Guidelines for Empirical Studies in Software Engineering involving Large Language Models

Sebastian Baltes
University of Bayreuth, Germany
sebastian.baltes@uni-bayreuth.de

Florian Angermeir
fortiss, Germany
BTH, Sweden
angermeir@fortiss.org

Chetan Arora
Monash University, Australia
chetan.arora@monash.edu

Marvin Muñoz Barón
Chunyang Chen
TU Munich, Germany
{marvin.munoz-baron,chun-
yang.chen}@tum.de

Lukas Böhme
Hasso-Plattner-Institut, Germany
University of Potsdam, Germany
lukas.boehme@hpi.de

Fabio Calefato
University of Bari, Italy
fabio.calefato@uniba.it

Neil Ernst
University of Victoria, Canada
nernst@uvic.ca

Davide Falessi
University of Rome Tor Vergata, Italy
falessi@ing.uniroma2.it

Brian Fitzgerald
Lero, Ireland
University of Limerick, Ireland
brian.fitzgerald@ul.ie

Davide Fucci
BTH, Sweden
davide.fucci@bth.se

Marcos Kalinowski
PUC Rio de Janeiro, Brazil
kalinowski@inf.puc-rio.br

Stefano Lambiase
Daniel Russo
Aalborg University, Denmark
{stla,daniel.russo}@cs.aau.dk

Mircea Lungu
IT University Copenhagen, Denmark
mlun@itu.dk

Lutz Prechelt
Freie Universität Berlin, Germany
prechelt@inf.fu-berlin.de

Paul Ralph
Dalhousie University, Canada
paulralph@dal.ca

Rijnard van Tonder
Independent, Antigua and Barbuda
rvantonder@gmail.com

Christoph Treude
SMU, Singapore
ctreude@smu.edu.sg

Stefan Wagner
TU Munich, Germany
stefan.wagner@tum.de

Abstract

Large language models (LLMs) are increasingly being integrated into software engineering (SE) research and practice, yet their non-determinism, opaque training data, and evolving architectures complicate the reproduction and replication of empirical studies. We present a community effort to scope this space, introducing a taxonomy of LLM-based study types together with eight guidelines for designing and reporting empirical studies involving LLMs. The guidelines present essential (MUST) criteria as well as desired (SHOULD) criteria and target transparency throughout the research process. Our recommendations, contextualized by our study types, are: (1) to declare LLM usage and role; (2) to report model versions, configurations, and fine-tuning; (3) to document tool architectures; (4) to disclose prompts and interaction logs; (5) to use human validation; (6) to employ an open LLM as a baseline; (7) to use suitable baselines, benchmarks, and metrics; and (8) to openly articulate limitations and mitigations. Our goal is to enable reproducibility and replicability despite LLM-specific barriers to open science. We maintain the study types and guidelines online as a living resource for the community to use and shape (llm-guidelines.org).

1 Introduction

In the short period since the release of ChatGPT in November 2022, large language models (LLMs) have changed the software engineering (SE) research landscape. Although there are numerous opportunities to use LLMs to support SE research and development tasks, solid science needs rigorous empirical evaluations to explore the effectiveness, performance, and robustness of using LLMs to automate different research and development tasks. LLMs can, for example, be used to support literature reviews, reduce development time, and generate software documentation. However, it is often unclear how valid, reproducible, and replicable empirical studies involving LLM are. This uncertainty poses significant challenges for researchers and practitioners who seek to draw reliable conclusions from empirical studies.

The importance of open science practices and documenting the study setup is not to be underestimated [45]. One of the primary risks in creating irreproducible and irreplicable results based on studies involving LLMs stems from the variability in model performance due to their inherent non-determinism, but also due to differences in configuration, training data, model architecture, or

evaluation metrics. Slight changes can lead to significantly different results. The lack of standardized benchmarks and evaluation protocols further hinders the reproducibility and replicability of study results. Without detailed information on the exact study setup, benchmarks, and metrics used, it is challenging for other researchers to replicate results using different models and tools. These issues highlight the need for clear guidelines and best practices for designing and reporting studies involving LLMs. While the SE research community has developed guidelines for conducting and reporting specific types of empirical studies such as controlled experiments (e.g., *Experimentation in Software Engineering* [131], *Guide to Advanced Empirical Software Engineering* [111]) or their replications (e.g., *A Procedure and Guidelines for Analyzing Groups of Software Engineering Replications* [106]), we believe that LLMs have specific intrinsic characteristics that require specific guidelines for researchers to achieve an acceptable level of reproducibility and replicability (see also our previous position paper [122]). For example, even if we knew the specific version of a commercial LLM used for an empirical study, the reported task performance could still change over time, since commercial models are known to evolve beyond version identifiers [21]. Moreover, commercial providers do not guarantee the availability of old model or tool versions indefinitely. In addition to version differences, LLM performance varies widely depending on configured parameters such as temperature. Therefore, not reporting the parameter settings severely impacts reproducibility. Even for "open" models such as *Mistral, DeepSeek or Llama*, we do not know how they were fine-tuned for specific tasks and what the exact training data was [41]. A general problem when evaluating LLMs' performance is that we do not know whether the solution to a certain problem was part of the training data or not.

So far, there are no holistic guidelines for conducting and reporting studies involving LLMs in SE research. With this community effort, we try to fill this gap. After outlining our *Scope*, we continue by introducing a taxonomy of *Study Types* before presenting eight *Guidelines* that the authors of this article co-developed. Also, for each study type and guideline, we identify relevant research exemplars, both within the SE literature, but also outside the discipline where relevant; this serves a useful purpose given the vast proliferation of research on this nascent topic. The most recent version is always available online.[1] Other researchers can suggest changes via a public GitHub repository.xw[2]

2 Scope

In this section, we outline the scope of the study types and guidelines we present.

SE as our Target Discipline. We target *software engineering* (SE) research conducted in academic or industry contexts. While other disciplines have started developing community guidelines for reporting empirical studies involving LLMs, SE has not seen a holistic effort in that direction. In healthcare, for example, Gallifant et al. presented the *TRIPOD-LLM* guidelines [39], which are centered around a checklist for "*good reporting of studies that are developing, tuning, prompt engineering or evaluating an LLM*" [39]. Although the checklist overlaps with our *Guidelines* (e.g., reporting

the LLM name and version, comparing performance between LLMs, humans and other benchmarks), they are healthcare-specific (e.g., they require to "*explain the healthcare context*" and "*therapeutic and clinical workflow*"). Instead of a checklist, we distinguish MUST and SHOULD criteria and provide brief *tl;dr* summaries highlighting our most important recommendations. SE research utilizes a broader range of empirical methods than many other disciplines. Therefore, we are convinced that SE-specific guidelines need to be developed in combination with a *Study Types* taxonomy such as ours. In SE, Sallou et al. have presented a vision paper on mitigating validity threats in LLM-based SE research. Although their guidelines partially overlap with ours (e.g., repeating experiments to handle output variability, considering data leakage), our guidelines cover a broader range of study types and provide more detailed advise. For example, we contextualize our guidelines by study type, distinguish LLMs and LLM-based tools, talk about models, baselines, benchmarks, and metrics, and explicitly address human validation. Moreover, our guidelines are the result of an extensive coordination process between a large number of experts in empirical SE, hence a first step towards true community guidelines.

Focus on Natural Language Use Cases. We want to clarify that our focus is on LLMs, that is, natural language use cases. Multimodal foundational models are beyond the scope of our study types and guidelines. We are aware that these foundational models have great potential to support SE research and practice. However, due to the diversity of artifacts that can be generated or used as input (e.g., images, audio, and video) and the more demanding hardware requirements, we deliberately focus on LLMs only. However, our guidelines could be extended in the future to include foundational models beyond natural language text.

Focus on Direct Tool or Research Support. Given the exponential growth in LLM usage across all research domains, we also want to define the research contexts in which our guidelines apply. LLMs are already widely used to support several aspects of the overall research process, from fairly simple tasks such as proof-reading, spell-checking, and text translation, to more complex activities such as data coding and synthesis of literature reviews. Regarding tools, we focus on use cases in the area of AI for software engineering (AI4SE), that is, studying the support and automation of SE tasks with the help of artificial intelligence (AI), more specifically LLMs (see Section *LLMs as Tools for Software Engineers*). For research support, we focus on empirical SE research supported by LLMs (see Section *LLMs as Tools for Software Engineering Researchers*). By research support, we mean the active involvement of LLMs in data collection, processing, or analysis. We consider LLMs supporting the study design or the writing process to be out of scope.

Researchers as our Target Audience. Third, our guidelines mainly target SE researchers planning, designing, conducting, and reporting empirical studies involving LLMs. Although researchers who review scientific articles written by others can also use our guidelines, for example, to check whether the authors adhere to the essential MUST requirements we present, reviewers are not our main target audience.

3 Methodology

The starting point for the development of these guidelines was the 2024 meeting of the *International Software Engineering Research Network* (ISERN) in Barcelona, Spain. The first and last author of this paper organized a session to discuss the need to develop guidelines for empirical studies involving LLMs. These discussions resulted in a position paper published at the *2nd International Workshop on Methodological Issues with Empirical Studies in Software Engineering* (WSESE 2025) [122]. The first and last author presented a preprint of that paper at the *2nd Copenhagen Symposium on Human-Centered Software Engineering AI* (CHASEAI 2024), forming a workstream to lead the collaborative refinement and extension of the preliminary guidelines presented in the position paper.

We held bi-weekly meetings, in which we decided on the structure of the study type taxonomy and the guidelines. During these discussions, we added a new study type (*LLMs for Synthesis*), refined the guidelines to *Report Model Version, Configuration, and Customizations* and to *Report Tool Architecture beyond Models*, extended the guideline to *Report Prompts, their Development, and Interaction Logs*, and added the new guidelines to *Use Suitable Baselines, Benchmarks, and Metrics* and to *Report Limitations and Mitigations*. Then, at least two co-authors were assigned to each study type and guideline. They refined and extended the content to align with the intended structure, followed by a peer review and refinement phase. Afterwards, the first author reviewed all sections and discussed potential changes with the topic leads. After completing the internal review, we invited external experts in empirical SE to review the study types and guidelines. After incorporating their feedback,[3] we compiled and published this article.

4 Study Types

The development of empirical guidelines for software engineering (SE) studies involving LLMs is crucial to ensuring the validity and reproducibility of results. However, such guidelines must be tailored to different study types that each pose unique challenges. Therefore, we developed a taxonomy of study types that we then use to contextualize the recommendations we provide.

Each study type section starts with a **description**, followed by **examples** from the SE research community and beyond as well as the **advantages** and **challenges** of using LLMs for the respective study type. The remainder of this section is structured as follows:

3.1 **LLMs as Tools for Software Engineering Researchers**
    3.1.1 *LLMs as Annotators*
    3.1.2 *LLMs as Judges*
    3.1.3 *LLMs for Synthesis*
    3.1.4 *LLMs as Subjects*
3.2 **LLMs as Tools for Software Engineers**
    3.2.1 *Studying LLM Usage in Software Engineering*
    3.2.2 *LLMs for New Software Engineering Tools*
    3.2.3 *Benchmarking LLMs for Software Engineering Tasks*

---

[3]Discussion: https://github.com/se-uhd/llm-guidelines-website/issues/89

4.1 LLMs as Tools for Software Engineering Researchers

LLMs can serve as powerful tools to help researchers conduct empirical studies. They can automate various tasks such as data collection, pre-processing, and analysis. For example, LLMs can apply predefined coding guides to qualitative datasets (*LLMs as Annotators*), assess the quality of software artifacts (*LLMs as Judges*), generate summaries of research papers (*LLMs for Synthesis*), or simulate human behavior in empirical studies (*LLMs as Subjects*). This can significantly reduce the time and effort required to conduct a study. However, besides the many advantages that all these applications bring, they also come with challenges such as potential threats to validity and implications for the reproducibility of study results.

4.1.1 *LLMs as Annotators.*

Description. Just like human annotators, LLMs can label artifacts based on a pre-defined coding guide. However, they can label data much faster than any human could. In qualitative data analysis, manually annotating ("coding") natural language text, e.g., in software artifacts, open-ended survey responses, or interview transcripts, is a time-consuming manual process [12]. LLMs can be used to augment or replace human annotations, provide suggestions for new codes (see Section *LLMs for Synthesis*), or even automate the entire qualitative data analysis process.

Example(s). Recent work in software engineering has begun to explore the use of LLMs for annotation tasks. Huang et al. [53] proposed an approach that utilizes multiple LLMs for joint annotation of mobile application reviews. They used three models of comparable size with an absolute majority voting rule (i.e., a label is only accepted if it receives more than half of the total votes from the models). Accordingly, the annotations fell into three categories: exact matches (where all models agreed), partial matches (where a majority agreed), and non-matches (where no majority was reached). The study by Ahmed et al. [1] examined LLMs as annotators in software engineering research across five datasets, six LLMs, and ten annotation tasks. They found that model-model agreement strongly correlates with human-model agreement, suggesting situations in which LLMs could effectively replace human annotators. Their research showed that for tasks where humans themselves disagreed significantly , models also performed poorly. Conversely, if multiple LLMs reach similar solutions independently, then LLMs are likely suitable for the annotation task. They proposed to use model confidence scores to identify specific samples that could be safely delegated to LLMs, potentially reducing human annotation effort without compromising inter-rater agreement.

Advantages. Recent research demonstrates several advantages of using LLMs as annotators, including their cost effectiveness and accuracy. LLM-based annotation can dramatically reduce costs compared to human labeling, with studies showing cost reductions of 50-96% on various natural language tasks [127]. For example, He et al. found in their study that GPT-4 annotation costs only $122.08 compared to $4,508 for a comparable MTurk pipeline [48]. Moreover, the LLM-based approach resulted in a completion time of just 2 days versus several weeks for the crowd-sourced approach.

LLMs consistently demonstrate strong performance, with Chat-GPT's accuracy exceeding crowd workers by approximately 25% on average [42], achieving impressive results in specific tasks such as sentiment analysis (65% accuracy) [144]. LLMs also show remarkably high inter-rater agreement, higher than crowd workers and trained annotators [42].

Challenges. Challenges of using LLMs as annotators include reliability issues, human-LLM interaction challenges, biases, errors, and resource considerations. Studies suggest that while LLMs show promise as annotation tools in SE research, their optimal use may be in augmenting rather than replacing human annotators [48, 127]. LLMs can negatively affect human judgment when labels are incorrect [52], and their overconfidence requires careful verification [123]. Moreover, in previous studies, LLMs have shown significant variability in annotation quality depending on the dataset and the annotation task [92]. Studies have shown that LLMs are especially unreliable for high-stakes labeling tasks [128] and that LLMs can have notable performance disparities between label categories [144]. Recent empirical evidence indicates that LLM consistency in text annotation often falls below scientific reliability thresholds, with outputs being sensitive to minor prompt variations [99]. Context-dependent annotations pose a specific challenge, as LLMs show difficulty in correctly interpreting text segments that require broader contextual understanding [48]. Although pooling of multiple outputs can improve reliability, this approach requires additional computational resources and still requires validation against human-annotated data. While generally cost-effective, LLM annotation requires careful management of per token charges, particularly for longer texts [127]. Furthermore, achieving reliable annotations may require multiple runs of the same input to enable majority voting [99], although the exact cost comparison between LLM-based and human annotation is controversial [48]. Finally, research has identified consistent biases in label assignment, including tendencies to overestimate certain labels and misclassify neutral content [144].

4.1.2 LLMs as Judges.

Description. LLMs can act as judges or raters to evaluate properties of software artifacts. For example, LLMs can be used to assess code readability, adherence to coding standards, or the quality of code comments. Judgment is distinct from the more qualitative task of assigning a code or label to typically unstructured text (see Section *LLMs as Annotators*). It is also different from using LLMs for general software engineering tasks, as discussed in Section *LLMs as Tools for Software Engineering Researchers*.

Example(s). Lubos et al. [77] leveraged Llama-2 to evaluate the quality of software requirements statements. They prompted the LLM with the text below, where the words in curly brackets reflect the study parameters:

```
Your task is to evaluate the quality of a
software requirement.
Evaluate whether the following requirement is
{quality_characteristic}.
{quality_characteristic} means:
{quality_characteristic_explanation}
The evaluation result must be: 'yes' or 'no'.
Request: Based on the following description of
the project:
```

```
{project_description}
Evaluate the quality of the following
requirement: {requirement}.
Explain your decision and suggest an improved version.
```

Lubos et al. evaluated the LLM's output against human judges, to assess how well the LLM matched experts. Agreement can be measured in many ways; this study used Cohen's kappa [23] and found moderate agreement for simple requirements and poor agreement for more complex requirements. However, human evaluation of machine judgments may not be the same as evaluation of human judgments and is an open area of research [35]. A crucial decision in studies focusing on LLMs as judges is the number of examples provided to the LLM. This might involve no tuning (zero-shot), several examples (few-shot), or providing many examples (closer to traditional training data). In the example above, Lubos et al. chose zero-shot tuning, providing no specific guidance besides the provided context; they did not show the LLM what a 'yes' or 'no' answer might look like. In contrast, Wang et al. [126] provided a rubric to an LLM when treating LLMs as a judge to produce interpretable scales (0 to 4) for the overall quality of acceptance criteria generated from user stories. They combined this with a lower-level reward stage to refine acceptance criteria, which significantly improved correctness, clarity, and alignment with the user stories.

Advantages. Depending on the model configuration, LLMs can provide relatively consistent evaluations. They can further help mitigate human biases and the general variability that human judges might introduce. This may lead to more reliable and reproducible results in empirical studies, to the extent that these models can be reproduced or checkpointed. LLMs can be much more efficient and scale more easily than the equivalent human approach. With LLM automation, entire datasets can be assessed, as opposed to subsets, and the assessment produced can be at the selected level of granularity, e.g., binary outputs ('yes' or 'no') or defined levels (e.g., 0 to 4). However, the main constraint that varies by model and budget is the input context size, i.e., the number of tokens one can pass into a model. For example, the upper bound of the context that can be passed to OpenAi's o1-mini model is 32k tokens.

