From Conversation to Evaluation: Benchmarking LLMs on Development Knowledge via SimpleDevQA

Jing Zhang[1,*], Lianghong Guo[1,*], Yanlin Wang[1,†], Terry Yue Zhuo[3], Yong Wang, Mingwei Liu[1], Jiachi Chen[6], Ensheng Shi [2], Yuchi Ma[2], Hongyu Zhang[4], Zibin Zheng[1]

[1]Sun Yat-Sen University, Zhuhai Key Laboratory of Trusted Large Language Models, [2]Huawei CodeArts Model Team, [3]Monash University, [4]Chongqing University, [5]The State Key Laboratory of Blockchain and Data Security, Zhejiang University

https://github.com/DeepSoftwareAnalytics/SimpleDevQA

Abstract

The Development Knowledge Question Answering (Dev Knowledge QA) task aims to provide accurate natural language answers to knowledge-seeking questions during software development. To investigate the importance of Dev Knowledge QA in AI-assisted software development and to what extent it has been explored, we conduct a preliminary analysis on real user–LLM dialogues from WildChat. Our findings indicate that Dev Knowledge QA plays a significant role in real-world software development scenarios, and these raw dialogues cannot be directly used to construct a Dev Knowledge QA benchmark. Existing Dev Knowledge QA benchmarks are limited in development knowledge scope and often not built from real user queries. To bridge this gap, we design a three-phase pipeline that transforms real-world dialogue into simple development knowledge-seeking QA pairs. Through this pipeline, we introduce SimpleDevQA, a multilingual Dev Knowledge QA benchmark inspired by real user dialogues. This dataset covers three languages (English, Chinese, and Russian), and focuses on questions with unique, short, and verifiable answers, making evaluation more accurate and simple. Extensive experiments with 18 mainstream LLMs show that closed-source models generally perform best on SimpleDevQA. We also find that RAG-based knowledge injection improves accuracy, and that Dev Knowledge QA performance correlates with both model confidence and code-generation capability. To facilitate the replication study, we have released our data and code at: `https://github.com/DeepSoftwareAnalytics/SimpleDevQA`.

1 Introduction

The Development Knowledge Question Answering (Dev Knowledge QA) task focuses on answering knowledge-based questions that arise during software development. Effectively responding to these diverse, knowledge-seeking questions is crucial for boosting developer productivity and enhancing software quality. With the increasing integration of LLMs into developer workflows, Dev Knowledge QA has emerged as a critical area for research and evaluation (da Silva et al., 2020; Zheng et al., 2025; Shi et al., 2025).

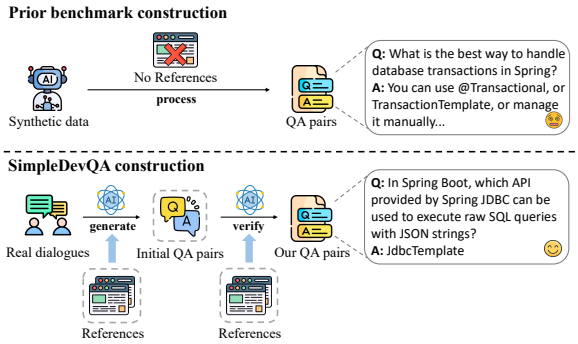

Figure 1: **SimpleDevQA vs. prior Dev Knowledge QA benchmarks.** Synthetic QA pairs (top) lacks references and may admit incorrect answers, making automated evaluation unreliable. Real dialogue-derived QA pairs produced by SimpleDevQA (bottom) has a single verifiable gold answer, supported by references, enabling reliable and accurate evaluation.

To investigate the importance of Dev Knowledge QA in AI-assisted software development scenarios, we conduct a preliminary study on WildChat (Zhao et al., 2024), a dataset including one million user–ChatGPT (Welsby and Cheung, 2023) conversations. Considering the large scale and topic diversity of WildChat, we filter and sample it to obtain WildChat-Dev-Lite, a software development-related subset of WildChat. We obtain three findings based on the preliminary study on the WildChat-Dev-Lite:

- The Dev Knowledge QA task accounts for the highest proportion (39.6%) of all tasks

[*]Jing Zhang and Lianghong Guo have equal contribution.
[†]Yanlin Wang is the corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 37649–37663
July 2-7, 2026

observed during user interactions with LLMs.
- In real Dev Knowledge QA dialogues, only 27.5% of dialogues focus on code understanding.
- Only 17.1% of real Dev Knowledge QA dialogues could be used for constructing a benchmark.

Through the preliminary study, we identify the importance of constructing a Dev Knowledge QA benchmark. However, existing Dev Knowledge QA benchmarks have the following two key problems:

P1: Limited development knowledge scope. Existing Dev Knowledge QA benchmarks mainly focus on code understanding (Liu and Wan, 2021; Lee et al., 2022; Chen et al., 2025; Hu et al., 2024; Liu et al., 2024b; Li et al., 2024c) and do not cover other knowledge related to the development process. For instance, CS1QA (Lee et al., 2022) primarily evaluates code understanding in programming education scenarios. However, based on Section 3.2 from our preliminary study, practical software development demands a much broader knowledge.

P2: Not built from real user queries. Some existing Dev Knowledge QA benchmarks (Liu and Wan, 2021; Liu et al., 2024b) are not built from real user queries from real-world development scenarios, which can not reflect the real demands of developers (Figure 1 top). For example, RepoQA (Liu et al., 2024b) focuses only on long-context tasks from code repositories, which fails to fully reflect genuine developer task demands in real development scenarios.

In this paper, we propose SimpleDevQA, a Dev Knowledge QA benchmark built from real developer dialogues, to address the aforementioned problems. Based on the findings in our preliminary study, we design and implement a three-phase data construction pipeline to convert real-world SE-related conversations into a Dev Knowledge QA benchmark. When constructing the benchmark, we focus only on software development knowledge-seeking questions that have a single correct answer, which helps us avoid the open-endedness of language models, following prior work (Wei et al., 2024; He et al., 2024). This is important because it makes measuring factuality much easier, though it may leave out questions that could have multiple valid responses (Zhang et al., 2025a; Guan et al., 2024). Additionally, each QA pair is also accompanied by multiple web-retrieved references, which can be used to verify the factual accuracy of the answer (Figure 1 down).

We summarize the main contributions as follows:

- We conduct a preliminary study to assess the importance of Dev Knowledge QA in AI-assisted software development.
- We propose SimpleDevQA, a multilingual Dev Knowledge QA benchmark inspired by real user dialogues.
- We construct a pipeline that converts real-world conversations into a Dev Knowledge QA benchmark.
- We conduct extensive experiments on SimpleDevQA and find that the performance of LLMs in Dev Knowledge QA.

2 Related Work

2.1 Dev Knowledge QA Benchmarks

Numerous Dev Knowledge QA benchmarks have emerged within the field of software engineering benchmark. s (Liu and Wan, 2021; Lee et al., 2022; Chen et al., 2025; Hu et al., 2024; Liu et al., 2024b; Li et al., 2024c,d).

