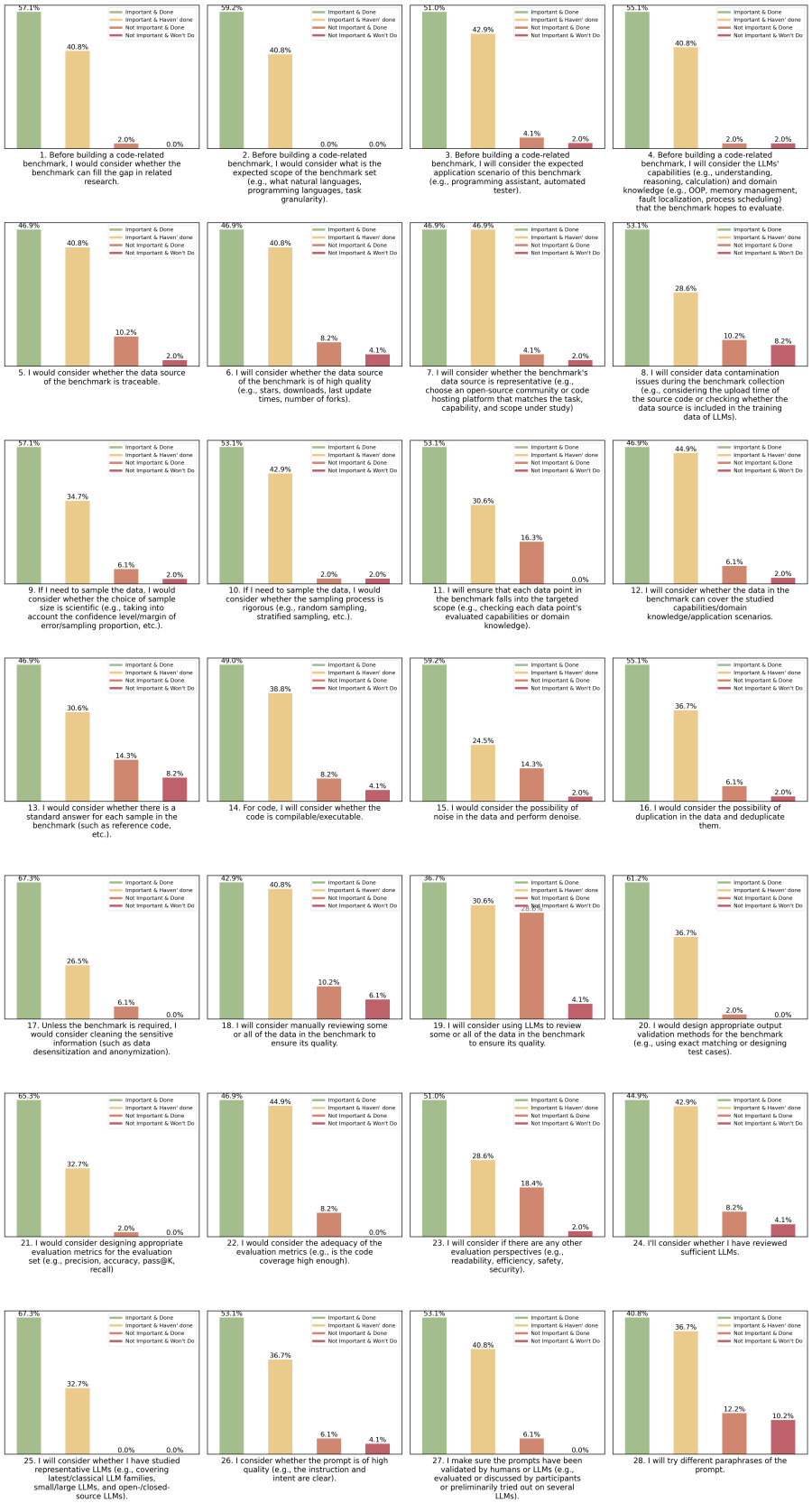

Figure 62. Results of Human Study (Questions 1 - 28

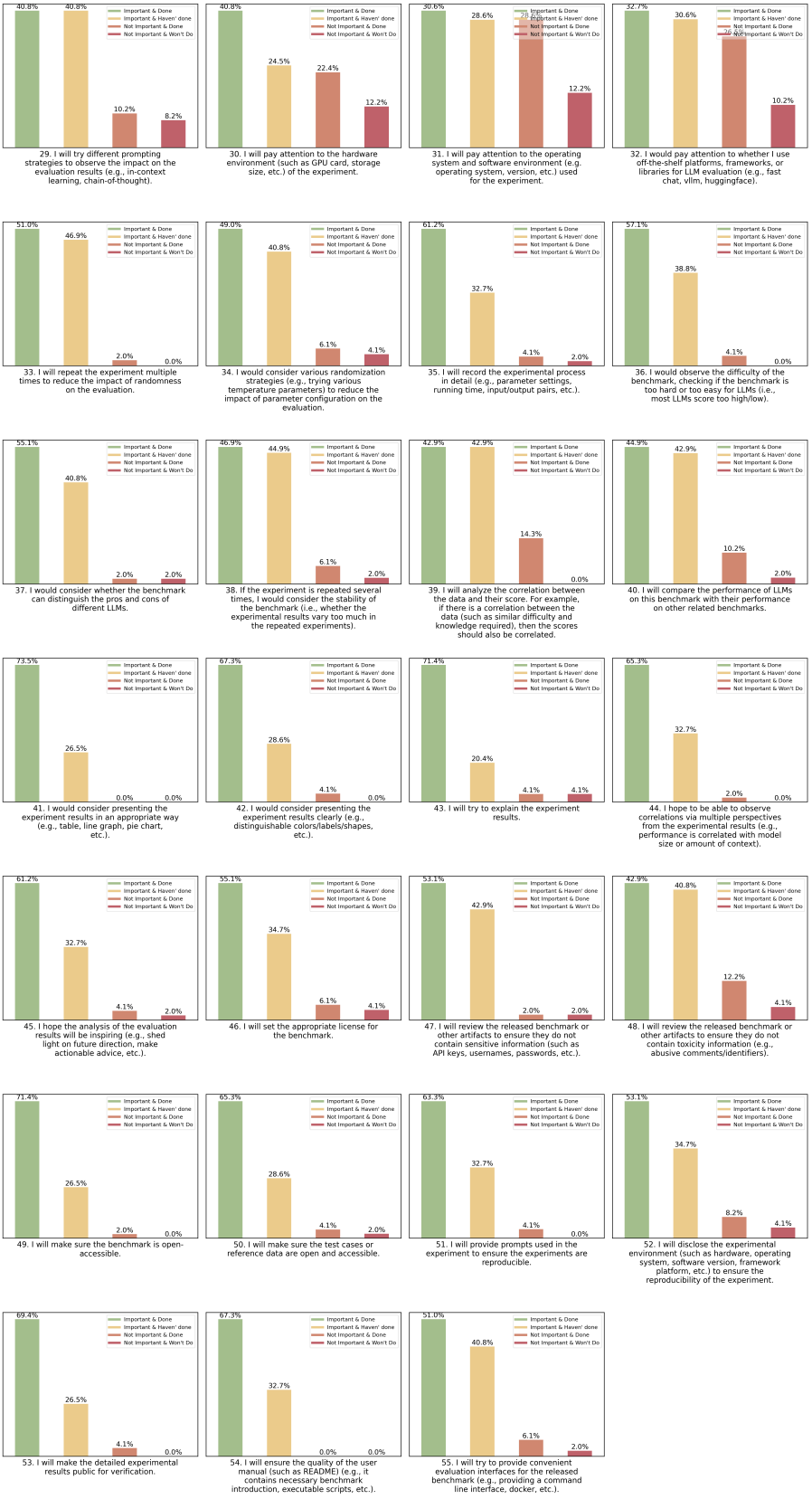

Figure 63. Results of Human Study (Questions 29 - 55

E. List of Studied Benchmarks (Focused Ones)

Code Generation: Five with top citations:

- HumanEval (Chen et al., 2021a)
- MBPP (Austin et al., 2021)
- CodeContest (Li et al., 2022)
- leetcodehardgym (Shinn et al., 2023)
- APPS (Hendrycks et al., 2021)

The latest one as of 31/12/2025:

- CIFE (Gunnu et al., 2025)

Defect Detection: Five with top citations:

- VulDeePecker (Li et al., 2018b)
- Devign (Zhou et al., 2019)
- Chromium and Debian (Chakraborty et al., 2022)
- μVulDeePecker (Zou et al., 2020)
- Synthetic Dataset (Hellendoorn et al., 2020)

The latest one as of 31/12/2025:

- Pushkar et al. (Pushkar et al., 2025a)

Program Repair: Five with top citations:

- Defects4J (Just et al., 2014)
- BFP (Tufano et al., 2019)
- MANYBUGS, INTROCLASS (Le Goues et al., 2015)
- HumanEval-Java (Jiang et al., 2023)
- QuixBugs (Prenner et al., 2022)

The latest one as of 31/12/2025:

- BackportBench (Zhong et al., 2025)

Code Summarization: Five with top citations:

- CODE-NN (Iyer et al., 2016)
- Java-small/med/large (Alon et al., 2019)
- code-summarization-public (Wan et al., 2018)
- HumanEvalPack (Muennighoff et al., 2024)
- Shrivastava et al. (Shrivastava et al., 2023b)

The latest one as of 31/12/2025:

- ContextCRBench (Hu et al., 2025a)

Code Reasoning: Five with top citations:

- LiveCodeBench (Jain et al., 2024)
- CRUXEval-X (Xu et al., 2024)
- xCodeEval (Khan et al., 2024)
- CodeQA (Liu & Wan, 2021)
- RepoQA (Liu et al., 2024b)

The latest one as of 31/12/2025:

- RE2-Bench (Liu et al., 2025a)

F. List of Studied Benchmarks (Full)

We collected and studied 672 code-related benchmarks. We then listed and grouped them by year.

2025:

- Web-Bench (Xu et al., 2025b)
- EFFIBENCH-X (Qing et al., 2025)
- EDIT-Bench (Chi et al., 2025)
- CIRCLE (Chua, 2025)
- CS101-Gold (Havare et al., 2025)
- OSS-Bench (Jiang et al., 2025b)
- CodeArena (Du et al., 2025b)
- RefactorCoderQA (Rahman et al., 2025b)
- SwiftEval (Petrukha et al., 2025)
- CoRe (Xie et al., 2026)
- CodeElo (Quan et al., 2025)
- LongCodeBench (Rando et al., 2025)
- DSCodeBench (Ouyang et al., 2025)
- RustEvo$\hat{2}$ (Liang et al., 2025)
- JITVUL (Yildiz et al., 2025)
- UA-Code-Bench (Syromiatnikov & Ruvinskaya, 2025)
- CodeJudgeBench (Jiang et al., 2025a)
- HumanEval-Judge, MBPP-Judge, BigCodeBench-Judge (Yang et al., 2025a)
- ContextCRBench (Hu et al., 2025a)
- LoCoBench (Qiu et al., 2025)
- CodeMixBench (Sheokand & Sawant, 2025)
- VERINA (Ye et al., 2025)
- SecureAgentBench (Chen et al., 2025a)
- ClassEval-T (Xue et al., 2025)
- ProjectEval (Liu et al., 2025d)
- IFEvalCode (Yang et al., 2025d)
- CodeARC (Wei et al., 2025a)
- TestGenEval (Jain et al., 2025)
- CodePromptEval (Khojah et al., 2025)
- FreshBrew (May et al., 2025)
- CLOVER (Xu et al., 2025a)
- THROWBENCH (Prenner & Robbes, 2025)
- QuanBench (Guo et al., 2025b)
- FrontendBench (Zhu et al., 2025a)
- SolEval (Peng et al., 2025d)
- SWE-Bench-Live (Zhang et al., 2025c)
- SWE-Bench++ (Wang et al., 2025f)
- Defects4Log (Wang et al., 2025i)
- SecVulEval (Ahmed et al., 2025)
- PATCHEVAL (Wei et al., 2025b)
- VulnRepairEval (Wang et al., 2025h)
- FullStackBench (Bytedance-Seed-Foundation-Code-Team et al., 2025)
- CodeIF (Yan et al., 2025b)
- AutoCodeBench (Chou et al., 2025)
- TransLibEval (Xue et al., 2026)
- SWE-Perf (He et al., 2025a)
- TRACY (Gong et al., 2025)
- UnLeakedTestbench (Huang et al., 2025)
- TypyBench (Dong et al., 2025)
- LogicCat (Liu et al., 2025f)
- BIRD-INTERACT (Huo et al., 2025)
- SQL-Synth (Wang et al., 2025b)
- Falcon (Luo et al., 2025a)
- CORGI (Li et al., 2026)
- GITS-Eval (Rondon et al., 2025)
- CWEval (Peng et al., 2025a)
- DI-BENCH (Zhang et al., 2025e)
- LINUXFLBENCH (Zhou et al., 2025)
- ClassEval (Du et al., 2023)
- Deep-Bench (Daghighfarsoodeh et al., 2025)
- MCMD (Tao et al., 2022)