Challenges. When relying on the judgment of LLMs, researchers must build a *reliable* process for generating judgment labels that considers the non-deterministic nature of LLMs and report the intricacies of that process transparently [109]. For example, the order of options has been shown to affect LLM outputs in multiple-choice settings [96]. In addition to reliability, other quality attributes to consider include the *accuracy* of the labels. For example, a reliable LLM might be reliably inaccurate and wrong. Evaluating and judging large numbers of items, for example, to perform fault localization on the thousands of bugs that large open-source projects have to deal with, comes with costs in clock time, compute time, and environmental sustainability. Evidence shows that LLMs can behave differently when reviewing their own outputs [93]. In more human-oriented datasets (such as discussions of pull requests) LLMs may suffer from well-documented biases and issues with fairness [37]. For tasks in which human judges disagree significantly, it is not clear if an LLM judge should reflect the majority opinion or act as an independent judge. The underlying statistical framework of an LLM usually pushes outputs towards the most likely (majority)

answer. There is ongoing research on how suitable LLMs are as independent judges. Questions about bias, accuracy, and trust remain [15]. There is reason for concern about LLMs judging student assignments or doing peer review of scientific papers [141]. Even beyond questions of technical capacity, ethical questions remain, particularly if there is some implicit expectation that a human is judging the output. Involving a human in the judgment loop, for example, to contextualize the scoring, is one approach [91]. However, the lack of large-scale ground truth datasets for benchmarking LLM performance in judgment studies hinders progress in this area.

4.1.3   LLMs for Synthesis.

Description. LLMs can support synthesis tasks in software engineering research by processing and distilling information from qualitative data sources. In this context, synthesis refers to the process of integrating and interpreting information from multiple sources to generate higher-level insights, identify patterns across datasets, and develop conceptual frameworks or theories. Unlike annotation (see Section *LLMs as Annotators*), which focuses on categorizing or labeling individual data points, synthesis involves connecting and interpreting these annotations to develop a cohesive understanding of the phenomenon being studied. Synthesis tasks can also mean generating synthetic datasets (e.g., source code, bug-fix pairs, requirements, etc.) that are then used in downstream tasks to train, fine-tune, or evaluate existing models or tools. In this case, the synthesis is done primarily using the LLM and its training data; the input is limited to basic instructions and examples.

Example(s). Published examples of applying LLMs for synthesis in the software engineering domain are still scarce. However, recent work has explored the use of LLMs for qualitative synthesis in other domains, allowing reflection on how LLMs can be applied for this purpose in software engineering [12]. Barros et al. [14] conducted a systematic mapping study on the use of LLMs for qualitative research. They identified examples in domains such as healthcare and social sciences (see, e.g., [28, 79]) in which LLMs were used to support qualitative analysis, including grounded theory and thematic analysis. Overall, the findings highlight the successful generation of preliminary coding schemes from interview transcripts, later refined by human researchers, along with support for pattern identification. This approach was reported not only to expedite the initial coding process but also to allow researchers to focus more on higher-level analysis and interpretation. However, Barros et al. emphasize that effective use of LLMs requires structured prompts and careful human oversight. Similarly, de Morais Leça et al. [27] conducted a systematic mapping study to investigate how LLMs are used in qualitative analysis and how they can be applied in software engineering research. Consistent with the study by Barros et al. [14], de Morais Leça et al. identified that LLMs are applied primarily in tasks such as coding, thematic analysis, and data categorization, reducing the time, cognitive demands, and resources required for these processes. Finally, El-Hajjami and Salinesi's work is an example of using LLMs to create synthetic datasets. They present an approach to generate synthetic requirements, showing that they "*can match or surpass human-authored requirements for specific classification tasks*" [34].

Advantages. LLMs offer promising support for synthesis in SE research by helping researchers process artifacts such as interview transcripts and survey responses, or by assisting literature reviews. Qualitative research in SE traditionally faces challenges such as limited scalability, inconsistencies in coding, difficulties in generalizing findings from small or context-specific samples, and the influence of the researchers' subjectivity on data interpretation [12]. The use of LLMs for synthesis can offer advantages in addressing these challenges [12, 14, 27]. LLMs can reduce manual effort and subjectivity, improve consistency and generalizability, and assist researchers in deriving codes and developing coding guides during the early stages of qualitative data analysis [12, 16]. LLMs can enable researchers to analyze larger datasets, identifying patterns across broader contexts than traditional qualitative methods typically allow. In addition, they can help mitigate the effects of human subjectivity. However, while LLMs streamline many aspects of qualitative synthesis, careful oversight remains essential.

Challenges. Although LLMs have the potential to automate synthesis, concerns about overreliance remain, especially due to discrepancies between AI- and human-generated insights, particularly in capturing contextual nuances [13]. Bano et al. [13] found that while LLMs can provide structured summaries and qualitative coding frameworks, they may misinterpret nuanced qualitative data due to a lack of contextual understanding. Other studies have echoed similar concerns [12, 14, 27]. In particular, LLMs cannot independently assess the validity of arguments. Critical thinking remains a human responsibility in qualitative synthesis. Peters and Chin-Yee have shown that popular LLMs and LLM-based tools tend to overgeneralize results when summarizing scientific articles, that is, they "*produce broader generalizations of scientific results than those in the original.*" Compared to human-authored summaries, "*LLM summaries were nearly five times more likely to contain broad generalizations*" [95]. In addition, LLMs can produce biased results, reinforcing existing prejudices or omitting essential perspectives, making human oversight crucial to ensure accurate interpretation, mitigate biases, and maintain quality control. Moreover, the proprietary nature of many LLMs limits transparency. In particular, it is unknown how the training data might affect the synthesis process. Furthermore, reproducibility issues persist due to the influence of model versions and prompt variations on the synthesis result.

4.1.4   ***LLMs as Subjects.***

Description. In empirical studies, data is collected from participants through methods such as surveys, interviews, or controlled experiments. LLMs can serve as *subjects* in empirical studies by simulating human behavior and interactions (we use the term "subject" since "participant" implies being human [6]). In this capacity, LLMs generate responses that approximate those of human participants, which makes them particularly valuable for research involving user interactions, collaborative coding environments, and software usability assessments. This approach enables data collection that closely reflects human reactions while avoiding the need for direct human involvement. To achieve this, prompt engineering techniques are widely employed, with a common approach being the use of the *Personas Pattern* [64], which involves tailoring LLM responses to align with predefined profiles or roles that emulate specific user

archetypes. Zhao et al. outline opportunities and challenges of using LLMs as research subjects in detail [140]. Furthermore, recent sociological studies have emphasized that, to be effectively utilized in this capacity, LLMs—including their agentic versions—should meet four criteria of algorithmic fidelity [5]. Generated responses should be: (1) indistinguishable from human-produced texts (e.g., LLM-generated code reviews should be comparable to those from real developers); (2) consistent with the attitudes and sociodemographic information of the conditioning context (e.g., LLMs simulating junior developers should exhibit different confidence levels, vocabulary, and concerns compared to senior engineers); (3) naturally aligned with the form, tone, and content of the provided context (e.g., responses in an agile stand-up meeting simulation should be concise, task-focused, and aligned with sprint objectives rather than long, formal explanations); and (4) reflective of patterns in relationships between ideas, demographics, and behavior observed in comparable human data (e.g., discussions on software architecture decisions should capture trade-offs typically debated by human developers, such as maintainability versus performance, rather than abstract theoretical arguments).

Example(s). LLMs can be used as subjects in various types of empirical studies, enabling researchers to simulate human participants. The broader applicability of LLM-based studies beyond software engineering has been compiled by Xu et al. [136], who examined various uses in social science research. Given the socio-technical nature of software development, some of these approaches are transferable to empirical software engineering research. For example, LLMs can be applied in survey and interview studies to impersonate developers responding to survey questionnaires or interviews, allowing researchers to test the clarity and effectiveness of survey items or to simulate responses under varying conditions, such as different levels of expertise or cultural contexts. For example, Gerosa et al. [40] explored persona-based interviews and multi-persona focus groups, demonstrating how LLMs can emulate human responses and behaviors while addressing ethical concerns, biases, and methodological challenges. Another example is usability studies, in which LLMs can simulate end-user feedback, providing insights into potential usability issues and offering suggestions for improvement based on predefined user personas. This aligns with the work of Bano et al. [11], who investigated biases in LLM-generated candidate profiles in SE recruitment processes. Their study, which analyzed both textual and visual inputs, revealed biases favoring male Caucasian candidates, lighter skin tones, and slim physiques, particularly for senior roles.

Advantages. Using LLMs as subjects can reduce the effort of recruiting human participants, a process that is often time-consuming and costly [78], by augmenting existing datasets or, in some cases, completely replacing human participants. In interviews and survey studies, LLMs can simulate diverse respondent profiles, enabling access to underrepresented populations, which can strengthen the generalizability of findings. In data mining studies, LLMs can generate synthetic data (see *LLMs for Synthesis*) to fill gaps where real-world data is unavailable or underrepresented.

Challenges. It is important that researchers are aware of the inherent biases [26] and limitations [47, 124] when using LLMs

as study subjects. Schröder et al. [110], who have studied discrepancies between LLM and human responses, even conclude that "*LLMs do not simulate human psychology and recommend that psychological researchers should treat LLMs as useful but fundamentally unreliable tools that need to be validated against human responses for every new application.*" One critical concern is construct validity. LLMs have been shown to misrepresent demographic group perspectives, failing to capture the diversity of opinions and experiences within a group. The use of identity-based prompts can reduce identities to fixed and innate characteristics, amplifying perceived differences between groups. These biases introduce the risk that studies which rely on LLM-generated responses may inadvertently reinforce stereotypes or misrepresent real-world social dynamics. Alternatives to demographic prompts can be employed when the goal is to broaden response coverage [124]. Beyond construct validity, internal validity must also be considered, particularly with regard to causal conclusions based on studies relying on LLM-simulated responses. Finally, external validity remains a challenge, as findings based on LLMs may not generalize to humans.

4.2 LLMs as Tools for Software Engineers

LLM-based assistants have become an essential tool for software engineers, supporting them in various tasks such as code generation and debugging. Researchers have studied how software engineers use LLMs (*Studying LLM Usage in Software Engineering*), developed new tools that integrate LLMs (*LLMs for New Software Engineering Tools*), and benchmarked LLMs for software engineering tasks (*Benchmarking LLMs for Software Engineering Tasks*).

4.2.1 Studying LLM Usage in Software Engineering.

Description. Studying how software engineers use LLMs and LLM-based tools is crucial to understand the current state of practice in SE. Researchers can observe software engineers' usage of LLM-based tools in the field, or study if and how they adopt such tools, their usage patterns, as well as perceived benefits and challenges. Surveys, interviews, observational studies, or analysis of usage logs can provide insights into how LLMs are integrated into development processes, how they influence decision making, and what factors affect their acceptance and effectiveness. Such studies can inform improvements for existing LLM-based tools, motivate the design of novel tools, or derive best practices for LLM-assisted software engineering. They can also uncover risks or deficiencies of existing tools.

Example(s). Based on a convergent mixed-methods study, Russo has found that early adoption of generative AI by software engineers is primarily driven by compatibility with existing workflows [104]. Khojah et al. investigated the use of ChatGPT (GPT-3.5) by professional software engineers in a week-long observational study [62]. They found that most developers do not use the code generated by ChatGPT directly but instead use the output as a guide to implement their own solutions. Azanza et al. conducted a case study that evaluated the impact of introducing LLMs on the onboarding process of new software developers [9] (GPT-3). Their study identified potential in the use of LLMs for onboarding, as it can allow newcomers to seek information on their own, without the need to "*bother*" senior colleagues. Jahic and Sami surveyed

participants from 15 software companies regarding their practices on LLMs in software engineering [55]. They found that the majority of study participants had already adopted AI for software engineering tasks; most of them used ChatGPT. Multiple participants cited copyright and privacy issues, as well as inconsistent or low-quality outputs, as barriers to adoption. Retrospective studies that analyze data generated while developers use LLMs can provide additional insights into human-LLM interactions. For example, researchers can employ data mining methods to build large-scale conversation datasets, such as the DevGPT dataset introduced by Xiao et al. [134]. Conversations can then be analyzed using quantitative [97] and qualitative [84] analysis methods.

Advantages. Studying the real-world usage of LLM-based tools allows researchers to understand the state of practice and guide future research directions. In field studies, researchers can uncover usage patterns, adoption rates, and the influence of contextual factors on usage behavior. Outside of a controlled study environment, researchers can uncover contextual information about LLM-assisted SE workflows beyond the specific LLMs being evaluated. This may, for example, help researchers generate hypotheses about how LLMs impact developer productivity, collaboration, and decision-making processes. In controlled laboratory studies, researchers can study specific phenomena related to LLM usage under carefully regulated, but potentially artificial conditions. Specifically, they can isolate individual tasks in the software engineering workflow and investigate how LLM-based tools may support task completion. Furthermore, controlled experiments allow for direct comparisons between different LLM-based tools. The results can then be used to validate general hypotheses about human-LLM interactions in SE.

Challenges. When conducting field studies in real-world environments, researchers have to ensure that their study results are "*dependable*" [117] beyond the traditional validity criteria such as internal or construct validity. The usage environment in a real-world context can often be extremely diverse. The integration of LLMs can range from specific LLMs based on company policies to the unregulated use of any available LLM. Both extremes may influence the adoption of LLM by software engineers, and hence need to be addressed in the study methodology. In addition, in longitudinal case studies, the timing of the study may have a significant impact on its result, as LLMs and LLM-based tools are rapidly evolving. Moreover, developers are still learning how to best make use of the new technology, and best practices are still being developed and established. Furthermore, the predominance of proprietary commercial LLM-based tools in the market poses a significant barrier to research. Limited access to telemetry data or other usage metrics restricts the ability of researchers to conduct comprehensive analyses of real-world tool usage. To make study results reliable, researchers must establish that their results are rigorous, for example, by using triangulation to understand and minimize potential biases [117]. When it comes to studying LLM usage in controlled laboratory studies, researchers may struggle with the inherent variability of LLM outputs. Since reproducibility and precision are essential for hypothesis testing, the stochastic nature of LLM responses—where identical prompts may yield different outputs across participants—can complicate the interpretation of experimental results and affect the reliability of study findings.

Finally, agentic software development tools such as Claude Code have configurable degrees of human involvement, and can run almost any terminal command. This autonomy together with the before-mentioned non-determinism make it hard to reproducibly study agentic tools in the field or in laboratory environments.

4.2.2 LLMs for New Software Engineering Tools.

Description. LLMs are being integrated into new tools that support software engineers in their daily tasks, e.g., to assist in code comprehension [138] and test case generation [107]. One way of integrating LLM-based tools into software engineers' workflows are GenAI agents. Unlike traditional LLM-based tools, these agents are capable of acting autonomously and proactively, are often tailored to meet specific user needs, and can interact with external environments [119, 130]. From an architectural perspective, GenAI agents can be implemented in various ways [130]. However, they generally share three key components: (1) a reasoning mechanism that guides the LLM (often enabled by advanced prompt engineering), (2) a set of tools to interact with external systems (e.g., APIs or databases), and (3) a user communication interface that extends beyond traditional chat-based interactions [101, 118, 142]. Researchers can also test and compare different tool architectures to increase artifact quality and developer satisfaction.

Example(s). Yan et al. proposed IVIE, a tool integrated into the VS Code graphical interface that generates and explains code using LLMs [138]. The authors focused more on the presentation, providing a user-friendly interface to interact with the LLM. Schäfer et al. [107] presented a large-scale empirical evaluation on the effectiveness of LLMs for automated unit test generation. They presented TestPilot, a tool that implements an approach in which the LLM is provided with prompts that include the signature and implementation of a function under test, along with usage examples extracted from the documentation. Richards and Wessel introduced a preliminary GenAI agent designed to assist developers in understanding source code by incorporating a reasoning component grounded in the theory of mind [101].

Advantages. From an engineering perspective, developing LLM-based tools is easier than implementing many traditional SE approaches such as static analysis or symbolic execution. Depending on the capabilities of the underlying model, it is also easier to build tools that are independent of a specific programming language. This enables researchers to build tools for a more diverse set of tasks. In addition, it allows them to test their tools in a wider range of contexts.

Challenges. Traditional approaches, such as static analysis, are deterministic. LLMs are not. Although the non-determinism of LLMs can be mitigated using configuration parameters and prompting strategies, this poses a major challenge. It can be challenging for researchers to evaluate the effectiveness of a tool, as minor changes in the input can lead to major differences in the performance. Since the exact training data is often not published by model vendors, a reliable assessment of tool performance for unknown data is difficult. From an engineering perspective, while open models are available, the most capable ones require substantial hardware resources. Using cloud-based APIs or relying on third-party providers

for hosting, while seemingly a potential solution, introduces new concerns related to data privacy and security.

4.2.3 Benchmarking LLMs for Software Engineering Tasks.

Description. Benchmarking is the process of evaluating an LLM's performance using standardized tasks and metrics, which requires high-quality reference datasets. LLM output is compared to a ground truth from the benchmark dataset using general metrics for text generation, such as *ROUGE*, *BLEU*, or *METEOR* [50], or task-specific metrics, such as *CodeBLEU* for code generation. For example, *HumanEval* [22] is often used to assess code generation, establishing it as a de facto standard.

Example(s). In SE, benchmarking may include the evaluation of an LLM's ability to produce accurate and reliable outputs for a given input, usually a task description, which may be accompanied by data obtained from curated real-world projects or from synthetic SE-specific datasets. Typical tasks include code generation, code summarization, code completion, and code repair, but also natural language processing tasks, such as anaphora resolution (i.e., the task of identifying the referring expression of a word or phrase occurring earlier in the text). *RepairBench* [112], for example, contains 574 buggy Java methods and their corresponding fixed versions, which can be used to evaluate the performance of LLMs in code repair tasks. This benchmark uses the *Plausible@1* metric (i.e., the probability that the first generated patch passes all test cases) and the *AST Match@1* metric (i.e., the probability that the abstract syntax tree of the first generated patch matches the ground truth patch). *SWE-Bench* [59] is a more generic benchmark that contains 2,294 SE Python tasks extracted from GitHub pull requests. To score the LLM's performance on the tasks, the benchmark validates whether the generated patch is applicable (i.e., successfully compiles) and calculates the percentage of passed test cases.