Existing Dev Knowledge QA benchmarks rely on questions grounded in a given code snippet, while overlooking broader development knowledge (Lee et al., 2022; Li et al., 2024c; Hu et al., 2024). For example, CodeQA (Liu and Wan, 2021) targets snippet-level syntax/API understanding, CoReQA (Chen et al., 2025) uses code-review history from GitHub issues/comments, RepoQA (Liu et al., 2024b) evaluates repository-level multi-turn and long-context understanding, and InfiBench (Li et al., 2024d) scales to free-form QA from high-quality Stack Overflow questions. The comparison between SimpleDevQA and other Dev Knowledge QA benchmarks is shown in Table 2.

2.2 Real-World Software QA Datasets

Numerous software QA datasets have been constructed from online QA communities (Wang et al., 2022; Dhingra et al., 2017; Wu et al., 2023). For example, the StaQC (Yao et al., 2018) focuses on matching natural language questions with relevant code snippets, the TechQA corpus (Castelli et al., 2019) consists of questions from a technical forum. However, these real QA datasets often yield verbose answers, contain subjective opinions, or present multiple solutions, making it difficult to directly assess LLMs' performance.

2.3 Factuality Benchmarks

Factual evaluation of LLMs has garnered academic attention, leading to the development of several representative benchmarks (Chern et al., 2023; Zhong et al., 2024; Huang et al., 2023; Srivastava et al., 2022; Zhang et al., 2025b). For example, TruthfulQA (Lin et al., 2021) assesses whether models generate answers that sound reasonable but are actually false. HallEval (Li et al., 2023) measures hallucination across QA, summarization, and dialogue tasks. Additionally, OpenAI recently released SimpleQA (Wei et al., 2024) and Chinese SimpleQA (He et al., 2024), which enable reliable factual accuracy assessment. However, current factual benchmarks remain limited to general domains, leaving professional fields like software development without dedicated evaluation frameworks.

3 Preliminary Study

To investigate the importance of the Dev Knowledge QA task in real-world development scenarios, we conduct a preliminary study on WildChat (Zhao et al., 2024). Given the diverse topics in WildChat's dialogues, we extract software development-related dialogues for our research. For the detailed procedure, we refer our readers to Appendix B.

3.1 Importance Analysis of Dev Knowledge QA

To investigate the importance of Dev Knowledge QA, we analyze the distribution of Dev Knowledge QA tasks in real-world development scenarios when users utilize LLMs. We manually filter dialogues from WildChat-Dev-Lite that are unrelated to software development and classify the real-world dialogues into 8 task categories. The detailed classification procedure is provided in the Appendix C. Then, we perform a statistical analysis on the classified dialogues in the Figure 2.

As illustrated in the Figure 2 below, we find that the Dev Knowledge QA task (39.6%) accounts for the highest proportion among all tasks, surpassing the code generation task (32.3%), illustrating the high popularity of the Dev Knowledge QA task in real-world development scenarios.

3.2 Topic Analysis of Dev Knowledge QA

Since previous Dev Knowledge QA benchmarks primarily focus on code understanding (Liu and Wan, 2021; Hu et al., 2024; Lee et al., 2022; Liu et al., 2024b), we analyze whether existing benchmarks can fully evaluate real Dev Knowledge QA capabilities.

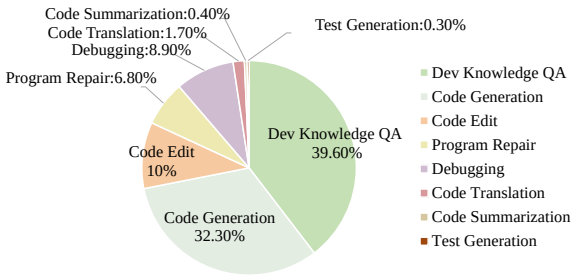

Figure 2: The distribution of SE tasks in WildChat-Dev-Lite dialogues.

Through manual annotation of 3,098 Dev Knowledge QA dialogues, we find that only 851 (approximately 27.5%) involve questions centered on specific code snippets. In contrast, the remaining 2,870 dialogues (approximately 72.5%) consisted of queries seeking factual information about other aspects of development knowledge, such as system design, database management, and computer architecture, etc. Thus, existing Dev Knowledge QA benchmarks can not fully evaluate real Dev Knowledge QA capabilities.

3.3 Challenge Analysis of Constructing Dev Knowledge QA Benchmark

We further investigate whether these real-world development-related dialogues can be used to construct a Dev Knowledge QA benchmark.

First, We randomly sample 1,000 Dev Knowledge QA dialogues for manual verification. Then, we manually verify each Dev Knowledge QA instance against three critical criteria proposed by previous research (Wei et al., 2024; He et al., 2024): (1) the question must have a single answer, (2) reference answers should not change over time, and (3) reference answers must be supported by evidence. The results show that only 17.11% of the dialogues are suitable for building a Dev Knowledge QA benchmark. Most dialogues cannot be used because many answers are incorrect, and even when correct, the questions often admit multiple valid responses, making it difficult to define a single reference answer. The cases are shown in Appendix G. This highlights the need for a dedicated benchmark that can automatically evaluate LLMs on Dev Knowledge QA in real-world development scenarios.

4 SimpleDevQA

Our preliminary study in the Section 3 shows that most real-world Dev Knowledge QA dialogues are unsuitable for benchmark construction. To address this, we design a three-stage benchmark construction framework that can convert real-world SE-related conversations into a Dev Knowledge QA benchmark. An overview of our pipeline is illustrated in Figure 3. First of all, we randomly sample 3,000 real-world conversations from WildChat-Dev to serve as the seed conversation.

4.1 Reference Collection

To ensure that the generated QA pairs are grounded in factual evidence while remaining faithful to the original user intent, we collect high-quality reference documents for each real-world developer dialogue in this stage.

Starting from the seed dataset, each dialogue is analyzed by GPT-4o-mini (OpenAI, 2024b) to identify core software engineering topics and then generate search queries that capture the main intent and technical focus of the dialogue. These queries are submitted to the Google Search API (Piasecki et al., 2018) to retrieve relevant web page URLs. For each dialogue, we collect 10 candidate web pages. Then, we use the Goose3 (Goose3 Contributors, 2024) tool to remove irrelevant content from the collected web pages. Finally, we collect 30,000 reference documents for the QA-pair Generation stage.

4.2 QA-pair Generation

To ensure both the realism and reliability of the constructed QA pairs, we generate Dev Knowledge QA pairs using the original real-world dialogue and its corresponding reference documents.

The generation process begins with the preprocessing of the reference content. For documents that are too long to be directly used as input for the LLM due to context window limits or irrelevant information, we first extract their main content. Then, all reference documents linked to the same dialogue are combined into a single reference text. This unified reference text, along with the original real-world dialogue, is fed into an LLM to produce candidate QA pairs. Details on the generation can be found in the Appendix D. Finally, we generate 9,000 QA pairs from the initial 3,000 Dev Knowledge dialogues.