- HumanEvalNext (Koohestani et al., 2025b)
- FEA-Bench (Li et al., 2025c)
- SolBench (Chen et al., 2025d)
- DEFECTS4J-TRANS (Li et al., 2025a)
- CASTLE (Dubniczky et al., 2025)
- CodeIF-Bench (Wang et al., 2025g)
- Multi-SWE-bench (Zan et al., 2025)
- SciReplicate-Bench (Xiang et al., 2025)
- SWE-PolyBench (Rashid et al., 2025)
- APIRAT (Wang et al., 2025a)
- LeetCodeDataset (Xia et al., 2025)
- CoCo-Bench (Yin et al., 2025)
- SecRepoBench (Shen et al., 2025)
- CodeFlowBench (Wang et al., 2026)
- SWE-smith (Yang et al., 2025c)
- WebGen-Bench (Lu et al., 2025)
- SWE-rebench (Badertdinov et al., 2025)
- ResearchCodeBench (Hua et al., 2025)
- WebUIBench (Lin et al., 2025b)
- SWE-Factory (Guo et al., 2026)
- SafeGenBench (Li et al., 2025d)
- Zeng et al. (Zeng et al., 2025a)
- SWE-MERA (Pavel et al., 2025)
- F2STRANS (Zhang et al., 2025d)
- VulCoCo (Bui et al., 2025)
- NoCode-bench (Deng et al., 2025b)
- MRG-Bench (Li, 2025)
- STEPWISE-CODEX-Bench (Yan et al., 2025a)
- CodeFuse-CR-Bench (Guo et al., 2025a)
- SWE-Mirror (Wang et al., 2025e)
- SWE-Bench-Pro (Deng et al., 2025c)
- MultiSpider-2.0 (Pham et al., 2025)
- MULocBench (Zhang et al., 2025i)
- TC-Bench (Luo et al., 2025b)
- Defects4C (Wang et al., 2025d)
- GDPR-Bench-Android (Ran et al., 2025)
- PRDBench (Fu et al., 2025b)
- CodeAlignBench (Mehralian et al., 2025)
- Go-UT-Bench (Pipalani et al., 2025)
- RGym (Shehada et al., 2025)
- CodeFuse-CommitEval (Zhang et al., 2025g)
- PACIFIC (Dreyfuss et al., 2025)
- NL2Repo-Bench (Ding et al., 2026)
- RE2-Bench (Liu et al., 2025a)
- CIFE (Gunnu et al., 2025)
- SWE-EVO (Thai et al., 2026)
- Multi-Docker-Eval (Fu et al., 2025a)
- CodeCriticBench (Zhang et al., 2025a)
- Cui et al. (Cui et al., 2025)
- CodeReviewQA (Lin et al., 2025a)
- BigO(Bench) (Chambon et al., 2025)
- SWR-Bench (Zeng et al., 2025b)
- UniCode (Zheng et al., 2025b)
- SusVibes (Zhao et al., 2025a)
- COFFE (Peng et al., 2025c)
- DependEval (Du et al., 2025a)
- SWA-Bench, SWEE-Bench (Vergopoulos et al., 2025)
- CVE-Bench (Zhu et al., 2025b)
- Obscura (Nikiema et al., 2025)
- BinaryLLMs-Eval (Shang et al., 2025)
- CodeAssistBench (Kim et al., 2025)
- SWE-QA (Peng et al., 2025b)
- RECODE-H (Miao et al., 2025)
- CoReQA (Chen et al., 2025b)
- HumanEval-V (Wang et al., 2025c)
- CAMA (He et al., 2025b)

- C2RUST-BENCH (Sirlanci et al., 2025)
- CRUST-Bench (Khatry et al., 2025)
- OSVBench (Li et al., 2025b)
- VADER (Liu et al., 2025b)
- CodeSense (Roy et al., 2025)
- CETBench (Oza et al., 2025)
- DesignBench (Xiao et al., 2025)
- CoQuIR (Geng et al., 2025)
- MultiCodeIF (Duan et al., 2025)
- CoreCodeBench (Fu et al., 2026)
- WebMMU (Awal et al., 2025)
- ProjectAnalyzer (Gnieciak & Szandala, 2025)
- VulGate (Safdar et al., 2025)
- text2SQL4PM (Yamate et al., 2025)
- E2EDev (Liu et al., 2025c)
- Chart2Code (Tang et al., 2026)
- RealClassEval (Rahman et al., 2025a)
- BackPortBench (Zhong et al., 2025)
- Sun et al. (Sun et al., 2025a)
- Pushkar et al. (Pushkar et al., 2025b)
- RepoTransBench (Wang et al., 2025j)
- DomainCodeBench (Zheng et al., 2025a)
- UniVul (Sun et al., 2025b)
- mHumanEval (Raihan et al., 2025)
- WebCode2M (Gui et al., 2025)
- CodeJudge-Eval (Zhao et al., 2025b)
- HumanEval-XL (Peng et al., 2024)
- Visual-SWE-bench (Zhang et al., 2025f)

2024:

- CodeEditorBench (Guo et al., 2024)
- MHPP (Dai et al., 2024)
- LiveCodeBench (Jain et al., 2024)
- CodeAgentBench (Zhang et al., 2024b)
- CruxEval (Gu et al., 2024)
- BigCodeBench (Zhuo et al., 2024)
- OOPEval (Wang et al., 2024c)
- DevEval (Li et al., 2024a)
- Long Code Arena (Bogomolov et al., 2024)
- CodeRAGBench (Wang et al., 2024f)
- ScenEval (Paul et al., 2024)
- AICoderEval (Xia et al., 2024b)
- VersiCode (Wu et al., 2024b)
- VHDL-Eval (Vijayaraghavan et al., 2024)
- NaturalCodeBench (Zhang et al., 2024d)
- CodeGuard+ (Fu et al., 2024)
- PECC (Haller et al., 2024)
- USACO (Shi et al., 2024b)
- ParEval (Nichols et al., 2024)
- MxEval (Athiwaratkun et al., 2022)
- MMCode (Li et al., 2024c)
- Plot2Code (Wu et al., 2024a)
- ChartMimic (Shi et al., 2024a)
- DebugBench (Tian et al., 2024)
- PythonIO (Zhang et al., 2024e)
- StaCCQA (Yang et al., 2024a)
- RepoQA (Liu et al., 2024b)
- PRIMEVUL (Ding et al., 2024a)
- VulDetectBench (Liu et al., 2024d)
- ProCQA (Li et al., 2024e)
- CoSQA+ (Gong et al., 2024a) (Huang et al., 2021)
- JavaBench (Cao et al., 2024a)
- HumanEvo (Zheng et al., 2024)
- REPOEXEC (Hai et al., 2024)
- EHR-SeqSQL (Ryu et al., 2024)
- BookSQL (Kumar et al., 2024)
- AMBROSIA (Saparina & Lapata, 2024)

- WUB, WCGB (Yun et al., 2024)
- RES-Q (LaBash et al., 2024)
- PythonSaga (Yadav et al., 2024a)
- Mercury (Du et al., 2024)
- ENAMEL (Qiu et al., 2024a)
- RealHumanEval (Mozannar et al., 2024)
- CoderUJB (Zeng et al., 2024)
- EvoEval (Xia et al., 2024a)
- ML-Bench (Liu et al., 2023c)
- VerilogEval (Pinckney et al., 2024)
- CodeApex (Fu et al., 2023)
- HumanEvalPack (Muennighoff et al., 2024)
- HumanEval+ (Liu et al., 2023a)
- HumanEval-X (Zheng et al., 2023a)
- XCodeEval (Khan et al., 2024)
- CoderEval (Yu et al., 2023)
- CodeXGLUE (Lu et al., 2021)
- VulnPatchPairs (Risse & Böhme, 2024)
- WikiSQL (Zhong et al., 2017)
- CrossCodeEval (Ding et al., 2023)
- SWE-bench (Jimenez et al., 2024)
- BAIRI et al. (Bairi et al., 2024)
- BioCoder (Tang et al., 2024)
- RepoBench (Liu et al., 2024c)
- NoFunEval (Singhal et al., 2024)
- CoCoMIC (Ding et al., 2024b)
- Java-small/med/large (Alon et al., 2019)
- FixEval (Haque et al., 2023)
- CommitBench (Schall et al., 2024)
- InfiAgent-DABench (Hu et al., 2024)
- InfiBench (Li et al., 2024d)
- Design2Code (Si et al., 2024)
- MatPlotBench (Yang et al., 2024c)
- EditEval (Li et al., 2024b)
- D1, D2, D3 (Huang et al., 2024)
- RepoEval (Liao et al., 2024)
- BetterTypes4Py, InferTypes4Py (Wei et al., 2023)
- HumanEval-Java (Jiang et al., 2023)
- PIE (Shypula et al., 2024)
- EvalGPTFix (Zhang et al., 2023a)
- EHRSQL (Lee et al., 2023)
- Spider2-V (Cao et al., 2024c)
- TESTEVAL (Wang et al., 2024d)
- ChatTester (Yuan et al., 2023a)
- Code Lingua (Pan et al., 2024a)
- EffiBench (HUANG et al., 2024)
- CRUXEval-X (Xu et al., 2024)
- DomainEval (Zhu et al., 2024)
- SWE-Bench+ (Aleithan et al., 2024)
- eyeballvul (Chauvin, 2024)
- ComplexCodeEval (Feng et al., 2024)
- BabelBench (Wang et al., 2024e)
- CRQBench (Dinella et al., 2024)
- R2C2-Coder (Deng et al., 2025a)
- SAFIM (Gong et al., 2024b)
- McEval (Chai et al., 2024)
- PolyHumanEval (Tao et al., 2024)
- RustRepoTrans (Ou et al., 2025)
- RAPID (Ma et al., 2024)
- BEAVER (Chen et al., 2025c)
- Spider2.0 (Lei et al., 2025)
- Wu et al. (Wu et al., 2024c)
- BigTable-0.2k (Zhang et al., 2024a)
- SWT-Bench (Mündler et al., 2025)
- CodeSecEval (Wang et al., 2024b)
- SWE-bench-java (Zan et al., 2024)