Advantages. Properly built benchmarks provide objective, reproducible evaluation across different tasks, enabling a comparison between different models (and versions). In addition, benchmarks built for specific SE tasks can help identify LLM weaknesses and support their optimization and fine-tuning for such tasks. They can foster open science practices by providing a common ground for sharing data (e.g., as part of the benchmark itself) and results (e.g., of models run against a benchmark). Benchmarks built using real-world data can help legitimize research results for practitioners, supporting industry-academia collaboration.

Challenges. Benchmark contamination, that is, the inclusion of the benchmark in the LLM training data [2], has recently been identified as a problem. The careful selection of samples and the creation of the corresponding input prompts is particularly important, as correlations between prompts may bias the benchmark results [113]. Although LLMs might perform well on a specific benchmark such as *HumanEval* or *SWE-bench*, they do not necessarily perform well on other benchmarks. Moreover, benchmarks usually do not capture the full complexity and diversity of software engineering work [19]. Another issue is that metrics used for benchmarks, such as *perplexity* or *BLEU-N*, do not necessarily reflect human judgment. Recently, Cao et al. [18] have proposed guidelines for the creation of LLM benchmarks related to coding

tasks, grounded in a systematic survey of existing benchmarks. In this process, they highlight current shortcomings related to reliability, transparency, irreproducibility, low data quality, and inadequate validation measures. For more details on benchmarks, see Section *Use Suitable Baselines, Benchmarks, and Metrics.*

5 Guidelines

Our guidelines focus on LLMs, that is, foundational models that use *text* as in- and output. We do not address multi-modal foundation models that support other media such as images. However, many of our recommendations apply to these models as well. We consider the extension of our guidelines to explicitly address multi-modal foundational models an important direction for future work. The main goal of our guidelines is to *enable reproducibility and replicability* of empirical studies involving LLMs in software engineering. While we consider LLM-based studies to have characteristics that differ from traditional empirical studies (e.g., their inherent non-determinism and the fact that truly open models are rare), previous guidelines regarding open science and empirical studies still apply. Although full reproducibility of LLM study results is very challenging given LLM's non-determinism, transparency on LLM usage, methods, data, and architecture, as suggested by our guidelines, is an essential prerequisite for future replication studies. The wording of our guidelines (MUST, SHOULD, MAY) follows RFC 2119 [86] and RFC 8174 [87]. Throughout the following sections, we mention the information we expect researchers to report in the *paper* and/or in the *supplementary material*. We are aware that different publication venues have different page limits and that not all aspects can be reported in the *paper*. If information MUST be reported in the *paper*, we explicitly mention this in the specific guidelines. Of course, it is better to report essential information that we expect in the *paper* in the *supplementary material* than not at all. The *supplementary material* SHOULD be published according to the *ACM SIGSOFT Open Science Policies* [44]. At the beginning of each section, we provide a brief *tl;dr* summary that lists the most important aspects of the corresponding guideline. In addition to our **recommendations**, we provide **examples** from the SE research community and beyond, as well as the **advantages** and potential **challenges** of following the respective guidelines. We conclude each guideline by linking it to the **study types**. The remainder of this section presents our eight guidelines and is structured as follows:

4.1 *Declare LLM Usage and Role*
4.2 *Report Model Version, Configuration, and Customizations*
4.3 *Report Tool Architecture beyond Models*
4.4 *Report Prompts, their Development, and Interaction Logs*
4.5 *Use Human Validation for LLM Outputs*
4.6 *Use an Open LLM as a Baseline*
4.7 *Use Suitable Baselines, Benchmarks, and Metrics*
4.8 *Report Limitations and Mitigations*

5.1 Declare LLM Usage and Role

tl;dr: Researchers MUST disclose any use of LLMs to support empirical studies in their paper. *They SHOULD report their purpose, automated tasks, and expected benefits.*

5.1.1 **Recommendations.** When conducting any kind of empirical study involving LLMs, researchers MUST clearly declare that an LLM was used (see *Scope* for what we consider relevant research support). This SHOULD be done in a suitable section of the *paper*, for example, in the introduction or research methods section. For authoring scientific articles, this transparency is, for example, required by the ACM Policy on Authorship: "*The use of generative AI tools and technologies to create content is permitted but must be fully disclosed in the Work*" [7]. Beyond generic authorship declarations and declarations that LLMs were used as part of the research process, researchers SHOULD report the exact purpose of using an LLM in a study, the tasks it was used to automate, and the expected benefits in the *paper*.

5.1.2 **Example(s).** The *ACM Policy on Authorship* suggests to disclose the usage of GenAI tools in the acknowledgments section of the *paper*, for example: *"ChatGPT was utilized to generate sections of this Work, including text, tables, graphs, code, data, citations"* [7]. Similarly, the acknowledgments section could also be used to disclose GenAI usage for other aspects of the research, if not explicitly described in other sections. The ACM policy further suggests: *"If you are uncertain about the need to disclose the use of a particular tool, err on the side of caution, and include a disclosure in the acknowledgments section of the Work"* [7]. For double-blind review, during which the acknowledgments section is usually hidden, researchers can add a temporary "AI Disclosure" section in the same place the acknowledgments section would appear. An example of an LLM disclosure beyond writing support can be found in a recent paper by Lubos et al. [77], in which they write in the methodology section:

> *"We conducted an LLM-based evaluation of requirements utilizing the Llama 2 language model with 70 billion parameters, fine-tuned to complete chat responses..."*

5.1.3 **Advantages.** Transparency in the use of LLMs helps other researchers understand the context and scope of the study, facilitating better interpretation and comparison of the results. Beyond this declaration, we recommend researchers to be explicit about the LLM version they used (see Section *Report Model Version, Configuration, and Customizations*) and the LLM's exact role (see Section *Report Tool Architecture beyond Models*).

5.1.4 **Challenges.** In general, we expect following our recommendations to be straightforward. One challenge might be authors' reluctance to disclose LLM usage for valid use cases because they fear that AI-generated content makes reviewers think that the authors' work is less original. However, the *ACM Policy on Authorship* is very clear in that any use of GenAI tools to create content MUST be disclosed. Before the introduction of LLMs, human proof-reading was possible and allowed and was not required to be declared. Our guidelines focus on research support beyond proof-reading and writing support (see *Scope*). However, over time, the threshold of what must be declared and what not will most likely evolve.

5.1.5 **Study Types.** Researchers MUST follow this guideline for all study types.

5.2 Report Model Version, Configuration, and Customizations

tl;dr: Researchers MUST report the exact LLM model or tool version, configuration, and experiment date in the paper. *For fine-tuned models, they MUST describe the fine-tuning goal, dataset, and procedure. Researchers SHOULD include default parameters, explain model choices, compare base- and fine-tuned model using suitable metrics and benchmarks, and share fine-tuning data and weights (or alternatively justify why they cannot share them).*

This guideline focuses on documenting the *model-specific* aspects of empirical studies involving LLMs. While Section *Report Tool Architecture beyond Models* addresses how LLMs are integrated into larger systems and tools, here we concentrate on the models themselves, their version, configuration parameters, and any direct customizations applied to the model (e.g., fine-tuning). Whether only this guideline or also the following one applies for a particular study depends on the exact study setup and tool architecture. In any event, as soon as an LLM is involved, the information outlined in this guideline is essential to enable reproducibility and replicability.

5.2.1 **Recommendations.** LLMs or LLM-based tools, especially those offered as-a-service, are frequently updated; different versions may produce different results for the same input. Moreover, configuration parameters such as temperature or seed values affect content generation. Therefore, researchers MUST document in the *paper* which model or tool version they used in their study, along with the date when the experiments were carried out and the configured parameters that affect output generation. Since default values might change over time, researchers SHOULD always report all configuration values, even if they used the defaults. Checksums and fingerprints MAY be reported since they identify specific versions and configurations. Depending on the study context, other properties such as the context window size (number of tokens) MAY be reported. Researchers SHOULD motivate in the *paper* why they selected certain models, versions, and configurations. Potential reasons can be monetary (e.g., no funding to integrate large commercial models), technical (e.g., existing hardware only supports smaller models), or methodological (e.g., planned comparison to previous work). Depending on the specific study context, additional information regarding the experiment or tool architecture SHOULD be reported (see Section *Report Tool Architecture beyond Models*).

A common customization approach for existing LLMs is fine-tuning. If a model was fine-tuned, researchers MUST describe the fine-tuning goal (e.g., improving the performance for a specific task), the fine-tuning procedure (e.g., full fine-tuning vs. Low-Rank Adaptation (LoRA), selected hyperparameters, loss function, learning rate, batch size, etc.), and the fine-tuning dataset (e.g., data sources, the preprocessing pipeline, dataset size) in the *paper*. Researchers SHOULD either share the fine-tuning dataset as part of the *supplementary material* or explain in the *paper* why the data cannot be shared (e.g., because it contains confidential or personal data that could not be anonymized). The same applies to the fine-tuned

model weights. Suitable benchmarks and metrics SHOULD be used to compare the base model with the fine-tuned model (see Section *Use Suitable Baselines, Benchmarks, and Metrics*).

In summary, our recommendation is to report:

(1) Model/tool name and version (MUST in *paper*); (2) All relevant configured parameters that affect output generation (MUST in *paper*); (3) Default values of all available parameters (SHOULD); (4) Checksum/fingerprint of used model version and configuration (MAY); (5) Additional properties such as context window size (MAY).

For fine-tuned models, additional recommendations apply:

(1) Fine-tuning goal (MUST in *paper*); (2) Fine-tuning dataset creation and characterization (MUST in *paper*); (3) Fine-tuning parameters and procedure (MUST in *paper*); (4) Fine-tuning dataset and fine-tuned model weights (SHOULD); (5) Validation metrics and benchmarks (SHOULD).

Commercial models (e.g., GPT-4o) or LLM-based tools (e.g., Chat-GPT) might not give researchers access to all required information. Our suggestion is to report what is available and openly acknowledge limitations that hinder reproducibility (see also Sections *Report Prompts, their Development, and Interaction Logs* and *Report Limitations and Mitigations*).

5.2.2 **Example(s).** Based on the documentation that OpenAI and Azure provide [82, 90], researchers might, for example, report:

> *"We integrated a* `gpt-4` *model in version* `0125-Preview` *via the Azure OpenAI Service, and configured it with a temperature of 0.7, top_p set to 0.8, a maximum token length of 512, and the seed value* `23487`. *We ran our experiment on 10th January 2025' (system fingerprint* `fp_6b68a8204b`*).*

Kang et al. provide a similar statement in their paper on exploring LLM-based bug reproduction [60]:

> *"We access OpenAI Codex via its closed beta API, using the code-davinci-002 model. For Codex, we set the temperature to 0.7, and the maximum number of tokens to 256."*

Our guidelines additionally suggest to report a checksum/fingerprint and exact dates, but otherwise this example is close to our recommendations.

Similar statements can be made for self-hosted models. However, when self-hosting models, the *supplementary material* can become a true replication package, providing specific instructions to reproduce study results. For example, for models provisioned using ollama, one can report the specific tag and checksum of the model being used, e.g., *"llama3.3, tag 70b-instruct-q8_0, checksum d5b5e1b84868."* Given suitable hardware, running the corresponding model in its default configuration is then as easy as executing one command in the command line (see also Section *Use an Open LLM as a Baseline*): `ollama run llama3.3:70b-instruct-q8_0`

An example of a study involving fine-tuning is Dhar et al.'s work [30]. They conducted an exploratory empirical study to assess whether LLMs can generate architectural design decisions. The authors detail the system architecture, including the decision-making framework, the role of the LLM in generating design decisions, and the interaction between the LLM and other components of the system (see Section *Report Tool Architecture beyond Models*). The authors provide information on the fine-tuning approach and datasets used for the evaluation, including the source of the architectural

decision records, preprocessing methods, and the criteria for data selection.

5.2.3 **Advantages.** Reporting of the information described above is a prerequisite for the verification, reproduction, and replication of LLM-based studies under the same or similar conditions. As mentioned before, LLMs are inherently non-deterministic. However, this cannot be an excuse to dismiss the verifiability and reproducibility of empirical studies involving LLMs. Although exact reproducibility is hard to achieve, researchers can do their best to come as close as possible to that gold standard. Part of that effort is reporting the information outlined in this guideline.

5.2.4 **Challenges.** Different model providers and modes of operating the models allow for varying degrees of information. For example, OpenAI provides a model version and a system fingerprint describing the backend configuration, which can also influence the output. However, in fact, the fingerprint is intended only to detect changes in the model or its configuration; one cannot go back to a certain fingerprint. As a beta feature, OpenAI lets users set a seed parameter to receive "(mostly) consistent output" [89]. However, the seed value does not allow for full reproducibility and the fingerprint changes frequently. While, as motivated above, open models significantly simplify re-running experiments, they also come with challenges in terms of reproducibility, as generated outputs can be inconsistent despite setting the temperature to 0 and using a seed value (see GitHub issue for Llama3).

5.2.5 **Study Types.** This guideline MUST be followed for all study types for which the researcher has access to (parts of) the model's configuration. They MUST always report the configuration that is visible to them, acknowledging the reproducibility challenges of commercial tools and models that are offered as-a-service. Depending on the specific study type, researchers SHOULD provide additional information on the architecture of a tool they built (see Section *Report Tool Architecture beyond Models*), prompts and interactions logs (see Section *Report Prompts, their Development, and Interaction Logs*), and specific limitations and mitigations (see Section *Report Limitations and Mitigations*).

For example, when *Studying LLM Usage in Software Engineering* by focusing on commercial tools such as ChatGPT or GitHub Copilot, researchers MUST be as specific as possible in describing their study setup. The configured model name, version, and the date when the experiment was conducted MUST always be reported. In those cases, reporting other aspects, such as prompts and interaction logs, is essential.

5.3 Report Tool Architecture Beyond Models

> *tl;dr: Researchers MUST describe the full architecture of LLM-based tools that they develop in their* paper. *This includes the role of the LLM, interactions with other components, and the overall system behavior. If autonomous agents are used, researchers MUST specify agent roles, reasoning frameworks, and communication flows. Hosting, hardware setup, and latency implications MUST be reported. For tools using retrieval or augmentation methods, data sources, integration mechanisms, and update and versioning*

> *strategies MUST be described. For ensemble architectures, the coordination logic between models MUST be explained. Researchers SHOULD include architectural diagrams and justify design decisions. Non-disclosed confidential or proprietary components MUST be acknowledged as reproducibility limitations.*

This section addresses the *system-level* aspects of LLM-based tools that researchers develop, complementing Section *Report Model Version, Configuration, and Customizations*, which focuses on model-specific details. While standalone LLMs require limited architectural documentation, a more detailed description is required for study setups and tool architectures that integrate LLMs with other components to create more complex systems. This section provides guidelines for documenting these broader architectures.

5.3.1 **Recommendations.** Oftentimes, LLM-based tool have *complex software layers* around the model (or models) that pre-processes data, prepares prompts, filters user requests, or post-processes responses. An example is ChatGPT, which allows users to select from different GPT models. GitHub Copilot allows users to select the same models, but its answers may significantly differ from ChatGPT, as GitHub Copilot automatically adds context from the software project in which it is used. Researchers can build their own tools using GPT models directly (*e.g.*, via the OpenAI API). The infrastructure and business logic around the bare model can significantly contribute to the performance of a tool for a given task. Therefore, researchers MUST clearly describe the tool architecture and what exactly the LLM (or ensemble of LLMs) contributes to the tool or method presented in a research paper.

If an LLM is used as a *standalone system*, for example, by sending prompts directly to a GPT-4o model via the OpenAI API without pre-processing the prompts or post-processing the responses, a brief explanation of this approach is usually sufficient. However, if LLMs are integrated into more *complex systems* with pre-processing, retrieval mechanisms, or autonomous agents, researchers MUST provide a detailed description of the system architecture in the *paper*. Aspects to consider are how the LLM interacts with other components such as databases, external APIs, and frameworks. If the LLM is part of an *agent-based system* that autonomously plans or executes tasks, researchers MUST describe its exact architecture, including the agents' roles (e.g., planner, executor, coordinator), whether it is a single-agent or multi-agent system, how it interacts with external tools and users, and the reasoning framework used (e.g., chain-of-thought, self-reflection, multi-turn dialogue).

For agent-based systems that use external tools (e.g., Claude Code and its subagents), researchers MUST discuss *agent behavior traceability* by clearly delineating three distinct components: (1) *LLM Input/Output*: Internal deliberation, planning, and interpretation performed by the model. (2) *External tool calls*: Specific invocations of APIs, databases, file systems, or other external services that the agent explicitly triggers (e.g., via the Model Context Protocol). (3) *User or system interactions*: Human-in-the-loop feedback, environment responses, or multi-agent communication. Researchers SHOULD report complete execution traces that show the sequence and causality between these components, including inputs and outputs of all external tool calls. This traceability is essential for determining whether task success is attributable to the

LLM's output and tool-calling capabilities, the external tools' functionality, or their interaction patterns. When full tool call logging is not feasible due to privacy or proprietary constraints, researchers SHOULD provide representative examples or anonymized traces that demonstrate the agent's decision-making process.

Researchers SHOULD provide a high-level architectural diagram to improve transparency. To improve clarity, researchers SHOULD explain design decisions, particularly regarding how the models were hosted and accessed (API-based, self-hosted, etc.) and which retrieval mechanisms were implemented (keyword search, semantic similarity matching, rule-based extraction, etc.). Researchers MUST NOT omit critical architectural details that could affect reproducibility, such as dependencies on proprietary tools that influence tool behavior. Especially for *time-sensitive measurements*, the previously mentioned description of the hosting environment is central, as it can significantly impact the results. Researchers MUST clarify whether local infrastructure or cloud services were used, including detailed infrastructure specifications and latency considerations.