4.3 QA-pair Filtering

Quality Verification. To improve the quality of SimpleDevQA, we separately verify the quality of both questions and answers. First, to improve the quality of questions, we filter out low-quality questions. Specifically, following previous studies(He et al., 2024), we use three core standards for QA pair quality verification: (1) Each question must have exactly one unambiguous and uncontroversial answer, avoiding open-ended questions; (2) Each question should exhibit sufficient complexity to evaluate LLMs' depth of understanding of software engineering knowledge; (3) Each question must have verifiable answers based on publicly available information by December 31, 2024. In this step, we apply LLM to filter out the samples that fail to meet any of these criteria. Second, to improve the quality of answers, we implement an LLM with Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) system to verify the factual correctness of answers. For our RAG system, we utilize LlamaIndex (Liu, 2022) to build the search tool and select Google Search (Piasecki et al., 2018) as the search engine. This tool is used to search related web pages as references for subsequent answer verification. Then, we use our RAG system to classify the QA pairs with incorrect answers. Finally, we manually correct the answers of these QA pairs. After Quality Verification, approximately 6,020 pairs (67%) were retained, with 33% discarded.

Difficulty Filtering. To increase the difficulty of SimpleDevQA, we employ LLMs to filter out low-difficulty QA pairs. Importantly, the notion of difficulty here is *model-centric*: a question is considered difficult if strong LLMs are likely to answer it incorrectly, rather than the *human-centric* notion that a difficult problem merely requires more time or effort to solve. Specifically, following previous research (He et al., 2024), we use four LLMs with strong general capability: GPT-4o (OpenAI, 2024b), Llama-3-70B-Instruct (Grattafiori et al., 2024), Qwen2.5-72B-Instruct (Qwen) and GLM-4-Plus (BigModel Team, 2024). If a question can be correctly answered by all four strong models, it is considered easy. After this step, we obtain 3,231 difficult QA pairs and 2,789 easy QA pairs.

Human Verification. In this stage, we conduct human verification to further enhance the quality of SimpleDevQA. We follow the criteria in Quality Verification to verify data quality. Each QA pair is evaluated independently by two annotators.

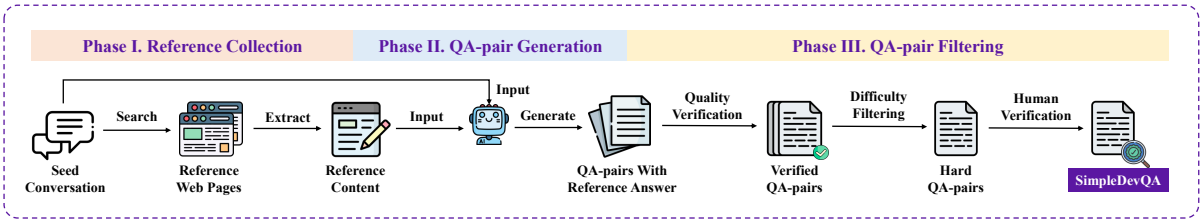

Figure 3: SimpleDevQA Construction Pipeline.

They first check whether it meets the predefined standards: if either annotator rejects it, the pair is removed. Then, annotators search for supporting evidence using search engines and provide answers with citations from authoritative sources (at least two URLs per answer). If the two annotators disagree, a third annotator reviews the case and makes the final decision, considering both previous evaluations. Only QA pairs with full agreement are kept to ensure high precision and consistency with the standards. Finally, we obtain 5,529 high-quality QA pairs.

5 Benchmark Characteristics

SimpleDevQA is a Dev Knowledge QA benchmark designed for the software development domain, containing 5,529 manually verified QA pairs. This includes 2,740 difficult QA pairs and 2,789 easy QA pairs. The feature statistics of our dataset are presented in the Appendix E. Additionally, to analyze the development knowledge evaluated in our benchmark, we manually classify the questions into syntactic and semantic categories following a prior study (see the Appendix E for details).

In summary, SimpleDevQA includes the following key characteristics:

Built from real user queries. The questions in SimpleDevQA are built from dialogues in the real-world development scenarios, which can reflect the real task demands of developers.

Diverse development knowledge. Based on the classification of the QA pairs, we find that SimpleDevQA covers a broader range of development knowledge domains, which more realistically reflects the development knowledge QA ability of LLMs.

Multilingual. SimpleDevQA incorporates three popular languages, including Chinese, English, and Russian. This dataset can be used to evaluate the multilingual Dev Knowledge QA capabilities of different LLMs.

Static. In SimpleDevQA, each QA pair's reference answers are grounded in stable development knowledge, which remains invariant over time or external changes. Each QA pair is equipped with reference web documents, which serve as static sources of knowledge. This design ensures long-term reproducibility for model evaluation,

Easy to evaluate. We retain only those questions with unambiguous answers, which ensures that we can evaluate the correctness of predicted answers simply using the LLM-as-judge method (Zhuo, 2024; Li et al., 2024a; Son et al., 2024; He et al., 2025), according to previous studies (He et al., 2024; Wei et al., 2024).

6 Experimental Setup

6.1 Experimental Details

Model Selection. In the evaluation, we choose 21 widely used LLMs. For the closed-source models, we include Claude-3.5-Haiku (Anthropic, 2024), o3-mini (OpenAI, 2024c), GPT-3.5-Turbo(Brown et al., 2020), DeepSeek-V3.2 (DeepSeek-AI et al., 2025), DeepSeek-V3 (Liu et al., 2024a), GPT-4o (Piasecki et al., 2018), and DeepSeek-R1. For the open-source models, we include Qwen3-30B-A3B-Instruct (Yang et al., 2025a), the Qwen2.5 series (7B, 14B, 32B) (Qwen), the InternLM2.5 series (7B, 20B) (Team, 2024), the Llama3 series (8B, 70B) (Grattafiori et al., 2024), the DeepSeek-Coder series (6.7B, V2-lite) (Guo et al., 2024), Qwen3-Coder-30B-A3B-Instruct (Yang et al., 2025a), the Qwen2.5-Coder series (7B, 32B) (Hui et al., 2024), and CodeLlama-7B-instruct (Roziere et al., 2023).

Experiment Settings. During LLM inference, we set the temperature to 0.7 and the top-p to 0.95. During evaluation, we set the temperature of the judge model to 0.5 and top-p to 1. This setting mitigates the risk of extreme bias from greedy decoding while avoiding excessive noise, thereby producing judgments that are both consistent and reasonably diverse (Li et al., 2024a; Gu et al., 2024). Nevertheless, it may introduce a certain degree of randomness and potential bias. To mitigate randomness,

we conduct each experiment three times and average the outcomes.

6.2 Evaluation Metrics

Grading Method: Following previous studies (Wei et al., 2024; He et al., 2024), we evaluate the correctness of the model's predicted answers by prompting a separate judge model with both the prediction and the reference answer to assign one of three labels: Correct, Not Attempted, or Incorrect. We here use GPT-4o-mini (OpenAI, 2024a) as the judge model.

Metrics: Following previous studies (Wei et al., 2024; He et al., 2024), we use these metrics to evaluate the performance of LLMs on SimpleDevQA:

- **Correct (CO):** The predicted answer includes the reference answer without contradiction.
- **Not Attempted (NA):** The model effectively does not attempt a full answer: either it refuses/admits its inability, or it gives only a partial response that fails to fully match the reference answer, and there are no contradictions with the reference answer.
- **Incorrect (IN):** The predicted answer contradicts the reference answer at any point, even if the contradiction is later corrected.
- **Correct Given Attempted (CGA):** The ratio of correct answers among all attempted answers.
- **F-score:** The harmonic mean of the overall percentage of correctly answered questions and Correct Given Attempted.
- **Average Tokens:** The average sum of input and output tokens per question that can evaluate LLM's potential invocation costs.