- TypeEvalPy/PyCG (Venkatesh et al., 2025)
- M²RC-EVAL (Liu et al., 2024a)
- MdEval (Liu et al., 2025e)
- CPP-UT-Bench (Bhargava et al., 2024)
- OBFUSEVAL (Zhang et al., 2025h)
- ExecRepoBench (Yang et al., 2024b)
- VulnLLMEval (Zibaeirad & Vieira, 2024)
- TestBench (Zhang et al., 2024c)
- Codev-Bench (Pan et al., 2024b)
- SWE-bench Multimodal (Yang et al., 2025b)
- HumanEval-V (Zhang et al., 2025b)
- HumanEval_T (Bradbury & More, 2025)
- TDD-Bench Verified (Ahmed et al., 2024)

2023:

- MCoNaLa (Wang et al., 2023b)
- MultiPL-E (Cassano et al., 2022)
- ODEX (Wang et al., 2022)
- TACO (Li et al., 2023b)
- DOTPROMPTS, MGDMICROBENCH (Agrawal et al., 2023)
- StudentEval (Babe et al., 2024)
- CodeTransOcean (Yan et al., 2023)
- G-TransEval (Jiao et al., 2023)
- AVATAR (Ahmad et al., 2023)
- RunBugRun (Prenner & Robbes, 2023)
- VulBench (Gao et al., 2023b)
- DiverseVul (Chen et al., 2023)
- Hellendoorn et al. (Hellendoorn et al., 2020)
- XSemPLR (Zhang et al., 2023b)
- BIRD (Li et al., 2023a)
- Stack-Repo (Shrivastava et al., 2023a)
- RepoEval (Liao et al., 2024)
- MTPB (Nijkamp et al., 2022)
- ARCADE (Yin et al., 2023)
- Shrivastava et al. (Shrivastava et al., 2023b)
- Grag et al. (Garg et al., 2022)
- GSM-HARD (Gao et al., 2023a)
- InferredBugs (Jin et al., 2023)
- LeetcodeHardGym (Shinn et al., 2023)
- APIBench (Patil et al., 2023)
- ClassEval (Du et al., 2023)
- CommitChronicle (Eliseeva et al., 2023)
- TeCo (Nie et al., 2023)
- TESTPILOT (Schäfer et al., 2024)

2022:

- AixBench (Hao et al., 2022)
- TypeBugs (Oh & Oh, 2022)
- XLCoST (Zhu et al., 2022)
- CS1QA (Lee et al., 2022)
- Chromium and Debian (Chakraborty et al., 2022)
- Spider-Realistic (Deng et al., 2021)
- Spider-SS (Gan et al., 2022)
- DSP (Chandel et al., 2022)
- CodeContest (Li et al., 2022)
- PandasEval, NumpyEval (Zan et al., 2022b)
- TorchDataEval, MonkeyEval, BeatNumEval (Zan et al., 2022a)
- DS-1000 (Lai et al., 2023)
- MCMD (Tao et al., 2022)
- ExeDS (Huang et al., 2022)
- QuixBugs (Prenner et al., 2022)
- ManyTypes4Py v0.7 (Mir et al., 2022)
- Lyra (Liang et al., 2022)

2021:

- SySeVR (Li et al., 2018a)

- Ling&Wu et al. (Ling et al., 2021)
- Chen et al. (Chen et al., 2021b)
- MBPP, MathQA-Python (Austin et al., 2021)
- HumanEval (Chen et al., 2021a)
- APPS (Hendrycks et al., 2021)
- Berabi et al. (Berabi et al., 2021)
- CrossVul (Nikitopoulos et al., 2021)
- PYPIBUGS, RANDOMBUGS (Allamanis et al., 2021)
- D2A (Zheng et al., 2021)
- CodeQA (Liu & Wan, 2021)
- Spider-DK (Gan et al., 2021b)
- KaggleDBQA (Lee et al., 2021)
- SEDE (Hazoom et al., 2021)
- Spider-Syn (Gan et al., 2021a)
- CoDesc (Hasan et al., 2021)
- Methods2Test (Tufano et al., 2022)
- Rozière et al. (Rozière et al., 2022)

2020:

- Lachaux&Roziere et al. (Rozière et al., 2020)
- μVulDeePecker (Zou et al., 2020)
- CosBench (Yan et al., 2020)
- PACS (Heyman & Cutsem, 2020)
- Criteria2SQL (Yu et al., 2020)
- SQUALL (Shi et al., 2020)
- Hu et al. (Hu et al., 2019)
- CodeSearchNet Challenge (Husain et al., 2019)
- MIMICSQL (Wang et al., 2020)
- Atlas (Watson et al., 2020)
- Liu et al. (Liu et al., 2022)
- Android (Agarwal et al., 2020)
- CCSD (Liu et al., 2021)

2019:

- BFP (Tufano et al., 2019)
- SARD (Lin et al., 2019)
- Spider (Yu et al., 2018)
- JuICe (Agashe et al., 2019)
- Nguyen et al. (Nguyen et al., 2019)
- Lin et al. (Lin et al., 2021)
- Zhou et al. (Zhou et al., 2019)
- CoSQL (Yu et al., 2019a)
- SParC (Yu et al., 2019b)
- Malik (Malik et al., 2019)
- LeClair (LeClair et al., 2019)

2018:

- CoNaLa (Yin et al., 2018)
- DeepCom (Hu et al., 2018a)
- TL-CodeSum (Hu et al., 2018b)
- code-summarization-public (Wan et al., 2018)
- Russell et al. (Russell et al., 2018)
- VulDeePecker (Li et al., 2018b)
- Lin et al. (Lin et al., 2018)
- StaQC (Yao et al., 2018)
- Advising (Finegan-Dollak et al., 2018)
- ConCode (Iyer et al., 2018)
- NNGen (Liu et al., 2018)
- Gu et al. (Gu et al., 2018)

2017:

- QuixBugs (Lin et al., 2017)
- the DeepFix dataset (Gupta et al., 2017)
- Barone et al. (Barone & Sennrich, 2017)

2016:

- CODE-NN (Iyer et al., 2016)
- Mou et al. (Mou et al., 2016)

2015:

- MANYBUGS, INTROCLASS (Le Goues et al., 2015)

2014:

- Defects4j (Just et al., 2014)
- BigCloneBench (Svajlenko et al., 2014)

G. Guideline

Finally, for ease of printing and use, we organized the guideline HOW2BENCH into a clear, color-coded checklist (4 pages in total) that is easy to print, attached at the end of the paper.

	HOW-TO-BENCH (1/4)		
	Phase 1. Benchmark Design	**Priority**	☑
1	**Consider whether the benchmark can fill the gap in related research.**	★★★	☐
2	**Consider what is the expected scope of the benchmark set** (e.g., what natural languages, programming languages, task granularity)**.**	★★★	☐
3	**Consider the expected application scenario of this benchmark** (e.g., programming assistant, automated tester).	★★★	☐
4	**Consider the LLMs' capabilities** (e.g., understanding, reasoning, calculation) **and domain knowledge** (e.g., OOP, memory management, fault localization, process scheduling) **that the benchmark hopes to evaluate.**	★★★	☐
	Phase 2. Benchmark Construction (1/2)	**Priority**	☑
5	**Consider whether the data source of the benchmark is traceable.**	★	☐
6	**Consider whether the data source of the benchmark is of high quality** (e.g., stars, downloads, last update times, number of forks).	★★	☐
7	**Consider whether the benchmark's data source is representative** (e.g., choose an open-source community or code hosting platform that matches the task, capability, and scope under study)	★	☐
8	**Consider data contamination issues during the benchmark collection** (e.g., considering the upload time of the source code or checking whether the data source is included in the training data of LLMs).	★	☐
9	**If data sampling is needed, consider whether the choice of sample size is scientific** (e.g., considering the confidence level/margin of error/sampling proportion, etc.).	★	☐
10	**If data sampling is needed, consider whether the sampling process is rigorous** (e.g., random sampling, stratified sampling, etc.).	★	☐
11	**Ensure each data point in the benchmark falls into the targeted scope** (e.g., checking each data point's evaluated capabilities or domain knowledge).	★	☐
12	**Consider whether the data in the benchmark can cover the studied capabilities/domain knowledge/application scenarios.**	★★★	☐
13	**Consider whether there is a standard answer for each sample in the benchmark** (such as reference code, etc.).	★★	☐
14	**For code, consider whether the code is compilable/executable.**	★	☐
15	**Consider the possibility of noise in the data and perform denoise.**	★★	☐
16	**Consider the possibility of duplication in the data and deduplicate them.**	★★	☐
17	**Clean the sensitive information** (such as data desensitization and anonymization) **unless the benchmark is deliberately designed so.**	★★	☐

