Empirical Software Engineering
(accepted manuscript)

Guidelines for Empirical Studies in Software Engineering involving Large Language Models

Sebastian Baltes · Florian Angermeir · Chetan Arora · Marvin Muñoz Barón · Chunyang Chen · Lukas Böhme · Fabio Calefato · Neil Ernst · Davide Falessi · Brian Fitzgerald · Davide Fucci · Junda He · Christoph Treude · Marcos Kalinowski · Stefano Lambiase · Daniel Russo · Mircea Lungu · Cristina Martinez Montes · Lutz Prechelt · Paul Ralph · Rijnard van Tonder · Stefan Wagner

Submitted: 26/08/2025
Revised: 27/02/2026, 05/05/2026, 11/06/2026

Sebastian Baltes
Heidelberg University, Germany
E-mail: sebastian.baltes@uni-heidelberg.de
(corresponding author)

Florian Angermeir
fortiss, Germany
Blekinge Institute of Technology, Sweden
E-mail: angermeir@fortiss.org

Chetan Arora
Monash University, Australia
E-mail: chetan.arora@monash.edu

Marvin Muñoz Barón · Chunyang Chen
Technical University of Munich, Germany
E-mail: {marvin.munoz-baron, chun-yang.chen}@tum.de

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

Fabio Calefato
University of Bari, Italy
E-mail: fabio.calefato@uniba.it

Neil Ernst
University of Victoria, Canada
E-mail: nernst@uvic.ca

Davide Falessi

Abstract Large Language Models (LLMs) are widely used in software engineering (SE) research and practice, yet their non-determinism, opaque training data, and rapidly evolving models threaten the reproducibility and replicability of empirical studies. We address this challenge through a collaborative effort of 22 researchers, presenting a taxonomy of seven study types that organizes how LLMs are used in SE research, together with eight guidelines for designing and reporting such studies. Each guideline distinguishes requirements (must) from recommendations (should) and is contextualized by the study types it applies to. Our guidelines recommend that researchers: (1) declare LLM usage and role; (2) report model versions, configurations, and customizations; (3) document the system and prompt design beyond the model; (4) report ses-

University of Rome Tor Vergata, Italy
E-mail: falessi@ing.uniroma2.it

Brian Fitzgerald
Lero, Ireland
University of Limerick, Ireland
E-mail: brian.fitzgerald@ul.ie

Davide Fucci
Blekinge Institute of Technology, Sweden
E-mail: davide.fucci@bth.se

Junda He · Christoph Treude
Singapore Management University, Singapore
E-mail: {jundahe, ctreude}@smu.edu.sg

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

Stefano Lambiase · Daniel Russo
Aalborg University in Copenhagen, Denmark
E-mail: {stla, daniel.russo}@cs.aau.dk

Mircea Lungu
IT University of Copenhagen, Denmark
E-mail: mlun@itu.dk

Cristina Martinez Montes
Chalmers University of Technology, Sweden
University of Gothenburg, Sweden
E-mail: montesc@chalmers.se

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

Paul Ralph
Dalhousie University, Canada
E-mail: paulralph@dal.ca

Rijnard van Tonder
Independent Researcher, Antigua and Barbuda
E-mail: rvantonder@gmail.com

Stefan Wagner
Technical University of Munich, Germany
E-mail: stefan.wagner@tum.de

sion traces, i.e., interaction logs and runtime traces; (5) use suitable baselines, benchmarks, and metrics; (6) include an open LLM as a baseline; (7) validate LLM outputs against human judgment; and (8) articulate limitations and mitigations. We complement the guidelines with an applicability matrix mapping guidelines to study types and a reporting checklist for authors and reviewers. We maintain the study types and guidelines online as a living resource for the community to use and shape (llm-guidelines.org).

Keywords software engineering, large language models, empirical research, meta-science, guidelines

1 Introduction

Since the release of ChatGPT in November 2022, large language models (LLMs) have been adopted widely across software engineering (SE) research [64], yet the reproducibility and replicability of empirical studies involving LLMs remains uncertain. Recent findings indicate low reproducibility in SE studies involving LLMs. Angermeir et al examined 85 LLM-centric ICSE and ASE 2024 papers [6]. Of the 18 papers that both used OpenAI models and provided artifacts, only five were complete enough to execute, and none fully reproduced the original results [6]. Evertz et al found at least one LLM-specific pitfall in each of 72 peer-reviewed security and SE papers from 2023–2024, and only 15.7% of the observed pitfalls were discussed [46].

LLM-based SE studies are hard to reproduce for three reasons. First, their inherent non-determinism causes variability across runs [24, 146], and slight changes can lead to substantially different results [2, 138]. Second, commercial models evolve beyond version identifiers, so reported performance can change over time [33]. Third, even for "open" models, training data and fine-tuning details often remain undisclosed [51]. Moreover, prompt formatting choices alone can shift accuracy by up to 76 percentage points [142], and configured parameters such as temperature affect output variability [124]. Hence, not reporting these settings directly affects reproducibility.

Traditional open science practice in SE has focused on releasing source code and datasets as a replication package. The empirical artifact analysis literature in SE focuses more on code repositories and data than on upstream artifacts such as requirements specifications and design documents [91]. With LLMs, code is often generated from prompts, context files, and other runtime inputs to the model, so reporting must shift left to cover these upstream artifacts in addition to the code and data they produce.

Although the SE research community has developed guidelines for conducting and reporting specific types of empirical studies such as controlled experiments (e.g., [143, 164]), their replications (e.g., [133]), or empirical studies in general (e.g., the *ACM SIGSOFT Empirical Standards* [118]), none of these address the LLM-specific aspects described above. Previously, a position paper highlighted these issues [155], but there was no comprehensive

community-developed guidance for designing and reporting empirical studies involving LLMs in SE.

Therefore, we present community-developed guidelines for designing and reporting studies involving LLMs in SE research, co-developed by 22 researchers. After outlining our *Scope*, we introduce a taxonomy of *Study Types*, then present eight *Guidelines*. We complement these with an applicability matrix mapping guidelines to study types and a reporting checklist for authors and reviewers. For each study type and guideline, we identify relevant examples, both within and outside of SE research. We maintain the study types and guidelines online as a living resource for the community to use and shape (llm-guidelines.org).

2 Scope and Conventions

Software Engineering as our Target Discipline: We target SE research because reporting guidelines from other disciplines do not address its specific needs (see *Related Reporting Guidelines* below). In particular, SE research employs a wide variety of empirical methods [118, 164]. We organize the LLM-involving subset of these methods into a taxonomy of *Study Types* (Section 4), and each of our *Guidelines* specifies its applicability per study type.

Focus on Text-Based Use Cases: While multi-modal foundation models that use or generate images, audio, or video may also support SE research and practice, we focus on textual use cases of LLMs (e.g., in natural language or programming languages). Many of our guidelines, particularly those concerning model reporting, prompt documentation, and reproducibility, are likely applicable to multi-modal settings as well, but we leave a systematic assessment of this applicability to future work.

Focus on Direct Development or Research Support: While researchers may use LLMs for many peripheral tasks (e.g., proofreading, spell-checking, translation), our guidelines focus on their direct role in empirical research and engineering practice. For engineers, we focus on the use of LLMs to automate SE tasks, that is, artificial intelligence (AI) for software engineering (AI4SE) (see *LLMs for SE*). This includes agentic systems that autonomously plan and execute multi-step tasks using LLMs (see *System and Prompt Design*). For researchers, we focus on the use of LLMs to automate empirical research tasks such as data collection, processing, or analysis (see *LLMs for Research*). The 2026 ACM Policy on Authorship likewise requires disclosing AI used in the research itself, not AI used only to assist with writing [11].

Researchers as our Target Audience: Our guidelines are intended to help SE researchers design, plan, conduct, and report empirical studies involving LLMs, and to support scholarly peer review of such studies. Each

guideline includes an *Advice for Reviewers* subsection with targeted assessment suggestions. Our guidelines focus on what to report and how. They complement but do not replace methodological guidance for designing specific types of empirical studies.

Reporting Locations: We use *paper* to refer to the manuscript PDF, including any appendices it contains. We use *supplementary material* to refer to any artifact, replication package, dataset, or other resource external to the manuscript PDF (e.g., hosted on Zenodo or Figshare). What belongs in the main body versus an appendix is a presentation choice for authors and venues; what belongs inside the manuscript versus outside it is the reporting-location decision the guidelines make. When a recommendation does not specify a location, either *paper* or *supplementary material* is acceptable.

Navigating These Guidelines: Our guidelines are structured to support different reading strategies depending on the reader's goal. *Researchers planning a new study* may start with the taxonomy of study types (see *Study Types*) to identify which types apply to their planned work, then consult Table 3 to determine which guidelines are requirements (**must**) and which are recommendations (**should**) for those study types. Each guideline section opens with a brief summary, allowing readers to quickly assess relevance before reading the full text. *Researchers writing up results* may start with the checklist in Appendix C, which organizes actionable items by typical paper sections (Introduction, Research Design and Methods, Results, etc.). Table 2 provides a quick reference to all eight guidelines, and Table 4 maps each guideline's rationale to its recommendations. *Reviewers* can use the *Advice for Reviewers* subsection at the end of each guideline for targeted guidance on assessing manuscripts. Table 3 helps reviewers identify which guidelines apply to the study type under review.

Related Reporting Guidelines: Reporting guidelines have a long tradition in healthcare research, where CONSORT [140] is the canonical standard for randomized clinical trials. Our reporting checklist (Appendix C) follows its structural template. LLM-specific reporting guidelines have appeared more recently. Outside SE, these include Gallifant et al's TRIPOD-LLM for healthcare [48], Navarro et al's HCI guidelines [104], and Kapoor et al's REFORMS for ML-based science [76]. Within SE, they include Sallou et al's vision paper on validity threats [132], our earlier workshop position paper [155], and Korn et al's prompt-reporting guideline for automated SE [82], derived from a literature review of ICSE, FSE, and ASE papers and a survey of 105 program committee members. Beyond LLM-specific work, the NeurIPS reproducibility checklist [115] prescribes per-submission disclosure for ML papers, covering items such as algorithm description, experimental setup, and statistical reporting.

Korn et al frame their recommendations as complementing ours. The items their 105 surveyed program committee members endorsed as essential align

with what *Version and Configuration*, *System and Prompt Design*, and *Limitations and Mitigations* already require. Navarro et al's HCI guidelines, by contrast, balance "*the practical realities of authors' time, cost, and page limitations*" with scientific concerns and HCI norms, while our *Guidelines* aim for comprehensive coverage and use **must** and **should** to express that balance. On prompt reporting, Navarro et al argue for selective disclosure based on each prompt's centrality to author claims, whereas *System and Prompt Design* recommends fuller disclosure with named exceptions for privacy, anonymity, or confidentiality concerns. On technical evaluation, Navarro et al recommend a "*modest technical evaluation of the LLM component on a dataset of representative inputs*" tailored to HCI research, whereas *Benchmarks and Metrics* asks for established benchmarks, traditional baselines, and inferential statistics where applicable. Peripheral LLM uses such as proofreading, spell-checking, and translation are entirely out of scope (see *Focus on Direct Development or Research Support* above).

Our guidelines name *paper* or *supplementary material* as the reporting location for each recommendation (see *Reporting Locations* above). In summary, our guidelines apply to SE empirical research broadly, organize recommendations around the taxonomy of seven study types introduced in *Study Types* with explicit per-study-type applicability, and pair each recommendation with a target reporting location in *paper* or *supplementary material*.

3 Methodology

The development of these guidelines was initiated at the 2024 meeting of the *International Software Engineering Research Network* (ISERN) in Barcelona, Spain, where the first and last author organized a session on guidelines for empirical studies involving LLMs. The resulting discussions led to a position paper [155]. A preprint of this paper was the basis for further discussions at the *2nd Copenhagen Symposium on Human-Centered Software Engineering AI* (CHASEAI 2024), where a workstream formed to collaboratively refine and extend the preliminary guidelines. Our process followed an established tradition of developing reporting guidelines through expert collaboration, as exemplified by CONSORT for clinical trials [23] and, more recently, TRIPOD-LLM for LLM-based prediction studies in medicine [48].

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 and extended the guideline sections (in particular *Version and Configuration*, *System and Prompt Design*, *Session Traces*) and added *Benchmarks and Metrics* and *Limitations and Mitigations*.

Then, at least two co-authors were assigned to each study type and guideline to refine and extend the content, followed by a peer review and refinement phase. Afterwards, the first author reviewed all sections and discussed potential changes with the topic leads. A requirement for co-authorship was that each

author contributed or reviewed content and, most importantly, read the complete guidelines and confirmed that they stand behind all recommendations, not only their own contributions.

To select illustrative examples for each study type and guideline, co-author pairs independently searched for relevant papers using academic databases. This initial pool was extended based on suggestions from other co-authors. Identified papers were assessed by one author and subsequently reviewed by a second author; papers that did not add new insights beyond those already covered were not included. Multiple review rounds were then carried out on the guidelines as a whole, during which selected papers were cross-checked by additional authors and, where necessary, further examples were added or replaced. The examples are intended to be illustrative, not the result of a systematic review; the involvement of multiple authors at different stages helped mitigate individual selection bias.

After completing the internal review, we invited external experts in empirical SE to review the study types and guidelines. During the review process of the *Empirical Software Engineering* journal, the structure and content were improved based on reviewers' suggestions. We self-assessed our guidelines against the quality dimensions of the *SIGSOFT Empirical Standards* [118]. This assessment is reported in the conclusion.

4 Study Types

The SE community needs guidelines to establish common standards for designing and reporting empirical studies involving LLMs. However, such guidelines must be tailored to different study types that each pose unique challenges. Therefore, we created a taxonomy of study types to contextualize our recommendations. We use the term *study type* to refer to categories of LLM involvement in empirical research, rather than to research methodologies (e.g., experiments, case studies). A single empirical study may involve multiple such study types. Each study type section starts with a **description**, followed by **examples** from the SE research community and beyond. The **advantages** and **challenges** of using LLMs are discussed in a summary subsection at the end of this section, synthesizing cross-cutting themes across all study types. See Table 1 for an overview of study types.

Table 1 Overview of study types.

Section	Title	Short Name
4.1	**LLMs as Tools for Software Engineering Researchers**	
4.1.1	LLMs as Annotators	Annotators
4.1.2	LLMs as Judges	Judges
4.1.3	LLMs for Synthesis	Synthesis
4.1.4	LLMs as Subjects	Subjects
4.2	**LLMs as Tools for Software Engineers**	
4.2.1	Studying LLM Usage in Software Engineering	Usage
4.2.2	LLMs for New Software Engineering Tools	Tools
4.2.3	Benchmarking LLMs for Software Engineering Tasks	Benchmarking

4.1 LLMs as Tools for Software Engineering Researchers

LLMs can serve as tools to help researchers conduct empirical studies. They can automate various tasks such as data collection, pre-processing, and analysis. For example, LLMs can perform qualitative coding on natural language text such as interview transcripts (*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*). If LLMs could complete these tasks well enough, they would reduce the time and effort required to conduct a study. However, these applications each pose challenges to validity and reproducibility.

4.1.1 LLMs as Annotators

In the annotator role, LLMs perform qualitative coding—the annotation of natural language text such as requirements, interview transcripts, or open-ended survey responses—that researchers would otherwise do by hand.

Description: Coding is a time-consuming manual process [18]. LLMs can augment this process, suggest new codes, and label artifacts based on a predefined coding guide much faster than humans can [61]. This covers both *closed coding*, where the LLM labels artifacts against a predefined coding guide, and *open coding*, where the LLM proposes new codes from the data. In closed coding, LLM labels can be assessed against those of human coders applying the same guide; in open coding, researchers review the resulting codebook for adequacy (e.g., level of abstraction, redundancies between codes). The extent to which, or under what conditions, LLMs can perform these tasks *effectively* remains an open research question [4], and whether they should be used at all for reflexive qualitative analysis is itself contested [74]. Indeed, measuring their effectiveness is practically and philosophically challenging. From an interpretivist philosophical perspective, one cannot measure the quality of analysis by comparing one (human or machine) analyst's work to another. From a realist perspective, triangulating across multiple human judges, LLM judges,

and other data sources (e.g., whether a pull request is marked as having resolved an issue) improves confidence in the findings but does not prove that any one judge is valid.

Examples: Huang et al used multiple LLMs for joint annotation of mobile application reviews [67]. 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). This approach slightly outperformed the best individual model tested. Meanwhile, Ahmed et al examined LLMs as annotators in SE research across five datasets, six LLMs, and ten annotation tasks [4]. They found that inter-model agreement strongly correlates with human-model agreement; models performed poorly in tasks where humans also frequently disagreed. 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.

4.1.2 LLMs as Judges

In the judge role, LLMs rate or rank artifacts on quality criteria, in contrast to the qualitative coding tasks of *LLMs as Annotators*.

Description: As judges, LLMs rate software artifacts along quality criteria (e.g., code readability, adherence to coding standards, comment quality) or rank candidate solutions against such criteria. The scoring rubric is usually embedded in the prompt, with either a numerical scale or a binary verdict.