7 Experimental Results

7.1 Performance Comparison Analysis

We evaluate 21 mainstream LLMs on the difficult QA pairs in SimpleDevQA. The results are presented in the Table 1. First, we can find that LLMs' performance on SimpleDevQA varies considerably, and o3-mini and DeepSeek-R1 perform best, achieving F-scores of 0.719 and 0.7, respectively. Second, at similar parameter scales, specialized code LLMs significantly outperformed general LLMs. For instance, while Qwen2.5-32B-Instruct scored only 0.551 on the F-score, Qwen2.5-Coder-32B-Instruct achieved 0.582. Third, we find that closed-source models generally outperformed open-source models, with closed-source models consistently achieving higher F-scores. For example, the open-source Llama-3.1-70B has an F-score of 0.544, while the closed-source Claude-3.5-Haiku achieved 0.623.

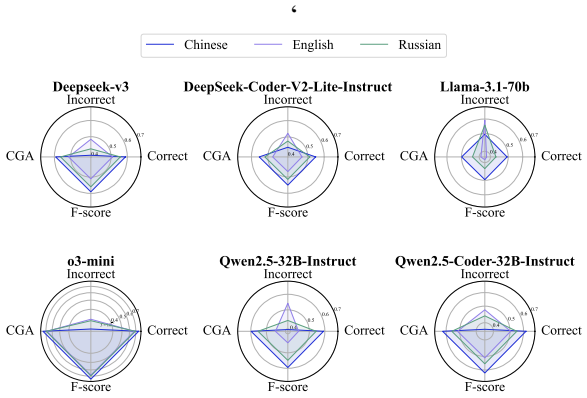

Figure 4: Results of different models between different language subsets.

Furthermore, we find that models' performance improves as their scale increases across many model series, such as the InternLM2.5 and Llama3 series. Finally, we find that Claude-3.5-Haiku achieves the highest score on the Not Attempt metric among all evaluated LLMs. For questions about which it is uncertain, it tends to abstain rather than provide an incorrect answer.

We also analyze the performance of six LLMs on the Chinese, English, and Russian subsets of SimpleDevQA, with the results detailed in Figure 4. From these experimental outcomes, we identify several key findings: First, o3-mini demonstrates the strongest performance, achieving the leading score across all three languages. Furthermore, most Chinese-developed models, such as the Qwen and DeepSeek series, perform best in Chinese. The result indicates that the model performs differently across languages.

> **Summary:** First, LLMs' performance in Dev Knowledge QA varies considerably. Second, there is room for growth in the open-source models' understanding of software development knowledge. Third, code-specific LLMs generally outperform general LLMs of similar scale.

7.2 Impacts of the Knowledge Injection Strategy

Previous studies (Xia et al., 2024; Li and Flanigan, 2024; Li et al., 2024b) have demonstrated that the Retrieval Augmentation Generation (Lewis et al., 2020) (RAG) strategy is one common method to inject factual knowledge into LLMs to improve

Model	Type	Correct	Incorrect	NA	CGA	F-score	Avg. Tokens
Closed-Source Large Language Models							
Claude-3.5-Haiku	General LLM	0.536	0.324	**0.14**	0.623	0.576	321.62
o3-mini	General LLM	**0.718**	0.278	0.004	0.721	**0.719**	229.86
GPT-3.5-Turbo	General LLM	0.677	0.307	0.016	0.688	0.682	117.05
DeepSeek-V3.2	General LLM	0.6	0.387	0.013	0.608	0.604	-
DeepSeek-V3	General LLM	0.56	0.432	0.007	0.564	0.562	527.57
GPT-4o	General LLM	0.619	0.373	0.008	0.624	0.622	254.81
DeepSeek-R1	General LLM	0.679	**0.262**	0.059	**0.722**	0.7	918.02
Open-Source Large Language Models							
Qwen3-30B-A3B-Instruct	General LLM	0.638	0.355	0.007	0.642	0.64	-
Qwen2.5-32B-Instruct	General LLM	0.543	0.442	0.015	0.551	0.547	309.92
Qwen2.5-14B-Instruct	General LLM	0.498	0.489	0.013	0.505	0.501	317.04
Qwen2.5-7B-Instruct	General LLM	0.445	0.534	0.021	0.454	0.45	360.53
InternLM2.5-7B-chat	General LLM	0.411	0.572	0.017	0.418	0.414	521.66
InternLM2.5-20B-chat	General LLM	0.466	0.517	0.017	0.474	0.47	458.14
Llama-3.1-8B	General LLM	0.449	0.535	0.016	0.456	0.453	304.89
Llama-3.1-70B	General LLM	0.538	0.451	0.011	0.544	0.541	304.82
CodeLlama-7B-Instruct	Code LLM	0.389	0.594	0.017	0.396	0.393	403.62
DeepSeek-Coder-V2-Lite-Instruct	Code LLM	0.518	0.472	0.01	0.523	0.521	403.87
DeepSeek-Coder-6.7B-Instruct	Code LLM	0.511	0.477	0.012	0.517	0.514	269.88
Qwen3-Coder-30B-A3B-Instruct	Code LLM	0.582	0.403	0.015	0.591	0.586	-
Qwen2.5-Coder-7B-Instruct	Code LLM	0.573	0.412	0.015	0.582	0.578	**116.9**
Qwen2.5-Coder-32B-Instruct	Code LLM	0.574	0.412	0.014	0.582	0.578	278.35

Table 1: Results of different models on difficult QA pairs from SimpleDevQA. NA is short for Not Attempted; CGA is short for Correct Given Attempted.

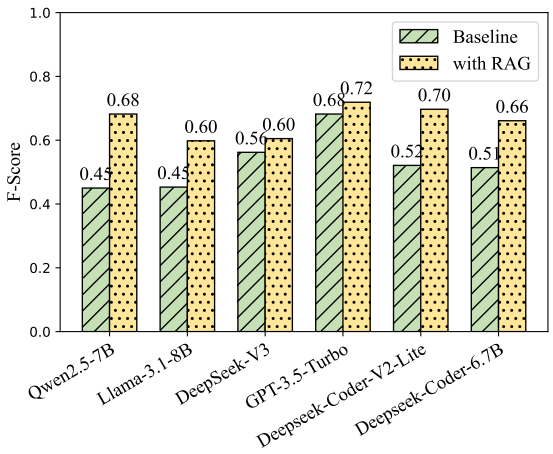

Figure 5: The effect of the knowledge injection strategy.

LLMs' factual accuracy on general QA tasks. In this section, we investigate its impact on LLMs' performances in the Dev Knowledge QA task. Following previous studies (He et al., 2024), we implement our RAG pipeline using LlamaIndex (Liu, 2022) and integrate the Google Search API (Google Developers, 2024) to retrieve relevant software-engineering knowledge from the web.