HOW-TO-BENCH (2/4)			
	Phase 2. Benchmark Construction (2 / 2)	**Priority**	☑
18	**Manually review some or all of the data in the benchmark to ensure its quality.**	★★	☐
19	**Use LLMs to review some or all of the data in the benchmark to ensure its quality.**	★	☐
20	**Design appropriate output validation methods for the benchmark** (e.g., using exact matching or designing test cases).	★★★	☐
21	**Design appropriate evaluation metrics for the evaluation set** (e.g., precision, accuracy, pass@K, recall).	★★★	☐
22	**Consider the adequacy of the evaluation metrics** (e.g., is the code coverage high enough).	★★★	☐
23	**Consider if there are any other evaluation perspectives** (e.g., readability, efficiency, safety, security).	★	☐
	Phase 3. Benchmark Evaluation	**Priority**	☑
24	**Consider whether sufficient LLMs are evaluated.**	★	☐
25	**Consider whether representative LLMs** (e.g., covering latest/classical LLM families, small/large LLMs, and open-/closed-source LLMs) **are evaluated.**	★	☐
26	**Consider whether the prompt is of high quality** (e.g., the instruction and intent are clear).	★★★	☐
27	**The prompts have been validated by humans or LLMs** (e.g., evaluated or discussed by participants or preliminarily tried out on several LLMs).	★	☐
28	**Try different paraphrases of the prompt.**	★	☐
29	**Try different prompting strategies to observe the impact on the evaluation results** (e.g., in-context learning, chain-of-thought).	★★	☐
30	**Pay attention to the hardware environment** (such as GPU card, storage size, etc.) of the experiment.	★★★	☐
31	**Pay attention to the operating system and software environment** (e.g. operating system, version, etc.) **used for the experiment.**	★★★	☐
32	**Pay attention to the off-the-shelf platforms, frameworks, or libraries for LLM evaluation** (e.g., fast chat, vllm, huggingface) **that are used.**	★★	☐
33	**Repeat the experiment multiple times to reduce the impact of randomness on the evaluation.**	★	☐
34	**Consider various randomization strategies** (e.g., trying various temperature parameters) **to reduce the impact of parameter configuration on the evaluation.**	★★	☐
35	**Record the experimental process in detail** (e.g., parameter settings, running time, input/output pairs, etc.).	★★★	☐

HOW-TO-BENCH (3/4)			
	Phase 4. Benchmark Analysis	**Priority**	☑
36	**Observe the difficulty of the benchmark, checking if the benchmark is too hard or too easy for LLMs** (i.e., most LLMs score too high/low).	★★	☐
37	**Consider whether the benchmark can distinguish the pros and cons of different LLMs.**	★	☐
38	**If the experiment is repeated several times, consider the stability of the benchmark** (i.e., whether the experimental results vary too much in the repeated experiments).	★	☐
39	**Analyze the correlation between the data and their score**. For example, if there is a correlation between the data (such as similar difficulty and knowledge required), then the scores should also be correlated.	★★	☐
40	Compare the performance of LLMs on this benchmark with their performance on **other related benchmarks**.	★	☐
41	**Consider presenting the experiment results in an appropriate way** (e.g., table, line graph, pie chart, etc.).	★★★	☐
42	**Consider presenting the experiment results clearly** (e.g., distinguishable colors/labels/shapes, etc.).	★★★	☐
43	**Explain the experiment results**.	★★★	☐
44	**Observe correlations via multiple perspectives from the experimental results** (e.g., performance is correlated with model size or amount of context).	★	☐

HOW-TO-BENCH (4/4)			
	Phase 5. Benchmark Release	**Priority**	☑
45	**The analysis of the evaluation results will be inspiring** (e.g., shed light on future direction, make actionable advice, etc.).	★	☐
46	**Set the appropriate license for the benchmark.**	★★★	☐
47	**Review the released benchmark or other artifacts to ensure they do NOT contain sensitive information** (e.g., API keys, usernames, passwords, etc.).	★★★	☐
48	**review the released benchmark or other artifacts to ensure they do NOT contain toxicity information** (e.g., abusive comments/identifiers).	★★★	☐
49	**Make sure the benchmark is open-accessible.**	★★★	☐
50	**Make sure the test cases or reference data are open and accessible.**	★★★	☐
51	**Provide prompts used in the experiment to ensure the experiments are reproducible.**	★★★	☐
52	**Disclose the experimental environment** (e.g., hardware, operating system, software version, framework platform) **to ensure the reproducibility of the experiment.**	★★★	☐
53	**Make the detailed experimental results public for verification.**	★★★	☐
54	**Ensure the quality of the user manual** such as README (e.g., it contains necessary benchmark introduction, executable scripts, etc.).	★★	☐
55	**Provide convenient evaluation interfaces for the released benchmark** (e.g., providing a command line interface, docker, etc.).	★★	☐