If *retrieval or augmentation methods* were used (e.g., retrieval-augmented generation (RAG), rule-based retrieval, structured query generation, or hybrid approaches), researchers MUST describe how external data is retrieved, stored, and integrated into the LLM's responses. This includes specifying the type of storage or database used (e.g., vector databases, relational databases, knowledge graphs) and how the retrieved information is selected and used. Stored data used for context augmentation MUST be reported, including details on data preprocessing, versioning, and update frequency. If this data is not confidential, an anonymized snapshot of the data used for context augmentation SHOULD be made available.

For *ensemble models*, in addition to following the *Report Model Version, Configuration, and Customizations* guideline for each model, the researchers MUST describe the architecture that connects the models. The *paper* MUST at least contain a high-level description, and details can be reported in the *supplementary material*. Aspects to consider include documenting the logic that determines which model handles which input, the interaction between models, and the architecture for combining outputs (e.g., majority voting, weighted averaging, sequential processing).

5.3.2 **Example(s).** Some empirical studies involving LLMs in SE have documented the architecture and supplemental data according to our guidelines. In the following, we provide two examples.

Schäfer et al. conducted an empirical evaluation of using LLMs for automated unit test generation [107]. The authors provide a comprehensive description of the system architecture, detailing how the LLM is integrated into the software development workflow to analyze codebases and produce the corresponding unit tests. The architecture includes components for code parsing, prompt formulation, interaction with the LLM, and integration of the generated tests into existing test suites. The paper also elaborates on the datasets utilized for training and evaluating the LLM's performance in unit test generation. It specifies the sources of code samples, the selection criteria, and the preprocessing steps undertaken to prepare the data.

A second example is Yan et al.'s IVIE tool [138], which integrates LLMs into the VS Code interface. The authors document the tool architecture, detailing the IDE integration, context extraction

from code editors, and the formatting pipeline for LLM-generated explanations. This documentation illustrates how architectural components beyond the core LLM affect the overall tool performance and user experience.

5.3.3 **Advantages.** Usually, researchers implement software layers around the bare LLMs, using different architectural patterns. These implementations significantly impact the performance of LLM-based tools and hence need to be documented in detail. Documenting the architecture and supplemental data of LLM-based systems enhances reproducibility and transparency [76]. In empirical software engineering studies, this is essential for experiment replication, result validation, and benchmarking. A clear documentation of the architecture and the supplemental data that was used enables comparison and upholds scientific rigor and accountability, fostering reliable and reusable research.

5.3.4 **Challenges.** Researchers face challenges in documenting LLM-based architectures, including proprietary APIs and dependencies that restrict disclosure, managing large-scale retrieval databases, and ensuring efficient query execution. They must also balance transparency with data privacy concerns, adapt to the evolving nature of LLM integrations, and, depending on the context, handle the complexity of multi-agent interactions and decision-making logic, all of which can impact reproducibility and system clarity.

5.3.5 **Study Types.** This guideline MUST be followed for all studies that involve tools with system-level components beyond bare LLMs, from lightweight wrappers that pre-process user input or post-process model outputs, to systems employing retrieval-augmented methods or complex agent-based architectures.

5.4 Report Prompts, their Development, and Interaction Logs

> *tl;dr: Researchers MUST publish all prompts, including their structure, content, formatting, and dynamic components. If full prompt disclosure is not feasible, for example, due to privacy or confidentiality concerns, summaries or examples SHOULD be provided. Prompt development strategies (e.g., zero-shot, few-shot), rationale, and selection process MUST be described. When prompts are long or complex, input handling and token optimization strategies MUST be documented. For dynamically generated or user-authored prompts, generation and collection processes MUST be reported. Prompt reuse across models and configurations MUST be specified. Researchers SHOULD report prompt revisions and pilot testing insights. To address model non-determinism and ensure reproducibility, especially when targeting SaaS-based commercial tools, full interaction logs (prompts and responses) SHOULD be included if privacy and confidentiality can be ensured.*

Prompts are critical for any study involving LLMs. Depending on the task, prompts may include various types of content, including instructions, context, and input data, and output indicators. For SE tasks, the context can include source code, execution traces, error messages, and other forms of natural language content. The output can be unstructured text or a structured format such as JSON. Prompts significantly influence the quality of the output of a

model, and understanding how exactly they were formatted and integrated into an LLM-based study is essential to ensure transparency, verifiability, and reproducibility. We remind the reader that these guidelines do not apply when LLMs are used solely for language polishing, paraphrasing, translation, tone or style adaptation, or layout edits (see *Scope*).

5.4.1 **Recommendations.** Researchers MUST report the all prompts that were used for an empirical study, including all instructions, context, input data, and output indicators. The only exception to this is if transcripts might identify anonymous participants or reveal personal or confidential information, for example, when working with industry partners. Prompts can be reported using a structured template that contains placeholders for dynamically added content (see Section *LLMs as Judges* for an example).

Moreover, specifying the exact output format of the prompts is crucial. For example, when using code snippets, researchers should specify whether they were enclosed in markdown-style code blocks (indented by four space characters, enclosed in triple backticks, etc.), if line breaks and whitespace were preserved, and whether additional annotations such as comments were included. Similarly, for other artifacts such as error messages, stack traces, researchers should explain how these were presented.

Researchers MUST also explain in the *paper* how they developed the prompts and why they decided to follow certain prompting strategies. If prompts from the early phases of a research project are unavailable, they MUST at least summarize the prompt evolution. However, given that prompts can be stored as plain text, there are established SE techniques such as version control systems that can be used to collect the required provenance information.

Prompt development is often iterative, involving collaboration between human researchers and AI tools. Researchers SHOULD report any instances in which LLMs were used to suggest prompt refinements, as well as how these suggestions were incorporated. Furthermore, prompts may need to be revised in response to failure cases where the model produced incorrect or incomplete outputs. Pilot testing and prompt evaluation are vital to ensure that prompts yield reliable results. If such testing was conducted, researchers SHOULD summarize key insights, including how different prompt variations affected output quality and which criteria were used to finalize the prompt design. A prompt changelog can help track and report the evolution of prompts throughout a research project, including key revisions, reasons for changes, and versioning (e.g., v1.0: initial prompt; v1.2: added output formatting; v2.0: incorporated examples of ideal responses). Since prompt effectiveness varies between models and model versions, researchers MUST make clear which prompts were used for which models in which versions and with which configuration (see Section *Report Model Version, Configuration, and Customizations*).

Researchers MUST specify whether zero-shot, one-shot, or few-shot prompting was used. For few-shot prompts, the examples provided to the model SHOULD be clearly outlined, along with the rationale for selecting them. If multiple versions of a prompt were tested, researchers SHOULD describe how these variations were evaluated and how the final design was chosen.

When dealing with extensive or complex prompt context, researchers MUST describe the strategies they used to handle input

length constraints. Approaches might include truncating, summarizing, or splitting prompts into multiple parts. Token optimization measures, such as simplifying code formatting or removing unnecessary comments, MUST also be documented if applied.

In cases where prompts are generated dynamically, such as through preprocessing, template structures, or retrieval-augmented generation (RAG), the process MUST be thoroughly documented. This includes explaining any automated algorithms or rules that influenced prompt generation. For studies involving human participants who create or modify prompts, researchers MUST describe how these prompts were collected and analyzed.

To ensure full reproducibility, researchers MUST make all prompts and prompt variations publicly available as part of their *supplementary material*. If the complete set of prompts cannot be included in the *paper*, researchers SHOULD provide summaries and representative examples. This also applies if complete disclosure is not possible due to privacy or confidentiality concerns. For prompts containing sensitive information, researchers MUST: (1) anonymize personal identifiers, (ii) replace proprietary code with placeholders, and (iii) clearly highlight modified sections.

When trying to verify results, even with the exact same prompts, decoding strategies, and parameters, LLMs can still behave non-deterministically. Non-determinism can arise from batching, input preprocessing, and floating point arithmetic on GPUs [20]. Thus, in order to enable other researchers to verify the conclusions that researchers have drawn from LLM interactions, researchers SHOULD report the full interaction logs (prompts and responses) as part of their *supplementary material*. Reporting this is especially important for studies targeting commercial software-as-a-service (SaaS) solutions such as ChatGPT. The rationale for this is similar to the rationale for reporting interview transcripts in qualitative research. In both cases, it is important to document the entire interaction between the interviewer and the participant. Just as a human participant might give different answers to the same question asked two months apart, the responses from ChatGPT can also vary over time (see also Section *LLMs as Subjects*). Therefore, keeping a record of the actual conversation is crucial for accuracy and context and shows depth of engagement for transparency.

5.4.2 **Example(s).** A paper by Anandayuvaraj et al. [4] is a good example of making prompts available online. In that paper, the authors analyze software failures reported in news articles and use prompting to automate tasks such as filtering relevant articles, merging reports, and extracting detailed failure information. Their online appendix contains all the prompts used in the study, providing transparency and supporting reproducibility.

Liang et al.'s paper is a good example of comprehensive prompt reporting. The authors make the exact prompts available in their *supplementary material* on Figshare, including details such as code blocks being enclosed in triple backticks. The *paper* thoroughly explains the rationale behind the prompt design and the data output format. It also includes an overview figure and two concrete examples, enabling transparency and reproducibility while keeping the main text concise. An example of reporting full interaction logs is the study by Ronanki et al. [102], for which they reported the full answers of ChatGPT and uploaded them to Zenodo.

5.4.3 **Advantages.** As motivated above, providing detailed documentation of the prompts improves the verifiability, reproducibility, and comparability of LLM-based studies. It allows other researchers to replicate the study under similar conditions, refine prompts based on documented improvements, and evaluate how different types of content (e.g., source code vs. execution traces) influence LLM behavior. This transparency also enables a better understanding of how formatting, prompt length, and structure impact results across various studies.

An advantage of reporting full interaction logs is that, while for human participants conversations often cannot be reported due to confidentiality, LLM conversations can. Detailed logs enable future reproduction/replication studies to compare results using the same prompts. This could be valuable for tracking changes in LLM responses over time or across different versions of the model. It would also enable secondary research to study how consistent LLM responses are over model versions and identify any variations in its performance for specific SE tasks.

5.4.4 **Challenges.** One challenge is the complexity of prompts that combine multiple components, such as code, error messages, and explanatory text. Formatting differences (e.g., Markdown vs. plain text) can affect how LLMs interpret input. Additionally, prompt length constraints may require careful context management, particularly for tasks involving extensive artifacts such as large codebases. Privacy and confidentiality concerns can hinder prompt sharing, especially when sensitive data is involved.

Not all systems allow the reporting of complete (system) prompts and interaction logs with ease. This hinders transparency and verifiability. In addition to our recommendation to *Use an Open LLM as a Baseline*, we recommend exploring open-source tools such as Continue [32]. For commercial tools, researchers MUST report all available information and acknowledge unknown aspects as limitations (see Section *Report Limitations and Mitigations*). Understanding suggestions of commercial tools such as GitHub Copilot might require recreating the exact state of the codebase at the time the suggestion was made, a challenging context to report. One solution could be to use version control to capture the exact state of the codebase when a recommendation was made, keeping track of the files that were automatically added as context.

5.4.5 **Study Types.** Reporting requirements vary depending on the study type. For the study type *LLMs for New Software Engineering Tools*, researchers MUST explain how prompts were generated and structured within the tool. When *Studying LLM Usage in Software Engineering*, especially for controlled experiments, exact prompts MUST be reported for all conditions. For observational studies, especially the ones targeting commercial tools, researchers MUST report the full interactions logs except for when transcripts might identify anonymous participants or reveal personal or confidential information. As mentioned before, if the complete interaction logs cannot be shared, e.g., because they contain confidential information, the prompts and responses MUST at least be summarized and described in the *paper*.

Researchers MUST report the complete prompts that were used, including all instructions, context, and input data. This applies across all study types, but specific focus areas vary. For *LLMs as Annotators*, researchers MUST document any predefined coding guides

or instructions included in prompts, as these influence how the model labels artifacts. For *LLMs as Judges*, researchers MUST include the evaluation criteria, scales, and examples embedded in the prompt to ensure a consistent interpretation. For *LLMs for Synthesis* tasks (e.g., summarization, aggregation), researchers MUST document the sequence of prompts used to generate and refine outputs, including follow-ups and clarification queries. For *LLMs as Subjects* (e.g., simulating human participants), researchers MUST report any role-playing instructions, constraints, or personas used to guide LLM behavior. For *Benchmarking LLMs for Software Engineering Tasks* studies using pre-defined prompts (e.g., *HumanEval, Crux-Eval*), researchers MUST specify the benchmark version and any modifications made to the prompts or evaluation setup. If prompt tuning, retrieval-augmented generation (RAG), or other methods were used to adapt prompts, researchers MUST disclose and justify those changes, and SHOULD make the relevant code publicly available.

5.5 Use Human Validation for LLM Outputs

> *tl;dr: If assessing the quality of generated artifacts is important and no reference datasets or suitable comparison metrics exist, researchers SHOULD use human validation for LLM outputs. If they do, they MUST define the measured construct (e.g., usability, maintainability) and describe the measurement instrument in the* paper. *Researchers SHOULD consider human validation early in the study design (not as an afterthought), build on established reference models for human-LLM comparison, and share their instruments as* supplementary material. *When aggregating LLM judgments, methods and rationale SHOULD be reported and inter-rater agreement SHOULD be assessed. Confounding factors SHOULD be controlled for, and power analysis SHOULD be conducted to ensure statistical robustness.*

5.5.1 **Recommendations.** Although, in principle, LLMs can automate many tasks in research and software development, which have traditionally been performed by humans, it is often essential to assess the outcome quality to determine whether the automation was successful. If, for a certain task, there are no reference datasets or suitable comparison metrics, researchers SHOULD rely on human judgment to validate the LLM outputs. Human judgment is particularly important if metrics and existing datasets alone cannot fully capture the target qualities relevant in the study context. Integrating human participants in the study design will require additional considerations, including a recruitment strategy, annotation guidelines, training sessions, or ethical approvals. Therefore, researchers SHOULD consider human validation early in the study design, not as an afterthought. In any way, they MUST clearly define the constructs that the human and LLM annotators evaluate [98]. When designing custom instruments to assess LLM output (e.g., questionnaires, scales), researchers SHOULD share their instruments in the *supplementary material*.

Validating the output of an LLM MAY involve the aggregation of inputs from multiple human judges. In these cases, researchers SHOULD clearly describe their aggregation method and document their reasoning. When multiple humans are annotating the same

artifact, researchers SHOULD validate the agreement between multiple validators with inter-rater reliability measures such as *Cohen's Kappa* or *Krippendorff's Alpha*. When human validation is used, additional confounding factors SHOULD be controlled for, e.g., by categorizing participants according to their level of experience or expertise. Where applicable, researchers SHOULD perform a power analysis to estimate the required sample size, ensuring sufficient statistical power in their experimental design.

Researchers SHOULD use established reference models to compare humans with LLMs. For example, the reference model of Schneider et al. [108] provides researchers with an overview of the design considerations for studies comparing LLMs with humans. If studies involve the annotation of software artifacts, and the goal is to automate the annotation process using LLM, researchers SHOULD follow systematic approaches to decide whether and how human annotators can be replaced. For example, Ahmed et al. [1] suggest a method that involves using a jury of three LLMs with 3 to 4 few-shot examples rated by humans, where the model-to-model agreement on all samples is determined using *Krippendorff's Alpha*. If the agreement is high (alpha *gt* 0.5), a human rating can be replaced with an LLM-generated one. In cases of low model-to-model agreement (alpha ≤ 0.5), they then evaluate the prediction confidence of the model, selectively replacing annotations where the model confidence is high (≥ 0.8).

Besides generating and modifying content, agentic software development tools such as Claude Code can autonomously call command-line tools or pull in additional information from MCP servers. For such tools, it makes sense to separate textual output (e.g., file changes) from output related to tool execution. When evaluating agentic tools, researchers SHOULD assess the feedback that users provided for proposed changes, report statistics on how frequently they accepted the content, and how they modified it. This agentic human-in-the-loop interaction approach is essentially a built-in human validation, even though the degrees of freedom are larger than in more traditional experiments that use LLMs directly.

5.5.2 **Example(s).** Ahmed et al. [1] evaluated how human annotations can be replaced by LLM generated labels. In their study, they explicitly report the calculated agreement metrics between the models and between humans. For example, they wrote that "model-model agreement is high, for all criteria, especially for the three large models (GPT-4, Gemini, and Claude). Table I in their study indicates that the mean Krippendorff's α is 0.68-0.76. Second, we see that human-model and human-human agreements are in similar ranges, 0.24-0.40 and 0.21-0.48 for the first three categories."

Xue et al. [137] conducted a controlled experiment in which they evaluated the impact of ChatGPT on the performance and perceptions of students in an introductory programming course. They used multiple measures to judge the impact of LLM from the human point of view. In their study, they recorded students' screens, evaluated their answers for the given tasks, and distributed a post-study survey to collect their opinions.

5.5.3 **Advantages.** For empirical studies involving LLMs, human validation helps ensure the reliability of study results, as LLMs can produce incorrect or biased outputs. For example, in natural language processing tasks, a large-scale study has shown that LLMs

have a significant variation in their results, which limits their reliability as a direct substitute for human judges [15]. Moreover, LLMs do not match human annotation in labeling tasks for natural language inference, position detection, semantic change, and hate speech detection [128].

Incorporating human judgment into the evaluation process adds a layer of quality control and increases the trustworthiness of the study findings, especially when explicitly reporting inter-rater reliability metrics [63]. Incorporating feedback from individuals in the target population strengthens external validity by grounding study findings in real-world usage scenarios, which may positively impact the transfer of study results to practice. Researchers might uncover additional opportunities to further improve the LLM or LLM-based tool based on the reported experiences.