The experimental results are presented in the Figure 5. First, we can find that after implementing the RAG strategy, all evaluated LLMs show significant performance improvements on SimpleDevQA, with an average improvement of 11.3%. Second, the RAG approach effectively reduces the performance gap between different LLMs. For instance, without RAG, the F-score gap between Qwen2.5-7B-Instruct and GPT-3.5-Turbo is as high as 23.2%, whereas after RAG integration, this gap decreases substantially to just 3.7%. The results show that smaller LLMs can achieve comparable performance to larger LLMs after the RAG strategy.

Summary: The knowledge injection strategy can improve LLMs' accuracy in Dev Knowledge QA, and can enable smaller LLMs to achieve performance comparable to that of larger LLMs.

7.3 Stated Confidence and Accuracy Correlation Analysis

According to previous studies (Wei et al., 2024; He et al., 2024), the factual QA benchmark like

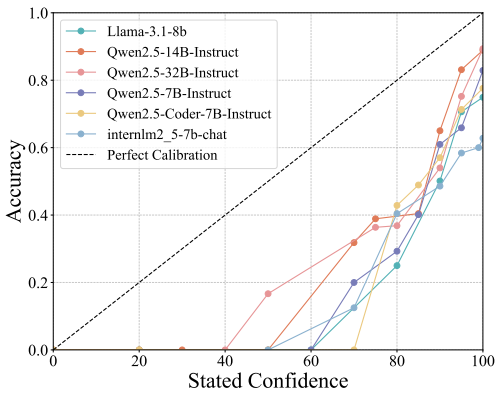

Figure 6: Calibration of model outputs using self-reported stated confidence scores.

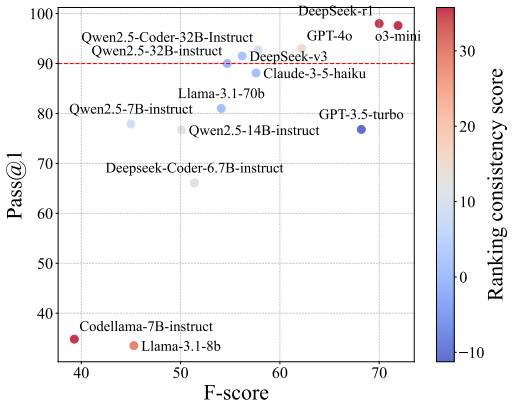

Figure 7: Results of different LLMs on SimpleDevQA and HumanEval.

SimpleQA (Wei et al., 2024) can not only evaluate the factual accuracy of LLM but also serve as a reliable calibration test, assessing the alignment between model confidence and factual accuracy. A well-calibrated LLM enables developers to judge the trustworthiness of their answers based on their stated confidence. Following the previous study, we conduct a calibration analysis by directly instructing the LLM to state its confidence after its answer when responding to a question (the prompt is seen in Appendix F).

As shown in Figure 6, the results indicate that all LLMs' QA accuracy increases as their confidence increases. This suggests that users can select answers for which LLMs express higher confidence, potentially leading to more accurate outcomes. However, all evaluated LLMs exhibit a degree of overconfidence. Specifically, their performance falls well below the Y=X ideal calibration line, indicating that these LLMs tend to overestimate the accuracy of their answers. These findings indicate that LLM calibration still requires substantial improvement.

Summary: The accuracy of LLMs shows a positive correlation with their stated confidence in Dev Knowledge QA. However, they tend to overestimate the accuracy of their answers.

7.4 Capability Correlation Study

Based on our preliminary study in Section 3.1, code generation and Dev Knowledge QA are the most common tasks when developers interact with LLMs. To investigate how LLM performance relates across these two tasks, we compare evaluated LLMs on two benchmarks: HumanEval (Chen et al., 2021) for code generation and SimpleDevQA for Dev Knowledge QA. We gather each model's HumanEval pass@1* score alongside its SimpleDevQA F-score and plot these paired metrics in a scatter plot.

Experimental results in Figure 7 demonstrate that LLMs that achieve higher pass@1 scores tend to also score higher on SimpleDevQA, indicating a strong relationship between code generation capability and development knowledge comprehension. Among them, o3-mini and DeepSeek-R1 exhibit outstanding performance on both datasets, demonstrating superior comprehensive capabilities. However, special cases exist. For example, Qwen2.5-7B-Instruct achieves nearly 80% pass@1 on code generation benchmarks, yet its F-score is only about 45%. This gap suggests that strong code generation ability does not guarantee equally strong performance on Dev Knowledge QA tasks.

Summary: Generally, LLMs with stronger code generation performance also exhibit stronger performance in Dev Knowledge QA, where o3-mini and DeepSeek-R1 show the best comprehensive performance.

8 Conclusions

In this paper, we propose SimpleDevQA, a multilingual Dev Knowledge QA benchmark built from real developer dialogues that covers broader development knowledge. Based on this dataset, we conduct extensive experiments. Experimental results show that first, closed-source models typically surpass open-source ones, and code LLMs generally outperform general LLMs of similar scale. Sec-

*We collect pass@1 from https://evalplus.github.io/leaderboard.html.

ond, the Retrieval-Augmented Generation (RAG) strategy can enable smaller models to achieve performance comparable to larger models. Third, we find that LLMs' accuracy generally increases with their stated confidence in SimpleDevQA. However, they tend to overestimate the accuracy of their answers. Finally, we find that LLMs with stronger code generation performance also exhibit stronger performance in the Dev Knowledge QA task.

By releasing SimpleDevQA to the community, we aim to help facilitate further research in software development. We hope that our benchmark will serve as a reliable tool for evaluating LLMs' understanding of development knowledge.

Limitations

We have identified the following limitations to our study.

Limited benchmark size and data imbalance. The current version contains only 5,529 instances across 10 domains and 3 languages, constrained by the limited size and coverage of the seed dataset during the data generation stage. Specifically, we sampled 3,000 real Dev Knowledge dialogues as the seed dataset, given the high cost of LLM inference and the labor-intensive manual verification required in the filtering stage. In addition, the dataset exhibits domain imbalance. Nearly half of the questions are in 53.7% of the topics are focused on "APIs & Frameworks". Importantly, this imbalance is a direct consequence of our construction method: SimpleDevQA is inspired by real user-LLM dialogues, and thus the resulting distribution naturally reflects developers' knowledge-seeking tendencies and demand patterns in practice. In this sense, the skew is not merely a limitation, but also evidence of the benchmark's realism. Nevertheless, the collection pipeline we provide is general for converting real-world dialogues into high-quality QA pairs. With this pipeline, additional data covering more languages and domains can be obtained. In future work, we plan to expand the seed dialogue pool to improve overall coverage and to examine whether scaling up real dialogue collection can naturally increase the volume of currently underrepresented categories, while still preserving the benchmark's real-world distributional characteristics.