Examples: Lubos et al used Llama-2 to evaluate the quality of software requirements statements [93]. They prompted the LLM with the text below, where the words in braces 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.
```

They evaluated LLM output against expert human judges and found moderate agreement for simple requirements and poor agreement for more complex requirements. In contrast, Wang et al used an LLM to generate acceptance criteria for user stories, and provided a rubric to an LLM to judge the generated acceptance criteria on interpretable scales (0 to 4) [158].

4.1.3 LLMs for Synthesis

In the synthesis role, LLMs integrate and interpret information from multiple sources to produce higher-level findings such as themes, patterns, or conceptual frameworks. They can also generate synthetic content (e.g., source code, bug-fix pairs, requirements) for downstream training or evaluation.

Description: Unlike annotation (see *LLMs as Annotators*), which focuses on categorizing or labeling individual data points, 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. LLMs may be able to support synthesis tasks in SE research by processing and distilling information from qualitative data sources. Although synthesis in the preceding notion refers to abstraction and interpretation across multiple data sources, the term is sometimes also used to refer to generating synthetic content (e.g., source code, bug-fix pairs, or requirements) that is 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.

Examples: Published examples of applying LLMs for synthesis in SE remain scarce. However, some recent work in other domains is instructive [18]. Barros et al conducted a systematic mapping study on using LLMs for qualitative research [20]. The included studies span fields such as healthcare, education, and cultural studies, where LLMs supported qualitative methods, such as grounded theory and thematic analysis, by aiding in pattern identification. In SE, de Morais Leça et al explored how LLMs have been applied for qualitative data analysis (QDA) and proposed general strategies and guidelines for their application [102]. Ornelas et al complemented this perspective by studying the opportunities and limitations of introducing LLM-based support into QDA, and by formulating recommendations for embedding human–AI collaboration across the thematic analysis phases [110]. Building on these, subsequent work has proposed hybrid frameworks combining LLM support with human-led QDA. Rasheed et al designed an LLM-driven multi-agent system that integrates AI with human decision-making to automate qualitative data analysis methods [121]. Their system generated initial codes, developed themes, and summarized text. Similarly, Montes et al compared the performance of humans and LLMs in coding, theme development, definition, and refinement, creating guidelines for a hybrid-LLM framework [101]. Finally, El-Hajjami and Salinesi's work is an example of using LLMs to create synthetic datasets. They presented an approach to generate synthetic requirements, showing that they "*can match or surpass human-authored requirements for specific classification tasks*" [45].

4.1.4 LLMs as Subjects

In the subject role, LLMs serve as virtual subjects, generating responses or behaviors that an empirical study would otherwise collect from human participants.

Description: In empirical studies, data is collected from participants through methods such as surveys, interviews, or controlled experiments. LLMs can serve as virtual *subjects* by simulating human behavior and interactions. If LLMs can generate responses that approximate those of human participants, they could be valuable for research involving user interactions, collaborative coding environments, and software usability assessments [174]. To achieve this, prompt engineering techniques are widely employed; for instance, the *Personas Pattern* [81] involves tailoring LLM responses to align with predefined profiles or roles that emulate specific user archetypes. To serve as virtual subjects, generated responses should be indistinguishable from human-produced texts, consistent with the attitudes and sociodemographic information of the conditioning context (e.g., junior vs. senior developers), naturally aligned with the form, tone, and content of the simulated scenario, and reflect patterns in relationships between ideas, demographics, and behavior observed in comparable human data [9].

Examples: Xu et al compiled a list of ways LLMs can support social science research, some of which transfer to empirical SE research [168]. For example, LLMs can emulate human responses and behaviors in simulated interviews and focus groups [50]. Similarly, Bano et al investigated biases in LLM-generated candidate profiles in SE recruitment processes [19]. They found biases favoring male candidates, lighter skin tones, and slim physiques, particularly for senior roles. LLMs may be able to simulate end-user feedback and behavior in usability studies, identify usability issues, and offer suggestions for improvement based on predefined user personas.

4.2 LLMs as Tools for Software Engineers

LLM-based assistants have become a widely adopted tool for software engineers [148], supporting them in various tasks such as code generation and debugging. Researchers have studied how software engineers use LLMs (*Studying LLM Usage*), developed new tools that integrate LLMs (*LLMs for Tools*), and benchmarked LLMs for software engineering tasks (*Benchmarking LLMs*).

4.2.1 Studying LLM Usage in Software Engineering

Studies of LLM usage examine how software engineers adopt and integrate LLM-based tools into their workflows.

Description: LLM usage studies focus on real-world settings, outside the controlled conditions of benchmarks and experiments. 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.

Examples: Based on a convergent mixed-methods study, Russo found that early adoption of generative AI by software engineers is primarily driven by compatibility with existing workflows [131]. Khojah et al investigated the use of ChatGPT (GPT-3.5) by professional software engineers in a week-long observational study [77]. 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's case study found that LLMs could "*enhance personalized, instant onboarding support; however, relying on proprietary external LLMs poses significant data privacy risks*" [14]. Jahic and Sami found that most participants from 15 software companies they surveyed had already adopted AI (especially ChatGPT) for SE tasks, but cited low-quality outputs, copyright issues, and the risk of proprietary code leaks as barriers to adoption [71]. 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 [166]. Conversations can then be analyzed using quantitative [116] and qualitative [100] analysis methods.

4.2.2 LLMs for New Software Engineering Tools

In this role, LLMs serve as components of new tools that assist software engineers (e.g., with code comprehension or test generation) or as autonomous agents that perform multi-step tasks on their behalf.

Description: New LLM-based tools support software engineers in their daily tasks, such as code comprehension [169] and test case generation [134]. One way of integrating LLM-based tools into software engineers' workflows is using GenAI agents. Unlike traditional LLM-based tools, these agents are capable of

acting autonomously and proactively, are often tailored to meet specific user needs (e.g., via context files or domain-specific tools), and can interact with external environments (e.g., file systems, shells, or web APIs) [162, 171]. From an architectural perspective, GenAI agents can be implemented in various ways [162], but at their core they run a control loop around the LLM (observe → inspect → choose → act) [120]. Each chosen action (e.g., a tool call) produces a result that the agent feeds back into the next iteration, until the task is complete. *CoALA* [151] offers a conceptual framework for organizing such agents. For coding agents specifically, Raschka identifies building blocks such as the repository context gathered before each call, the constructed prompt and its tool definitions, structured session memory, and delegation to bounded subagents [120]. Because these architectures vary, researchers can test and compare them to study how design choices affect downstream performance.

Examples: Yan et al proposed *IVIE*, a tool integrated into the VS Code graphical interface that generates and explains code using LLMs [169]. The authors focused more on the presentation, providing a user-friendly interface to interact with the LLM. Schäfer et al presented a large-scale empirical evaluation on the effectiveness of LLMs for automated unit test generation [134]. 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 [125]. Takerngsaksiri et al presented *HULA*, a multi-agent system deployed in Atlassian JIRA that lets engineers refine LLM-generated coding plans and source code, and reported acceptance and modification rates from real users [152].

4.2.3 Benchmarking LLMs for Software Engineering Tasks

Benchmarking studies measure LLM performance on standardized SE tasks against reference outputs and shared metrics.

Description: In a benchmark, the reference outputs serve as ground truth, and the metrics measure how well an LLM's outputs match them. Typical tasks include code generation, code summarization, code completion, and code repair [171], 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). Metrics may include general metrics for text generation, such as *ROUGE*, *BLEU*, or *METEOR* [64], or task-specific metrics, such as *CodeBLEU* for code generation. Benchmarking requires high-quality reference datasets.

Examples: In SE, benchmarking may include evaluating an LLM's ability to produce accurate and reliable outputs for a given input (usually a task description, possibly accompanied by data obtained from curated real-world projects or from synthetic SE-specific datasets). *RepairBench* [144], 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. It 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* [73] is a more generic benchmark that contains 2,294 SE Python tasks extracted from GitHub pull requests. To score the LLM's task performance, the benchmark validates whether the generated patch successfully compiles and calculates the percentage of passed test cases. Meanwhile, *HumanEval* [34] is often used to assess code generation.

4.3 Advantages and Challenges

Using LLMs in empirical SE research, whether as tools for researchers or for software engineers, offers several potential advantages but also raises fundamental challenges that cut across the study types described above.

Advantages: The primary advantage of using LLMs for research tasks is *speed, cost reduction, and scalability.* LLMs can annotate, judge, synthesize, and simulate faster and at lower cost than human researchers, with studies showing cost reductions of 50–96% on various natural language tasks [159] (e.g., He et al found that GPT-4 annotation required only two days and 122.08 USD compared to several weeks and 4,508 USD for a comparable MTurk pipeline [61]). Similarly, augmenting or replacing human participants with LLM-generated virtual participants would reduce recruitment effort [94]. This efficiency unlocks scalability: qualitative research traditionally does not scale well to large samples, but LLMs let researchers process more text than human-only coding allows and support larger judgment datasets.

LLMs can also *automate* tasks such as coding qualitative data, assessing artifact quality, and generating summaries, reducing cognitive demands and resources required for qualitative research. Some studies suggest that LLMs may improve *consistency*: ChatGPT's accuracy exceeded crowd workers by approximately 25%, and LLMs can achieve higher inter-rater agreement than crowd workers and trained annotators [52]. However, both human and LLM judges exhibit systematic biases, such as preferring outputs with fake citations or rich formatting [32], and no compelling evidence supports claims that LLMs are less biased than human annotators.

LLMs could potentially provide *access to otherwise-inaccessible research contexts. If* virtual participants' behavior is sufficiently similar to human participants, LLMs could access underrepresented and hard-to-reach populations, strengthen generalizability and inclusiveness, impute missing data

(see *LLMs for Synthesis*), and enable research that is ethically problematic with real humans (e.g., questions that would force a human to relive past trauma do not harm an LLM).

Studying real-world LLM-based tools allows researchers to *understand the state of practice*, uncovering usage patterns, adoption rates, and contextual factors, and *generate hypotheses* about how LLMs affect developer productivity, collaboration, and decision-making. From an engineering perspective, developing LLM-based tools *requires less task-specific engineering* than traditional SE approaches such as static analysis or symbolic execution, because a single model can handle diverse inputs without language-specific parsers or hand-crafted rules. Good benchmarks provide *standardized evaluation and model comparison*, reducing research effort and supporting open science by providing common ground for sharing data and results. Benchmarks built for specific SE tasks can help identify LLM weaknesses and support optimization and fine-tuning.

Challenges: The stochastic nature of LLM responses, where identical prompts may yield different outputs, *undermines replicability* across all study types, complicating interpretation of experimental results and test-retest reliability. More broadly, *reliability* is a persistent concern: Like any measurement instrument, *LLMs as Judges* and *LLMs as Annotators* should exhibit validity, test-retest reliability, inter-rater reliability, minimal error, and measurement invariance [119]. However, LLMs show substantial variability depending on the dataset and task [21, 111]. They are sensitive to prompt variations [122] and option order [114], can behave differently when reviewing their own outputs [112], and can be unreliable for high-stakes labeling [160]. Crucially, *reliability does not imply validity*: a reliable LLM might be reliably inaccurate [175]. For tasks with no single correct answer, the statistical framework pushes outputs toward the most likely answer, which may not be the best one [79, 173]. See *Version and Configuration* and *Limitations and Mitigations* for reporting guidance on replicability and reliability.

LLMs and LLM-based tools *evolve rapidly*, and so do practitioners' workflows around them. This combination complicates longitudinal comparisons and can quickly render study findings obsolete. Findings tied to a specific model version may not transfer to later releases even within the same model family, complicating reproducibility and longitudinal comparisons.

The prevalence of *proprietary tools and opaque training data* limits researchers' ability to assess and mitigate biases, and enables benchmark contamination [5]. LLMs exhibit *bias* in multiple forms: tendencies to overestimate certain labels [176], well-documented fairness issues [47], and the potential to reinforce prejudices. When simulating human participants, LLMs are "*likely to both misportray and flatten the representations of demographic groups*" [156]. See *Open LLM* and *Limitations and Mitigations* for mitigation strategies.

Evaluation difficulty is inherent in studying LLM-based tools and benchmarks. Benchmarks may lack construct validity [119], usually do not capture the full complexity of software engineering work [30], and may

lead to *overconfidence* and overfitting [16]. LLMs are also susceptible to adversarial manipulation where semantics-preserving changes may deceive them into accepting flawed artifacts [60]. See *Benchmarks and Metrics* for detailed guidance on benchmark design, including Cao et al's guidelines for coding task benchmarks [29].

A fundamental challenge is *philosophical and methodological incongruence* with qualitative research. In a recent open letter, 419 experienced qualitative researchers argued "*that analytic approaches such as reflexive thematic analysis are human research practices requiring a subjective, positioned, and reflexive researcher and therefore the use of GenAI in such approaches is not methodologically congruent*" [74]. LLMs lack the capacity for genuinely reflexive qualitative analysis because they operate on statistical prediction without understanding the meaning of the data being analyzed [17, 101]. Effective use requires structured prompts and careful human oversight [20]. See *Human Validation* and *Limitations and Mitigations* for guidance.

There is *insufficient evidence* for the effectiveness of LLMs in most research roles. No compelling evidence exists that LLMs can accurately simulate human participants [137] or reliably judge most relevant properties of SE artifacts. Gathering such evidence for each specific usage may be quite difficult [58]. LLMs may be better suited for augmenting rather than replacing human researchers [61, 159], but even then, limited evidence exists that augmentation increases *effectiveness* as well as *efficiency*. When LLM outputs are incorrect, they can negatively *affect human judgment* [66]. See *Human Validation* for mitigation strategies.

Majority voting across multiple outputs improves reliability [122] but increases *cost and environmental impact*. While open models are available, the most capable ones require substantial hardware; relying on *cloud-based APIs* introduces concerns related to data privacy, security, and replicability. See *Limitations and Mitigations* for sustainability considerations.

Field study findings face *generalizability* challenges because outcomes may be highly sensitive to individual differences, usage patterns, goals, and contexts. Field studies must be "*dependable*" [150] beyond traditional validity criteria, which complicates methodology; see *Limitations and Mitigations* for a detailed threat taxonomy.

5 Guidelines

A primary goal of our guidelines is to *enable reproducibility and replicability* of empirical SE studies involving LLMs. As repeating LLM-focused research to verify results lies somewhere between the ACM's definitions of reproducibility (different team, same research artifacts) and replicability (different team, different research artifacts) [1] due to potential model changes and incomplete research artifacts (e.g., unreported prompts, configurations, or seeds), we follow Angermeir et al and use the terms interchangeably [6]. While previous guidelines regarding open science and empirical studies still apply, LLM-

specific characteristics (e.g. inherent non-determinism [146, 172], opaque and proprietary models) present additional replicability challenges, which, in turn, demand new guidance.

Each guideline below begins with a brief **summary**, followed by its **rationale**, **recommendations**, **examples**, **benefits**, and **challenges**, with links to the relevant **study types**. The *rationale* articulates the underlying principle, i.e., why the guideline matters, while the *recommendations* provide concrete, actionable practices. Table 4 in the appendix provides a compact overview of this mapping.

The guidelines further contain **advice for reviewers** and close with a **see also** list linking related guidelines. Broadly, reviewers should use our guidelines to help them interrogate the extent to which a manuscript's authors have done what was practically possible to improve reproducibility. We must neither accept research absent reasonable efforts to improve reproducibility, nor reject research for failing to obtain an impossible goal. We borrow this principle of "*reasonable efforts*" from the SIGSOFT Empirical Standards [118], where it applies to methodological rigor more generally. Of course, papers should acknowledge their limitations, but to determine whether these limitations are reasonable, reviewers should ask "have the authors done what they could to minimize limitations?" Our guidelines attempt to capture what "*reasonable efforts*" practically means for LLM-based studies.

To distinguish essential criteria from recommendations, our guidelines use two tiers. A **must** criterion is a *requirement*. Studies that intend to follow these guidelines are expected to meet all **must** criteria. A **should** criterion represents a desired practice that strengthens a study's rigor or transparency. However, there may be valid reasons to deviate in particular circumstances (e.g., resource constraints, inapplicability to a specific study context or type). Nonetheless, authors should briefly justify any deviation from a **should** criterion that could compromise a study's validity or reproducibility. The distinction between **must** and **should** also reflects *what* the criterion mandates, not only its severity. A **must** criterion is typically a disclosure obligation such as reporting model versions, publishing prompts, describing architectures, and discussing limitations, so that readers can independently evaluate the choices authors made. A **should** criterion is typically a methodological recommendation such as which baselines to include and which validation strategies or statistical analyses to apply. Table 2 lists our eight guidelines. Table 3 in the appendix provides an overview of how each guideline applies to each study type. Cells marked ● indicate that the guideline is a **must** requirement for the study type and cells marked ○ indicate that the guideline **should** be followed. Cells marked – indicate that the guideline is typically not applicable or not feasible for the study type.

The following sections indicate which information we expect researchers to report, and whether it should be in the *paper* or *supplementary material*. Where a publication venue's page limits hinder reporting all expected elements in the *paper*, it is better to report essential information in the *supplementary*

Table 2 Overview of guidelines.

Section	Title	Short Name
5.1	Declare LLM Usage and Role	Declare Usage
5.2	Report Model Version, Configuration, and Customizations	Model Version
5.3	Report System and Prompt Design	Design
5.4	Report Session Traces	Traces
5.5	Use Suitable Baselines, Benchmarks, and Metrics	Benchmarks & Metrics
5.6	Use an Open LLM as a Baseline	Open LLM
5.7	Use Human Validation for LLM Outputs	Human Validation
5.8	Report Limitations and Mitigations	Limitations

material than not at all. The *supplementary material* **should** be published according to the *ACM SIGSOFT Open Science Policies* [55].

5.1 Declare LLM Usage and Role

> *Summary*: Researchers **must** disclose any use of LLMs to support empirical studies, specifying which LLM was used, how it was used, and where in the research process it was employed. This disclosure **should** appear in a suitable section of the *paper*. They **should** report the exact purpose, the tasks that were automated, and the expected benefits in the *paper*. When the LLM is central to the study, the declaration **should** be prominent and detailed in the methodology section; for tangential uses, a brief statement in the methodology section suffices.

5.1.1 ***Rationale:***

Transparency about LLM involvement is a prerequisite for informed assessment of a study's scope, limitations, and potential biases. Without explicit disclosure, readers cannot evaluate how the LLM's characteristics may have influenced the research process or its outcomes.

5.1.2 ***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; Cheng et al argue specifically for the methods section, since acknowledgments are easily missed at the end of a paper [36]. The ACM Policy on Authorship requires authors to describe in the methods section any use of AI in the research itself [11].

Beyond generic declarations, 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*. A sufficient declaration specifies not only *that* an LLM was used, but also *which* LLM (name and version), *how* it was used (e.g., as an annotator, code generator, or judge), and *where* in the research process it was employed (e.g., data collection, analysis, or synthesis).

When the LLM is central to the study (e.g., as the main tool being evaluated or as a core component of the research method), the declaration **should** be prominent and detailed, appearing in the methodology section with cross-references to the specific guidelines that apply (e.g., Sections *Version and Configuration*, *System and Prompt Design*, and *Session Traces*). When the LLM's role is more tangential (e.g., used for a single preprocessing step), a brief but explicit statement in the methodology section is sufficient. When a study assigns multiple distinct roles to LLMs (e.g., one model generates evaluation data while another scores outputs), each role **should** be declared separately. In each case, the disclosure **must** be specific enough for readers to assess how the LLM's involvement may affect the study's validity and reproducibility.

5.1.3 ***Examples:***

The *ACM Policy on Authorship* requires reporting AI-generated artifacts such as code, datasets, and figures where they underlie a study's conclusions [11]. Reporting in the methods section keeps this disclosure visible under double-blind review, where end-of-paper acknowledgments are often removed. An example of an LLM disclosure beyond writing support can be found in a recent paper by Lubos et al [93], 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..."*

A more contemporary declaration could similarly state:

> *"We used Claude Opus 4.7 via the Anthropic API to synthesize themes from interview transcripts, with all prompts and conversation logs published as supplementary material."*

Golnari et al's DevBench paper illustrates separate disclosure of multiple LLM roles: *GPT-4o* generated the synthetic benchmark instances, nine models (including *Claude 4 Sonnet*, *GPT-4.1*, *DeepSeek-V3*, and *Ministral-3B*) were the evaluation subjects, and *o3-mini* was the LLM judge that scored completions for relevance and helpfulness [53].

5.1.4 ***Benefits:***

Transparency in the use of LLMs helps other researchers understand the context and scope of the study, supporting interpretation and comparison of the results. Realizing these benefits requires reporting the LLM's exact role and version (see *Version and Configuration*).

5.1.5 ***Challenges:***

Declaring LLM usage requires only a brief statement and no additional experiments, making compliance 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. In fact, there is evidence suggesting that AI disclosure can negatively affect trust in authors [135]. However, the *ACM Policy on Authorship* requires disclosure of AI used in the research itself, not AI used only to assist with writing [11]. Our guidelines focus on such use (see *Scope*), not on proofreading or writing support.

5.1.6 ***Study Types:***

Researchers **must** follow this guideline for all study types. The specific focus of the declaration varies by study type. For *LLMs as Annotators*, *LLMs as Judges*, *LLMs for Synthesis*, and *LLMs as Subjects*, researchers **must** declare the specific role assigned to the LLM (e.g., annotator, judge, synthesizer, or simulated participant). For *Studying LLM Usage*, researchers **must** clarify which LLM(s) the observed participants used and under which conditions. For *LLMs for Tools*, researchers **must** declare the LLM's role within the tool architecture and its contribution to the tool's functionality. For *Benchmarking LLMs*, researchers **must** declare which LLMs were benchmarked and for which tasks.

5.1.7 ***Advice for Reviewers:***

The most common problem with disclosure is incompleteness or vagueness about how the LLM was used. If the paper says "we used LLM X to help with task Y" without specifying how, reviewers should request clarification. Such requests are typically minor revisions unless the missing details may reveal methodological problems.

If undisclosed LLM use is suspected, the reviewer should consult their editor or program chair. When the evidence is conclusive, the key question is the degree to which undisclosed use affects the study's contribution, ranging from negligible (e.g., word choice in a single sentence) to severe (e.g., generating the reported data or results).

5.1.8 ***See Also:***

→ Section 5.2 (Report Model Version, Configuration, and Customizations): The disclosure is incomplete without naming the specific model and version.
→ Section 5.3 (Report System and Prompt Design): When the LLM lives inside a tool or agent, authors must also describe that tool's architecture and prompts.

→ Section 5.4 (Report Session Traces): Session traces show what the LLM did during the study.

5.2 Report Model Version, Configuration, and Customizations

Summary: Researchers **must** report the exact LLM model or tool version, configuration, and date of study execution in the *paper*. When using quantized models, researchers **should** report the quantization level and method. For fine-tuned models, they **must** describe the fine-tuning goal, dataset, and procedure in the *paper*. 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 as *supplementary material* (or alternatively justify in the *paper* why they cannot share them).

5.2.1 ***Rationale:***

LLMs and LLM-based tools are frequently updated, and configuration parameters such as temperature or seed values affect content generation. This guideline focuses on documenting the *model-specific* aspects of empirical studies involving LLMs, concentrating on the models themselves, their version, configuration parameters, and customizations (e.g., fine-tuning). While the *System and Prompt Design* section addresses system-level integration and the authored artifacts (including prompts) that the model uses on each call, the information outlined here is essential for reproducibility whenever an LLM is involved.

5.2.2 ***Recommendations:***

Researchers **must** document in the *paper* which model or tool version they used in their study, along with the date of study execution and the parameters they configured that affect output generation. Since default values might change over time, researchers **should** report all configuration values, even if they used the defaults. Checksums and fingerprints **should** 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) **should** be reported. When using quantized models, researchers **should** report the quantization level (e.g., 4-bit, 8-bit) and method (e.g., GPTQ or AWQ), as different quantization approaches produce different outputs, affecting both output quality and reproducibility. Researchers **should** motivate in the *paper* why they selected certain models, versions, and configurations. Reasons may be monetary, technical, 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.

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 versus Low-Rank Adaptation (LoRA), selected hyperparameters, loss function, learning rate, and batch size), 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.

In summary, researchers **must** report in the *paper* at minimum (1) the exact model or tool name and version, (2) all parameters they configured that affect output generation, (3) the date of study execution, and, for fine-tuned models, (4) the fine-tuning goal, dataset characterization, approach (e.g., full fine-tuning vs. LoRA), and hyperparameters. Beyond these requirements, researchers **should** additionally report default parameter values, checksums or fingerprints, model properties relevant to the study (e.g., context-window size), and quantization level and method where applicable. For fine-tuned models, they **should** also share the dataset and model weights as *supplementary material* (unless legal or privacy constraints prevent disclosure) and report validation metrics and benchmarks.

Commercial models (e.g., GPT-5) or LLM-based tools (e.g., ChatGPT) might not give researchers access to all required information. For these tools, researchers **should** report what is available and openly acknowledge limitations that hinder reproducibility.

5.2.3 ***Examples:***

Based on the documentation that OpenAI and Azure provide [98, 107], 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. The system fingerprint was* `fp_6b68a8204b`*.*

Kang et al provide a similar statement in their paper on exploring LLM-based bug reproduction [75]:

> *"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 recommend reporting a checksum/fingerprint and exact dates; otherwise, this example is close to our recommendations.

Dhar et al assessed whether LLMs can generate architectural design decisions [40], detailing the system architecture and the LLM's role within it. They provide information on the fine-tuning approach and datasets, including the source of architectural decision records, preprocessing methods, and data selection criteria.

For self-hosted models, the *supplementary material* can become a true replication package. For example, for models provisioned using ollama, one can report the specific tag and checksum, e.g., *"llama3.3, tag 70b-instruct-q8_0, checksum d5b5e1b84868."* Given suitable hardware, running the model is then as easy as executing the following command:

```
ollama run llama3.3:70b-instruct-q8_0
```

5.2.4 ***Benefits:***

Reporting the model version, configuration, and date of study execution is a prerequisite for the verification and replication of LLM-based studies. While LLMs are inherently non-deterministic, this cannot excuse dismissing reproducibility. Although exact reproducibility is hard to achieve, the recommendations above help researchers come as close as possible to that standard.

5.2.5 ***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, 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*" [106]. However, the seed value does not allow for full reproducibility and the fingerprint changes frequently. Although, as motivated above, open models substantially 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). Setting the temperature to 0 configures greedy decoding (always selecting the most probable next token), which minimizes output variability but can degrade quality by producing repetitive text and missing higher-quality responses [62].

Even with a temperature of 0, full determinism is rarely guaranteed. Floating-point arithmetic on GPUs causes slight numerical differences that cascade into divergent token selections [172], and Sparse Mixture-of-Experts routing amplifies this effect [31]. Silent backend changes in commercial APIs produce different outputs over time [33], and even self-hosted open models with identical settings do not always yield consistent outputs [12]. Researchers **should not** treat a temperature of 0 as a guarantee of reproducibility, but as one measure among several, including fixed seed values [106], system fingerprints, and archiving of raw outputs. When a temperature of 0 is chosen

primarily for reproducibility, this motivation **should** be stated explicitly, along with an acknowledgment of its potential impact on output quality.

5.2.6 ***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 system and prompt design (see *System and Prompt Design*), session traces (see *Session Traces*), and specific limitations and mitigations (see *Limitations and Mitigations*).

For example, when *Studying LLM Usage* 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 of study execution **must** always be reported. See *System and Prompt Design* for prompt reporting and *Session Traces* for interaction logs.

For *LLMs as Annotators*, *LLMs as Judges*, and *LLMs for Synthesis*, researchers **must** report the model configuration used for the respective annotation, judging, or synthesis tasks, including temperature and other sampling parameters that affect output variability. For *LLMs as Subjects*, researchers **must** report any persona-related configuration settings and parameters that configure the simulated behavior. For *LLMs for Tools*, researchers **must** report the configuration for each model integrated in the tool's architecture, including any model-specific parameter choices. For *Benchmarking LLMs*, researchers **must** report the configuration for all benchmarked models to enable fair cross-model comparisons.

5.2.7 ***Advice for Reviewers:***

Missing version, configuration, or parameter information is typically a minor revision request. Before concluding that information is absent, reviewers should check appendices and supplementary materials, as details are sometimes reported there rather than in the main text. Rejection over missing details is rarely warranted unless the omissions obscure deeper methodological problems.

5.2.8 ***See Also:***

→ Section 5.3 (Report System and Prompt Design): Beyond the model itself, authors must also document the architecture and prompts that use it.
→ Section 5.4 (Report Session Traces): Session traces show what the reported model and configuration produced at runtime.
→ Section 5.5 (Use Suitable Baselines, Benchmarks, and Metrics): Benchmark comparisons require identifying the exact model version under test.

→ Section 5.6 (Use an Open LLM as a Baseline): An open model gives full version visibility, which some commercial products do not.
→ Section 5.8 (Report Limitations and Mitigations): Hidden or shifting commercial versions become reproducibility threats in their own right.

5.3 Report System and Prompt Design

Summary: Researchers **must** describe in the *paper* the full architecture of LLM-based tools, from standalone uses to agentic systems, including the LLM's role and its interactions with other components. Hosting and access **should** be described; for time-sensitive measurements, researchers **must** clarify whether local infrastructure or cloud services were used. Researchers **must** publish all prompts as *supplementary material*, including prompt templates with representative instances and the dynamic generation process where applicable, and **must** include representative examples in the *paper*; sensitive content **must** be anonymized. If full prompt disclosure is not feasible, summaries or examples **should** be provided. The prompting strategy and prompt reuse across models and configurations **must** be specified, and how the prompts were developed **should** be described; for few-shot prompts, how the examples were selected **must** be explained in the *paper*. Researchers **must** describe the configuration mechanisms used (e.g., context files, skills, subagents) and summarize in the *paper* which tools were exposed; all configuration artifacts and the complete tool catalog (schemas, definitions, Model Context Protocol (MCP) servers) **should** be published as *supplementary material*. For agentic systems, researchers **must** specify the agents' roles, reasoning frameworks, and communication flows; where external tools are used, the model's reasoning, tool calls, and interactions with users or the environment **must** be reported separately. For retrieval-augmented generation (RAG) and similar methods, researchers **must** describe how external data was retrieved, stored, and integrated. Where legally possible, the implementation **should** be open-sourced; non-disclosed proprietary components **must** be acknowledged as reproducibility limitations.

5.3.1 ***Rationale:***

LLM-based studies rest on artifacts that researchers design, author, or configure before any model is invoked: *software layers* that pre-process data, prepare prompts, filter user requests, or post-process responses, and the *context* those layers feed into each call [49]. Context includes prompt templates, context files, tool and skill schemas, and retrieval mechanisms that bring external data into model invocations. For example, ChatGPT and GitHub Copilot use the same underlying models, but their outputs differ substantially because

Copilot automatically adds project context. Researchers can also build tools using models directly via APIs.

Prompts are central to any LLM-based study [142]. A *prompt* is a concrete input to an LLM that guides its output [139]. Depending on the task, a prompt may include instructions (e.g., "*classify the following bug report*"), task context, input data, and output format specifications (e.g., "*respond as JSON with fields 'category' and 'justification'*"), with outputs ranging from unstructured text to structured formats such as JSON [39]. A *prompt template* is a parameterized structure containing static elements (e.g., instructions, output format specifications) and placeholders for variable content (e.g., source code under analysis) that are filled in at runtime to construct concrete prompts [139]. In automated studies, researchers typically design prompt templates from which individual prompts are then instantiated. Prompts substantially influence a model's output, so how they are formatted and integrated into an LLM-based study is essential for transparency, verifiability, and reproducibility. Sclar et al question the methodological validity of comparing models with "*an arbitrarily chosen, fixed prompt format*", because their research has shown that the performance of different prompt formats only weakly correlates between different models [142].

This guideline covers the architecture, prompts, and configuration used in an LLM-based study, complementing *Version and Configuration* (model-specific details) and *Session Traces* (runtime behavior). It does not apply when LLMs are used solely for language polishing, paraphrasing, translation, tone or style adaptation, or layout edits (see *Scope*).

5.3.2 ***Recommendations:***

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, including any dependencies on proprietary tools that affect reproducibility. Researchers **should** justify substantive architectural choices where alternatives existed (e.g., why a particular agentic framework or tool catalog was selected). Researchers **should** describe how the models were hosted and accessed. For *time-sensitive measurements*, the hosting choice (e.g., self-hosting on local hardware, an aggregator such as OpenRouter, or a vendor API such as the OpenAI API) can substantially affect results, so researchers **must** clarify whether local infrastructure or cloud services were used, including the specific hardware for local hosting (e.g., GPU model and VRAM) or the service tier for cloud APIs (latency reporting is covered under *Benchmarks and Metrics*). Where legally possible (e.g., not restricted by industry-partner agreements), researchers **should** release the source code of their implementation under an open-source license.

Reporting requirements scale with system complexity: minimal for standalone LLMs, detailed for pipelines, agentic systems, and any context files or tool schemas they depend on. In the topic-specific paragraphs that follow, the *paper* **must** contain a high-level description of any reported component, with full details provided as *supplementary material*.

Prompt Reporting. Researchers **must** report all prompts used in an empirical study, including instructions, task context, input data, and output indicators. The complete set **must** be made publicly available as *supplementary material*, with representative examples in the *paper* itself. When confidentiality (e.g., industry-partner agreements) prevents full publication, researchers **should** publish summaries and representative examples instead. For prompts that can be partially shared, researchers **must** anonymize personal identifiers, replace proprietary code with placeholders, and clearly highlight modified sections. When prompt templates are used, researchers **must** report them alongside representative instances, specifying which parts are static and which are dynamically filled. When prompts are generated dynamically, researchers **must** document the code or rules that assemble each prompt from runtime inputs. Researchers **should** specify the exact formatting of prompts, including how code snippets were enclosed (e.g., triple backticks), whether whitespace was preserved, and how other artifacts such as error messages and stack traces were formatted. For studies involving human participants who create or modify prompts, researchers **should** describe how these prompts were collected and analyzed.

Prompt Development. Researchers **should** 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, researchers **should** at least summarize the prompt evolution. 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 and how these suggestions were incorporated. A prompt changelog can track prompt evolution, including revisions and reasons for changes (e.g., v1.0: initial prompt; v2.0: added few-shot examples). Because 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.

Prompting Strategy and Input Handling. Researchers **must** specify whether zero-shot, one-shot, or few-shot prompting was used. For few-shot prompts, researchers **must** explain in the *paper* how the examples were selected and **should** include the concrete examples in the *supplementary material*. 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 **should** describe the strategies they used to handle input length constraints (e.g., truncating, summarizing, or splitting prompts into multiple parts). Token optimization measures, such as simplifying code formatting or removing unnecessary comments, **should** also be documented if applied.

Pipelines and Complex Systems. If the LLM is used in a *standalone setup*, with prompts sent directly to a model via an API and no pre-processing of

inputs or post-processing of outputs, researchers **must** state this explicitly. If the LLM is part of a *complex system* (e.g., with pre-processing or post-processing stages), researchers **must** describe each component's role and the data flow between them. For systems using retrieval-augmented generation (RAG) or related methods (e.g., rule-based retrieval, structured query generation, or hybrid approaches), researchers **must** additionally describe how external data was retrieved, stored (e.g., in vector databases, knowledge graphs), and selected for inclusion in the model's context. The data used for retrieval **should** also be reported, including its preprocessing, versioning, and update frequency. If not confidential, an anonymized snapshot **should** be made available as *supplementary material*. For *ensemble models*, in addition to following the *Version and Configuration* guideline for each model, researchers **should** describe the architecture connecting them: the routing logic that determines which model handles which input, model interactions, and the output combination strategy (e.g., majority voting, weighted averaging, sequential processing).

Agentic Systems. If the LLM is part of an *agentic system* that autonomously plans or executes tasks, researchers **must** additionally describe the agents' roles (e.g., planner, executor, coordinator), whether the system is single-agent or multi-agent, how the agents interact with external tools and users, and the reasoning framework used (e.g., chain-of-thought, self-reflection, multi-turn dialogue). For agentic systems that use external tools (e.g., Claude Code and its subagents), researchers **must** distinguish three kinds of activity: (1) the model's reasoning, planning, and outputs; (2) tool calls (e.g., to APIs, databases, file systems, or Model Context Protocol (MCP) servers); (3) interactions with users, the environment, or other agents. Reporting these separately lets readers understand whether a result came from the model, a tool, or their interaction. How to record the runtime traces of each is covered in *Session Traces*.

Context Files and Agent Configuration. Researchers can tailor agentic tools through *configuration mechanisms* such as context files, skills, subagents, hooks, settings, and rules [49]. A *configuration artifact* is a concrete instance of such a mechanism: a single file (e.g., a context file or a subagent file) or a directory bundling several files (e.g., a skill folder containing `SKILL.md` alongside scripts, references, and assets). Since configuration mechanisms steer agent behavior in the same way as system prompts, researchers **must** describe which configuration mechanisms were used and **should** publish all configuration artifacts as *supplementary material*. Configuration mechanisms and their artifacts **must** be reported with the same level of detail as general prompts, including their development process and any iterations. Because context files are version-controlled, their evolution across a study is recoverable from the project's Git history and from the runtime traces reported under *Session Traces*. Where subagents are used, researchers **should** also describe the delegation pattern, including which subagent handles which task and how control is returned to the orchestrator.

Tool Catalog and MCP Servers. On each call, agentic systems expose a set of tools to the model along with their schemas (e.g., parameter names, types, and natural-language descriptions). Tools differ in granularity. An editor agent might expose one tool per editing operation (e.g., `read_file`, `apply_edit`), while Claude Code defines a single `Bash` tool (`PowerShell` on Windows) through which the agent runs arbitrary shell commands such as `grep` or `ls` [8]. The catalog, the wording of descriptions, and the order in which tools are presented all influence which tool the model selects, so the exact serialized form matters for reproducibility. The same applies to the list of MCP servers made available to the model. Researchers **must** summarize in the *paper* which tools were exposed to the model, and **should** include a complete list with names and purposes, tool schemas, and the names of any connected MCP servers as *supplementary material*.

5.3.3 ***Examples:***

Schäfer et al evaluated LLMs for automated unit test generation, providing a detailed description of the system architecture including code parsing, prompt formulation, LLM interaction, and test suite integration [134]. They also detailed the datasets used, including sources, selection criteria, and preprocessing steps.

A second example is Yan et al's *IVIE* tool [169], 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.

Liang et al's paper is a good example of comprehensive prompt reporting [89]. The authors make the exact prompts available in their *supplementary material* on Figshare, including details such as code blocks enclosed in triple backticks. The *paper* explains the rationale behind the prompt design and the data output format, and it includes an overview figure and two concrete examples, keeping the main text concise while remaining reproducible.

5.3.4 ***Benefits:***

Documenting the tool architecture and hosting infrastructure of LLM-based systems strengthens reproducibility and transparency, enabling experiment replication, result validation, and cross-study comparison. Prompt documentation provides similar benefits, letting other researchers replicate studies, refine prompts, and evaluate how content and formatting choices influence LLM behavior. Reporting configuration mechanisms, the tool catalog, and MCP servers additionally lets other researchers reconstruct the configured context, often the dominant factor in agent behavior.

5.3.5 ***Challenges:***

Documenting LLM-based architectures involves challenges such as proprietary APIs and dependencies that restrict disclosure, managing large-scale retrieval databases, and ensuring efficient query execution. Researchers must also balance transparency with data privacy concerns, adapt to the evolving nature of LLM integrations, and handle the complexity of multi-agent interactions and decision-making logic, all of which can impact reproducibility and system clarity. Prompts themselves are challenging to document because they often combine multiple components such as code, error messages, and explanatory text, and privacy or confidentiality concerns can hinder sharing.

Not all systems allow reporting of complete (system) prompts, context files, and tool schemas. For commercial tools, researchers **must** report all available information and acknowledge unknown aspects as limitations. Disclosure practices vary across vendors: some publish their system prompts, others keep them proprietary. Where prompts are published, they also change between releases [163], so researchers **should** record the tool version and date of use even when the prompt content itself is unavailable. 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, which is a challenging context to report. One solution is 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. We also recommend exploring open-source tools such as *OpenCode* [108], which expose more of the configuration that controls agent behavior.

5.3.6 ***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 agentic architectures. It also **must** be followed by all studies that use concrete prompts or prompt templates.

For *LLMs for Tools*, this guideline is of primary importance: researchers **must** describe the tool's full architecture, explain how prompts were generated and structured within the tool, report the context files, tool catalog, and skill definitions used, and document how the role of each LLM fits into the overall system behavior. For *Benchmarking LLMs*, researchers **must** describe the evaluation harness and infrastructure when it goes beyond bare model API calls (e.g., custom sandboxing, orchestration layers, or post-processing pipelines), and the harness **should** be usable with open models; see [7] for a practitioner account of evaluation harness design for agentic systems. Researchers using pre-defined prompts (e.g., *HumanEval*, *SWE-Bench*) **must** specify the benchmark version and any modifications made to the prompts or evaluation setup. If prompt tuning, RAG, or other methods were used to adapt prompts, researchers **must** disclose and justify

those changes, and **should** make the relevant code publicly available. For *Studying LLM Usage*, researchers **should** describe the tool architecture of the studied tool to the extent that it is accessible, as architectural details may influence observed usage patterns. For controlled experiments under *Studying LLM Usage*, exact prompts **must** be reported for all conditions. 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** report the evaluation criteria, scales, and examples embedded in the prompt to ensure 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 *LLMs as Annotators*, *LLMs as Judges*, *LLMs for Synthesis*, and *LLMs as Subjects*, if the research setup involves a custom pipeline (e.g., RAG for annotation or chained prompts for synthesis), the architecture **should** also be reported.

5.3.7 ***Advice for Reviewers:***

As with other guidelines, missing architectural or prompt information is typically a minor revision request unless so much is missing that methodological rigor cannot be assessed. Reviewers may ask authors to move key details from supplements into the paper body to ensure the main text is self-contained.

Regarding proprietary or confidential components, three principles apply: (1) empirically evaluating an opaque artifact is a valid scientific contribution; (2) the less available and inspectable the artifact, the weaker the contribution; (3) *scientific instruments* including tools, metrics, scales, and experimental materials must be fully disclosed, even when the object under study cannot be.

A challenging aspect of prompt reporting is the description of prompt development, which is inherently iterative and creative. Reviewers should *not* expect justification of each word choice or post-hoc rationalized accounts of prompt generation. Instead, reviewers should focus on whether (1) a new research team could *use* (not reproduce) exactly the same prompts in exactly the same way, and (2) potential biases or validity issues are transparent.

5.3.8 ***See Also:***

→ Section 5.2 (Report Model Version, Configuration, and Customizations): Every architecture runs on at least one specific model; name its version.
→ Section 5.4 (Report Session Traces): Architecture and prompts are static; session traces show how they behave at runtime.
→ Section 5.7 (Use Human Validation for LLM Outputs): Human validation often complements automated metrics for tool outputs.

→ Section 5.6 (Use an Open LLM as a Baseline): Open models let other researchers run the reported prompts and schemas on the exact same model weights.
→ Section 5.8 (Report Limitations and Mitigations): When a tool hides internal prompts or schemas, authors must report the gap as a limitation.

5.4 Report Session Traces