5.5.4 **Challenges.** Measurement through human validation can be challenging. Ensuring that the operationalization of a desired construct and the method of measuring it are appropriate requires a deep understanding of (1) the construct, (2) construct validity in general, and (3) systematic approaches for designing measurement instruments [114]. Human judgment is subjective, which can lead to variability between judgments due to differences in experience, expertise, interpretations, and biases among evaluators [80]. For example, Hicks et al. found that "the struggle with adapting to AI-assisted work is more common for the racial minority developer cohort. That group also rated the quality of the output of AI-assisted coding tools significantly lower than other groups" [49]. Such biases can be mitigated with a careful selection and combination of multiple judges, but cannot be completely controlled for. Recruiting participants as human validators will always incur additional resources compared to machine-generated measures. Researchers must weigh the cost and time investment incurred by the recruitment process against the potential benefits for the validity of their study results.

5.5.5 **Study Types.** For *Studying LLM Usage in Software Engineering*, researchers SHOULD carefully reflect on the validity of their evaluation criteria. In studies where there are arguably objective evaluation criteria (e.g., in *Benchmarking LLMs for Software Engineering Tasks*), there is little need for human validation, since the benchmark has (hopefully) been validated. Where criteria are more unclear (e.g., *LLMs as Annotators*, *LLMs as Subjects*, or *LLMs for Synthesis*), human validation is important, and this guideline should be followed. When using *LLMs as Judges*, it is common to start with humans who co-create intial rating criteria. In developing *LLMs for New Software Engineering Tools*, human validation of the LLM output is critical when the output is supposed to match human expectations.

5.6 Use an Open LLM as a Baseline

> *tl;dr: Researchers* SHOULD *include an open LLM as a baseline when using commercial models and report inter-model agreement. By open LLM, we mean a model that everyone with the required hardware can deploy and operate. Such models are usually "open weight." A full replication package with step-by-step instructions* SHOULD *be provided as part of the* supplementary material.

5.6.1 **Recommendations.** Empirical studies using LLMs in SE, especially those that target commercial tools or models, SHOULD incorporate an open LLM as a baseline and report established metrics for inter-model agreement (see Section *Use Suitable Baselines, Benchmarks, and Metrics*). We acknowledge that including an open LLM baseline might not always be possible, for example, if the study involves human participants, and letting them work on the tasks using two different models might not be feasible. Using an open model as a baseline is also not necessary if the use of the LLM is tangential to the study goal.

Open models allow other researchers to verify research results and build upon them, even without access to commercial models. A comparison of commercial and open models also allows researchers to contextualize model performance. Researchers SHOULD provide a complete replication package as part of their *supplementary material*, including clear step-by-step instructions on how to verify and reproduce the results reported in the paper.

Open LLMs are available on platforms such as *Hugging Face*. Depending on their size and the available compute power, open LLMs can be hosted on a local computer or server using frameworks such as *Ollama* or *LM Studio*. They can also be run on cloud-based services such as *Together AI* or on large hyperscalers such as AWS, Azure, Alibaba Cloud, and Google Cloud.

The term "open" can have different meanings in the context of LLMs. Widder et al. discuss three types of openness: transparency, reusability, and extensibility [129]. They also discuss what openness in AI can and cannot provide. Moreover, the *Open Source Initiative* (OSI) [88] provides a definition of open-source AI that serves as a useful framework for evaluating the openness of AI models. In simple terms, according to OSI, open-source AI means that one has access to everything needed to use the AI, which includes that it is possible to understand, modify, share, retrain, and recreate it.

Besides open models, researchers MAY explore whether their study goal can be achieved with existing open-source tools. Tools such as Continue, Cline, and opencode offer open alternatives to commercial tools such as GitHub Copilot and Claude Code. Using open tools instead of their closed-source counterparts allows researchers to instrument the tools, understand their architecture (in particular how exactly they leverage LLMs), and collect detailed telemetry data.

5.6.2 **Example(s).** Numerous studies have adopted open LLMs as baseline models. For example, Wang et al. evaluated seven advanced LLMs, six of which were open-source, testing 145 API mappings drawn from eight popular Python libraries across 28,125 completion prompts aimed at detecting deprecated API usage in code completion [125]. Moumoula et al. compared four LLMs on a cross-language code clone detection task [85]. Three evaluated models were open-source. Gonçalves et al. fine-tuned the open LLM *LLaMA* 3.2 on a refined version of the *DiverseVul* dataset to benchmark vulnerability detection performance [43]. Many papers have used CodeBERT as an LLM, which is a bimodal (code + NL) Transformer pre-trained by Microsoft Research and released under the MIT license [17, 115, 133, 139]. Its model weights, source code, and data-processing scripts are all openly published on GitHub [81].

5.6.3 **Advantages.** Using a true open LLM as a baseline improves the reproducibility of scientific research by providing full access

to model architectures, training data, and parameter settings (see Section *Report Model Version, Configuration, and Customizations*), thereby allowing independent reconstruction and verification of experimental results. Moreover, by adopting an open-source baseline, researchers can directly compare novel methods against established performance metrics without the variability introduced by proprietary systems (see also Section *Use Suitable Baselines, Benchmarks, and Metrics*). The transparent nature of these models allows for detailed inspection of data processing pipelines and decision-making routines, which is essential for identifying potential sources of bias and delineating model limitations (see Section *Report Limitations and Mitigations*). Furthermore, unlike closed-source alternatives, which can be withdrawn or altered without notice, open LLMs ensure long-term accessibility and stability, preserving critical resources for future studies. Finally, the licensing requirements associated with open-source implementations lower financial barriers, making advanced language models attainable for research groups operating under constrained budgets.

5.6.4 **Challenges.** Open-source LLMs face several notable challenges (see also Section *Report Limitations and Mitigations*). First, they often lag behind the most advanced proprietary systems in common benchmarks, making it difficult for researchers to demonstrate clear improvements when evaluating new methods using open LLMs alone (see Section *Use Suitable Baselines, Benchmarks, and Metrics*). Additionally, deploying and experimenting with these models typically requires substantial hardware resources, in particular high-performance GPUs, which may be beyond reach for many academic groups (see Section *Report Tool Architecture beyond Models*).

The notion of "openness" itself remains in flux: although numerous models are described as open, many release only the trained weights without disclosing the underlying training data or methodological details (see Section *Report Model Version, Configuration, and Customizations*), a practice sometimes referred to as "open weight" openness [41]. To address this gap, in our recommendations, we have referenced the open-source AI definition proposed by the OSI as an initial framework for what constitutes genuinely open-source AI software [88].

Finally, unlike managed cloud APIs provided by proprietary vendors, installing, configuring, and fine-tuning open-source models can be technically demanding. Documentation is often sparse or fragmented, placing a high barrier to entry for researchers without specialized engineering support (see also Section *Report Limitations and Mitigations*).

5.6.5 **Study Types.** When evaluating *LLMs for New Software Engineering Tools*, researchers SHOULD use an open LLM as a baseline whenever it is technically feasible; if integration proves too complex, they SHOULD report the initial benchmarking results of open models (see Section *Use Suitable Baselines, Benchmarks, and Metrics*). In formal benchmarking studies and controlled experiments (see Section *Benchmarking LLMs for Software Engineering Tasks*), an open LLM MUST be one of the models under evaluation. For observational studies in which using an open LLM is impossible, investigators SHOULD explicitly acknowledge its absence and discuss how this limitation might affect their conclusions (see Section *Report Limitations and Mitigations*). Finally, when using *LLMs for*

Synthesis, where an LLM serves to explore qualitative data or contrast alternative interpretations, researchers MAY report results of an open LLM.

5.7 Use Suitable Baselines, Benchmarks, and Metrics

tl;dr: Researchers MUST justify all benchmark and metric choices in the paper and SHOULD summarize benchmark structure, task types, and limitations. Where possible, traditional (non-LLM) baselines SHOULD be used for comparison. Researchers MUST explain why the selected metrics are suitable for the specific study. They SHOULD report established metrics to make study results comparable, but can report additional metrics that they consider appropriate. Due to the inherent non-determinism of LLMs, experiments SHOULD be repeated; the result distribution SHOULD then be reported using descriptive statistics.

5.7.1 **Recommendations.** Benchmarks, baselines, and metrics play an important role in assessing the effectiveness of LLMs or LLM-based tools. Benchmarks are model- and tool-independent standardized tests used to assess the performance of LLMs on specific tasks such as code summarization or code generation. A benchmark consists of multiple standardized test cases, each with at least one task and a corresponding expected result. Metrics are used to quantify performance on benchmark tasks, enabling a comparison. A baseline represents a reference point for the measured LLM. Since LLMs require substantial hardware resources, baselines serve as a comparison to assess their performance against traditional algorithms with lower computational costs.

When selecting benchmarks, it is important to fully understand the benchmark tasks and the expected results because they determine what the benchmark actually assesses. Researchers MUST briefly summarize why they selected certain benchmarks in the *paper*, They SHOULD also summarize the structure and tasks of the selected benchmark(s), including the programming language(s) and descriptive statistics such as the number of contained tasks and test cases. Researchers SHOULD also discuss the limitations of the selected benchmark(s). For example, many benchmarks focus heavily on isolated Python functions. This assesses a very specific part of software development, which is certainly not representative of the full breadth of software engineering work [19].

Researchers MAY include an example of a task and the corresponding test case(s) to illustrate the structure of the benchmark. If multiple benchmarks exist for the same task, researchers SHOULD compare performance between benchmarks. When selecting only a subset of all available benchmarks, researchers SHOULD use the most specific benchmarks given the context.

The use of LLMs might not always be reasonable if traditional approaches achieve similar performance. For many tasks for which LLMs are being evaluated, there exist traditional non-LLM-based approaches (e.g., for program repair) that can serve as a baseline. Even if LLM-based tools perform better, the question is whether the resources consumed justify the potentially marginal improvements. Researchers SHOULD always check whether such traditional baselines exist and, if they do, compare them with the LLM or LLM-based tool using suitable metrics.

To compare traditional and LLM-based approaches or different LLM-based tools, researchers SHOULD report established metrics whenever possible, as this allows secondary research. They can, of course, report additional metrics that they consider appropriate. In any event, researchers MUST argue why the selected metrics are suitable for the given task or study.

If a study evaluates an LLM-based tool that is supposed to support humans, a relevant metric is the acceptance rate, meaning the ratio of all accepted artifacts (e.g., test cases, code snippets) in relation to all artifacts that were generated and presented to the user. Another way of evaluating LLM-based tools is calculating inter-model agreement (see also Section Use an Open LLM as a Baseline). This allows researchers to assess how dependent a tool's performance is on specific models and versions. Metrics used to measure inter-model agreements include general agreement (percentage), *Cohen's kappa*, and *Scott's Pi coefficient*.

LLM-based generation is non-deterministic by design. Due to this non-determinism, researchers SHOULD repeat experiments to statistically assess the performance of a model or tool, e.g., using the arithmetic mean, confidence intervals, and standard deviations.

From a measurement perspective, researchers SHOULD reflect on the theories, values, and measurement models on which the benchmarks and metrics they have selected for their study are based. For example, a large open dataset of software bugs is a collection of bugs according to a certain theory of what constitutes a bug, and the values and perspective of the people who labeled the dataset. Reflecting on the context in which these labels were assigned and discussing whether and how the labels generalize to a new study context is crucial.

5.7.2 **Example(s).** According to Hou et al., main problem types for LLMs are classification, recommendation and generation problems. Each of these problem types requires a different set of metrics. Hu et al. conducted a comprehensive structured literature review on LLM benchmarks related to software engineering tasks. They analyzed 191 benchmarks and categorized them according to the specific task they address, making the paper a valuable resource for identifying existing metrics. Metrics used to assess generation tasks include *BLEU*, *pass@k*, *Accuracy*, *Accuracy@k*, and *Exact Match*. The most common metric for recommendation tasks is *Mean Reciprocal Rank*. For classification tasks, classical machine learning metrics such as *Precision*, *Recall*, *F1-score*, and *Accuracy* are often reported.

In the following, we briefly discuss two common metrics used for generation tasks: *BLEU-N* and *pass@k*. *BLEU-N* [94] is a similarity score based on n-gram precision between two strings, ranging from 0 to 1. Values close to 0 represent dissimilar content, values closer to 1 represent similar content, indicating that a model is more capable of generating the expected output. *BLEU-N* has multiple variations. *CodeBLEU* [100] and *CrystalBLEU* [33] are the most notable ones tailored to code. They introduce additional heuristics such as AST matching. As mentioned, researchers MUST motivate why they chose a certain metric or variant thereof for their particular study.

The metric *pass@k* reports the likelihood that a model correctly completes a code snippet at least once within k attempts. To our knowledge, the basic concept of *pass@k* was first reported by Kulal et al. to evaluate code synthesis [65]. They named the concept *success rate at B*, where B denotes the trial "budget." The term

pass@k was later popularized by Chen et al. as a metric for code generation correctness [22]. The exact definition of correctness varies depending on the task. For code generation, correctness is often defined based on test cases: A passing test implies that the solution is correct. The resulting pass rates range from 0 to 1. A pass rate of 0 indicates that the model was unable to generate a single correct solution within k tries; a rate of 1 indicates that the model successfully generated at least one correct solution in k tries.

The *pass@k* metric is defined as: $1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}$, where n is the total number of generated samples per prompt, c is the number of correct samples among n, and k is the number of samples.

Choosing an appropriate value for k depends on the downstream task of the model and how end-users interact with it. A high pass rate for *pass@1* is highly desirable in tasks where the system presents only one solution or if a single solution requires high computational effort. For example, code completion depends on a single prediction, since end users typically see only a single suggestion. Pass rates for higher k values (e.g., 2, 5, 10) indicate how well a model can solve a given task in multiple attempts. The *pass@k* metric is frequently referenced in papers on code generation [46, 54, 71, 103]. However, *pass@k* is not a universal metric suitable for all tasks. For instance, for creative or open-ended tasks, such as comment generation, it is not a suitable metric since more than one correct result exists.

Two benchmarks used for code generation are *HumanEval* (available on GitHub) [94] and *MBPP* (available on Hugging Face) [8]. Both benchmarks consist of code snippets written in Python sourced from publicly available repositories. Each snippet consists of four parts: a prompt containing a function definition and a corresponding description of what the function should accomplish, a canonical solution, an entry point for execution, and test cases. The input of the LLM is the entire prompt. The output of the LLM is then evaluated against the canonical solution using metrics or against a test suite. Other benchmarks for code generation include *ClassEval* (available on GitHub) [31], *LiveCodeBench* (available on GitHub) [56], and *SWE-bench* (available on GitHub) [59]. An example of a code translation benchmark is *TransCoder* [66] (available on GitHub).

5.7.3 **Advantages.** Benchmarks are an essential tool for assessing model performance for SE tasks. Reproducible benchmarks measure and assess the performance of models for a specific SE tasks, enabling comparison. That comparison enables progress tracking, for example, when researchers iteratively improve a new LLM-based tool and test it against benchmarks after significant changes. For practitioners, leaderboards, i.e., published benchmark results of models, support the selection of models for downstream tasks.

5.7.4 **Challenges.** A general challenge with benchmarks for LLMs is that the most prominent ones, such as *HumanEval* and *MBPP*, use Python, introducing a bias toward this specific programming language and its idiosyncrasies. Since model performance is measured against these benchmarks, researchers often optimize for them. As a result, performance may degrade if programming languages other than Python are used.

Many closed-source models, such as those released by *OpenAI*, achieve exceptional performance on certain tasks but lack

transparency and reproducibility [31, 70, 145]. Benchmark leaderboards, particularly for code generation, are led by closed-source models [31, 145]. While researchers SHOULD compare performance against these models, they must consider that providers might discontinue them or apply undisclosed pre- or post-processing beyond the researchers' control (see *Use an Open LLM as a Baseline*).

The challenges of individual metrics include that, for example, *BLEU-N* is a syntactic metric and therefore does not measure semantic or structural correctness. Thus, a high *BLEU-N* score does not directly indicate that the generated code is executable. While alternatives exist, they often come with their own limitations. For example, *Exact Match* is a strict measurement that does not account for functional equivalence of syntactically different code. Execution-based metrics such as *pass@k* directly evaluate correctness by running test cases, but they require a setup with an execution environment. When researchers observe unexpected values for certain metrics, the specific results should be investigated in more detail to uncover potential problems. These problems can be related to formatting, since code formatting is known to influence metrics such as *BLEU-N* or *Exact Match.*

Another challenge to consider is that metrics usually capture one specific aspect of a task or solution. For example, metrics such as *pass@k* do not reflect qualitative aspects of the generated source code, including its maintainability or readability. However, these aspects are critical for many downstream tasks. Moreover, benchmarks are isolated test sets and may not fully represent real-world applications. For example, benchmarks such as *HumanEval* synthesize code based on written specifications. However, such explicit descriptions are rare in real-world applications. Thus, evaluating model performance with benchmarks might not reflect real-world tasks and end-user performance.

Finally, benchmark data contamination [135] continues to be a major challenge. In many cases, the LLM training data is not released. However, the benchmark itself could be part of the training data. Such benchmark contamination may lead to the model remembering the solution from the training data rather than solving the new task based on the input data. This leads to artificially high performance on benchmarks. However, for unforeseen scenarios, the model might perform much worse.

5.7.5 **Study Types.** This guideline MUST be followed for all study types that automatically evaluate the performance of LLMs or LLM-based tools. The design of a benchmark and the selection of appropriate metrics are highly dependent on the specific study type and research goal. Recommending specific metrics for specific study types is beyond the scope of these guidelines, but Hu et al. provide a good overview of existing metrics for evaluating LLMs [51]. In addition, our Section *Use Human Validation for LLM Outputs* provides an overview of the integration of human evaluation in LLM-based studies. For *LLMs as Annotators*, the research goal might be to assess which model comes close to a ground truth dataset created by human annotators. Especially for open annotation tasks, selecting suitable metrics to compare LLM-generated and human-generated labels is important. In general, annotation tasks can vary significantly. Are multiple labels allowed for the same sequence? Are the available labels predefined, or should the LLM generate a set of labels independently? Due to this task dependence, researchers

MUST justify their metric choice, explaining what aspects of the task it captures together with known limitations. If researchers assess a well-established task such as code generation, they SHOULD report standard metrics such as *pass@k* and compare the performance between models. If non-standard metrics are used, researchers MUST state their reasoning.