Potential bias from model generation and judgment. The limitation is that the use of LLMs in both benchmark construction and evaluation may introduce bias. In benchmark construction, the QA pairs are generated by an LLM using the original dialogues and references as input. Although we apply a multi-stage filtering process to ensure correctness, this process may still introduce subtle biases, such as phrasing that reflects the LLM's style rather than organic developer language. In evaluation, we set the temperature of the judge model = 0.5 and top-p = 1 to balance determinism and diversity. While this reduces the risk of extreme bias from greedy decoding, inherent randomness remains even after multiple runs, which may still affect the consistency, reproducibility, and introduce potential bias (Ye et al., 2024; Chen et al., 2024; Zheng et al., 2023). And we conducted a human evaluation and measured agreement with the LLM judge. The Cohen's Kappa score (Mchugh, 2012) between the LLM and human evaluation reaches 0.83 with an overall accuracy of 0.91. These results indicate a high level of agreement (Kappa > 0.8), supporting the high reliability of the LLM judge.

Limited scope of evaluated baselines. While our current study provides comprehensive evaluations using zero-shot and RAG frameworks, the scope of our baseline comparisons remains somewhat restricted. Specifically, a limitation is the absence of baseline, such as deep research or agentic search. In the future, we will include deep research and agentic search as baselines to further understand the potential of our proposed benchmark.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No.92582202, No.92582110, No.62302534), and GMCC-SYSU Joint Lab for Smart Applications.

References

2023. Confidence interval. Wikipedia, The Free Encyclopedia. [Online; accessed May 31, 2025].

2023. Sample size calculator. https://www.surveysystem.com/sscalc.htm. Accessed: 2025-05-31.

Anthropic. 2024. Claude Haiku. Accessed: 2025-05-28.

BigModel Team. 2024. GLM-4 Usage Guide. Accessed: 2025-05-28.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are

few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Vittorio Castelli, Rishav Chakravarti, Saswati Dana, Anthony Ferritto, Radu Florian, Martin Franz, Dinesh Garg, Dinesh Khandelwal, Scott McCarley, Mike McCawley, Mohamed Nasr, Lin Pan, Cezar Pendus, John Pitrelli, Saurabh Pujar, Salim Roukos, Andrzej Sakrajda, Avirup Sil, Rosario Uceda-Sosa, and 2 others. 2019. The techqa dataset. *Preprint*, arXiv:1911.02984.

Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. 2024. Humans or llms as the judge? a study on judgement biases. *arXiv preprint arXiv:2402.10669*.

Jialiang Chen, Kaifa Zhao, Jie Liu, Chao Peng, Jierui Liu, Hang Zhu, Pengfei Gao, Ping Yang, and Shuiguang Deng. 2025. Coreqa: Uncovering potentials of language models in code repository question answering. *Preprint*, arXiv:2501.03447.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

I Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, Pengfei Liu, and 1 others. 2023. Factool: Factuality detection in generative ai–a tool augmented framework for multi-task and multi-domain scenarios. *arXiv preprint arXiv:2307.13528*.

José Wellington Franco da Silva, Amanda Drielly Pires Venceslau, Juliano Efson Sales, José Gilvan Rodrigues Maia, Vládia Célia Monteiro Pinheiro, and Vânia Maria Ponte Vidal. 2020. A short survey on end-to-end simple question answering systems. *Artificial Intelligence Review*, 53(7):5429–5453.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Bhuwan Dhingra, Kathryn Mazaitis, and William W Cohen. 2017. Quasar: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*.

Google Developers. 2024. Custom Search API Documentation. Accessed: 2025-05-28.

Goose3 Contributors. 2024. Goose3: A Python Library for Web Content Extraction. Accessed: 2025-05-28.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, and 1 others. 2024. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

Melody Y Guan, Manas Joglekar, Eric Wallace, Saachi Jain, Boaz Barak, Alec Helyar, Rachel Dias, Andrea Vallone, Hongyu Ren, Jason Wei, and 1 others. 2024. Deliberative alignment: Reasoning enables safer language models. *arXiv preprint arXiv:2412.16339*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Junda He, Jieke Shi, Terry Yue Zhuo, Christoph Treude, Jiamou Sun, Zhenchang Xing, Xiaoning Du, and David Lo. 2025. From code to courtroom: Llms as the new software judges. *arXiv preprint arXiv:2503.02246*.

Yancheng He, Shilong Li, Jiaheng Liu, Yingshui Tan, Weixun Wang, Hui Huang, Xingyuan Bu, Hangyu Guo, Chengwei Hu, Boren Zheng, and 1 others. 2024. Chinese simpleqa: A chinese factuality evaluation for large language models. *arXiv preprint arXiv:2411.07140*.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79.

Ruida Hu, Chao Peng, Jingyi Ren, Bo Jiang, Xiangxin Meng, Qinyun Wu, Pengfei Gao, Xinchen Wang, and Cuiyun Gao. 2024. Coderepoqa: A large-scale benchmark for software engineering question answering. *arXiv preprint arXiv:2412.14764*.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, and 1 others. 2023. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems*, 36:62991–63010.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Klaus Krippendorff. 2011. Computing krippendorff's alpha-reliability.

Changyoon Lee, Yeon Seonwoo, and Alice Oh. 2022. Cs1qa: A dataset for assisting code-based question answering in an introductory programming course. *arXiv preprint arXiv:2210.14494*.

Krystal M Lewis and Peter Hepburn. 2010. Open card sorting and factor analysis: a usability case study. *The electronic library*, 28(3):401–416.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Changmao Li and Jeffrey Flanigan. 2024. Rac: Efficient llm factuality correction with retrieval augmentation. *arXiv preprint arXiv:2410.15667*.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. 2024a. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*.

Jiarui Li, Ye Yuan, and Zehua Zhang. 2024b. Enhancing llm factual accuracy with rag to counter hallucinations: A case study on domain-specific queries in private knowledge-bases. *arXiv preprint arXiv:2403.10446*.

Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023. Halueval: A large-scale hallucination evaluation benchmark for large language models. *arXiv preprint arXiv:2305.11747*.

Linyi Li, Shijie Geng, Zhenwen Li, Yibo He, Hao Yu, Ziyue Hua, Guanghan Ning, Siwei Wang, Tao Xie, and Hongxia Yang. 2024c. Infibench: Evaluating the question-answering capabilities of code large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 128668–128698. Curran Associates, Inc.

Linyi Li, Shijie Geng, Zhenwen Li, Yibo He, Hao Yu, Ziyue Hua, Guanghan Ning, Siwei Wang, Tao Xie, and Hongxia Yang. 2024d. Infibench: Evaluating the question-answering capabilities of code large language models. *Advances in Neural Information Processing Systems*, 37:128668–128698.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Chenxiao Liu and Xiaojun Wan. 2021. Codeqa: A question answering dataset for source code comprehension. *arXiv preprint arXiv:2109.08365*.

Jerry Liu. 2022. LlamaIndex.

Jiawei Liu, Jia Le Tian, Vijay Daita, Yuxiang Wei, Yifeng Ding, Yuhan Katherine Wang, Jun Yang, and Lingming Zhang. 2024b. Repoqa: Evaluating long context code understanding. *arXiv preprint arXiv:2406.06025*.

Dung Nguyen Manh, Thang Phan Chau, Nam Le Hai, Thong T Doan, Nam V Nguyen, Quang Pham, and Nghi DQ Bui. 2024. Codemmlu: A multi-task benchmark for assessing code understanding capabilities of codellms. *arXiv preprint arXiv:2410.01999*.

Mary L Mchugh. 2012. Interrater reliability: the kappa statistic. *Biochemia Medica*, 22(3):276–282.