Summary: To address model non-determinism and ensure reproducibility, especially when targeting SaaS-based commercial tools, researchers **should** include full interaction logs (prompts and responses) as *supplementary material* if privacy and confidentiality can be ensured. Traces containing sensitive information **must** be anonymized. For agentic systems, interaction logs **should** cover human-in-the-loop exchanges with the agent, including feedback and approval decisions. Researchers **should** report the complete runtime trace as *supplementary material*, covering both external tool invocations (tool name, arguments, result, ordering) and which configured artifacts (skills, context files, subagents reported under *System and Prompt Design*) were activated, so that readers can attribute task outcomes to the model, the tools, or their interaction pattern. Where tool-native trace formats are used, the file format and tool version **must** be described; an open format with a documented schema **should** be preferred. Developed plans **should** be reported as *supplementary material* if available. When full trace disclosure is not feasible, representative or anonymized examples **should** be provided, and unobservable aspects of commercial-tool runs **must** be acknowledged as reproducibility limitations.

5.4.1 ***Rationale:***

A *session* is any bounded period of activity during which an LLM is invoked. Sessions cover one-shot prompts, batch runs, multi-turn conversations, and agentic runs that span many tool calls. A *session trace* records everything that crosses the LLM boundary or is produced around the model during that period: prompts received, responses returned, tool calls the model made together with their arguments and results, plans the model developed, and which of the statically configured artifacts (e.g., skills, context files, subagents, tools) were actually picked up at runtime.

Two kinds of session trace are important to capture. An *interaction log* captures what a human can observe at the LLM's interface: the prompts sent in and the responses returned, whether the prompts come from a human user or, in non-interactive runs, from a calling harness. A *runtime trace* records the LLM's internal activity: each call to an external tool (e.g., APIs, file systems, databases, MCP servers, subagents) or activation of a configured artifact (e.g., context files, skills), with the tool or artifact identified, the arguments passed,

and the result returned. For agentic runs, both kinds matter, because neither tells the full story on its own.

Even with the exact same prompts, decoding strategies, and parameters, LLMs can behave non-deterministically. Non-determinism can arise from probabilistic sampling and, even with greedy decoding (temperature = 0), from batching and floating-point arithmetic on GPUs [172], and from Mixture-of-Experts routing [31]. Verifying conclusions drawn from such interactions therefore depends as much on runtime traces as on the system design itself. This matters particularly for studies targeting commercial software-as-a-service (SaaS) solutions such as ChatGPT, and for agentic runs where behavior depends on how the model chose among many possible tool calls.

The rationale is similar to reporting interview transcripts in qualitative research. Just as a human participant might give different answers to the same question asked two months apart, the responses from tools such as ChatGPT can also vary over time, and the trace of an agent's decisions on any given run is often not reproducible at a later date. This guideline addresses runtime reporting and complements *System and Prompt Design*, which covers the static artifacts that determine the model's input.

5.4.2 ***Recommendations:***

Interaction Logs. Researchers **should** report the full interaction logs (prompts sent to the LLM and responses returned) as part of their *supplementary material*. For agentic systems, interaction logs cover the human-facing exchanges with the agent, including human-in-the-loop feedback, approval or rejection decisions, and iterative refinements. These **should** also be reported as *supplementary material* so that readers can reconstruct the sequence of exchanges and assess human oversight decisions. For traces containing sensitive information, researchers **must** anonymize personal identifiers, replace proprietary code with placeholders, and clearly highlight modified sections.

Runtime Traces. When an LLM calls out to external tools (e.g., APIs, file systems, databases, MCP servers, subagents) or activates configured artifacts reported under *System and Prompt Design* (e.g., context files, skills, subagents), this runtime activity forms a *runtime trace* distinct from the interaction log. Researchers **should** report the complete runtime trace as *supplementary material*, including for each entry the tool or artifact name, arguments (if any), result, and ordering relative to surrounding interaction-log entries. This lets readers attribute task success to the model, the external tools, or their interaction pattern, and distinguish artifacts that were configured from those that actually influenced a given run. Researchers **should** use an open format with a documented schema. Emerging standards such as the OpenTelemetry GenAI semantic conventions [109] or OpenInference [10] are preferred where they fit. Where tool-native formats are used (e.g., Claude Code's session transcripts, LangGraph's state logs), researchers **must** describe the file format and report the tool version.

Agentic Plans. For agentic systems that autonomously plan and execute tasks, researchers **should** report any plans the system exposes as *supplementary material*. In Claude Code, for example, a plan is a short Markdown document the user can open and edit during a session, listing the proposed steps and the files or commands the agent intends to touch. Other frameworks such as LangGraph keep plans inside the agent's internal execution state. All reported traces **must** be made publicly available as *supplementary material*, subject to privacy and confidentiality constraints. When full trace logging is not feasible, researchers **should** provide representative examples or anonymized traces.

5.4.3 ***Examples:***

An example of reporting full interaction logs is the study by Ronanki et al, for which the authors reported the full answers of ChatGPT and uploaded them to Zenodo [127]. For agentic systems, Bouzenia and Pradel unified the runtime trajectories of three SE agents (*RepairAgent*, *AutoCodeRover*, *OpenHands*) into a custom thought-action-result format and released the resulting 120 trajectories with 2,822 LLM interactions as a public dataset [27].

5.4.4 ***Benefits:***

Unlike human participant conversations, which often cannot be reported because of confidentiality, LLM interaction logs can be shared. This enables reproduction studies, tracking of response changes over time or across model versions, and secondary research on LLM consistency for specific SE tasks.

For agentic systems, reporting runtime traces alongside interaction logs lets readers follow the model's reasoning, the tool calls it made, and the order of those calls. Runtime traces complement the static configuration reported under *System and Prompt Design* by showing which of the configured artifacts were activated on a given run.

5.4.5 ***Challenges:***

Not all systems allow reporting of complete interaction logs with ease, and this hinders transparency and verifiability. For commercial tools, researchers **must** report all available information and acknowledge unknown aspects as limitations. Tool-call traces for commercial SaaS agents are often opaque: the user sees the final response but not the sequence of internal tool calls. When this is the case, authors **should** document what was and was not observable. For agents running locally or via open-source tools, these traces are usually more accessible and **should** be reported whenever available. Agent frameworks differ in whether and how they log agent-to-agent communication, so reporting practices vary across studies.

5.4.6 ***Study Types:***

Runtime trace reporting requirements depend heavily on the study type and on the accessibility of the underlying system. The general expectation is that interaction logs are reported as *supplementary material* whenever feasible; runtime traces are additionally expected when agentic execution or evaluation harnesses go beyond direct API calls.

For *Studying LLM Usage*, especially observational studies targeting commercial tools, researchers **must** report the full interaction logs except when transcripts might identify anonymous participants or reveal personal or confidential information. If 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*. For *LLMs for Tools* that use agentic execution, researchers **should** report runtime traces for the runs used to evaluate the tool. For *Benchmarking LLMs* that use agent-based harnesses, researchers **should** report runtime traces for representative runs, letting readers understand how task success depends on the agent's decision sequence rather than the model's raw output alone. For *LLMs as Annotators*, *LLMs as Judges*, *LLMs for Synthesis*, and *LLMs as Subjects*, when the research setup involves multi-turn interaction or agentic orchestration, researchers **should** report the corresponding interaction logs and, where applicable, runtime traces.

5.4.7 ***Advice for Reviewers:***

As with other guidelines, missing trace information is typically a minor revision request unless so much is missing that methodological rigor cannot be assessed. Reviewers should recognize that complete trace reporting is easier for local and open-source setups than for commercial SaaS agents. When reviewing commercial-tool studies, reviewers should focus on whether authors have reported everything the system exposed and have been explicit about what remains unobservable.

5.4.8 ***See Also:***

→ Section 5.3 (Report System and Prompt Design): Session traces show runtime behavior; system and prompt design covers the static artifacts that produce it.
→ Section 5.2 (Report Model Version, Configuration, and Customizations): A trace cannot be reproduced or compared across studies without the model identity that produced it.
→ Section 5.7 (Use Human Validation for LLM Outputs): Agentic traces produce outputs that often warrant human validation.
→ Section 5.6 (Use an Open LLM as a Baseline): Open models let other researchers reproduce a reported trace on the exact same model weights.

→ Section 5.8 (Report Limitations and Mitigations): When a tool hides parts of the trace, authors must report the gap as a limitation.

5.5 Use Suitable Baselines, Benchmarks, and Metrics

Summary: Researchers **must** justify all benchmark and metric choices in the *paper*, **must** discuss their reliability and validity (especially construct validity), and **should** summarize benchmark structure, task types, and limitations. They **should** operationally define the phenomenon being measured, justify the sampling strategy used to select problems for inclusion in the benchmark, isolate the target capability from confounders where possible, and perform an error analysis. When creating or releasing a benchmark, data sources and collection dates **must** be disclosed for each release. Where possible, traditional (non-LLM) baselines **should** be used for comparison. Researchers **must** explain in the *paper* why the selected metrics are suitable for the specific study; prior adoption in related work alone does not constitute sufficient justification. 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. If comparing models or tools, researchers **should** use appropriate inferential statistics rather than relying solely on summary statistics. Researchers **should** justify the number of experiment repetitions, for example through a power analysis or by monitoring convergence of descriptive statistics. Latency **must** be reported when it can affect study outcomes (e.g., interactive user studies, latency comparisons).

5.5.1 ***Rationale:***

Meaningful evaluation requires well-understood, valid measurement instruments. Without justified benchmarks and metrics, claims about LLM performance lack the rigor needed for scientific comparison and cumulative progress. A *benchmark* is a "*standard tool for the competitive evaluation and comparison of competing systems or components according to specific characteristics, such as performance, dependability, or security*" [80]. A *metric* "*is a method, algorithm, or procedure for assigning one or more numbers to a phenomenon*" [119]. A baseline is a reference point, enabling comparison of LLMs against traditional algorithms with lower computational costs.

5.5.2 ***Recommendations:***

Benchmark and metric selection requires understanding the benchmark tasks, what exactly is being measured, and how it relates to the (often latent) variables researchers actually care about. When one or more metrics or benchmarks are used, researchers **must** briefly justify in the *paper* why each selected benchmark and metric is suitable for the given task or study, and **must** discuss the reliability and validity—especially construct validity—of those choices (see *Limitations and Mitigations*). Beyond these requirements, researchers **should**:

- provide an operational definition of the phenomenon the benchmark is intended to measure (e.g., functional correctness, code maintainability, vulnerability detection), including its scope and any sub-components [22];
- 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;
- describe and justify the sampling strategy used to select problems for inclusion in the benchmark (e.g., function-completion problems in *HumanEval*, GitHub issues in *SWE-bench*, vulnerable functions in *DiverseVul*); if non-probability sampling (e.g., convenience) is used, discuss its implications for the generalizability of conclusions [15, 22];
- discuss the limitations of the selected benchmark(s) (e.g., widely used benchmarks such as *HumanEval* [34] and *MBPP* [13] only test short Python functions, which is not representative of the full breadth of SE work [30]);
- 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 both performance and design choices (e.g., which tasks are included, how outputs are scored, what aspect of the phenomenon is covered) across benchmarks [22]. When selecting only a subset of all available benchmarks, researchers **should** use the most specific benchmarks given the context. When adapting an existing benchmark, researchers **should** document what was changed and why, and **should** report performance on both the original and the adapted version where feasible [22].

Benchmark scores often confound the target capability with unrelated capabilities the task happens to require. For code translation benchmarks, prompt formatting alone can shift performance by up to 40% [29, 59], conflating prompt format sensitivity with translation capability. More broadly, output format compliance (e.g., specific JSON schemas or unit-test conventions) and general instruction-following capability are routinely bundled into the aggregate benchmark score [22]. Researchers **should** identify which capabilities a benchmark conflates, isolate the target capability where possible (e.g., by reporting per-subtask breakdowns or by adopting benchmarks designed to test the target capability in isolation), and acknowledge remaining confounders as threats to construct validity.

After running a benchmark, researchers **should** perform an error analysis by categorizing the failures observed and reporting the relative frequency of each category. If failures concentrate on inputs that demand capabilities other than the target capability (e.g., reading across many files rather than fixing the bug the benchmark targets), this is a construct-validity threat and **should** be reported alongside the primary scores [22].

In addition to disclosing data sources and collection dates (see *Challenges* below), researchers creating or releasing a new benchmark **should** adopt concrete contamination-prevention mechanisms: maintaining a held-out subset of items for ongoing, uncontaminated evaluation; embedding canary strings, that is, unique markers that downstream tools can later search for in model outputs to detect inclusion in training data; and investigating whether the benchmark's source materials may already appear in common LLM training corpora [22, 29]. Researchers using existing benchmarks **should** additionally discuss contamination as a study limitation (see *Limitations and Mitigations*).

Furthermore, researchers **should** check whether a less resource-intensive approach (e.g., for static analysis tasks or program repair) can serve as a baseline. If so, the LLM or LLM-based tool **should** be compared with such baselines using suitable metrics. Even if LLM-based tools outperform baselines, researchers **should** discuss whether the resources consumed justify the (often marginal) improvements [96].

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 report additional metrics that they consider appropriate. We briefly discuss common metrics in the *Examples* subsection below. As mentioned, researchers **must** motivate why they chose a certain metric or variant thereof for their particular study. Prior adoption alone does not constitute sufficient justification; researchers **must not** justify metric choices solely by citing their use in prior work. Latency **must** be reported when it can affect study outcomes.

Due to LLM non-determinism, researchers **should** repeat experiments and report descriptive statistics of model or tool performance (e.g., arithmetic mean, median, confidence intervals, standard deviations) [2, 24]. If comparing models or tools, researchers **should** use appropriate inferential statistics with effect sizes rather than relying solely on differences in means or other summary statistics. Suitable choices include hypothesis tests such as the Mann-Whitney U test, McNemar's test for binary outcomes [84], and bootstrap-based comparisons. For choosing among these and related tests, Dror et al provide a decision tree based on the distributional assumptions of the test statistic and the size of the test set [41]. When scores vary across raters or across runs of the same scorer (e.g., multiple human raters or repeated runs of an LLM judge), researchers **should** report the distribution of ratings per item rather than only aggregated point estimates or exact-match agreement, since aggregation can mask systematic disagreement [22].

The number of required repetitions depends on factors such as the study type, the variability of the task, and the desired precision of estimates. As

with sample sizes for human validation (see *Human Validation*), researchers **should** justify their chosen number of repetitions, for example, through a power analysis [24, 38] or by monitoring the convergence of descriptive statistics across incremental runs [25]. A pilot study can help estimate the expected variability and inform this decision.

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, labeling phenomena as "bugs" in a large open dataset reflects a certain theory of what constitutes a bug, as well as the values and perspectives 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.

Researchers building or releasing new SE benchmarks **should** consult operational checklists. Cao et al provide *HOW2BENCH*, a 55-item checklist covering benchmark design, construction, evaluation, analysis, and release [29]. Bean et al provide a complementary domain-agnostic checklist organized around the construct-validity recommendations summarized above [22].

5.5.3 ***Examples:***

Common Metrics. Two common metrics used for generation tasks are *BLEU-N* and *pass@k*. *BLEU-N* [113] was originally developed for evaluating machine translation quality by measuring modified n-gram precision (with a brevity penalty) between a candidate and reference text, ranging from 0 (dissimilar) to 1 (similar). It has been widely adopted in SE for code generation tasks, though its validity in this context is debatable. An n-gram overlap does not capture functional correctness, and syntactically different code can be semantically equivalent (see *Challenges* below). Code-specific variations attempt to address these limitations. *CodeBLEU* [123] augments n-gram overlap with syntactic (AST) and data-flow matching, whereas *CrystalBLEU* [44] ignores n-grams that recur across unrelated programs, such as boilerplate syntax and common API calls, because they inflate overlap scores without indicating genuine similarity.

The metric *pass@k* reports the probability that at least one of k generated solutions for a task passes the task's tests. In their pseudocode-to-code study, Kulal et al counted a test example as solved if best-first search over one candidate code line per pseudocode line produced an accepted program within B trials. Each trial compiled one candidate program and, if compilation succeeded, ran public tests [85]. Chen et al defined *pass@k* for code generation and estimated it from $n \geq k$ samples, treating a sample as correct when it passes the task's unit tests [34].

For a single task, the estimator for *pass@k* is:

$$\text{pass@}k = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}},$$

where:

- n is the total number of samples generated per task (with $n \geq k$),
- c is the number of correct samples among the n, and
- k is the number of attempts considered, drawn from the n generated samples without replacement.

A benchmark reports the mean of the task-level *pass@k* estimates. The choice of k depends on the downstream task. The metric *pass@1* is critical for single-suggestion scenarios such as code completion, while higher k values (e.g., 2, 5, 10) assess multi-attempt capability and are commonly reported in technical reports of code LLMs (e.g., Code Llama [129], DeepSeek-Coder [57], Qwen2.5-Coder [68], StarCoder [88]). However, *pass@k* is not a universal metric suitable for all generation tasks. It requires a binary notion of correctness, making the metric appropriate for code synthesis evaluated via unit tests, but unsuitable for open-ended generation tasks such as comment generation, where multiple valid outputs exist.

A complementary perspective from industry practice is $pass^k$ (also written *pass^k*), which measures the probability that *all* k trials succeed rather than *at least one* [7]. While *pass@k* increases with k, $pass^k$ decreases, making it useful for assessing reliability in deployment scenarios where consistent success matters (e.g., customer-facing agents).

If a study evaluates an LLM-based tool for supporting 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, which reveals how much a tool's performance depends on specific models and versions. Metrics used to measure inter-model agreements include general agreement (percentage), *Cohen's* κ, and *Krippendorff's* α (see *Human Validation* for recommended thresholds and best practices for measuring agreement).