5.8 Report Limitations and Mitigations

tl;dr: Researchers MUST transparently report study limitations, including the impact of non-determinism and generalizability constraints. The paper MUST specify whether generalization across LLMs or across time was assessed, and discuss model and version differences. Authors MUST describe measurement constructs and methods, disclose any data leakage risks, and avoid leaking evaluation data into LLM improvement pipelines. They MUST provide model outputs, discuss sensitive data handling, ethics approvals, and justify LLM usage in light of its resource demands.

When using LLMs for empirical studies in SE, researchers face unique challenges and potential limitations that can influence the validity, reliability, and reproducibility of their findings [105]. It is important to openly discuss these limitations and explain how their impact was mitigated. All this is relative to the current capabilities of LLMs and the current state of the art in terms of tool architectures. If and how the performance of LLMs in a particular study context will improve with future model generations or new architectural patterns is beyond what researchers can and should discuss as part of a paper's limitation section. Nevertheless, risk management and threat mitigation are important tasks during the design of an empirical study. They should not happen as an afterthought.

5.8.1 **Recommendations.** A cornerstone of open science is the ability to reproduce research results. Although the inherent non-deterministic nature of LLM is a strength in many use cases, its impact on *reproducibility* is problematic. To enable reproducibility, researchers SHOULD disclose a replication package for their study. They SHOULD perform multiple repetitions of their experiments (see *Use Suitable Baselines, Benchmarks, and Metrics*) to account for non-deterministic outputs. In some cases, researchers MAY reduce the output variability by setting the temperature to a value close to 0 and setting a fixed seed value. However, configuring a lower temperature can negatively impact task performance, and not all models allow the configuration of seed values. Besides the inherent non-determinism, the behavior of an LLM depends on many external factors such as prompt variations and model evolution. To ensure reproducibility, researchers SHOULD follow all previous guidelines and further discuss the generalization challenges outlined below.

Although the topic of *generalizability* is not new, it has gained new relevance with increasing interest in LLMs. In LLM-based studies, generalizability boils down to two main concerns: (1) First, are the results specific to an LLM, or can they be achieved with other LLMs as well? (2) Second, will these results still be valid in the future? If generalizability to other LLMs is not in the scope of the research, this MUST be clearly explained in the *paper* or, if generalizability is in scope, researchers MUST compare their results

or subsets of the results (e.g., due to computational cost) with other LLMs that are, e.g., similar in size, to assess the generalizability of their findings (see also Section *Use an Open LLM as a Baseline*). Multiple studies (e.g., [21, 69]) found that the performance of proprietary LLMs (e.g., GPT) decreased over time for certain tasks using the same model version (e.g. GPT-4o). Reporting the model version and configuration is not sufficient in such cases. To date, the only way to mitigate this limitation is the usage of an open LLM having a versioning schema and archiving (see *Use an Open LLM as a Baseline*). Hence, researchers SHOULD employ open LLMs to establish a reproducible baseline and, if the use of an open LLM is not possible, researchers SHOULD test and report their results over an extended period of time as a proxy of the stability of the results over time.

Data leakage, contamination, or overfitting occurs when information outside the training data influences model performance. With the growing reliance on big datasets, the risks of inter-dataset duplication increases (see, e.g., [3, 74]). In the context of LLMs for SE, this can manifest itself, for example, as training data samples that appear in the fine-tuning or evaluation datasets, potentially compromising the validity of the evaluation results [75]. Moreover, ChatGPT's functionality to "improve the model for everyone" can result in unintentional data leakage. Hence, to ensure the validity of the evaluation results, researchers SHOULD carefully curate the fine-tuning and evaluation datasets to prevent inter-dataset duplication and MUST NOT leak their fine-tuned or evaluation datasets into LLM improvement processes. When publishing the results, researchers SHOULD of course still follow open science practices and publish the datasets as part of their *supplementary material*. If information about the training data of the employed LLM is available, researchers SHOULD evaluate the inter-dataset duplication and MUST discuss potential data leakage in the *paper*. When training an LLM from scratch, researchers MAY consider using open datasets such as *RedPajama (Together AI)* [25], which are already built with deduplication in mind (with the positive side effect of potentially improving performance [68]).

Conducting studies with LLMs is a resource-intensive process. For self-hosted LLMs, the respective hardware must be provided, and for managed LLMs, the service *costs* must be considered. The challenge becomes more pronounced as LLMs grow larger, research architectures become more complex, and experiments become more computationally expensive. For example, multiple repetitions to assess model or tool performance (see *Use Suitable Baselines, Benchmarks, and Metrics*) multiply the cost and impact *scalability*. Consequently, resource-intensive research remains predominantly the domain of private companies or well-funded research institutions, hindering researchers with limited resources in reproducing, replicating, or extending study results. Hence, for transparency reasons, researchers SHOULD report the cost associated with executing an LLM-based study. If the study used self-hosted LLMs, researchers SHOULD report the specific hardware used. If the study used managed LLMs, the service cost SHOULD be reported. To ensure the validity and reproducibility of the research results, researchers MUST provide the LLM outputs as evidence (see Section *Report Prompts, their Development, and Interaction Logs*). Depending on the architecture (see Section *Report Tool Architecture beyond Models*), these outputs SHOULD be reported on different architectural

levels (e.g., outputs of individual LLMs in multi-agent systems). Additionally, researchers SHOULD include a subset of the employed validation dataset, selected using an accepted sampling strategy, to allow partial replication of the results.

Sensitive data can range from personal to proprietary data, each with its own set of *ethical concerns*. As mentioned above, a big threat to proprietary LLMs and sensitive data is its use for model improvement. Hence, using sensitive data can lead to privacy and intellectual property (IP) violations. Another threat is the implicit bias of LLMs that could lead to discrimination or unfair treatment of individuals or groups. To mitigate these concerns, researchers SHOULD minimize the sensitive data used in their studies, MUST follow applicable regulations (e.g., GDPR) and individual processing agreements, SHOULD create a data management plan outlining how the data is handled and protected against leakage and discrimination, and MUST apply for approval from the ethics committee of their organization (if required).

Although *metrics* such as *BLEU* or *ROUGE* are commonly used to evaluate the performance of LLMs (see Section *Use Suitable Baselines, Benchmarks, and Metrics*), they may not capture other relevant SE-specific aspects such as functional correctness or runtime performance of automatically generated source code [73]. Given the high resource demand of LLMs, in addition to traditional metrics such as accuracy, precision, and recall or more contemporary metrics such as *pass@k*, *resource consumption* has to become a key indicator of performance. While research has predominantly focused on energy consumption during the early phases of building LLMs (e.g., data center manufacturing, data acquisition, training), inference, that is LLM usage, often becomes similarly or even more resource-intensive [29, 36, 58, 83, 132]. Hence, researchers SHOULD aim for lower resource consumption on the model side. This can be achieved by selecting smaller (e.g., GPT-4o-mini instead of GPT-4o) or newer models or by employing techniques such as model pruning, quantization, or knowledge distillation [83]. Researchers MAY further reduce resource consumption by restricting the number of queries, input tokens, or output tokens [83], by using different prompt engineering techniques (e.g., zero-shot prompts seem to emit less CO2 than chain-of-thought prompts), or by carefully sampling smaller datasets for fine-tuning and evaluation instead of using large datasets. Reducing resource consumption involves a trade-off with our recommendation to perform multiple experimental runs to account for non-determinism, as suggested in *Use Suitable Baselines, Benchmarks, and Metrics*. To report the environmental impact of a study, researchers SHOULD use software such as CodeCarbon or Experiment Impact Tracker to track and quantify the carbon footprint of the study or report an estimate of the carbon footprint through tools such as MLCO2 Impact. They SHOULD detail the LLM version and configuration as described in *Report Model Version, Configuration, and Customizations*, state the hardware or hosting provider of the model as described in *Report Tool Architecture beyond Models* and report the total number of requests, accumulated input and output tokens. Researchers MUST justify why LLMs were chosen over existing approaches and discuss the achieved performance in relation to their potentially higher resource consumption.

5.8.2 **Example(s).** An example highlighting the need for caution around *replicability* is the study of Staudinger et al. [116] who

attempted to replicate an LLM study. The authors were unable to reproduce the exact results, even though they saw similar trends as in the original study.

To analyze whether the results of proprietary LLMs transfer to open LLMs, Staudinger et al. benchmarked previous results using GPT-3.5 and GPT4 against Mistral and Zephyr [116]. They found that open-source models could not deliver the same performance as proprietary models. This paper is also an example of a study reporting *costs*: "120 USD in API calls for GPT 3.5 and GPT 4, and 30 USD in API calls for Mistral AI. Thus, the total LLM cost of our reproducibility study was 150 USD". Tinnes et al. is an example of balancing dataset size between the need for manual semantic analysis and computational resource consumption [121].

Individual studies have already begun to highlight the uncertainty about the *generalizability* of their results in the future. Jesse et al. acknowledge the issue that LLMs evolve over time and that this evolution could impact the study results [57]. Since a lot of research in SE evolves around code, inter-dataset code duplication has been extensively researched over the years to curate de-duplicated datasets (see, e.g., [3, 61, 74, 75]). The issue of inter-dataset duplication has also attracted interest in other disciplines. For example, in biology, Lakiotaki et al. acknowledge and address the overlap between multiple common disease datasets [67]. In the domain of code generation, Coignion et al. evaluated the performance of LLMs to produce LeetCode [24]. To mitigate the issue of inter-dataset duplication, they only used LeetCode problems published after 2023-01-01, reducing the likelihood of LLMs having seen those problems before. Further, they discuss the performance differences of LLMs on different datasets in light of potential inter-dataset duplication. Zhou et al. performed an empirical evaluation of data leakage in 83 software engineering benchmarks [143]. Although most benchmarks suffer from minimal leakage, very few showed a leakage of up to 100%. The authors found a high impact of data leakage on the performance evaluation. A starting point for studies that aim to assess and mitigate inter-dataset duplication are the Falcon LLMs, for which parts of its training data are available on Hugging Face [120]. Through this dataset, it is possible to reduce the overlap between the traning and evaluation data, improving the validity of the evaluation results. A starting point to prevent active data leakage into a LLM improvement process is to ensure that the data is not used to train the model (e.g., via OpenAI's data control functionality) [10].

Finally, bias can occur in LLM training datasets, resulting in various types of discrimination. Gallegos et al. propose metrics to quantify biases in various tasks (e.g., text generation, classification, question answering) [38].

5.8.3 **Advantages.** The *reproduction* or *replication* of study results under similar conditions by different parties greatly increases the validity of the results. Independent verification is of particular importance for studies involving LLMs, due to the non-determinism of their outputs and the potential for biases in their training, fine-tuning, and evaluation datasets. Mitigating threats to *generalizability* of a study through the integration of an open LLM as a baseline or the reporting of results over an extended period of time can increase the validity, reliability, and replicability of a study's results. Assessing and mitigating the effects of *inter-dataset duplication*

strengthens a study's validity and reliability, as it prevents overly optimistic performance estimates that do not apply to previously unknown samples. Reporting the *costs* of performing a study not only increases transparency but also provides context for secondary literature. Providing *replication packages* with LLM output and data samples for partial replicability are paramount steps toward open and inclusive research in light of resource inequality between researchers. Considering and justifying the usage of LLMs over other approaches can lead to more efficient and sustainable solutions for SE problems. Reporting the *environmental impact* of LLM usage also sets the stage for more sustainable research practices in SE.

5.8.4 **Challenges.** With commercial LLMs evolving over time, the *generalizability* of results to future model versions is uncertain. Employing open LLMs as a baseline can mitigate this limitation, but may not always be feasible due to computational cost. Most LLM providers do not publicly offer information about their training data, impeding the assessment of *inter-dataset duplication* effects. Consistently keeping track of and reporting the *costs* involved in a research endeavor is challenging. Building a coherent *replication package* that includes LLM outputs and samples for partial replicability requires additional effort and resources. Finding the right *metrics* to evaluate the performance of LLMs in SE for specific tasks is challenging. Defining all requirements beforehand to ensure the use of suitable metrics can be a challenge, especially in exploratory research. Our Section *Use Suitable Baselines, Benchmarks, and Metrics* and its references can serve as a starting point. Ensuring *compliance* across jurisdictions is difficult with different regions having different regulations and requirements (e.g., GDPR and the AI Act in the EU, CCPA in California). Selecting datasets and models with fewer *biases* is challenging, as biases are often unknown. Measuring or estimating the *environmental impact* of a study is challenging and might not always be feasible. Especially in exploratory research, the impact is hard to estimate in advance, making it difficult to justify the usage of LLMs over other approaches.

5.8.5 **Study Types.** Limitations and mitigations SHOULD be discussed for all study types in relation to the context of the individual study. This section provides a starting point and lists aspects along which researchers can reflect on the limitations of their study design and potential mitigations that exist.

6 Conclusion

This paper organizes the landscape of empirical studies involving LLMs in software engineering along seven study types and distills eight guidelines aimed at improving their validity, transparency, and reproducibility. Our guidelines recommend that researchers (1) explicitly declare when and how LLMs are used; (2) report model versions, configuration, and fine-tuning details; (3) describe the complete tool architecture beyond the model; (4) release prompts, their development, and, where possible, interaction logs; (5) validate LLM outputs with humans; (6) include an open LLM as a baseline; (7) select appropriate baselines, benchmarks, and metrics; and (8) report study limitations and mitigations, including costs, potential biases, and environmental impact. Adopting these practices will not eliminate LLM non-determinism, but it will make claims auditable, and results easier to replicate and compare across studies.

By consistently applying our recommendations, the community can produce more reliable evidence on what LLM can and cannot deliver for software engineering research and practice. Our guidelines complement existing empirical software engineering guidance while addressing LLM-specific challenges. Because models and tools evolve, we maintain the guidelines as a living resource and invite the community to refine and extend them: llm-guidelines.org.

Acknowledgments

References

[1] Toufique Ahmed, Premkumar T. Devanbu, Christoph Treude, and Michael Pradel. 2025. Can LLMs Replace Manual Annotation of Software Engineering Artifacts?. In *22nd IEEE/ACM International Conference on Mining Software Repositories, MSR@ICSE 2025, Ottawa, ON, Canada, April 28-29, 2025*. IEEE, 526–538. https://doi.org/10.1109/MSR66628.2025.00086

[2] Sanchit Ahuja, Varun Gumma, and Sunayana Sitaram. 2024. Contamination Report for Multilingual Benchmarks. *CoRR* abs/2410.16186 (2024). https://doi.org/10.48550/ARXIV.2410.16186 arXiv:2410.16186

[3] Miltiadis Allamanis. 2019. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2019*, Hidehiko Masuhara and Tomas Petricek (Eds.). ACM, 143–153. https://doi.org/10.1145/3359591.3359735

[4] Dharun Anandayuvaraj, Matthew Campbell, Arav Tewari, and James C Davis. 2024. FAIL: Analyzing Software Failures from the News Using LLMs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 506–518.

[5] Lisa P. Argyle, Ethan C. Busby, Nancy Fulda, Joshua Gubler, Christopher Michael Rytting, and David Wingate. 2022. Out of One, Many: Using Language Models to Simulate Human Samples. *CoRR* abs/2209.06899 (2022). https://doi.org/10.48550/ARXIV.2209.06899 arXiv:2209.06899

[6] American Psychological Association. 2018. APA Dictionary of Psychology: subject. https://dictionary.apa.org/subject. Accessed 2025-08-15.

[7] Association for Computing Machinery. 2023. ACM Policy on Authorship. https://www.acm.org/publications/policies/new-acm-policy-on-authorship. Accessed 2025-01-13.

[8] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *CoRR* abs/2108.07732 (2021). arXiv:2108.07732 https://arxiv.org/abs/2108.07732

[9] Maider Azanza, Juanan Pereira, Arantza Irastorza, and Aritz Galdos. 2024. Can LLMs Facilitate Onboarding Software Developers? An Ongoing Industrial Case Study. In *36th International Conference on Software Engineering Education and Training, CSEE&T 2024*. IEEE, 1–6. https://doi.org/10.1109/CSEET62301.2024.10662989

[10] Simone Balloccu, Patrícia Schmidtová, Mateusz Lango, and Ondrej Dusek. 2024. Leak, Cheat, Repeat: Data Contamination and Evaluation Malpractices in Closed-Source LLMs. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2024 - Volume 1: Long Papers, St. Julian's, Malta, March 17-22, 2024*, Yvette Graham and Matthew Purver (Eds.). Association for Computational Linguistics, 67–93. https://aclanthology.org/2024.eacl-long.5

[11] Muneera Bano, Hashini Gunatilake, and Rashina Hoda. 2025. What Does a Software Engineer Look Like? Exploring Societal Stereotypes in LLMs. *arXiv* (2025). arXiv:2501.03569 [cs.SE] https://arxiv.org/abs/2501.03569

[12] Muneera Bano, Rashina Hoda, Didar Zowghi, and Christoph Treude. 2024. Large language models for qualitative research in software engineering: exploring opportunities and challenges. *Autom. Softw. Eng.* 31, 1 (2024), 8. https://doi.org/10.1007/S10515-023-00407-8

[13] Muneera Bano, Didar Zowghi, and Jon Whittle. 2023. Exploring Qualitative Research Using LLMs. *arXiv* (2023). arXiv:2306.13298 [cs.SE] https://arxiv.org/abs/2306.13298

[14] Cauã Ferreira Barros, Bruna Borges Azevedo, Valdemar Vicente Graciano Neto, Mohamad Kassab, Marcos Kalinowski, Hugo Alexandre D. do Nascimento, and Michelle C. G. S. P. Bandeira. 2024. Large Language Model for Qualitative Research – A Systematic Mapping Study. *arXiv* (2024). arXiv:2411.14473 [cs.CL] https://arxiv.org/abs/2411.14473

[15] Anna Bavaresco, Raffaella Bernardi, Leonardo Bertolazzi, Desmond Elliott, Raquel Fernández, Albert Gatt, Esam Ghaleb, Mario Giulianelli, Michael Hanna, Alexander Koller, André F. T. Martins, Philipp Mondorf, Vera Neplenbroek, Sandro Pezzelle, Barbara Plank, David Schlangen, Alessandro Suglia, Aditya K. Surikuchi, Ece Takmaz, and Alberto Testoni. 2024. LLMs instead of Human Judges? A Large Scale Empirical Study across 20 NLP Evaluation Tasks. *CoRR* abs/2406.18403 (2024). https://doi.org/10.48550/ARXIV.2406.18403 arXiv:2406.18403

[16] Courtni Byun, Piper Vasicek, and Kevin D. Seppi. 2023. Dispensing with Humans in Human-Computer Interaction Research. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems, CHI EA 2023*, Albrecht Schmidt, Kaisa Väänänen, Tesh Goyal, Per Ola Kristensson, and Anicia Peters (Eds.). ACM, 413:1–413:26. https://doi.org/10.1145/3544549.3582749

[17] Yuchen Cai, Aashish Yadavally, Abhishek Mishra, Genesis Montejo, and Tien N. Nguyen. 2024. Programming Assistant for Exception Handling with CodeBERT. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 94:1–94:13. https://doi.org/10.1145/3597503.3639188

[18] Jialun Cao, Yuk-Kit Chan, Zixuan Ling, Wenxuan Wang, Shuqing Li, Mingwei Liu, Chaozheng Wang, Boxi Yu, Pinjia He, Shuai Wang, et al. 2025. How Should I Build A Benchmark? *arXiv preprint arXiv:2501.10711* (2025).