Meta AI. 2024. Meta llama-3-8b-instruct. Accessed: 2025-05-31.

Changan Niu, Chuanyi Li, Bin Luo, and Vincent Ng. 2022. Deep learning meets software engineering: A survey on pre-trained models of source code. *arXiv preprint arXiv:2205.11739*.

OpenAI. 2024a. GPT-4o-mini. Accessed: 2025-05-29.

OpenAI. 2024b. Hello GPT-4o. Accessed: 2025-05-28.

OpenAI. 2024c. OpenAI o3 Mini. Accessed: 2025-05-28.

Jan Piasecki, Marcin Waligora, and Vilius Dranseika. 2018. Google search as an additional source in systematic reviews. *Science and engineering ethics*, 24:809–810.

Team Qwen. Qwen2. 5: A party of foundation models, september 2024. *URL https://qwenlm. github. io/blog/qwen2*, 5.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Ensheng Shi, Yanlin Wang, Fengji Zhang, Bei Chen, Hongyu Zhang, Yanli Wang, Daya Guo, Lun Du, Shi Han, Dongmei Zhang, and Hongbin Sun. 2025. Sotana: An open-source software engineering instruction-tuned model. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, pages 1–12.

Guijin Son, Hyunwoo Ko, Hoyoung Lee, Yewon Kim, and Seunghyeok Hong. 2024. Llm-as-a-judge & reward model: What they can and cannot do. *arXiv preprint arXiv:2409.11239*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, and 1 others. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

I Team. 2024. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*.

valpy. 2024. Prompt classification model. Accessed: 2025-05-28.

Zhiruo Wang, Shuyan Zhou, Daniel Fried, and Graham Neubig. 2022. Execution-based evaluation for open-domain code generation. *arXiv preprint arXiv:2212.10481*.

Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. 2024. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*.

Philip Welsby and Bernard MY Cheung. 2023. Chatgpt.

Di Wu, Xiao-Yuan Jing, Hongyu Zhang, Yang Feng, Haowen Chen, Yuming Zhou, and Baowen Xu. 2023. Retrieving api knowledge from tutorials and stack overflow based on natural language queries. *ACM Transactions on Software Engineering and Methodology*, 32(5):1–36.

Peng Xia, Kangyu Zhu, Haoran Li, Hongtu Zhu, Yun Li, Gang Li, Linjun Zhang, and Huaxiu Yao. 2024. Rule: Reliable multimodal rag for factuality in medical vision language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1081–1093.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Shuo Yang, Jiachi Chen, and Zibin Zheng. 2023. Definition and detection of defects in nft smart contracts. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 373–384.

Shuo Yang, Xingwei Lin, Jiachi Chen, Qingyuan Zhong, Lei Xiao, Renke Huang, Yanlin Wang, and Zibin Zheng. 2025b. Hyperion: Unveiling dapp inconsistencies using llm and dataflow-guided symbolic execution. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 2125–2137.

Ziyu Yao, Daniel S Weld, Wei-Peng Chen, and Huan Sun. 2018. Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference*, pages 1693–1703.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, and 1 others. 2024. Justice or prejudice? quantifying biases in llm-as-a-judge. *arXiv preprint arXiv:2410.02736*.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, and 1 others. 2025a. Siren's song in the ai ocean: A survey on hallucination in large language models. *Computational Linguistics*, pages 1–46.

Ziyao Zhang, Chong Wang, Yanlin Wang, Ensheng Shi, Yuchi Ma, Wanjun Zhong, Jiachi Chen, Mingzhi Mao, and Zibin Zheng. 2025b. Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):481–503.

Ziyin Zhang, Chaoyu Chen, Bingchang Liu, Cong Liao, Zi Gong, Hang Yu, Jianguo Li, and Rui Wang. 2023. Unifying the perspectives of nlp and software engineering: A survey on language models for code. *arXiv preprint arXiv:2311.07989*.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. Wildchat: 1m chatgpt interaction logs in the wild. *arXiv preprint arXiv:2405.01470*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Zibin Zheng, Kaiwen Ning, Qingyuan Zhong, Jiachi Chen, Wenqing Chen, Lianghong Guo, Weicheng Wang, and Yanlin Wang. 2025. Towards an understanding of large language models in software engineering tasks. *Empirical Software Engineering*, 30(2):50.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2024. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2299–2314, Mexico City, Mexico. Association for Computational Linguistics.

Terry Yue Zhuo. 2024. Ice-score: Instructing large language models to evaluate code. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2232–2242.

A Comparison of existing Dev Knowledge QA benchmarks

The comparison between SimpleDevQA and other Dev Knowledge QA benchmarks is shown in Table 2. We find that SimpleDevQA is not only

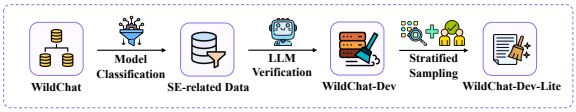

Figure 8: Data processing workflow for extracting software development–related conversations.

grounded in real user queries but also covers diverse knowledge required throughout the software development process.

B Extraction of the software development-related dialogues

Following WildChat (Zhao et al., 2024), we use a prompt-classification model (valpy, 2024) to filter the dataset and obtain 376,888 dialogues related to software engineering. To further enhance data quality, we employ Llama3-Instruct-8B (Meta AI, 2024) to process the dialogues, filtering out conversations irrelevant to software development. This step results in the WildChat-Dev dataset, which includes 103,112 dialogues focused on development. To facilitate our manual annotation and analysis of real dialogues, we apply stratified sampling to 103,112 dialogues based on user language, dialogue length, and the interacted model within the data, which results in the WildChat-Dev-Lite dataset. To compute the sample size, we follow previous studies (Yang et al., 2023, 2025b), use the random sampling method based on the confidence interval (wik, 2023). We set a confidence interval of 1 and a confidence level of 95%, and compute that the sample size is 8,786 (sur, 2023). At the end, we obtain the WildChat-Dev-Lite dataset, which includes 8,786 real dialogues. The whole data processing workflow is shown in Figure 8.

C Classification of the real dialogues

To analyze the distribution of software engineering–related tasks in real-world dialogues, we apply the following pipeline to categorize real-world dialogues. First, we manually filter dialogues from WildChat-Dev-Lite that are unrelated to software development. Then, we use the open card sorting method (Lewis and Hepburn, 2010) to summarize the tasks involved in the dialogues from WildChat-Dev-Lite, and the final eight categories are inspired by prior works (Zhang et al., 2023; Hou et al., 2024; Niu et al., 2022), as shown in Table 3. Finally, we manually classify the remaining dialogues into 8 task categories. Specifically, we first conduct a pilot study in which three annotators jointly label a small sample of dialogues and discuss the criteria. During the annotation process, two annotators independently label each dialogue. They then cross-check the results and resolve discrepancies through discussion. If no agreement can be reached, a third annotator is involved, and final decisions are made via majority voting. To assess annotation reliability, we calculate Krippendorff's alpha (Krippendorff, 2011), which is 0.908, indicating strong agreement. The final average annotation accuracy is 0.924.