Common problem types for LLM-based studies are classification, recommendation, and generation, each requiring different metrics [64]. Hu et al categorized 191 LLM benchmarks by SE task, providing a valuable reference [65]. For an overview of code foundation models, agents, and their evaluation, see Yang et al [171]. From a practitioner perspective, Anthropic categorize evaluation approaches for LLM-based agents into code-based graders (e.g., test suite execution, static analysis), model-based graders (e.g., rubric-scored LLM judgments), and human graders [7]. This taxonomy may help researchers systematically design evaluation strategies for agent-based tools. Common metrics include *BLEU*, *pass@k*, *Accuracy@k*, and *Exact Match* for generation; *Mean Reciprocal Rank* for recommendation; and *Precision*, *Recall*, *F1-score*, and *Accuracy* for classification.

Benchmark Examples. Benchmarks used for code generation include *HumanEval* (available on GitHub) [34], *MBPP* (available on Hugging Face) [13], *ClassEval* (available on GitHub) [42], *LiveCodeBench* (available on GitHub) [72], and *SWE-bench* (available on GitHub) [73]. An example of a code translation benchmark is *TransCoder* [128] (available on GitHub). Golnari et al's

DevBench (available on GitHub) synthesized 1,800 code completion instances from developer telemetry, rather than collecting them from public sources [53]. For evaluating LLMs as agents, *AgentBench* (available on GitHub) [92] evaluates LLM agents across eight environments, including an operating system shell, a database, and a web browser.

Most benchmarks focus on generation tasks, but benchmarks for classification and recommendation also exist. For classification, *DiverseVul* (available on GitHub) [35] provides vulnerable and non-vulnerable functions evaluated using standard classification metrics. For recommendation, *CodeSearchNet* (available on GitHub) [69] contains code-documentation pairs evaluated using *Mean Reciprocal Rank*.

5.5.4 ***Benefits:***

Established benchmarks and metrics let researchers assess new systems against shared reference points rather than against hundreds of competing systems, showing which approaches work best on those benchmarks. Researchers can also track progress by iteratively improving a new LLM-based tool and retesting it against the same benchmarks after substantial changes. For practitioners, leaderboards support selecting models for downstream tasks, and baselines clarify how much LLMs improve over non-LLM alternatives.

5.5.5 ***Challenges:***

Computer scientists commonly use simple metrics (e.g., lines of code, CPU time) as proxies for complex, multidimensional latent variables (e.g., system size, environmental sustainability), without empirically validating that the metrics capture the intended construct [117]. Unlike fields such as psychology where measurement theory has a longer tradition [26], many AI metrics and benchmarks lack both theoretical underpinnings and empirical validation of their construct, measurement, and ecological validity.

These validity issues are widespread, not isolated. In a survey of 572 code benchmarks released between 2014 and 2025, Cao et al found that 84.2% did not consider test suite coverage when constructing test cases, 64.0% reported single-pass evaluations without controlling for randomness, and 82.5% did not address data contamination [29]. Bean et al reported comparable patterns in a parallel review of 445 LLM benchmarks from ML and NLP venues [22]. As a result, model performance on these benchmarks can reflect benchmark artifacts rather than the capability the benchmark claims to test.

For example, *pass@k* is intended to measure a model's *functional correctness*, that is, its capability to generate code that produces the expected output for a given specification. However, functional correctness is only one dimension of code quality. *pass@k* does not capture maintainability, readability, security, or efficiency of the generated code, all of which are critical for downstream use. Furthermore, "correctness" is defined entirely by test suites, whose coverage is itself unvalidated. A solution that passes all provided tests may still be

incorrect for untested inputs. Whether a test suite adequately operationalizes correctness for a given task is a subjective judgment that is rarely examined.

Similarly, *HumanEval* is intended to measure a model's capability to *synthesize short Python functions from docstring specifications*. This operationalizes a narrow slice of "code generation capability": it covers neither multi-file tasks, nor debugging, nor the use of existing codebases. These tasks dominate real-world software engineering [30]. Notably, neither *pass@k* nor *HumanEval* has undergone rigorous empirical validation of its construct validity. Their widespread adoption rests on face validity and convenience rather than on evidence that they reliably measure what researchers intend them to measure [29]. *HumanEval* also contains implementation, documentation, and test case bugs in its original release, which directly affect score interpretation [90]; comparable issues have been documented in other widely-used code benchmarks [29].

Benchmarks and metrics also have generalizability problems. Prominent LLM benchmarks such as *HumanEval* and *MBPP* use Python, so researchers can optimize for Python's idiosyncrasies, producing gains that do not generalize to other languages. Similarly, the metric *BLEU-N* is a syntactic metric, so code can score highly without being executable. The metric *Exact Match*, meanwhile, does not account for functional equivalence of syntactically different code. Both *BLEU-N* and *Exact Match* are influenced by code formatting, which confounds their intended use.

Execution-based metrics such as *pass@k* directly evaluate correctness by running test cases, but they require an execution environment. When metric values look surprising, examine the specific test cases driving them: common causes include outliers, bugs in test suites or scoring code, and items that exercise capabilities other than the one being measured.

Finally, benchmark data contamination, where the benchmark itself is part of the training data, may lead to artificially high performance if the model remembers the solution from the training data rather than deriving it from the input [167] (see *Limitations and Mitigations*). Therefore, for proprietary LLMs that do not release their training data, researchers **should** consider using human validation, curating new data, or refactoring existing data [167].

To mitigate contamination, researchers can create new benchmark datasets by collecting data after a specified cutoff date; researchers **must** disclose data sources and collection dates for each release. However, this temporal approach requires continuous updates as new models may include the benchmark in future training data. Alternatively, keeping the benchmark private prevents inclusion in training sets, but requires trust in the benchmark creator and a system to execute it without leaking data. A third strategy is to synthesize benchmark instances rather than draw them from public sources, which avoids training-data contamination at the cost of requiring an instance generator and a validation pipeline. Researchers can also evaluate how contaminated their benchmark is [37].

5.5.6 ***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 [65].

For *Benchmarking LLMs*, this guideline is of primary importance: researchers **must** use established benchmarks or rigorously justify the creation of new ones, **must** report standard metrics to enable cross-study comparison, and **should** report an error analysis alongside the primary scores so that readers can assess whether reported gains reflect the target capability or other capabilities the task happens to require. When researchers compare LLMs or tools on latency, they **must** report the infrastructure used to produce the measurements (*System and Prompt Design*). 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 widely. 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. For *LLMs for Tools*, researchers **should** benchmark the tool against suitable baselines using appropriate metrics that capture the tool's intended contribution. For *LLMs as Judges*, researchers **should** report inter-rater agreement metrics and validity measures to demonstrate the reliability and quality of LLM judgments. For *LLMs for Synthesis*, researchers **should** specify metrics for comparing synthesized outputs (e.g., coverage, faithfulness) and justify their appropriateness for the synthesis task. For *Studying LLM Usage*, researchers **should** justify the measurement instruments and metrics used for studying LLM usage patterns, including any survey scales or behavioral measures. In interactive setups where response times can influence participant behavior, researchers **must** report observed latency. For *LLMs as Subjects*, researchers **should** compare simulated and real human responses using appropriate metrics to assess simulation fidelity. 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.

5.5.7 ***Advice for Reviewers:***

Reviewers should expect manuscripts to: (1) clearly identify the constructs or variables the study aims to measure (e.g., LLM performance, quality of generated code), including independent, dependent, and control variables; (2) present their measurement model, i.e., which metrics, benchmarks, or baselines are used and how they relate to the target constructs; (3) justify the selection of metrics, benchmarks, and baselines; (4) discuss *in detail* the assumptions, reliability, and validity (*especially construct validity*) of each benchmark and metric; (5) articulate any limitations regarding construct and measurement validity.

As with other guidelines, missing information about baselines or metrics is typically a revision request. However, vague descriptions that conflate broad concerns (e.g., effectiveness, quality) with specific counting methods should be questioned. Ubiquity of a benchmark or metric does not imply validity or appropriateness for a given context. Manuscripts should convey a solid understanding of construct and measurement validity by explaining and justifying their measurement models.

5.5.8 ***See Also:***

→ Section 5.6 (Use an Open LLM as a Baseline): Using an open LLM as a baseline supports reproducible benchmark comparisons across studies.
→ Section 5.7 (Use Human Validation for LLM Outputs): Human evaluation captures aspects of LLM outputs that automated metrics cannot measure.
→ Section 5.8 (Report Limitations and Mitigations): Every benchmark and metric choice carries construct-validity threats that authors must discuss.

5.6 Use an Open LLM as a Baseline

Summary: Researchers **should** include an open LLM as a baseline when using commercial models and report inter-model agreement. We follow the OSI definition of open-source AI: access to everything needed to understand, modify, share, retrain, and recreate the model. Many models release only trained weights without training data or methodological details ("open weight"). Researchers **should** ensure the open-LLM baseline is independently reproducible from their *supplementary material*.

5.6.1 ***Rationale:***

Reproducibility depends on access to the model under study. When research relies exclusively on proprietary models, other researchers cannot independently verify or build upon the findings. Including an open LLM as a baseline ensures that at least part of the study can be fully replicated.

5.6.2 ***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 *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** ensure the open-LLM baseline is independently reproducible from their *supplementary material*.

Open LLMs are available from hubs such as *Hugging Face*. They can be self-hosted with frameworks such as *Ollama* or *LM Studio*, accessed through cloud services such as *Together AI*, AWS, Azure, and Google Cloud, or routed through aggregators such as *OpenRouter* that expose many providers behind a single API. For agentic setups, open-source tools such as *Continue*, *Cline*, and *opencode* [108] publish their full agent code, system prompts, and tool catalog. By contrast, vendor-hosted services such as *GitHub Copilot* and *Claude Code* expose only parts of their tooling (e.g., editor extensions, hook examples) and keep their deployed agent loops, system prompts, and tool catalogs proprietary.

The term "open" can have different meanings in the context of LLMs. Widder et al discuss three types of openness (i.e., transparency, reusability, and extensibility) and what openness can and cannot provide [161]. The *Open Source Initiative* (OSI) [105] defines open-source AI as having access to everything needed to understand, modify, share, retrain, and recreate the model.

5.6.3 ***Examples:***

An increasing number of 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 [157]. Moumoula et al compared four LLMs on a cross-language code clone detection task [103]. 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 [54]. Golnari et al evaluated nine code completion models on the *DevBench* benchmark and included three open-weight models (*DeepSeek-V3*, *DeepSeek-V3.1*, and *Ministral-3B*) alongside commercial frontier models, releasing benchmark, evaluation scripts, and per-model raw completions under an MIT license [53]. *CodeBERT*, a bimodal transformer pre-trained by Microsoft Research, is published with model weights, source code, and data processing scripts on GitHub [97]. It has been used as an open baseline across diverse SE tasks, including exploit code generation [170], vulnerability detection [165], code clone detection [147], and programming assistance for exception handling [28].

5.6.4 ***Benefits:***

A true open LLM baseline improves reproducibility by exposing model architectures, parameter settings, and ideally training data, enabling independent verification of results. Such baselines also let researchers compare novel methods against a stable reference point, since proprietary models can silently change between tests. They also allow inspection of training data (when released) and model behavior, helping identify biases and limitations. Unlike closed-source alternatives, which can be withdrawn or silently updated, open LLMs remain available for future studies. They typically avoid the per-use API fees that can constrain research groups with limited budgets.

5.6.5 ***Challenges:***

Open-source LLMs face several challenges:

- *Definitional inconsistency.* Many models release only trained weights without training data or methodological details ("open weight" openness) [51], which is why we reference the OSI definition in our recommendations [105].
- *Performance gap.* Open models often lag behind the most advanced proprietary "frontier" models in common benchmarks, making it difficult to demonstrate clear improvements when evaluating new methods using open LLMs alone.
- *Hardware demands.* Deploying and experimenting with these models typically requires substantial hardware resources, in particular high-performance GPUs that may be beyond reach for many academic groups.
- *Operational complexity.* Unlike APIs provided by proprietary vendors (e.g., the OpenAI API), installing, configuring, and fine-tuning open-source models can be technically demanding.

5.6.6 ***Study Types:***

This guideline applies primarily to study types in which the researcher controls which LLM is used. For *Benchmarking LLMs*, an open LLM **should** be one of the models under evaluation, so that the reported scores can be independently re-run. Where this is not feasible, researchers **should** justify the omission and, per *System and Prompt Design*, ensure the evaluation harness can be used with open models. For controlled experiments under *Studying LLM Usage*, an open LLM **should** be one of the models under test; if the experimental design requires capabilities that only a specific commercial model exhibits, researchers **should** acknowledge this as a limitation. When evaluating *LLMs for 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. For *LLMs as Annotators* and *LLMs as Judges*, researchers **should** compare annotation or judgment quality from open vs. commercial models to assess the extent to which results depend on a specific proprietary model. For *LLMs for Synthesis*, researchers **should** compare synthesis results from open and commercial models to evaluate robustness of the findings. For *Studying LLM Usage*, using an open LLM as a baseline is often not feasible when the study observes participants using specific commercial tools; in such cases, investigators **should** explicitly acknowledge its absence and discuss how this limitation might affect their conclusions. For *LLMs as Subjects*, using an open LLM as a baseline may similarly be impractical if the study design requires the capabilities of a specific model; researchers **should** acknowledge this limitation when applicable.

5.6.7 ***Advice for Reviewers:***

Reviewers should distinguish between LLM use that is central to the research (e.g., building LLM-driven SE tools, using LLMs for synthesis) and use that is tangential (e.g., generating recruiting materials). An open LLM baseline is expected only when LLM use is central; otherwise, its absence need not be justified. When an open baseline is expected, reviewers should look for either the use of an open LLM or a convincing argument for why it is impractical. If authors claim openness, some justification for that characterization is appropriate. Where practical, reviewers should examine whether the replication package contains sufficiently detailed instructions. Performance differences between open and proprietary models are beyond the study authors' control. Reviewers should focus on whether the open baseline serves the study's methodological purpose rather than on its absolute performance.

5.6.8 ***See Also:***

→ Section 5.2 (Report Model Version, Configuration, and Customizations): Authors must report version, configuration, and parameters for open models, not just commercial ones.
→ Section 5.3 (Report System and Prompt Design): Self-hosting an open model adds hardware, hosting, and infrastructure to document.
→ Section 5.5 (Use Suitable Baselines, Benchmarks, and Metrics): Inter-model agreement metrics make open-vs-commercial comparisons interpretable.
→ Section 5.8 (Report Limitations and Mitigations): When using an open LLM as a baseline is impractical, authors must report this as a limitation.

5.7 Use Human Validation for LLM Outputs

Summary: 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, and not as an afterthought. They **should** build on established reference models for human-LLM comparison. When developing or adapting measurement instruments, researchers **must** share them. When LLMs replace, rather than augment, humans in research tasks such as annotating software artifacts, coding interview transcripts, or simulating study participants, researchers **must** explain whether and how the replacement is justified and **should** ground it with inter-model and model-to-human agreement. For qualitative coding of interpretive data, they **should** justify why an LLM is methodologically appropriate. When aggregating LLM judgments, methods and rationale **should** be reported and inter-rater agreement **should** be assessed. Confounding factors **should** be controlled for; where applicable, researchers may perform a power analysis to estimate the required sample size. For value-laden or culturally contingent constructs, researchers **should** describe rater demographics beyond expertise and discuss potential demographic biases. When evaluating agentic tools, user feedback on the agent's proposed actions **should** be assessed and acceptance statistics reported.

5.7.1 ***Rationale:***

Even with well-justified benchmarks and metrics (*Benchmarks and Metrics*), automated measurement captures only the constructs the benchmark was designed to measure. Subjective constructs and constructs that no current benchmark covers require human assessment.

5.7.2 ***Recommendations:***

When LLMs automate research or software development tasks previously performed by humans, the LLM's performance needs to be assessed. Where no benchmark adequately operationalizes the target construct, researchers **should** validate LLM outputs against human judgment.

Study Design Considerations. Studies that include human participants need 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. Authors **must** clearly define in the *paper* the constructs that the human and LLM annotators evaluate [117]. When designing custom instruments to assess LLM output (e.g., questionnaires, scales), researchers **must** share these instruments.

Replacing Human Judgment. LLMs may be used to replace, rather than augment, humans in research tasks such as annotating software artifacts, coding interview transcripts, or simulating participants in user studies. In such cases, researchers **must** explain whether and how the replacement is justified and **should** follow systematic approaches to support this judgment, e.g., showing that a jury of three LLMs exhibits inter-model agreement at the same threshold researchers would require for human-to-human agreement, such as Krippendorff's $\alpha > 0.8$. Such agreement is necessary but not sufficient. High inter-model agreement can reflect shared model biases rather than valid annotation, so researchers **should** additionally validate a sample against human experts before treating LLM outputs as a substitute for human work [4, 83]. For qualitative coding of interpretive data such as open-ended developer interview responses, agreement metrics do not show that an LLM can perform the interpretive work that the task requires. Researchers **should** justify why the LLM is methodologically appropriate (see *LLMs for Research*).

Subjective Judgment and Agreement. When the judgment is subjective (i.e., depends on the judge's values or theories), the same artifacts **should** be judged independently by both the LLM and a panel of human experts, and the LLM judgments **should** then be compared against an aggregated human reference. Researchers **should** clearly describe their aggregation method and reasoning. Researchers **should** use established reference models to compare humans with LLMs. For example, Schneider et al outline design considerations for studies comparing LLMs with humans [136].

One systematic approach to building the human reference is to randomly order the objects requiring judgment and put them in groups small enough to judge in one to three hours. Two or three human experts review the first group together, simultaneously judging, discussing, and creating a set of decision rules documenting their reasoning. Then, the experts iterate among rounds of:

1. independently rating one group of objects,

2. calculating inter-rater agreement (IRA) or reliability (IRR),
3. meeting to discuss disagreements and reach consensus,
4. updating the decision rules so the disagreement will not recur.

The goal is to reach a target IRA or IRR (e.g., Krippendorff's $\alpha > 0.8$), indicating sufficient decision rules. Once this target is reached and sustained for two to three groups, it is permissible to continue with a single rater. Researchers **should** report measures of IRA or IRR, preferably broken down by round, and **should** apply thresholds consistent with established standards. Krippendorff recommends discarding data with $\alpha < 0.667$, considers $0.667 \leq \alpha < 0.8$ sufficient only for tentative conclusions, and requires $\alpha \geq 0.8$ for reliable data [83].