[19] Satish Chandra. 2025. Benchmarks for AI in Software Engineering (BLOG@CACM). https://cacm.acm.org/blogcacm/benchmarks-for-ai-in-software-engineering/. Accessed 2025-08-24.

[20] Sherman Chann. 2023. Non-determinism in GPT-4 is caused by Sparse MoE. https://152334h.github.io/blog/non-determinism-in-gpt-4/. Accessed 2025-01-13.

[21] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. How is ChatGPT's behavior changing over time? *CoRR* abs/2307.09009 (2023). https://doi.org/10.48550/ARXIV.2307.09009 arXiv:2307.09009

[22] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[23] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (April 1960), 37–46. https://doi.org/10.1177/001316446002000104

[24] Tristan Coignion, Clément Quinton, and Romain Rouvoy. 2024. A Performance Study of LLM-Generated Code on Leetcode. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering, EASE 2024, Salerno, Italy, June 18-21, 2024*. ACM, 79–89. https://doi.org/10.1145/3661167.3661221

[25] Together Computer. 2023. RedPajama: an Open Dataset for Training Large Language Models. https://github.com/togethercomputer/RedPajama-Data.

[26] Rachel Crowell. 2023. Why AI's diversity crisis matters, and how to tackle it. *Nature Career Feature* (2023). https://doi.org/10.1038/d41586-023-01689-4

[27] Matheus de Morais Leça, Lucas Valença, Reydne Santos, and Ronnie de Souza Santos. 2024. Applications and Implications of Large Language Models in Qualitative Analysis: A New Frontier for Empirical Software Engineering. *arXiv* (2024). arXiv:2412.06564 [cs.SE] https://arxiv.org/abs/2412.06564

[28] Stefano De Paoli. 2024. Performing an inductive thematic analysis of semistructured interviews with a large language model: An exploration and provocation on the limits of the approach. *Social Science Computer Review* 42, 4 (2024), 997–1019.

[29] Alex de Vries. 2023. The growing energy footprint of artificial intelligence. *Joule* 7, 10 (2023), 2191–2194.

[30] Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. 2024. Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical Study. In *21st IEEE International Conference on Software Architecture, ICSA 2024, Hyderabad, India, June 4-8, 2024*. IEEE, 79–89. https://doi.org/10.1109/ICSA59870.2024.00016

[31] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2023. ClassEval: A

Manually-Crafted Benchmark for Evaluating LLMs on Class-level Code Generation. *CoRR* abs/2308.01861 (2023). https://doi.org/10.48550/ARXIV.2308.01861 arXiv:2308.01861

[32] Ty Dunn. 2023. It's time to collect data on how you build software. https://blog.continue.dev/its-time-to-collect-data-on-how-you-build-software/. Accessed 2025-08-15.

[33] Aryaz Eghbali and Michael Pradel. 2022. CrystalBLEU: Precisely and Efficiently Measuring the Similarity of Code. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022*. ACM, 28:1–28:12. https://doi.org/10.1145/3551349.3556903

[34] Abdelkarim El-Hajjami and Camille Salinesi. 2025. How Good Are Synthetic Requirements? Evaluating LLM-Generated Datasets for AI4RE. *CoRR* abs/2506.21138 (2025). https://doi.org/10.48550/ARXIV.2506.21138 arXiv:2506.21138

[35] Aparna Elangovan, Jongwoo Ko, Lei Xu, Mahsa Elyasi, Ling Liu, Sravan Bodapati, and Dan Roth. 2024. Beyond correlation: The impact of human uncertainty in measuring the effectiveness of automatic evaluation and LLM-as-a-judge. *CoRR* abs/2410.03775 (2024). https://doi.org/10.48550/ARXIV.2410.03775 arXiv:2410.03775

[36] Zhenxiao Fu, Fan Chen, Shan Zhou, Haitong Li, and Lei Jiang. 2024. LLMCO2: Advancing Accurate Carbon Footprint Prediction for LLM Inferences. *CoRR* abs/2410.02950 (2024). https://doi.org/10.48550/ARXIV.2410.02950 arXiv:2410.02950

[37] Isabel O. Gallegos, Ryan A. Rossi, Joe Barrow, Md. Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen Ahmed. 2024. Bias and Fairness in Large Language Models: A Survey. *Computational Linguistics* 50 (2024), 1097–1179. Issue 3. https://doi.org/10.1162/coli_a_00524

[38] Isabel O. Gallegos, Ryan A. Rossi, Joe Barrow, Md. Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K. Ahmed. 2023. Bias and Fairness in Large Language Models: A Survey. *CoRR* abs/2309.00770 (2023). https://doi.org/10.48550/ARXIV.2309.00770 arXiv:2309.00770

[39] Jack Gallifant, Majid Afshar, Saleem Ameen, Yindalon Aphinyanaphongs, Shan Chen, Giovanni Cacciamani, Dina Demner-Fushman, Dmitriy Dligach, Roxana Daneshjou, Chrystinne Fernandes, Lasse Hyldig Hansen, Adam Landman, Lisa Lehmann, Liam G. McCoy, Timothy Miller, Amy Moreno, Nikolaj Munch, David Restrepo, Guergana Savova, Renato Umeton, Judy Wawira Gichoya, Gary S. Collins, Karel G. M. Moons, Leo A. Celi, and Danielle S. Bitterman. 2025. The TRIPOD-LLM reporting guideline for studies using large language models. *Nature Medicine* 31, 1 (01 Jan 2025), 60–69. https://doi.org/10.1038/s41591-024-03425-5

[40] Marco Aurélio Gerosa, Bianca Trinkenreich, Igor Steinmacher, and Anita Sarma. 2024. Can AI serve as a substitute for human subjects in software engineering research? *Autom. Softw. Eng.* 31, 1 (2024), 13. https://doi.org/10.1007/S10515-023-00409-6

[41] Elizabeth Gibney. 2024. Not all 'open source' AI models are actually open. *Nature News* (2024). https://doi.org/10.1038/d41586-024-02012-5

[42] Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. 2023. ChatGPT Outperforms Crowd-Workers for Text-Annotation Tasks. *CoRR* abs/2303.15056 (2023). https://doi.org/10.48550/ARXIV.2303.15056 arXiv:2303.15056

[43] José Gonçalves, Miguel Silva, Bernardo Cabral, Tiago Dias, Eva Maia, Isabel Praça, Ricardo Severino, and Luís Lino Ferreira. 2025. Evaluating LLaMA 3.2 for Software Vulnerability Detection. *arXiv preprint arXiv:2503.07770* (2025).

[44] Daniel Graziotin. 2024. *acmsigsoft/open-science-policies: v1.0.0*. https://doi.org/10.5281/zenodo.10796477

[45] Odd Erik Gundersen, Odd Cappelen, Martin Mølnå, and Nicklas Grimstad Nilsen. 2024. The Unreasonable Effectiveness of Open Science in AI: A Replication Study. *CoRR* abs/2412.17859 (2024). https://doi.org/10.48550/ARXIV.2412.17859 arXiv:2412.17859

[46] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming - The Rise of Code Intelligence. *CoRR* abs/2401.14196 (2024). https://doi.org/10.48550/ARXIV.2401.14196 arXiv:2401.14196

[47] Jacqueline Harding, William D'Alessandro, N. G. Laskowski, and Robert Long. 2024. AI language models cannot replace human research participants. *AI Soc.* 39, 5 (2024), 2603–2605. https://doi.org/10.1007/S00146-023-01725-X

[48] Zeyu He, Chieh-Yang Huang, Chien-Kuang Cornelia Ding, Shaurya Rohatgi, and Ting-Hao Kenneth Huang. 2024. If in a Crowdsourced Data Annotation Pipeline, a GPT-4. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024*, Florian 'Floyd' Mueller, Penny Kyburz, Julie R. Williamson, Corina Sas, Max L. Wilson, Phoebe O. Toups Dugas, and Irina Shklovski (Eds.). ACM, 1040:1–1040:25. https://doi.org/10.1145/3613904.3642834

[49] Catherine M Hicks, Carol S Lee, and Kristen Foster-Marks. 2025. The New Developer: AI Skill Threat, Identity Change & Developer Thriving in the Transition to AI-Assisted Software Development. *PsyArXiv* (March 2025). https://doi.org/10.31234/osf.io/2gej5_v2

[50] 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

[51] Xing Hu, Feifei Niu, Junkai Chen, Xin Zhou, Junwei Zhang, Junda He, Xin Xia, and David Lo. 2025. Assessing and Advancing Benchmarks for Evaluating Large Language Models in Software Engineering Tasks. *arXiv* (2025). https://arxiv.org/abs/2505.08903

[52] Fan Huang, Haewoon Kwak, and Jisun An. 2023. Is ChatGPT better than Human Annotators? Potential and Limitations of ChatGPT in Explaining Implicit Hate Speech. In *Companion Proceedings of the ACM Web Conference 2023, WWW 2023*, Ying Ding, Jie Tang, Juan F. Sequeda, Lora Aroyo, Carlos Castillo, and Geert-Jan Houben (Eds.). ACM, 294–297. https://doi.org/10.1145/3543873.3587368

[53] Jiangping Huang, Bochen Yi, Weisong Sun, Bangrui Wan, Yang Xu, Yebo Feng, Wenguang Ye, and Qinjun Qin. 2024. Enhancing Review Classification Via LLM-Based Data Annotation and Multi-Perspective Feature Representation Learning. *SSRN Electronic Journal* (2024), 1–15. https://doi.org/10.2139/ssrn.5002351

[54] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-Coder Technical Report. *CoRR* abs/2409.12186 (2024). https://doi.org/10.48550/ARXIV.2409.12186 arXiv:2409.12186

[55] Jasmin Jahic and Ashkan Sami. 2024. State of Practice: LLMs in Software Engineering and Software Architecture. In *21st IEEE International Conference on Software Architecture, ICSA 2024 - Companion, Hyderabad, India, June 4-8, 2024*. IEEE, 311–318. https://doi.org/10.1109/ICSA-C63560.2024.00059

[56] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. *CoRR* abs/2403.07974 (2024). https://doi.org/10.48550/ARXIV.2403.07974 arXiv:2403.07974

[57] Kevin Jesse, Toufique Ahmed, Premkumar T. Devanbu, and Emily Morgan. 2023. Large Language Models and Simple, Stupid Bugs. In *20th IEEE/ACM International Conference on Mining Software Repositories, MSR 2023*. IEEE, 563–575. https://doi.org/10.1109/MSR59073.2023.00082

[58] Peng Jiang, Christian Sonne, Wangliang Li, Fengqi You, and Siming You. 2024. Preventing the Immense Increase in the Life-Cycle Energy and Carbon Footprints of LLM-Powered Intelligent Chatbots. *Engineering* 40 (2024), 202–210. https://doi.org/10.1016/j.eng.2024.04.002

[59] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=VTF8yNQM66

[60] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023*. IEEE, 2312–2323. https://doi.org/10.1109/ICSE48619.2023.00194

[61] Anjan Karmakar, Miltiadis Allamanis, and Romain Robbes. 2023. JEMMA: An extensible Java dataset for ML4Code applications. *Empir. Softw. Eng.* 28, 2 (2023), 54. https://doi.org/10.1007/S10664-022-10275-7

[62] Ranim Khojah, Mazen Mohamad, Philipp Leitner, and Francisco Gomes de Oliveira Neto. 2024. Beyond Code Generation: An Observational Study of ChatGPT Usage in Software Engineering Practice. *Proc. ACM Softw. Eng.* 1, FSE (2024), 1819–1840. https://doi.org/10.1145/3660788

[63] Qusai Khraisha, Sophie Put, Johanna Kappenberg, Azza Warraitch, and Kristin Hadfield. 2024. Can large language models replace humans in systematic reviews? Evaluating GPT-4's efficacy in screening and extracting data from peer-reviewed and grey literature in multiple languages. *Research Synthesis Methods* 15, 4 (2024), 616–626. https://doi.org/10.1002/jrsm.1715 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/jrsm.1715

[64] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, and Xin Zhou. 2023. Better Zero-Shot Reasoning with Role-Play Prompting. *CoRR* abs/2308.07702 (2023). https://doi.org/10.48550/ARXIV.2308.07702 arXiv:2308.07702

[65] Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy Liang. 2019. SPoC: Search-based Pseudocode to Code. *CoRR* abs/1906.04908 (2019). arXiv:1906.04908 http://arxiv.org/abs/1906.04908

[66] Marie-Anne Lachaux, Baptiste Rozière, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised Translation of Programming Languages. *CoRR* abs/2006.03511 (2020). arXiv:2006.03511 https://arxiv.org/abs/2006.03511

[67] Kleanthi Lakiotaki, Nikolaos Vorniotakis, Michail Tsagris, Georgios Georgakopoulos, and Ioannis Tsamardinos. 2018. BioDataome: a collection of uniformly preprocessed and automatically annotated datasets for data-driven biology. *Database J. Biol. Databases Curation* 2018 (2018), bay011. https://doi.org/10.1093/DATABASE/BAY011

[68] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating Training Data Makes Language Models Better. In *Proceedings of the 60th Annual Meeting*

Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8, Article 220 (Dec. 2024), 79 pages. https://doi.org/10.1145/3695988

of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 8424–8445. https://doi.org/10.18653/V1/2022.ACL-LONG.577

[69] David Li, Kartik Gupta, Mousumi Bhaduri, Paul Sathiadoss, Sahir Bhatnagar, and Jaron Chong. 2024. Comparing GPT-3.5 and GPT-4 Accuracy and Drift in Radiology Diagnosis Please Cases. *Radiology* 310, 1 (2024), e232411. https://doi.org/10.1148/radiol.232411 arXiv:https://doi.org/10.1148/radiol.232411

[70] Jia Li, Ge Li, Xuanming Zhang, Yunfei Zhao, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, and Yongbin Li. 2024. EvoCodeBench: An Evolving Code Generation Benchmark with Domain-Specific Evaluations. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/hash/6a059625a6027aca18302803743abaa2-Abstract-Datasets_and_Benchmarks_Track.html

[71] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Moustafa-Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *CoRR* abs/2305.06161 (2023). https://doi.org/10.48550/ARXIV.2305.06161 arXiv:2305.06161

[72] Jenny T Liang, Carmen Badea, Christian Bird, Robert DeLine, Denae Ford, Nicole Forsgren, and Thomas Zimmermann. 2024. Can gpt-4 replicate empirical software engineering research? *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1330–1353.

[73] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html

[74] Cristina V. Lopes, Petr Maj, Pedro Martins, Vaibhav Saini, Di Yang, Jakub Zitny, Hitesh Sajnani, and Jan Vitek. 2017. DéjàVu: a map of code duplicates on GitHub. *Proc. ACM Program. Lang.* 1, OOPSLA (2017), 84:1–84:28. https://doi.org/10.1145/3133908

[75] José Antonio Hernández López, Boqi Chen, Mootez Saad, Tushar Sharma, and Dániel Varró. 2025. On Inter-Dataset Code Duplication and Data Leakage in Large Language Models. *IEEE Trans. Software Eng.* 51, 1 (2025), 192–205. https://doi.org/10.1109/TSE.2024.3504286

[76] Qinghua Lu, Liming Zhu, Xiwei Xu, Zhenchang Xing, and Jon Whittle. 2024. Toward Responsible AI in the Era of Generative AI: A Reference Architecture for Designing Foundation Model-Based Systems. *IEEE Softw.* 41, 6 (2024), 91–100. https://doi.org/10.1109/MS.2024.3406333

[77] Sebastian Lubos, Alexander Felfernig, Thi Ngoc Trang Tran, Damian Garber, Merfat El Mansi, Seda Polat Erdeniz, and Viet-Man Le. 2024. Leveraging LLMs for the Quality Assurance of Software Requirements. In *32nd IEEE International Requirements Engineering Conference, RE 2024, Reykjavik, Iceland, June 24-28, 2024*, Grischa Liebel, Irit Hadar, and Paola Spoletini (Eds.). IEEE, 389–397. https://doi.org/10.1109/RE59067.2024.00046

[78] Kashumi Madampe, John Grundy, Rashina Hoda, and Humphrey O. Obie. 2024. The struggle is real! The agony of recruiting participants for empirical software engineering studies. In *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Liverpool, UK, September 2-6, 2024*. IEEE, 417–422. https://doi.org/10.1109/VL/HCC60511.2024.00065

[79] Walter S Mathis, Sophia Zhao, Nicholas Pratt, Jeremy Weleff, and Stefano De Paoli. 2024. Inductive thematic analysis of healthcare qualitative interviews using open-source large language models: How does it compare to traditional methods? *Computer Methods and Programs in Biomedicine* 255 (2024), 108356.