Prompt Template for QA-pair Generation

###Task Description###

You are an expert software programmer. Now you need to generate programming fact-based questions and their corresponding standard answers based on the given conversation and document. The QA pairs must meet the following requirements:
1. **Relevance to Software Engineering**: The questions must relate to concrete programming knowledge or concepts in software engineering, such as algorithms, data structures, design patterns, libraries, APIs, or best practices. Avoid subjective or opinion-based questions.
2. **Language Consistency**: The language used in the generated question-answer pairs must be consistent with the language used in the conversation.
3. **Clarity and Precision**: Each question should have one and only one clear and undisputed answer. Avoid ambiguous or open-ended questions.
4. **Timeless Answers**: The answers must be timeless and not subject to change. Avoid questions about roles, events that may change over time.
5. **Conciseness**: Answers should be as concise as possible while remaining accurate.
6. **Educational Value**: Questions should have a certain level of difficulty to pose a challenge and be educational for learning software engineering concepts. The generated questions cannot be easily answered correctly

###Output Format###

[Format]

###Examples of High-Quality QA Pairs###

[Example]

###Input###

Here is the given conversation: {conversation}, document: {document}

Figure 9: The prompt template for generating QA pairs.

D Generation of QA pairs

We use GPT-4o (OpenAI, 2024b) to generate Dev Knowledge QA pairs. The prompts for generating QA pairs are presented in Figure 9. Specifically, we request that LLM generate 3 QA pairs for one dialogue at the same time. Besides, we provide 10 demonstrations to enhance the quality of generated Dev Knowledge QA pairs from LLMs.

E Characteristics of SimpleDevQA

SimpleDevQA covers the three most prevalent languages from the WildChat-Dev-Lite dataset: English, Chinese, and Russian. Specifically, it includes 2,032 English QA pairs, 2,071 Chinese QA pairs, and 1,426 Russian QA pairs.

Additionally, questions in our benchmark have a length of 27.74 tokens [†] , while answers have

[†] https://github.com/openai/tiktoken.

Benchmark	Year	Size	Data Source	Built from Real User Queries	Diverse Dev Knowledge
CodeQA (Liu and Wan, 2021)	2021	190,000	Code Snippets	✗	✗
CS1QA (Lee et al., 2022)	2022	9,237	Textbooks, Edu Materials	✓	✗
RepoQA (Liu et al., 2024b)	2024	500	Real Code Repositories	✗	✗
InfiBench (Li et al., 2024d)	2024	234	Real questions	✓	✗
CoReQA (Chen et al., 2025)	2025	1,563	Real issues and comments	✓	✗
SimpleDevQA (ours)	2025	5,529	Real User Dialogues, Web	✓	✓

Table 2: Comparison of existing Dev Knowledge QA benchmarks.

Task Category	Task Description
Dev Knowledge QA	QA that questions knowledge during development
Code Generation	Generating code examples based on requirements
Code Debugging	Identifying errors in existing code
Program Repair	Fixing defective or buggy code
Code Translation	Converting code from source to target language
Code Editing	Modifying or enhancing functionality of existing code
Comment Generation	Generating explanatory comments for code
Test Generation	Creating test cases for code

Table 3: Description of software engineering-related tasks in real scenarios

Prompt Template for Confidence Output

###Task Description###

Please provide your best answer to the given question and indicate your confidence in the answer using a score from 0 to 100. Please respond in the following JSON format:

```
{
   "answer": "your answer",
   "confidence score": your confidence score
}
```

###Input###

Given a question: {question}

Figure 10: The prompt for guiding LLM to output confidence.

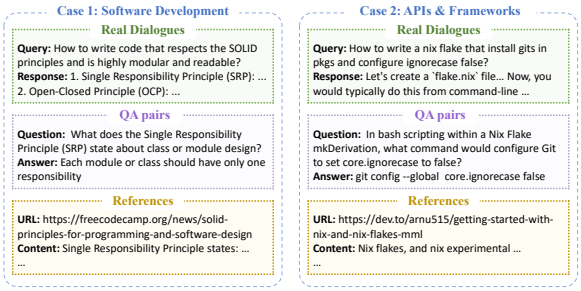

Figure 11: Case studies in SimpleDevQA.

a length of 7.99 tokens on average, as shown in Table 4. In addition to these, each QA pair is also accompanied by multiple web-retrieved references, which include corresponding webpage URLs and text snippets that can be used to verify the factual accuracy of the answers.

To categorize the knowledge domains covered by the QA pairs in the dataset, we manually classify the questions using a taxonomy adapted from a previous study (Manh et al., 2024). For ambiguous or cross-category queries, annotators were instructed to prioritize the category that is most central to answering the question, while recording secondary categories when applicable for internal auditing. The classification includes:

- **Syntactic questions:** These focus on programming language grammar and API usage, such as language-specific functions or common library usage, etc.
- **Semantic questions:** These target more abstract programming concepts, such as algorithms, data structures, and object-oriented principles, etc.

The classification results, shown in Table 4, indicate that the benchmark contains 4,135 syntactic questions and over 1,394 semantic questions, covering 9 distinct development knowledge domains.

F Analysis of stated confidence

We instruct the LLM to state its confidence after its answer when responding to a question for the specific prompt, as shown in Figure 10.

G Case Studies

In our dataset, we categorize and analyze cases in SimpleDevQA. We identify the following aspects that highlight the advantages of SimpleDevQA.

Simple and Accurate Evaluation. A key challenge in using Dev Knowledge QA derived from real-world dialogues is that the resulting QA pairs are often *hard to evaluate reliably*. As shown in Figure 11, real user queries are frequently open-ended (e.g., "How to write code that respects SOLID..."), and the corresponding responses may mix subjective guidance with multiple factual statements and multiple valid solution paths. Consequently, even when a model produces a semantically correct response, it may differ substantially in

(a) Knowledge Domain.

Category	Domain	Size
Syntactic knowledge	APIs & Frameworks	2967
	Programming language syntax	1168
Semantic knowledge	Software development & engineering	496
	Algorithms & Data structures	359
	Database management & SQL	224
	Computer organization & architecture	147
	Object-oriented programming	83
	System design	78
	Compiler design	5
	Computer networks	3

(b) Language Usage.

Language	Ratio
English	36.75%
Chinese	37.46%
Russian	25.79%

(c) Average Token Length.

QA	Length
Question	27.74
Answer	7.99

Table 4: SimpleDevQA Statistics.

wording, structure, or chosen solution route, making automated evaluation unreliable. SimpleDevQA mitigates this issue by focusing on knowledge-seeking questions with short, unambiguous, and verifiable answers through a multi-step curation process. Moreover, each QA pair is accompanied by a supporting reference, which provides explicit evidence for adjudication and further improves the accuracy and consistency of LLM-as-judge evaluation.

Broad Development Knowledge Coverage. SimpleDevQA is designed to reflect development knowledge needs beyond code-centric understanding. For instance, Figure 11 includes questions about software engineering principles (e.g., SRP under the SOLID principles) and practical workflow knowledge, such as configuring Git via terminal commands in real development environments. These examples cover *Software Development & Engineering* (software principles, version control practices) and *APIs & Frameworks*, demonstrating that the benchmark evaluates the broader knowledge developers routinely rely on during end-to-end software development.