Confounding factors **should** be discussed and, where feasible, controlled for (e.g., by categorizing participants according to their level of experience or expertise). For value-laden or culturally contingent constructs (e.g., judging code-style appropriateness or comment helpfulness), researchers **should** describe rater demographics beyond expertise (e.g., geographic, linguistic, or professional background) and discuss potential demographic biases in rater recruitment and instructions [22]. Where applicable, researchers may perform a power analysis [38, 43] to estimate the required sample size, ensuring sufficient statistical power in their experimental design. Although established sample-size guidance for LLM-human comparisons is limited, related fields commonly use 100 comparisons without further justification [153].

Agentic Tools. Agentic human-in-the-loop interaction is a built-in human validation, with larger degrees of freedom than traditional experiments that use LLMs directly. Besides generating and modifying content, agentic systems such as *Claude Code* (see *System and Prompt Design*) can autonomously call command-line tools or pull in additional information from MCP servers. When evaluating agentic tools, researchers **should** assess the feedback that users provided on the agent's proposed actions (e.g., file edits, command executions), report statistics on how frequently they accepted the proposals, and how they modified them.

5.7.3 ***Examples:***

Ahmed et al proposed a systematic method for deciding whether LLMs can replace human annotators on a given task, using inter-model agreement as an initial screening criterion (with a threshold of $\alpha > 0.5$) and model confidence for sample-level decisions [4]. However, their threshold is well below the levels generally considered acceptable for inter-rater agreement. Notably, while three LLMs exhibited higher inter-model agreement than human annotators on some tasks, human-model agreement remained low on others. This illustrates that high inter-model reliability does not guarantee alignment with human judgment, reinforcing the need for human validation before assuming that LLM annotations are valid. Moreover, a high inter-model agreement could

merely indicate that the models share systematic biases and hence reliably agree on the wrong answer.

Hymel and Johnson evaluated ChatGPT's capability to generate requirements documents by comparing an LLM-generated and a human-generated document based on the same business use case [70]. Domain experts reviewed both documents and attempted to distinguish their origin. For agentic tools, Takerngsaksiri et al reported acceptance and modification rates from real users for *HULA*, an agentic plan-and-code system deployed in Atlassian JIRA [152].

5.7.4 ***Benefits:***

Validation against human judgments builds confidence in the LLM's accuracy and validity [78]. Reporting IRA or IRR supports this validation but does not stand in for it, since even high agreement may reflect shared biases rather than valid judgment. When human reviewers disagree with the LLM, the disagreement points to concrete improvements in the prompts, the context provided to the LLM, or the operational definition of the construct.

5.7.5 ***Challenges:***

Assessing LLM performance with respect to a construct such as code maintainability, answer correctness, or annotation reliability is challenging because ensuring that a construct is defined well and operationalized using an appropriate measurement model requires a deep understanding of (1) the construct, (2) construct validity in general, and (3) instrumentation [119, 145]. Comparing an LLM to human judges is typically slower and more expensive than machine-generated measures. More fundamentally, neither human judgment nor machine-generated measures provides an objective ground truth against which LLM accuracy can be firmly determined. Human preference ratings under-represent properties such as factuality and faithfulness, and assertively phrased outputs receive systematically fewer flagged errors from crowdworkers [63]. Human validation is worth its added cost only when automated measures alone fail to capture the constructs the study evaluates.

Human judgments exhibit variability due to differences in experience, expertise, interpretations, and personal biases [95]. In pairwise judgment of LLM outputs, both human and LLM judges shift their preferences toward answers carrying fake references or richly formatted content, regardless of correctness [32]. When diverse humans rate items reliably given clear decision rules, we assume that reliability implies validity, but it does not. Measuring reliability is *much* easier than measuring validity, and often the best researchers can do is argue conceptually for why their judges, decision rules, and constructs *should* produce valid ratings.

5.7.6 ***Study Types:***

This guideline applies to all study types, although the need for human validation varies. *Replacing Human Judgment* introduces a conditional **must** that applies particularly to *LLMs as Annotators*, *LLMs as Judges*, *LLMs for Synthesis*, and *LLMs as Subjects*, where the LLM substitutes for human input. For *LLMs as Annotators*, researchers **should** validate LLM-generated annotations against human annotators to assess labeling quality and identify systematic biases. When using *LLMs as Judges*, researchers **should** co-create initial rating criteria with humans and validate a sample of LLM judgments against human expert assessments. For *LLMs for Synthesis*, researchers **should** employ human oversight to verify that qualitative interpretations and synthesized outputs faithfully represent the underlying data. For *LLMs as Subjects*, researchers **should** validate simulated responses against real human data to assess the fidelity of the simulation. For *Studying LLM Usage*, researchers **should** carefully reflect on the validity of their evaluation criteria and validate subjective assessments with human experts. For *LLMs for Tools* whose output is supposed to match human expectations, researchers **should** validate the LLM output against human judgment. For *Benchmarking LLMs*, there is less need for human validation when using extensively validated and widely-used benchmarks, but researchers **should** employ human validation when creating or adapting new benchmarks.

5.7.7 ***Advice for Reviewers:***

Human validation may be the most challenging of our guidelines to assess because it often requires evaluating conceptual arguments. If LLM output is validated only by comparison with other LLMs, reviewers should look for *quantitative empirical evidence* that such comparison is reliable *and* valid. High inter-model agreement alone is insufficient, as reliability does not imply validity. Similarly, reviewers should expect evidence that any employed benchmarks are reliable and valid. Absent such evidence, human validation is warranted. A single human judge is appropriate only when judgments depend on widely accepted theories and involve limited value conflict (e.g., tagging method names containing abbreviations). For multiple judges, reviewers should expect IRA/IRR improvement techniques as described in the recommendations above (experienced raters, organized rounds, consensus meetings, updated decision rules). Low IRA or IRR (e.g., Krippendorff's $\alpha < 0.8$) without these techniques is a concern. Conversely, if authors have followed best practices and still obtained mediocre results (e.g., $0.66 < \alpha < 0.8$), this should be noted as a limitation. Beyond reliability, reviewers should expect authors to explain conceptually why their human judgments should be valid, considering construct definitions, decision rules, and judge expertise. As with other guidelines, missing information is typically a revision request. Absent judge instructions, instruments, decision rules, or construct definitions may prevent assessment of rigor and validity, while missing recruitment details are less crit-

ical. Clarification requests about construct definitions are routine and should not alone warrant rejection.

5.7.8 ***See Also:***

→ Section 5.5 (Use Suitable Baselines, Benchmarks, and Metrics): Human evaluation complements automated metrics where benchmarks cannot sufficiently capture the target construct.
→ Section 5.2 (Report Model Version, Configuration, and Customizations): Human-validation findings characterize only the specific model and configuration that produced the outputs.
→ Section 5.3 (Report System and Prompt Design): Prompt and architecture choices produce the outputs that humans then validate.
→ Section 5.4 (Report Session Traces): Stored interaction logs and runtime traces are the artifacts human reviewers examine.
→ Section 5.8 (Report Limitations and Mitigations): Rater demographics, threshold choices, and the validity of human judgments are limitations to discuss.

5.8 Report Limitations and Mitigations

Summary: 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** discuss potential data leakage effects and their impact on results, including the risk of evaluation data entering model improvement pipelines, and **must** describe how the quality of subjective results was ensured. For studies involving sensitive data, they **must** discuss data governance mechanisms. They **should** justify LLM usage in light of its resource demands. Mitigation strategies such as replication packages, human validation, longitudinal re-runs, triangulation, and sensitivity analysis **should** be employed and reported where applicable. Where full data sharing is not possible, a subset of the validation data **should** be included to enable partial replication.

5.8.1 ***Rationale:***

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 [132]. Researchers must openly discuss these limitations and explain how their impact was mitigated. These limitations are relative to current LLM capabilities and tool architectures; speculating about future improvements is beyond the scope of a paper's limitation section. Nevertheless, risk management and threat mitigation **should** be planned during study design, not as an afterthought.

5.8.2 ***Recommendations:***

Researchers **must** clearly present the limitations of their work without defensiveness or obfuscation. These limitations may concern external, internal, and construct validity; reliability and reproducibility; and ethical, regulatory, and environmental concerns.

We follow the standard SE convention of organizing limitations by external, internal, and construct validity, plus reliability [118, 130], extended below with ethical, regulatory, and environmental concerns specific to LLM research. For studies that adopt qualitative analytic methods (the use of LLMs in reflexive qualitative analysis is itself contested, see *LLMs as Annotators*), researchers **should** use the trustworthiness criteria of credibility, transferability, dependability, and confirmability instead [56]. When deterministic reproducibility is structurally unattainable (e.g., SaaS-based models with opaque versioning), researchers **should** adopt the same trustworthiness criteria to substantiate dependability and confirmability of findings.

External Validity. The primary threats to external validity are:

- *Cross-model transfer limitations*: Results obtained with one LLM or family of LLMs may not generalize to others due to differences in training data, architecture, and post-training procedures (e.g., fine-tuning and reinforcement learning from human feedback); see *Open LLM* for using an open LLM as a comparison baseline.
- *Configuration sensitivity*: Results may not generalize beyond the specific configuration(s) tested (e.g., decoding parameters, system prompt, or context settings); see *System and Prompt Design* for an overview.
- *Tool-architecture specificity*: Tools built around vendor-specific APIs or features (e.g., function calling, structured output, or context-window size) may not transfer to other models without substantial re-engineering (see *System and Prompt Design*).
- *Limited domain coverage*: Studies often focus on a narrow set of programming languages, task types, or application domains, limiting generalizability to other SE contexts.

- *Limited participant and rater diversity*: Study participants (e.g., developers) and human raters validating LLM outputs may not represent the broader population in terms of expertise, geographic location, or cultural background (see *Human Validation* for guidance on rater diversity in value-laden constructs).
- *Research-to-practice gap*: Developers using the same tools outside researcher-supervised conditions may obtain results that differ from those reported in the study.
- *Cross-time instability*: Performance of proprietary models can change over time, leading to non-generalizable study outcomes [33, 87] (see *Version and Configuration* for version and fingerprint reporting that enables tracking such drift).

Generalizability is particularly critical for proprietary and non-deterministic systems whose behavior is subject to drift (i.e., silent changes in model output over time). Researchers **must** discuss the limitations and mitigations of external validity. Mitigations include *triangulation* across multiple models (e.g., proprietary and open), independent datasets, and complementary metrics; *sensitivity analysis* that varies LLM configurations, prompts, architecture decisions, datasets, and where applicable participant backgrounds; and performing *longitudinal re-runs* (see *Reliability & Reproducibility* below), which also helps detect cross-time instability.

Internal Validity. The primary threats to internal validity are:

- *Data leakage and contamination*: Inter-dataset duplication can produce training-evaluation overlap, yielding overly optimistic results (see *Benchmarks and Metrics*).
- *Evaluation data entering model-improvement pipelines*: Evaluation samples can unintentionally feed retraining or fine-tuning, especially in longitudinal studies involving LLMs.
- *Incomplete architecture, prompt, or pipeline reporting*: Undisclosed components introduce hidden confounders (see *System and Prompt Design*).

As transparency on training data is limited for LLMs, researchers **must** discuss potential data leakage effects and their impact on results. Concrete mitigations for contamination (e.g., post-cutoff benchmark construction, held-out subsets, canary strings) are discussed in *Benchmarks and Metrics*.

Construct Validity. The primary threats to construct validity are:

- *Metric-construct mismatch*: Traditional metrics such as BLEU or ROUGE may miss SE-specific aspects such as functional correctness or behavioral equivalence (see *Benchmarks and Metrics*).
- *Construct under-specification*: If a construct lacks an operational definition, neither automated metrics nor human raters can apply it consistently.
- *Reliability without validity*: High inter-rater or inter-model agreement does not imply that the measurement captures the intended construct; a reliable LLM can be reliably inaccurate (see *Human Validation*).

- *Benchmark scope limitations*: Benchmarks commonly ignore runtime behaviors, security implications, readability, testability, and maintainability, yielding results that may not transfer to realistic development settings.
- *Capability confounding*: Benchmark performance can blend the target capability with unrelated capabilities such as output format compliance, instruction following, or prompt format sensitivity, inflating apparent scores (see *Benchmarks and Metrics*).
- *Over-reliance on benchmark-specific metrics*: Optimizing for a single benchmark may produce dataset-specific shortcuts that pass the benchmark without exhibiting the capability it was designed to test, overstating real-world utility.
- *Prompt sensitivity*: Small changes in instructions, formatting, or in-context examples can substantially shift what the LLM appears to measure, making the operationalized construct unstable across prompt variants (see *System and Prompt Design*).
- *Judge biases*: LLM and human judges exhibit systematic biases such as position bias, verbosity bias, and preferences for richly formatted or citation-rich outputs regardless of correctness (see *Human Validation*).

If constructs are based on subjective interpretations, purely automated metrics are insufficient. Researchers **must** discuss how they ensured quality of subjective results, similarly to qualitative research. The primary mitigation is *human validation* of subjective constructs following quality criteria known from qualitative research (see *Human Validation*).

Reliability & Reproducibility. The primary threats to reliability and reproducibility are:

- *Non-deterministic outputs*: Identical prompts and configurations can yield different outputs across runs due to factors such as floating-point arithmetic, batching, and stochastic decoding strategies.
- *Infrastructure dependence*: Results may vary depending on the hardware, software stack, and hosting environment used; vendor-imposed quotas, throttling, or pricing changes can further prevent re-running experiments at the original scale, making exact replication challenging across different infrastructure setups.
- *Resource inequality*: LLM research is resource-intensive and remains predominantly in the domain of private companies or well-funded research institutions [3, 141], excluding researchers from under-resourced institutions.

Researchers **must** discuss the measures taken to increase reliability and reproducibility. However, non-deterministic reproducibility is not inherently disqualifying. The trustworthiness criteria introduced above apply particularly to SaaS-based LLM research, where providers frequently deprecate model versions without guaranteeing stable behavior. Mitigations include providing *replication packages* that cover prompt and architecture specifications, model

outputs, and representative examples for partial replicability (ideally accompanied by an implementation using an open model for long-term stability), and performing *longitudinal re-runs* with statistical analyses. When deterministic reproduction is structurally impossible, researchers **should** consider *methodological trustworthiness measures* such as triangulation, reflexivity, audit trails, and peer debriefing as complementary measures.

Ethical & Regulatory Boundaries. The primary concerns for ethical and regulatory matters are:

- *Use of sensitive or proprietary data*: Studies involving proprietary code, confidential business data, or personally identifiable information may face restrictions on data sharing that limit reproducibility.
- *Jurisdictional obligations*: Data protection regulations such as GDPR or CCPA and institutional policies may impose constraints on data collection, processing, and sharing in LLM-based studies.
- *Implicit model bias*: Especially for qualitative research, LLMs might "*reinforce dominant paradigms and biases*" and "*identify, replicate and reinforce dominant language and patterns*" [74] (see *Human Validation*).

Studies involving sensitive data **must** discuss data governance mechanisms tailored toward LLM environments, compliant with applicable jurisdictional obligations. If applicable, researchers **should** discuss how model biases potentially impact the study outcomes and how those biases were evaluated.

Environmental & Sustainability Constraints. The primary environmental and sustainability concerns are:

- *Energy consumption*: With growing model size, the environmental impact of experiments with LLMs increases, and the substantial energy costs of LLM experiments warrant consideration in study design [149].
- *Trade-off between repetition and sustainability*: Repeating experiments increases reliability but also energy consumption, requiring trade-offs during study design.

Researchers **should** justify the LLM's resource consumption against the benefits over traditional approaches. Mitigations include *energy preservation* and *cost accounting*. *Energy preservation* involves selecting smaller or newer, less resource-intensive models and applying techniques such as input/output token reduction, model pruning, quantization, or knowledge distillation [99] where feasible. Carbon footprint estimation is desirable, but still difficult. *Cost accounting* tracks resource consumption by reporting tokens, service costs, or hardware specifications.

5.8.3 ***Examples:***

Sallou et al catalog three categories of LLM-specific threats to validity (i.e., closed-source models, implicit data leakage, and reproducibility) and pair each with concrete mitigation strategies (e.g., versioned model archives, metamorphic test data, multiple replication runs with variability metrics, and detailed execution metadata) [132]. Du et al pair each threat in their *ClassEval* evaluation with a concrete mitigation: manually constructing the benchmark with multiple annotators to limit data leakage, piloting prompts on held-out tasks to control for prompt sensitivity, and reporting greedy-decoding results to control for non-determinism [42].

5.8.4 ***Benefits:***

Transparent reporting of limitations and mitigations helps readers calibrate confidence in the findings, makes explicit which threats were addressed and which remain open, and documents design decisions that other authors can borrow or refine. It also keeps a paper's claims proportionate to its evidence.

5.8.5 ***Challenges:***

Identifying limitations one is not already aware of is the hardest part of writing a threats section, particularly for methodological threats outside the team's primary expertise. Publication and reviewing norms can pressure authors to downplay weaknesses, while page limits make exhaustive treatment impractical. Threat lists that recite generic LLM-research issues (e.g., model bias, non-determinism, or contamination) without showing how each one applies to specific design choices in this study leave reviewers unable to tell which risks actually applied.

The threats to validity framework itself is contested within SE. Verdecchia et al argue that threats sections too often read as "*laundry-lists*" [154], Lago et al corroborated this empirically across a decade of ICSE Distinguished Paper Award winners [86], and Robillard et al argue for refocusing the discussion on study design trade-offs rather than the standard validity categories [126].

5.8.6 ***Study Types:***

Researchers **must** follow this guideline for all study types. Transparently reporting limitations and mitigations is a universal requirement, but specific concerns vary by study type. For *LLMs as Annotators*, researchers **must** discuss potential biases in label assignment, label reliability limitations, and sensitivity of annotations to prompt wording and model choice. For *LLMs as Judges*, researchers **must** address measurement validity concerns, known biases such as position bias or verbosity bias, and the extent to which LLM judgments align with human expert assessments. For *LLMs for Synthesis*, researchers **must** discuss the risk of contextual misinterpretation, potential loss of nuance in

summarized or aggregated outputs, and reflexivity limitations inherent in using an LLM for qualitative interpretation. For *LLMs as Subjects*, researchers **must** discuss the fundamental inability of LLMs to truly simulate human behavior, the risk of stereotype amplification, and the limited ecological validity of simulated responses. For *Studying LLM Usage*, researchers **must** discuss generalizability constraints across different tools and user populations, and acknowledge how observed usage patterns may not transfer to other contexts. For *LLMs for Tools*, researchers **must** discuss replicability constraints arising from dependencies on commercial models, the impact of model updates on tool behavior, and limitations of the evaluation setup. For *Benchmarking LLMs*, researchers **must** discuss potential data contamination, benchmark scope limitations, capability confounding, and the extent to which benchmark performance generalizes to real-world tasks.