[80] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and Inter-rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice. *Proc. ACM Hum. Comput. Interact.* 3, CSCW (2019), 72:1–72:23. https://doi.org/10.1145/3359174

[81] Microsoft. 2023. CodeBERT on GitHub. https://github.com/microsoft/CodeBERT. Accessed 2025-08-15.

[82] Microsoft. 2025. Azure OpenAI Service models. https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models. Accessed 2025-01-13.

[83] Narcis Eduard Mitu and George Teodor Mitu. 2024. The Hidden Cost of AI: Carbon Footprint and Mitigation Strategies. *Available at SSRN 5036344* (2024).

[84] Suad Mohamed, Abdullah Parvin, and Esteban Parra. 2024. Chatting with AI: Deciphering Developer Conversations with ChatGPT. In *21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024*, Diomidis Spinellis, Alberto Bacchelli, and Eleni Constantinou (Eds.). ACM, 187–191. https://doi.org/10.1145/3643991.3645078

[85] Micheline Bénédicte Moumoula, Abdoul Kader Kabore, Jacques Klein, and Tegawendé Bissyande. 2024. Large Language Models for cross-language code clone detection. *arXiv preprint arXiv:2408.04430* (2024).

[86] Network Working Group. 1997. RFC 2119. https://www.rfc-editor.org/rfc/rfc2119. Accessed 2025-08-15.

[87] Network Working Group. 2017. RFC 8174. https://www.rfc-editor.org/rfc/rfc8174. Accessed 2025-08-15.

[88] Open Source Initiative (OSI). 2025. Open Source AI Definition 1.0. https://opensource.org/ai/open-source-ai-definition. Accessed 2025-08-26.

[89] OpenAI. 2023. How to make your completions outputs consistent with the new seed parameter. https://cookbook.openai.com/examples/reproducible_outputs_with_the_seed_parameter. Accessed 2025-01-13.

[90] OpenAI. 2025. OpenAI API Introduction. https://platform.openai.com/docs/api-reference/chat/streaming. Accessed 2025-01-13.

[91] Qian Pan, Zahra Ashktorab, Michael Desmond, Martin Santillan Cooper, James Johnson, Rahul Nair, Elizabeth Daly, and Werner Geyer. 2024. Human-Centered Design Recommendations for LLM-as-a-Judge. In *ACL 2024 Workshop HuCLLM*. arXiv. https://doi.org/10.48550/arXiv.2407.03479 arXiv:2407.03479 [cs]

[92] Nicholas Pangakis, Samuel Wolken, and Neil Fasching. 2023. Automated Annotation with Generative AI Requires Validation. *CoRR* abs/2306.00176 (2023). https://doi.org/10.48550/ARXIV.2306.00176 arXiv:2306.00176

[93] Arjun Panickssery, Samuel Bowman, and Shi Feng. 2024. LLM Evaluators Recognize and Favor Their Own Generations. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 68772–68802. https://proceedings.neurips.cc/paper_files/paper/2024/file/7f1f0218e45f5414c79c0679633e47bc-Paper-Conference.pdf

[94] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. ACL, 311–318. https://doi.org/10.3115/1073083.1073135

[95] Uwe Peters and Benjamin Chin-Yee. 2025. Generalization bias in large language model summarization of scientific research. *R. Soc. Open Sci.* 12, 12241776 (2025). https://doi.org/10.1098/rsos.241776

[96] Pouya Pezeshkpour and Estevam Hruschka. 2024. Large Language Models Sensitivity to The Order of Options in Multiple-Choice Questions. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (Eds.). Association for Computational Linguistics, 2006–2017. https://doi.org/10.18653/V1/2024.FINDINGS-NAACL.130

[97] Md. Fazle Rabbi, Arifa I. Champa, Minhaz Fahim Zibran, and Md. Rakibul Islam. 2024. AI Writes, We Analyze: The ChatGPT Python Code Saga. In *21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024*, Diomidis Spinellis, Alberto Bacchelli, and Eleni Constantinou (Eds.). ACM, 177–181. https://doi.org/10.1145/3643991.3645076

[98] Paul Ralph and Ewan D. Tempero. 2018. Construct Validity in Software Engineering Research and Software Metrics. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering, EASE2018*, Austen Rainer, Stephen G. MacDonell, and Jacky W. Keung (Eds.). ACM, 13–23. https://doi.org/10.1145/3210459.3210461

[99] Michael V. Reiss. 2023. Testing the Reliability of ChatGPT for Text Annotation and Classification: A Cautionary Remark. *CoRR* abs/2304.11085 (2023). https://doi.org/10.48550/ARXIV.2304.11085 arXiv:2304.11085

[100] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. *CoRR* abs/2009.10297 (2020). arXiv:2009.10297 https://arxiv.org/abs/2009.10297

[101] Jonan Richards and Mairieli Wessel. 2024. What You Need is what You Get: Theory of Mind for an LLM-Based Code Understanding Assistant. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2024*. IEEE, 666–671. https://doi.org/10.1109/ICSME58944.2024.00070

[102] Krishna Ronanki, Christian Berger, and Jennifer Horkoff. 2023. Investigating ChatGPT's potential to assist in requirements elicitation processes. In *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 354–361.

[103] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet,

Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. *CoRR* abs/2308.12950 (2023). https://doi.org/10.48550/ARXIV.2308.12950 arXiv:2308.12950

[104] Daniel Russo. 2024. Navigating the Complexity of Generative AI Adoption in Software Engineering. *ACM Transactions on Software Engineering and Methodology* 33, 5 (2024), 1–50.

[105] June Sallou, Thomas Durieux, and Annibale Panichella. 2024. Breaking the silence: the threats of using llms in software engineering. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 102–106.

[106] Adrian Santos, Sira Vegas, Markku Oivo, and Natalia Juristo. 2021. A Procedure and Guidelines for Analyzing Groups of Software Engineering Replications. *IEEE Trans. Software Eng.* 47, 9 (2021), 1742–1763. https://doi.org/10.1109/TSE.2019.2935720

[107] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2024. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. *IEEE Trans. Software Eng.* 50, 1 (2024), 85–105. https://doi.org/10.1109/TSE.2023.3334955

[108] Kurt Schneider, Farnaz Fotrousi, and Rebekka Wohlrab. 2025. A Reference Model for Empirically Comparing LLMs with Humans. In *Proceedings of the 47th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS2025)*. IEEE.

[109] Kayla Schroeder and Zach Wood-Doughty. 2024. Can You Trust LLM Judgments? Reliability of LLM-as-a-Judge. *CoRR* abs/2412.12509 (2024). https://doi.org/10.48550/ARXIV.2412.12509 arXiv:2412.12509

[110] Sarah Schröder, Thekla Morgenroth, Ulrike Kuhl, Valerie Vaquet, and Benjamin Paaßen. 2025. Large Language Models Do Not Simulate Human Psychology. *arXiv* (2025). https://arxiv.org/abs/2508.06950

[111] Forrest Shull, Janice Singer, and Dag I. K. Sjøberg (Eds.). 2008. *Guide to Advanced Empirical Software Engineering*. Springer. https://doi.org/10.1007/978-1-84800-044-5

[112] André Silva and Martin Monperrus. 2024. RepairBench: Leaderboard of Frontier Models for Program Repair. *arXiv preprint arXiv:2409.18952* (2024).

[113] Charlotte Siska, Katerina Marazopoulou, Melissa Ailem, and James Bono. 2024. Examining the robustness of LLM evaluation to the distributional assumptions of benchmarks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 10406–10421. https://doi.org/10.18653/V1/2024.ACL-LONG.560

[114] Dag I. K. Sjøberg and Gunnar Rye Bergersen. 2023. Construct Validity in Software Engineering. *IEEE Trans. Software Eng.* 49, 3 (2023), 1374–1396. https://doi.org/10.1109/TSE.2022.3176725

[115] Tim Sonnekalb, Bernd Gruner, Clemens-Alexander Brust, and Patrick Mäder. 2022. Generalizability of Code Clone Detection on CodeBERT. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022*. ACM, 143:1–143:3. https://doi.org/10.1145/3551349.3561165

[116] Moritz Staudinger, Wojciech Kusa, Florina Piroi, Aldo Lipani, and Allan Hanbury. 2024. A Reproducibility and Generalizability Study of Large Language Models for Query Generation. In *Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region, SIGIR-AP 2024, Tokyo, Japan, December 9-12, 2024*, Tetsuya Sakai, Emi Ishita, Hiroaki Ohshima, Faegheh Hasibi, Jiaxin Mao, and Joemon M. Jose (Eds.). ACM, 186–196. https://doi.org/10.1145/3673791.3698432

[117] Gail M Sullivan and Joan Sargeant. 2011. Qualities of qualitative research: part I. *J Grad Med Educ* 3, 4 (Dec. 2011), 449–452.

[118] Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2024. Cognitive Architectures for Language Agents. *Trans. Mach. Learn. Res.* 2024 (2024). https://openreview.net/forum?id=1i6ZCvflQJ

[119] Wannita Takerngsaksiri, Jirat Pasuksmit, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Ruixiong Zhang, Fan Jiang, Jing Li, Evan Cook, Kun Chen, and Ming Wu. 2024. Human-In-the-Loop Software Development Agents. *arXiv preprint arXiv:2411.12924* (2024).

[120] Technology Innovation Institute. 2023. falcon-refinedweb (Revision 184df75). https://huggingface.co/datasets/tiiuae/falcon-refinedweb. https://doi.org/10.57967/hf/0737

[121] Christof Tinnes, Alisa Welter, and Sven Apel. 2025. Software Model Evolution with Large Language Models: Experiments on Simulated, Public, and Industrial Datasets. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025*. IEEE, 950–962. https://doi.org/10.1109/ICSE55347.2025.00112

[122] Stefan Wagner, Marvin Muñoz Barón, Davide Falessi, and Sebastian Baltes. 2025. Towards Evaluation Guidelines for Empirical Studies Involving LLMs. In *IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering, WSESE@ICSE 2025, May 3, 2025*. IEEE, 24–27. https://doi.org/10.1109/WSESE66602.2025.00011

[123] Mengting Wan, Tara Safavi, Sujay Kumar Jauhar, Yujin Kim, Scott Counts, Jennifer Neville, Siddharth Suri, Chirag Shah, Ryen W. White, Longqi Yang, Reid Andersen, Georg Buscher, Dhruv Joshi, and Nagu Rangan. 2024. TnT-LLM: Text Mining at Scale with Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024*, Ricardo Baeza-Yates and Francesco Bonchi (Eds.). ACM, 5836–5847. https://doi.org/10.1145/3637528.3671647

[124] Angelina Wang, Jamie Morgenstern, and John P. Dickerson. 2024. Large language models cannot replace human participants because they cannot portray identity groups. *CoRR* abs/2402.01908 (2024). https://doi.org/10.48550/ARXIV.2402.01908 arXiv:2402.01908

[125] Chong Wang, Kaifeng Huang, Jian Zhang, Yebo Feng, Lyuye Zhang, Yang Liu, and Xin Peng. 2024. How and why llms use deprecated apis in code completion? an empirical study. *arXiv preprint arXiv:2406.09834* (2024).

[126] Fanyu Wang, Chetan Arora, Yonghui Liu, Kaicheng Huang, Chakkrit Tantithamthavorn, Aldeida Aleti, Dishan Sambathkumar, and David Lo. 2025. Multi-Modal Requirements Data-based Acceptance Criteria Generation using LLMs. *arXiv* (2025). arXiv:2508.06888 [cs.SE] https://arxiv.org/abs/2508.06888

[127] Shuohang Wang, Yang Liu, Yichong Xu, Chenguang Zhu, and Michael Zeng. 2021. Want To Reduce Labeling Cost? GPT-3 Can Help. In *Findings of the ACL: EMNLP 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 4195–4205. https://doi.org/10.18653/V1/2021.FINDINGS-EMNLP.354

[128] Xinru Wang, Hannah Kim, Sajjadur Rahman, Kushan Mitra, and Zhengjie Miao. 2024. Human-LLM Collaborative Annotation Through Effective Verification of LLM Labels. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024*, Florian 'Floyd' Mueller, Penny Kyburz, Julie R. Williamson, Corina Sas, Max L. Wilson, Phoebe O. Toups Dugas, and Irina Shklovski (Eds.). ACM, 303:1–303:21. https://doi.org/10.1145/3613904.3641960

[129] David Gray Widder, Meredith Whittaker, and Sarah Myers West. 2024. Why 'open'AI systems are actually closed, and why this matters. *Nature* 635, 8040 (2024), 827–833.

[130] Julia Wiesinger, Patrick Marlow, and Vladimir Vuskovic. 2025. Agents. Whitepaper. *Google DeepMind* (Feb. 2025). https://gemini.google.com Contributors: Evan Huang, Emily Xue, Olcan Sercinoglu, Sebastian Riedel, Satinder Baveja, Antonio Gulli, Anant Nawalgaria.

[131] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2024. *Experimentation in Software Engineering, Second Edition*. Springer. https://doi.org/10.1007/978-3-662-69306-3

[132] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga Behram, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin S. Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim M. Hazelwood. 2022. Sustainable AI: Environmental Implications, Challenges and Opportunities. In *Proceedings of the Fifth Conference on Machine Learning and Systems, MLSys 2022*, Diana Marculescu, Yuejie Chi, and Carole-Jean Wu (Eds.). mlsys.org. https://proceedings.mlsys.org/paper_files/paper/2022/hash/462211f67c7d858f663355eff93b745e-Abstract.html

[133] Yuying Xia, Haijian Shao, and Xing Deng. 2024. VulCoBERT: A CodeBERT-Based System for Source Code Vulnerability Detection. In *2024 International Conference on Generative Artificial Intelligence and Information Security, GAIIS 2024, Kuala Lumpur, Malaysia, May 10-12, 2024*. ACM. https://doi.org/10.1145/3665348.3665391

[134] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. In *21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024*, Diomidis Spinellis, Alberto Bacchelli, and Eleni Constantinou (Eds.). ACM, 227–230. https://doi.org/10.1145/3643991.3648400

[135] Cheng Xu, Shuhao Guan, Derek Greene, and M. Tahar Kechadi. 2024. Benchmark Data Contamination of Large Language Models: A Survey. *CoRR* abs/2406.04244 (2024). https://doi.org/10.48550/ARXIV.2406.04244 arXiv:2406.04244

[136] Ruoxi Xu, Yingfei Sun, Mengjie Ren, Shiguang Guo, Ruotong Pan, Hongyu Lin, Le Sun, and Xianpei Han. 2024. AI for social science and social science of AI: A survey. *Inf. Process. Manag.* 61, 2 (2024), 103665. https://doi.org/10.1016/J.IPM.2024.103665

[137] Yuankai Xue, Hanlin Chen, Gina R. Bai, Robert Tairas, and Yu Huang. 2024. Does ChatGPT Help With Introductory Programming?An Experiment of Students Using ChatGPT in CS1. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training, SEET@ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 331–341. https://doi.org/10.1145/3639474.3640076

[138] Litao Yan, Alyssa Hwang, Zhiyuan Wu, and Andrew Head. 2024. Ivie: Lightweight Anchored Explanations of Just-Generated Code. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024*, Florian 'Floyd' Mueller, Penny Kyburz, Julie R. Williamson, Corina Sas, Max L. Wilson, Phoebe O. Toups Dugas, and Irina Shklovski (Eds.). ACM, 140:1–140:15. https://doi.org/10.1145/3613904.3642239

[139] Guang Yang, Yu Zhou, Xiang Chen, Xiangyu Zhang, Tingting Han, and Taolue Chen. 2023. ExploitGen: Template-augmented exploit code generation based

on CodeBERT. *J. Syst. Softw.* 197 (2023), 111577. https://doi.org/10.1016/J.JSS.2022.111577

[140] Chenguang Zhao, Meirewuti Habule, and Wei Zhang. 2025. Large language models (LLMs) as research Subjects: Status, opportunities and challenges. *New Ideas in Psychology* 79 (2025), 101167. https://doi.org/10.1016/j.newideapsych.2025.101167

[141] Ruiyang Zhou, Lu Chen, and Kai Yu. 2024. Is LLM a Reliable Reviewer? A Comprehensive Evaluation of LLM on Automatic Paper Reviewing Tasks. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, Nicoletta Calzolari, Min-Yen Kan, Véronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (Eds.). ELRA and ICCL, 9340–9351. https://aclanthology.org/2024.lrec-main.816

[142] Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. 2023. Agents: An Open-source Framework for Autonomous Language Agents. *CoRR* abs/2309.07870 (2023). https://doi.org/10.48550/ARXIV.2309.07870 arXiv:2309.07870

[143] Xin Zhou, Martin Weyssow, Ratnadira Widyasari, Ting Zhang, Junda He, Yunbo Lyu, Jianming Chang, Beiqi Zhang, Dan Huang, and David Lo. 2025. LessLeak-Bench: A First Investigation of Data Leakage in LLMs Across 83 Software Engineering Benchmarks. *arXiv* (2025). arXiv:2502.06215 [cs.SE] https://arxiv.org/abs/2502.06215

[144] Yiming Zhu, Peixian Zhang, Ehsan ul Haq, Pan Hui, and Gareth Tyson. 2023. Can ChatGPT Reproduce Human-Generated Labels? A Study of Social Computing Tasks. *CoRR* abs/2304.10145 (2023). https://doi.org/10.48550/ARXIV.2304.10145 arXiv:2304.10145

[145] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro von Werra. 2024. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. *CoRR* abs/2406.15877 (2024). https://doi.org/10.48550/ARXIV.2406.15877 arXiv:2406.15877