5.8.7 ***Advice for Reviewers:***

Reviewers should verify that the limitation section is comprehensive and appropriate for the specific study type, checking that: (1) limitations address the specific threats relevant to the study type (e.g., label reliability for annotation studies, simulation fidelity for studies using LLMs as subjects); (2) mitigations are concrete and correspond to identified limitations rather than being generic statements; (3) the impact of LLM non-determinism on findings is discussed; (4) generalizability constraints, across models, configurations, time periods, and populations, are acknowledged. When important limitations are missing, reviewers should request they be added. The absence of a limitation section, or one that is formulaic or insufficiently specific, is a more serious concern than any individual missing limitation and may warrant a major revision.

5.8.8 ***See Also:***

→ Section 5.2 (Report Model Version, Configuration, and Customizations): Authors can re-run with different models or configurations to check whether the results depend on those specific choices.
→ Section 5.3 (Report System and Prompt Design): Triangulation across architectures and prompts is one mitigation strategy.
→ Section 5.4 (Report Session Traces): Stored session traces serve as a baseline against which authors can monitor LLM behavior drift over time.
→ Section 5.5 (Use Suitable Baselines, Benchmarks, and Metrics): Benchmark and metric choices are one source of construct-validity threats authors must discuss.
→ Section 5.6 (Use an Open LLM as a Baseline): An open LLM baseline mitigates cross-model transfer concerns by providing an independently reproducible comparison.
→ Section 5.7 (Use Human Validation for LLM Outputs): When automated metrics cannot validly measure a construct, human validation is an alternative.

6 Conclusion

LLM non-determinism, opaque training data, and rapidly evolving models threaten the reproducibility and replicability of empirical SE studies. This paper presents a taxonomy of seven study types that organizes how LLMs are used in SE research, together with eight guidelines for designing and reporting such studies. Each guideline distinguishes requirements (**must**) from recommendations (**should**) and is contextualized by the study types it applies to. Our guidelines recommend that researchers: (1) declare LLM usage and role; (2) report model versions, configurations, and customizations; (3) document the system and prompt design beyond the model; (4) report session traces, i.e., interaction logs and runtime traces; (5) use suitable baselines, benchmarks, and metrics; (6) include an open LLM as a baseline; (7) validate LLM outputs against human judgment; and (8) articulate limitations and mitigations. An applicability matrix maps guidelines to study types, and a reporting checklist (Appendix C) supports authors and reviewers. The study types and guidelines were derived collaboratively by 22 co-authors, following a process similar to that of guidelines in other disciplines (see, e.g., [23]).

Following the quality dimensions of the *SIGSOFT Empirical Standards* [118], we consider our guidelines (1) *comprehensive*, as they cover seven study types that we identified and were derived from 22 co-authors' collective expertise; (2) *useful*, as they provide actionable recommendations with an applicability matrix and a concrete reporting checklist (Appendix C); (3) *well-argued*, as each guideline pairs a dedicated rationale with concrete examples from published SE studies; and (4) *integrated with previous work*, as they build on and complement established empirical SE standards while addressing LLM-specific challenges. Adopting these practices will not eliminate LLM non-determinism, but it would make claims auditable and results easier to replicate and compare across studies. Because models and tools evolve, we maintain the study types and guidelines as a living resource and invite the community to refine and extend them (llm-guidelines.org).

Acknowledgments

Our study types and guidelines are based on discussion sessions with researchers at the *2024 International Software Engineering Research Network* (ISERN) meeting and at the *2nd Copenhagen Symposium on Human-Centered Software Engineering AI* (supported by the Carlsberg Foundation with grant CF24-0693 and the Alfred P. Sloan Foundation with grant G-2024-22586 to Daniel Russo). We thank Steffen Herbold and Alexander Serebrenik for their feedback on the guidelines. LLM-based tools, including ChatGPT (GPT-5 and GPT-5.5) and Claude Code (Opus 4.6, 4.7, and 4.8; Fable 5), were used interactively for language editing and structural revision suggestions across all sections of this paper. All LLM-generated suggestions were reviewed and revised by the authors.

Appendix A Applicability Matrix

Table 3 Applicability of guidelines to study types. ● = the guideline's core recommendations **must** be followed for this study type, ○ = **should** be followed, – = are not directly applicable. Each guideline's study-type-specific guidance is detailed in the corresponding subsection.

	Annotators	Judges	Synthesis	Subjects	Usage	Tools	Benchmarking
Declare Usage	●	●	●	●	●	●	●
Model Version	●	●	●	●	●	●	●
Design	●	●	●	●	○	●	●
Traces	○	○	○	○	●	○	○
Benchmarks & Metrics	●	○	○	○	○	○	●
Open LLM	○	○	○	–	–	○	○
Human Validation	○	○	○	○	○	○	○
Limitations	●	●	●	●	●	●	●

Appendix B Rationale and Recommendations

Table 4 Rationale and key recommendations for each guideline. ● = **must**, ○ = **should**.

Guideline	Rationale	Core Recommendations
Declare Usage	Transparency enables informed assessment of scope and limitations.	● Declare which LLM, how it was used, and where in the research process.
Model Version	Reproducibility requires precise identification of the system used in a study.	● Report exact version, date, configuration, and fine-tuning details. ○ Report defaults, checksums, and quantization; motivate model choice; acknowledge commercial-model reproducibility limits.
Design	Static artifacts determine the model's input on every call and must be documented in full.	● Describe system and agent architecture, infrastructure, prompts, agent configuration, tool catalog, and retrieval mechanisms. ○ For LLM usage, describe the tool architecture to the extent accessible.
Traces	Runtime traces make LLM and agent behavior verifiable despite non-determinism and tool opacity.	● For studies of LLM usage, share full interaction logs subject to privacy constraints. ○ Otherwise, share interaction logs, runtime traces, and plans where feasible.
Benchmarks & Metrics	Meaningful evaluation requires reasoned valid measurement.	● Justify metric and benchmark choices; discuss their validity. ○ Define the phenomenon and sampling strategy; isolate confounders; analyze errors; repeat experiments and report distributions.
Open LLM	Reproducibility depends on access to the model under study.	○ Include an open LLM as a baseline; ensure it is independently reproducible from supplementary material; for benchmarks, design the harness for use with open models.
Human Validation	Automated metrics alone cannot ensure validity of subjective constructs.	● Define the measured construct and share custom measurement instruments. ● When LLMs replace humans in research tasks, explain whether and how the replacement is justified. ○ When LLMs replace humans in research tasks, ground the replacement with inter-model and model-to-human agreement. ○ Validate against human judgment with inter-rater reliability; describe rater demographics for value-laden constructs.
Limitations	Honest acknowledgment of threats strengthens a study.	● Discuss threats to internal validity (data leakage), reliability (non-determinism), construct and external validity. ○ Employ mitigations where possible.

Appendix C Reporting Checklist

The following checklist, inspired by CONSORT [140], summarizes actionable items from the guidelines. The checklist is organized along typical paper sections. Items marked ● are requirements (**must**), and items marked ○ are recommendations (**should**). Each item references its source guideline by short name. Items annotated with *paper* or *supplementary material* indicate where we expect the information to be reported. Unmarked items may be reported either in the paper or as supplementary material. Items prefixed with a bracketed tag apply only to studies with that characteristic (e.g., `[fine-tuning]`, `[agents]`).Beyond these characteristic-tagged items, each guideline's *Study Types* subsection lists study-type-specific recommendations where applicable.

Introduction

● Disclose any use of LLMs in the empirical study, specifying which LLM, how, and where it was used (Declare Usage).
○ Report in the *paper* the purpose of using LLMs, the tasks they automate, and the expected benefits (Declare Usage).

Research Design and Methods

Model Selection and Configuration

● Report in the *paper* the exact LLM model or tool version, the configuration, and the date of study execution (Model Version).
● `[fine-tuning]` For fine-tuned models, describe in the *paper* the fine-tuning goal, the dataset, and the procedure (Model Version).
○ Report default parameters and explain model and version choices (Model Version).
○ Report checksums and additional model properties where available; for commercial tools, openly acknowledge their reproducibility limits (Model Version).
○ `[quantization]` For quantized models, report the quantization level (e.g., 4-bit, 8-bit) and method (e.g., GPTQ or AWQ) (Model Version).
○ `[fine-tuning]` Compare base and fine-tuned models using suitable metrics and benchmarks; share fine-tuning data and weights as *supplementary material* (or justify in the *paper* why they cannot be shared) (Model Version).
○ `[commercial-models]` Include an open LLM as a baseline when using commercial models and report inter-model agreement (Open LLM).

System and Prompt Design

● Describe in the *paper* the full architecture of LLM-based tools, including the role of the LLM, interactions with other components, and overall system behavior (Design).
● Specify whether zero-shot, one-shot, or few-shot prompting was used (Design).
● Specify prompt reuse across models and configurations (Design).

● If the LLM was used in a standalone setup, with prompts sent directly to a model and no pre- or post-processing, state this explicitly (Design).
● Publish all prompts or, when using templates, prompt templates with representative instances, including their structure, content, formatting, and variable components, as *supplementary material*; include representative examples in the *paper* (Design).
● `[few-shot]` For few-shot prompts, explain in the *paper* how the examples were selected (Design).
● `[dynamic-prompts]` For dynamically generated prompts, document the code or rules that assemble each prompt from runtime inputs (Design).
● `[restricted-sharing]` For partially shared prompts, anonymize personal identifiers, replace proprietary code with placeholders, and clearly highlight modified sections (Design).
● `[context-files]` Describe in the *paper* any configuration mechanisms used (e.g., context files such as `CLAUDE.md` or `AGENTS.md`, skills, subagents, hooks, settings, rules) (Design).
● `[tool-use]` Summarize in the *paper* which tools were exposed to the model (Design).
● `[agents]` If autonomous agents are used, specify agent roles, reasoning frameworks, and communication flows (Design).
● `[agents]` For agentic systems that use external tools, distinguish the model's reasoning and outputs, its tool calls, and its interactions with users, the environment, or other agents (Design).
● `[context-augmentation]` For retrieval-augmented generation (RAG) or related methods, describe how external data was retrieved, stored, and selected for inclusion in the model's context (Design).
● `[benchmarking]` Describe the evaluation harness and infrastructure when it goes beyond bare model API calls (e.g., custom sandboxing, orchestration layers, or post-processing pipelines) (Design).
● `[benchmarking]` For pre-defined prompts from existing benchmarks, specify the benchmark version, and disclose and justify any modifications to the prompts or evaluation setup (Design).
● `[latency-sensitive]` For time-sensitive measurements, clarify whether local infrastructure or cloud services were used, including the specific hardware for local hosting (e.g., GPU model and VRAM) or the service tier for cloud APIs (Design).
○ Justify substantive architectural choices where alternatives existed (e.g., agentic framework, tool catalog) (Design).
○ Describe how the models were hosted and accessed (Design).
○ Describe prompt development rationale and selection process (Design).
○ Report prompt evolution and any LLM-suggested refinements (Design).
○ Where legally possible, release the source code of the implementation under an open-source license (Design).
○ `[few-shot]` Include the concrete few-shot examples in the *supplementary material* (Design).

○ `[participant-prompts]` For user-authored prompts, describe how they were collected and analyzed (Design).
○ `[long-prompts]` Document input handling and token optimization strategies when prompts are long or complex (Design).
○ `[restricted-sharing]` If full prompt disclosure is not feasible, provide summaries or examples (Design).
○ `[ensemble]` For ensemble architectures, explain in the *paper* the coordination logic between models (Design).
○ `[context-augmentation]` Report data preprocessing, versioning, and update frequency for stored data used for context augmentation (Design).
○ `[context-files]` Include all configuration artifacts (context files, skill folders, subagent files, hooks, settings, rules) as *supplementary material* (Design).
○ `[tool-use]` Include the tool catalog (names with purposes), tool schemas, and connected MCP servers as *supplementary material* (Design).
○ `[benchmarking]` Design the evaluation harness so it is usable with open models (Design).

Session Traces

● Where tool-native trace formats are used (e.g., Claude Code's session transcripts, LangGraph's state logs), describe the file format and report the tool version (Traces).
● `[restricted-sharing]` For traces containing sensitive information, anonymize personal identifiers, replace proprietary code with placeholders, and clearly highlight modified sections (Traces).
○ Use an open trace format with a documented schema where it fits (e.g., the OpenTelemetry GenAI semantic conventions or OpenInference) (Traces).
○ Include full interaction logs (prompts and responses) as *supplementary material* if privacy and confidentiality can be ensured (Traces).
○ `[agents]` For agentic systems, include interaction logs covering human-in-the-loop exchanges with the agent (feedback, approvals, refinements) as *supplementary material* (Traces).
○ `[agents]` For agentic systems, report the complete runtime trace as *supplementary material*, including for each entry the tool or artifact name, arguments, result, and ordering, and which configured artifacts (skills, context files, subagents) were activated (Traces).
○ `[agents]` For agentic systems, report any plans the system exposes as *supplementary material* (Traces).

Benchmarks and Metrics

● Justify in the *paper* all benchmark and metric choices (Benchmarks & Metrics).
● Discuss in the *paper* the reliability and validity, especially construct validity, of the selected benchmarks and metrics (Benchmarks & Metrics).
● Explain in the *paper* why the selected metrics are suitable for the specific study; prior adoption in related work alone is not sufficient justification (Benchmarks & Metrics).

● [latency-sensitive] Report latency when it can affect study outcomes (e.g., interactive user studies, latency comparisons) (Benchmarks & Metrics).
● [new-benchmark] For new or updated benchmarks, disclose data sources and collection dates for each release (Benchmarks & Metrics).
○ Provide an operational definition of the phenomenon the benchmark is intended to measure, including its scope and any sub-components (Benchmarks & Metrics).
○ Summarize benchmark structure, task types, and limitations (Benchmarks & Metrics).
○ Identify the capabilities a benchmark conflates with the target phenomenon, isolate the target where possible, and acknowledge remaining confounders as construct-validity threats (Benchmarks & Metrics).
○ Perform an error analysis: categorize the failures observed and report their relative frequency; report failures that cluster on confounding capabilities as construct-validity threats (Benchmarks & Metrics).
○ Describe and justify the sampling strategy used to select problems for inclusion in the benchmark (Benchmarks & Metrics).
○ Justify the number of experiment repetitions, for example through a power analysis or by monitoring convergence of descriptive statistics (Benchmarks & Metrics).
○ [non-probability-sampling] For non-probability sampling (e.g., convenience), discuss the implications for the generalizability of conclusions (Benchmarks & Metrics).
○ [new-benchmark] For new or released benchmarks, adopt contamination-prevention mechanisms: held-out subset, canary strings, and pre-exposure investigation against common training corpora (Benchmarks & Metrics).
○ [multi-rater-scoring] For ratings that vary across raters or runs (human raters, LLM-as-judge), report the distribution of ratings per item rather than only aggregated point estimates (Benchmarks & Metrics).

Human Validation

● [human-validation] If using human validation, define in the *paper* the measured construct (e.g., usability, maintainability) and describe the measurement instrument (Human Validation).
● [human-validation] When developing or adapting measurement instruments, share them (Human Validation).
● [human-validation] When LLMs replace humans in research tasks, explain whether and how the replacement is justified (Human Validation).
○ Consider human validation early in the study design and build on established reference models for human-LLM comparison (Human Validation).
○ [human-validation] When LLMs replace humans in research tasks, report the systematic approach used to justify the replacement, including inter-model and model-to-human agreement (Human Validation).
○ [human-validation] Validate LLM judgments against human judgment, report aggregation methods, and assess human-LLM agreement (Human Validation).

○ `[human-validation]` Discuss and, where feasible, control for confounding factors (Human Validation).
○ `[human-validation]` `[subjective-constructs]` For value-laden or culturally contingent constructs, describe rater demographics beyond expertise and discuss potential demographic biases (Human Validation).
○ `[agents]` When evaluating agentic tools, assess the feedback users provided on the agent's proposed actions, and report how frequently proposals were accepted and how they were modified (Human Validation).

Reproducibility, Ethics, and Resources

● `[restricted-sharing]` For studies involving sensitive data, discuss data governance mechanisms compliant with applicable jurisdictional obligations (Limitations).
○ Justify LLM usage in light of its resource demands (Limitations).
○ Ensure the open-LLM baseline is independently reproducible from the *supplementary material* (Open LLM).
○ `[restricted-sharing]` Where full sharing of prompts, traces, or datasets is not feasible, share representative examples for partial replicability (Limitations).

Results

○ Repeat experiments due to the inherent non-determinism of LLMs and report the result distribution using descriptive statistics (Benchmarks & Metrics).
○ Use traditional (non-LLM) baselines for comparison where possible (Benchmarks & Metrics).
○ Report established metrics to make study results comparable; additional metrics may be reported where appropriate (Benchmarks & Metrics).
○ `[comparing-models]` If comparing models or tools, use appropriate inferential statistics (e.g., hypothesis tests, effect sizes) rather than relying solely on summary statistics (Benchmarks & Metrics).

Limitations and Threats to Validity

● Discuss potential data leakage effects and their impact on results, and describe how the quality of subjective results was ensured (Limitations).
● Transparently report study limitations, including the impact of non-determinism and generalizability constraints (Limitations).
● Specify whether generalization across LLMs or across time was assessed, and discuss model and version differences (Limitations).
● `[restricted-sharing]` Acknowledge non-disclosed confidential or proprietary components as reproducibility limitations (Design).
○ Employ and report strategies to mitigate identified validity and reproducibility threats, such as replication packages, human validation, longitudinal re-runs, triangulation, and sensitivity analysis (Limitations).

7 Statements and Declarations

Competing Interests: The authors declare no competing interests. Multiple co-authors are members of the EMSE Editorial Board, which is disclosed here for transparency.

Authorship: Each author contributed or reviewed content and, most importantly, read the complete guidelines and confirmed that they stand behind all recommendations, not only their own contributions.

Funding: No funding was received for writing this article. However, the *2nd Copenhagen Symposium on Human-Centered Software Engineering AI*, which played a central role in forming the team of co-authors, was supported by the Carlsberg Foundation with grant CF24-0693 and the Alfred P. Sloan Foundation with grant G-2024-22586 to Daniel Russo.

Ethical Approval: No ethics approval was required for writing this article.

Data Availability: This article has no associated data. The guidelines are hosted online and maintained on GitHub.

References

1. ACM Publications Board (2020) Artifact review and badging – current. `https://www.acm.org/publications/policies/artifact-review-and-badging-current`, accessed 2025-01-13
2. Agarwal R, Schwarzer M, Castro PS, Courville AC, Bellemare MG (2021) Deep reinforcement learning at the edge of the statistical precipice. In: Ranzato M, Beygelzimer A, Dauphin YN, Liang P, Vaughan JW (eds) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, pp 29304–29320, URL `https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html`
3. Ahmed N, Wahed M, Thompson NC (2023) The growing influence of industry in AI research. Science 379(6635):884–886, DOI 10.1126/science.ade2420
4. Ahmed T, Devanbu PT, Treude C, Pradel M (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, pp 526–538, DOI 10.1109/MSR66628.2025.00086
5. Ahuja S, Gumma V, Sitaram S (2024) Contamination report for multilingual benchmarks. CoRR abs/2410.16186, DOI 10.48550/ARXIV.2410.16186, URL `https://doi.org/10.48550/arXiv.2410.16186`, `2410.16186`
6. Angermeir F, Amougou M, Kreitz M, Bauer A, Linhuber M, Fucci D, Constante FM, Méndez D, Gorschek T (2025) Reflections on the reproducibility of commercial LLM performance in empirical software engineer-

ing studies. CoRR abs/2510.25506, DOI 10.48550/ARXIV.2510.25506, URL https://doi.org/10.48550/arXiv.2510.25506, 2510.25506
7. Anthropic (2025) Demystifying Evals for AI Agents. https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents, accessed 2026-03-19
8. Anthropic (2026) Claude Code Tools Reference. https://code.claude.com/docs/en/tools-reference, accessed 2026-04-28
9. Argyle LP, Busby EC, Fulda N, Gubler J, Rytting CM, Wingate D (2022) Out of one, many: Using language models to simulate human samples. CoRR abs/2209.06899, DOI 10.48550/ARXIV.2209.06899, URL https://doi.org/10.48550/arXiv.2209.06899, 2209.06899
10. Arize AI (2026) OpenInference: AI observability and evaluation specification. https://github.com/Arize-ai/openinference, accessed 2026-05-07
11. Association for Computing Machinery (2026) ACM Policy on Authorship. https://www.acm.org/publications/policies/new-acm-policy-on-authorship, accessed 2026-06-01
12. Atil B, Aykent S, Chittams A, Fu L, Passonneau RJ, Radcliffe E, Rajagopal GR, Sloan A, Tudrej T, Ture F, Wu Z, Xu L, Baldwin B (2024) Non-determinism of "deterministic" LLM settings. CoRR abs/2408.04667, URL https://arxiv.org/abs/2408.04667, 2408.04667
13. Austin J, Odena A, Nye MI, Bosma M, Michalewski H, Dohan D, Jiang E, Cai CJ, Terry M, Le QV, Sutton C (2021) Program synthesis with large language models. CoRR abs/2108.07732, URL https://arxiv.org/abs/2108.07732, 2108.07732
14. Azanza M, Pereira J, Irastorza A, Galdos A (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, pp 1–6, DOI 10.1109/CSEET62301.2024.10662989
15. Baltes S, Ralph P (2022) Sampling in software engineering research: a critical review and guidelines. Empir Softw Eng 27(4):94, DOI 10.1007/s10664-021-10072-8, URL https://doi.org/10.1007/s10664-021-10072-8
16. Banerjee S, Agarwal A, Singh E (2024) The vulnerability of language model benchmarks: Do they accurately reflect true LLM performance? CoRR abs/2412.03597, URL https://arxiv.org/abs/2412.03597, 2412.03597
17. Bano M, Zowghi D, Whittle J (2023) Exploring qualitative research using llms. CoRR abs/2306.13298, DOI 10.48550/ARXIV.2306.13298, URL https://doi.org/10.48550/arXiv.2306.13298, 2306.13298
18. Bano M, Hoda R, Zowghi D, Treude C (2024) Large language models for qualitative research in software engineering: exploring opportunities and challenges. Autom Softw Eng 31(1):8, DOI 10.1007/S10515-023-00407-8, URL https://doi.org/10.1007/s10515-023-00407-8

19. Bano M, Gunatilake H, Hoda R (2025) What does a software engineer look like? exploring societal stereotypes in llms. In: 47th IEEE/ACM International Conference on Software Engineering: Software Engineering in Society, ICSE-SEIS 2025, Ottawa, ON, Canada, April 27 - May 3, 2025, IEEE, pp 173–184, DOI 10.1109/ICSE-SEIS66351.2025.00023, URL https://doi.org/10.1109/ICSE-SEIS66351.2025.00023
20. Barros CF, Azevedo BB, Neto VVG, Kassab M, Kalinowski M, do Nascimento HAD, Bandeira MCGSP (2025) Large language model for qualitative research: A systematic mapping study. In: IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering, WSESE@ICSE 2025, Ottawa, ON, Canada, May 3, 2025, IEEE, pp 48–55, DOI 10.1109/WSESE66602.2025.00015, URL https://doi.org/10.1109/WSESE66602.2025.00015
21. Bavaresco A, Bernardi R, Bertolazzi L, Elliott D, Fernández R, Gatt A, Ghaleb E, Giulianelli M, Hanna M, Koller A, Martins AFT, Mondorf P, Neplenbroek V, Pezzelle S, Plank B, Schlangen D, Suglia A, Surikuchi AK, Takmaz E, Testoni A (2024) LLMs instead of human judges? A large scale empirical study across 20 NLP evaluation tasks. CoRR abs/2406.18403, DOI 10.48550/ARXIV.2406.18403, URL https://doi.org/10.48550/arXiv.2406.18403, 2406.18403
22. Bean AM, Kearns RO, Romanou A, Hafner FS, Mayne H, Batzner J, Foroutan N, Schmitz C, Korgul K, Batra H, Deb O, Beharry E, Emde C, Foster T, Gausen A, Grandury M, Han S, Hofmann V, Ibrahim L, Kim H, Kirk HR, Lin F, Liu GKM, Luettgau L, Magomere J, Rystrøm J, Sotnikova A, Yang Y, Zhao Y, Bibi A, Bosselut A, Clark R, Cohan A, Foerster J, Gal Y, Hale SA, Raji ID, Summerfield C, Torr PHS, Ududec C, Rocher L, Mahdi A (2025) Measuring what matters: Construct validity in large language model benchmarks. In: Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, Datasets and Benchmarks Track, URL https://arxiv.org/abs/2511.04703
23. Begg C, Cho M, Eastwood S, Horton R, Moher D, Olkin I, Pitkin R, Rennie D, Schulz KF, Simel D, Stroup DF (1996) Improving the quality of reporting of randomized controlled trials. The CONSORT statement. JAMA 276(8):637–639, DOI 10.1001/jama.276.8.637
24. Bjarnason BH, Silva A, Monperrus M (2026) On randomness in agentic evals. CoRR abs/2602.07150, DOI 10.48550/ARXIV.2602.07150, URL https://doi.org/10.48550/arXiv.2602.07150, 2602.07150
25. Blackwell RE, Barry J, Cohn AG (2024) Towards reproducible LLM evaluation: Quantifying uncertainty in LLM benchmark scores. CoRR abs/2410.03492, DOI 10.48550/ARXIV.2410.03492, URL https://doi.org/10.48550/arXiv.2410.03492, 2410.03492
26. Borsboom D (2005) Measuring the Mind: Conceptual Issues in Contemporary Psychometrics. Cambridge University Press, DOI 10.1017/CBO9780511490026

27. Bouzenia I, Pradel M (2025) Understanding software engineering agents: A study of thought-action-result trajectories. In: 40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16-20, 2025, IEEE, pp 2846–2857, DOI 10.1109/ASE63991.2025.00234, URL https://doi.org/10.1109/ASE63991.2025.00234
28. Cai Y, Yadavally A, Mishra A, Montejo G, Nguyen TN (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, pp 94:1–94:13, DOI 10.1145/3597503.3639188
29. Cao J, Chan YK, Ling Z, Wang W, Li S, Liu M, Qiao R, Han Y, Wang C, Yu B, He P, Wang S, Zheng Z, Lyu MR, Cheung SC (2026) Rigor, reliability, and reproducibility matter: A decade-scale survey of 572 code benchmarks. CoRR abs/2501.10711, URL https://arxiv.org/abs/2501.10711, 2501.10711
30. Chandra S (2025) Benchmarks for AI in Software Engineering (BLOG@CACM). https://cacm.acm.org/blogcacm/benchmarks-for-ai-in-software-engineering/, accessed 2025-08-24
31. Chann S (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
32. Chen G, Chen S, Liu Z, Jiang F, Wang B (2024) Humans or llms as the judge? A study on judgement bias. In: Al-Onaizan Y, Bansal M, Chen Y (eds) Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024, Association for Computational Linguistics, pp 8301–8327, DOI 10.18653/V1/2024.EMNLP-MAIN.474, URL https://doi.org/10.18653/v1/2024.emnlp-main.474
33. Chen L, Zaharia M, Zou J (2024) How is ChatGPT's behavior changing over time? Harvard Data Science Review 6(2), DOI 10.1162/99608f92.5317da47
34. Chen M, Tworek J, Jun H, Yuan Q, de Oliveira Pinto HP, Kaplan J, Edwards H, Burda Y, Joseph N, Brockman G, Ray A, Puri R, Krueger G, Petrov M, Khlaaf H, Sastry G, Mishkin P, Chan B, Gray S, Ryder N, Pavlov M, Power A, Kaiser L, Bavarian M, Winter C, Tillet P, Such FP, Cummings D, Plappert M, Chantzis F, Barnes E, Herbert-Voss A, Guss WH, Nichol A, Paino A, Tezak N, Tang J, Babuschkin I, Balaji S, Jain S, Saunders W, Hesse C, Carr AN, Leike J, Achiam J, Misra V, Morikawa E, Radford A, Knight M, Brundage M, Murati M, Mayer K, Welinder P, McGrew B, Amodei D, McCandlish S, Sutskever I, Zaremba W (2021) Evaluating large language models trained on code. CoRR abs/2107.03374, URL https://arxiv.org/abs/2107.03374, 2107.03374
35. Chen Y, Ding Z, Alowain L, Chen X, Wagner DA (2023) DiverseVul: A new vulnerable source code dataset for deep learning based vulnerability detection. In: Proceedings of the 26th International Symposium on

Research in Attacks, Intrusions and Defenses, RAID 2023, Hong Kong, China, October 16-18, 2023, ACM, pp 654–668, DOI 10.1145/3607199.3607242

36. Cheng A, Calhoun A, Reedy G (2025) Artificial intelligence-assisted academic writing: recommendations for ethical use. Advances in Simulation 10(1):26, DOI 10.1186/s41077-025-00350-6, URL https://doi.org/10.1186/s41077-025-00350-6
37. Choi HK, Khanov M, Wei H, Li Y (2025) How contaminated is your benchmark? quantifying dataset leakage in large language models with kernel divergence. CoRR abs/2502.00678, DOI 10.48550/ARXIV.2502.00678, URL https://doi.org/10.48550/arXiv.2502.00678, 2502.00678
38. Cohen J (1992) A power primer. Psychological Bulletin 112(1):155–159, DOI 10.1037/0033-2909.112.1.155
39. DAIRAI (2024) Elements of a Prompt. https://www.promptingguide.ai/introduction/elements, accessed 2026-04-28
40. Dhar R, Vaidhyanathan K, Varma V (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, pp 79–89, DOI 10.1109/ICSA59870.2024.00016
41. Dror R, Baumer G, Shlomov S, Reichart R (2018) The hitchhiker's guide to testing statistical significance in natural language processing. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Volume 1: Long Papers, Association for Computational Linguistics, pp 1383–1392, DOI 10.18653/v1/P18-1128, URL https://aclanthology.org/P18-1128/
42. Du X, Liu M, Wang K, Wang H, Liu J, Chen Y, Feng J, Sha C, Peng X, Lou Y (2024) Evaluating large language models in class-level code generation. In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024, ACM, pp 81:1–81:13, DOI 10.1145/3597503.3639219, URL https://doi.org/10.1145/3597503.3639219
43. Dybå T, Kampenes VB, Sjøberg DIK (2006) A systematic review of statistical power in software engineering experiments. Inf Softw Technol 48(8):745–755, DOI 10.1016/J.INFSOF.2005.08.009, URL https://doi.org/10.1016/j.infsof.2005.08.009
44. Eghbali A, Pradel M (2022) CrystalBLEU: Precisely and efficiently measuring the similarity of code. In: 37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, ACM, pp 28:1–28:12, DOI 10.1145/3551349.3556903
45. El-Hajjami A, Salinesi C (2025) How good are synthetic requirements? evaluating LLM-generated datasets for AI4RE. CoRR abs/2506.21138, DOI 10.48550/ARXIV.2506.21138, URL https://doi.org/10.48550/arXiv.2506.21138, 2506.21138

46. Evertz J, Risse N, Neuer N, Müller A, Normann P, Sapia G, Gupta S, Pape D, Shaw S, Srivastav D, Wressnegger C, Quiring E, Eisenhofer T, Arp D, Schönherr L (2026) Chasing shadows: Pitfalls in LLM security research. In: 33rd Annual Network and Distributed System Security Symposium, NDSS 2026, San Diego, California, USA, February 23-27, 2026, The Internet Society, URL https://www.ndss-symposium.org/ndss-paper/chasing-shadows-pitfalls-in-llm-security-research/
47. Gallegos IO, Rossi RA, Barrow J, Tanjim MM, Kim S, Dernoncourt F, Yu T, Zhang R, Ahmed NK (2024) Bias and fairness in large language models: A survey. Comput Linguistics 50(3):1097–1179, DOI 10.1162/COLI_A_00524, URL https://doi.org/10.1162/coli_a_00524
48. Gallifant J, Afshar M, Ameen S, Aphinyanaphongs Y, Chen S, Cacciamani G, Demner-Fushman D, Dligach D, Daneshjou R, Fernandes C, Hansen LH, Landman A, Lehmann L, McCoy LG, Miller T, Moreno A, Munch N, Restrepo D, Savova G, Umeton R, Gichoya JW, Collins GS, Moons KGM, Celi LA, Bitterman DS (2025) The TRIPOD-LLM reporting guideline for studies using large language models. Nature Medicine 31(1):60–69, DOI 10.1038/s41591-024-03425-5
49. Galster M, Mohsenimofidi S, Lulla JL, Abubakar MA, Treude C, Baltes S (2026) Configuring agentic AI coding tools: An exploratory study. In: Proceedings of the 3rd ACM International Conference on AI-powered Software (AIware 2026)
50. Gerosa MA, Trinkenreich B, Steinmacher I, Sarma A (2024) Can AI serve as a substitute for human subjects in software engineering research? Autom Softw Eng 31(1):13, DOI 10.1007/S10515-023-00409-6, URL https://doi.org/10.1007/s10515-023-00409-6
51. Gibney E (2024) Not all 'open source' AI models are actually open. Nature News DOI 10.1038/d41586-024-02012-5
52. Gilardi F, Alizadeh M, Kubli M (2023) ChatGPT outperforms crowd workers for text-annotation tasks. Proceedings of the National Academy of Sciences 120(30):e2305016120, DOI 10.1073/pnas.2305016120
53. Golnari PA, Kumarappan A, Wen W, Liu X, Ryan G, Sun Y, Fu S, Nallipogu E (2026) Devbench: A realistic, developer-informed benchmark for code generation models. CoRR abs/2601.11895, DOI 10.48550/ARXIV.2601.11895, URL https://doi.org/10.48550/arXiv.2601.11895, 2601.11895
54. Gonçalves J, Silva M, Cabral B, Dias T, Maia E, Praça I, Severino R, Ferreira LL (2025) Evaluating llama 3.2 for software vulnerability detection. In: Praça I, Bernardi S, Inácio PRM (eds) Cybersecurity - 9th European Interdisciplinary Cybersecurity Conference, EICC 2025, Rennes, France, June 18-19, 2025, Proceedings, Springer, Communications in Computer and Information Science, pp 38–51, DOI 10.1007/978-3-031-94855-8_3, URL https://doi.org/10.1007/978-3-031-94855-8_3
55. Graziotin D (2024) acmsigsoft/open-science-policies: v1.0.0. DOI 10.5281/zenodo.10796477

56. Guba EG (1981) Criteria for assessing the trustworthiness of naturalistic inquiries. ECTJ 29(2):75–91, DOI 10.1007/BF02766777
57. Guo D, Zhu Q, Yang D, Xie Z, Dong K, Zhang W, Chen G, Bi X, Wu Y, Li YK, Luo F, Xiong Y, Liang W (2024) Deepseek-coder: When the large language model meets programming - the rise of code intelligence. CoRR abs/2401.14196, DOI 10.48550/ARXIV.2401.14196, URL https://doi.org/10.48550/arXiv.2401.14196, 2401.14196
58. Harding J, D'Alessandro W, Laskowski NG, Long R (2024) AI language models cannot replace human research participants. AI Soc 39(5):2603–2605, DOI 10.1007/S00146-023-01725-X, URL https://doi.org/10.1007/s00146-023-01725-x
59. He J, Rungta M, Koleczek D, Sekhon A, Wang FX, Hasan S (2024) Does prompt formatting have any impact on LLM performance? CoRR abs/2411.10541, DOI 10.48550/ARXIV.2411.10541, URL https://doi.org/10.48550/arXiv.2411.10541, 2411.10541
60. He J, Shi J, Zhuo TY, Treude C, Sun J, Xing Z, Du X, Lo D (2026) LLM-as-a-judge for software engineering: Literature review, vision, and the road ahead. ACM Transactions on Software Engineering and Methodology DOI 10.1145/3797276, URL https://doi.org/10.1145/3797276
61. He Z, Huang C, Ding CC, Rohatgi S, Huang TK (2024) If in a crowdsourced data annotation pipeline, a GPT-4. In: Mueller FF, Kyburz P, Williamson JR, Sas C, Wilson ML, Dugas POT, Shklovski I (eds) Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, ACM, pp 1040:1–1040:25, DOI 10.1145/3613904.3642834
62. Holtzman A, Buys J, Du L, Forbes M, Choi Y (2020) The curious case of neural text degeneration. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, URL https://openreview.net/forum?id=rygGQyrFvH
63. Hosking T, Blunsom P, Bartolo M (2024) Human feedback is not gold standard. In: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, OpenReview.net, URL https://openreview.net/forum?id=7W3GLNImfS
64. Hou X, Zhao Y, Liu Y, Yang Z, Wang K, Li L, Luo X, Lo D, Grundy J, Wang H (2024) Large language models for software engineering: A systematic literature review. ACM Trans Softw Eng Methodol 33(8):220:1–220:79, DOI 10.1145/3695988, URL https://doi.org/10.1145/3695988
65. Hu X, Niu F, Chen J, Zhou X, Zhang J, He J, Xia X, Lo D (2025) Assessing and advancing benchmarks for evaluating large language models in software engineering tasks. CoRR abs/2505.08903, DOI 10.48550/ARXIV.2505.08903, URL https://doi.org/10.48550/arXiv.2505.08903, 2505.08903
66. Huang F, Kwak H, An J (2023) Is ChatGPT better than human annotators? potential and limitations of ChatGPT in explaining implicit hate

speech. In: Ding Y, Tang J, Sequeda JF, Aroyo L, Castillo C, Houben G (eds) Companion Proceedings of the ACM Web Conference 2023, WWW 2023, ACM, pp 294–297, DOI 10.1145/3543873.3587368

67. Huang J, Yi B, Sun W, Wan B, Xu Y, Feng Y, Ye W, Qin Q (2024) Enhancing review classification via LLM-based data annotation and multi-perspective feature representation learning. SSRN Electronic Journal pp 1–15, DOI 10.2139/ssrn.5002351, URL https://ssrn.com/abstract=5002351
68. Hui B, Yang J, Cui Z, Yang J, Liu D, Zhang L, Liu T, Zhang J, Yu B, Dang K, Yang A, Men R, Huang F, Ren X, Ren X, Zhou J, Lin J (2024) Qwen2.5-coder technical report. CoRR abs/2409.12186, DOI 10.48550/ARXIV.2409.12186, URL https://doi.org/10.48550/arXiv.2409.12186, 2409.12186
69. Husain H, Wu H, Gazit T, Allamanis M, Brockschmidt M (2019) CodeSearchNet challenge: Evaluating the state of semantic code search. CoRR abs/1909.09436, URL http://arxiv.org/abs/1909.09436, 1909.09436
70. Hymel C, Johnson H (2025) Analysis of llms vs human experts in requirements engineering. CoRR abs/2501.19297, DOI 10.48550/ARXIV.2501.19297, URL https://doi.org/10.48550/arXiv.2501.19297, 2501.19297
71. Jahic J, Sami A (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, pp 311–318, DOI 10.1109/ICSA-C63560.2024.00059
72. Jain N, Han K, Gu A, Li W, Yan F, Zhang T, Wang S, Solar-Lezama A, Sen K, Stoica I (2024) LiveCodeBench: Holistic and contamination free evaluation of large language models for code. CoRR abs/2403.07974, DOI 10.48550/ARXIV.2403.07974, URL https://doi.org/10.48550/arXiv.2403.07974, 2403.07974
73. Jimenez CE, Yang J, Wettig A, Yao S, Pei K, Press O, Narasimhan KR (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, URL https://openreview.net/forum?id=VTF8yNQM66
74. Jowsey T, Braun V, Clarke V, Lupton D, Fine M (2025) We reject the use of generative artificial intelligence for reflexive qualitative research. Qualitative Inquiry DOI 10.1177/10778004251401851, URL https://doi.org/10.1177/10778004251401851
75. Kang S, Yoon J, Yoo S (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, pp 2312–2323, DOI 10.1109/ICSE48619.2023.00194
76. Kapoor S, Cantrell EM, Peng K, Pham TH, Bail CA, Gundersen OE, Hofman JM, Hullman J, Lones MA, Malik MM, Nanayakkara P, Poldrack RA, Raji ID, Roberts M, Salganik MJ, Serra-Garcia M, Stewart BM, Vandewiele G, Narayanan A (2024) REFORMS: Consensus-based

recommendations for machine-learning-based science. Science Advances 10(18):eadk3452, DOI 10.1126/sciadv.adk3452, URL https://doi.org/10.1126/sciadv.adk3452

77. Khojah R, Mohamad M, Leitner P, de Oliveira Neto FG (2024) Beyond code generation: An observational study of ChatGPT usage in software engineering practice. Proc ACM Softw Eng 1(FSE):1819–1840, DOI 10.1145/3660788
78. Khraisha Q, Put S, Kappenberg J, Warraitch A, Hadfield K (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):616–626, DOI 10.1002/jrsm.1715, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jrsm.1715
79. Kirk R, Mediratta I, Nalmpantis C, Luketina J, Hambro E, Grefenstette E, Raileanu R (2023) Understanding the effects of RLHF on LLM generalisation and diversity. CoRR abs/2310.06452, URL https://arxiv.org/abs/2310.06452, 2310.06452
80. von Kistowski J, Arnold JA, Huppler K, Lange K, Henning JL, Cao P (2015) How to build a benchmark. In: John LK, Smith CU, Sachs K, Lladó CM (eds) Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, January 31 - February 4, 2015, ACM, pp 333–336, DOI 10.1145/2668930.2688819, URL https://doi.org/10.1145/2668930.2688819
81. Kong A, Zhao S, Chen H, Li Q, Qin Y, Sun R, Zhou X, Wang E, Dong X (2024) Better zero-shot reasoning with role-play prompting. In: Duh K, Gómez-Adorno H, Bethard S (eds) Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024, Association for Computational Linguistics, pp 4099–4113, DOI 10.18653/V1/2024.NAACL-LONG.228, URL https://doi.org/10.18653/v1/2024.naacl-long.228
82. Korn A, Zaruchas L, Arora C, Metzger A, Smolka S, Wang F, Vogelsang A (2026) Reporting LLM prompting in automated software engineering: A guideline based on current practices and expectations. CoRR abs/2601.01954, DOI 10.48550/ARXIV.2601.01954, URL https://doi.org/10.48550/arXiv.2601.01954, to appear at FORGE 2026, co-located with ICSE 2026, Rio de Janeiro, Brazil, 2601.01954
83. Krippendorff K (2018) Content Analysis: An Introduction to Its Methodology, 4th edn. SAGE Publications
84. Kübler JM, Budhathoki K, Kleindessner M, Zhou X, Yin J, Khetan A, Karypis G (2026) When LLMs get significantly worse: A statistical approach to detect model degradations. In: Proc. of the International Conference on Learning Representations (ICLR)
85. Kulal S, Pasupat P, Chandra K, Lee M, Padon O, Aiken A, Liang P (2019) Spoc: Search-based pseudocode to code. In: Wallach HM,

Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox EB, Garnett R (eds) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp 11883–11894, URL https://proceedings.neurips.cc/paper/2019/hash/7298332f04ac004a0ca44cc69ecf6f6b-Abstract.html

86. Lago P, Runeson P, Song Q, Verdecchia R (2024) Threats to validity in software engineering - hypocritical paper section or essential analysis? In: Franch X, Daneva M, Martínez-Fernández S, Quaranta L (eds) Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2024, Barcelona, Spain, October 24-25, 2024, ACM, pp 314–324, DOI 10.1145/3674805.3686691, URL https://doi.org/10.1145/3674805.3686691
87. Li D, Gupta K, Bhaduri M, Sathiadoss P, Bhatnagar S, Chong J (2024) Comparing GPT-3.5 and GPT-4 accuracy and drift in radiology diagnosis please cases. Radiology 310(1):e232411, DOI 10.1148/radiol.232411
88. Li R, Allal LB, Zi Y, Muennighoff N, Kocetkov D, Mou C, Marone M, Akiki C, Li J, Chim J, Liu Q, Zheltonozhskii E, Zhuo TY, Wang T, Dehaene O, Davaadorj M, Lamy-Poirier J, Monteiro J, Shliazhko O, Gontier N, Meade N, Zebaze A, Yee M, Umapathi LK, Zhu J, Lipkin B, Oblokulov M, Wang Z, V RM, Stillerman JT, Patel SS, Abulkhanov D, Zocca M, Dey M, Zhang Z, Fahmy N, Bhattacharyya U, Yu W, Singh S, Luccioni S, Villegas P, Kunakov M, Zhdanov F, Romero M, Lee T, Timor N, Ding J, Schlesinger C, Schoelkopf H, Ebert J, Dao T, Mishra M, Gu A, Robinson J, Anderson CJ, Dolan-Gavitt B, Contractor D, Reddy S, Fried D, Bahdanau D, Jernite Y, Ferrandis CM, Hughes S, Wolf T, Guha A, von Werra L, de Vries H (2023) Starcoder: may the source be with you! Trans Mach Learn Res 2023, URL https://openreview.net/forum?id=KoFOg41haE
89. Liang JT, Badea C, Bird C, DeLine R, Ford D, Forsgren N, Zimmermann T (2024) Can GPT-4 replicate empirical software engineering research? Proc ACM Softw Eng 1(FSE):1330–1353, DOI 10.1145/3660767, URL https://doi.org/10.1145/3660767
90. Liu J, Xia CS, Wang Y, Zhang L (2023) Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In: Oh A, Naumann T, Globerson A, Saenko K, Hardt M, Levine S (eds) Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, URL http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html
91. Liu M, Huang X, He W, Xie Y, Zhang JM, Jing X, Chen Z, Ma Y (2024) Research artifacts in software engineering publications: Status and trends. J Syst Softw 213:112032, DOI 10.1016/J.JSS.2024.112032, URL https://doi.org/10.1016/j.jss.2024.112032
92. Liu X, Yu H, Zhang H, Xu Y, Lei X, Lai H, Gu Y, Ding H, Men K, Yang K, Zhang S, Deng X, Zeng A, Du Z, Zhang C, Shen S, Zhang T, Su Y,

Sun H, Huang M, Dong Y, Tang J (2024) Agentbench: Evaluating llms as agents. In: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, OpenReview.net, URL https://openreview.net/forum?id=zAdUB0aCTQ

93. Lubos S, Felfernig A, Tran TNT, Garber D, Mansi ME, Erdeniz SP, Le V (2024) Leveraging LLMs for the quality assurance of software requirements. In: Liebel G, Hadar I, Spoletini P (eds) 32nd IEEE International Requirements Engineering Conference, RE 2024, Reykjavik, Iceland, June 24-28, 2024, IEEE, pp 389–397, DOI 10.1109/RE59067.2024.00046
94. Madampe K, Grundy J, Hoda R, Obie HO (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, pp 417–422, DOI 10.1109/VL/HCC60511.2024.00065
95. McDonald N, Schoenebeck S, Forte A (2019) Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI practice. Proc ACM Hum Comput Interact 3(CSCW):72:1–72:23, DOI 10.1145/3359174
96. Menzies T (2025) The case for compact AI. Commun ACM 68(9):6–7, DOI 10.1145/3746057
97. Microsoft (2023) CodeBERT on GitHub. https://github.com/microsoft/CodeBERT, accessed 2025-08-15
98. Microsoft (2025) Azure OpenAI Service models. https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models, accessed 2025-01-13
99. Mitu NE, Mitu GT (2024) The hidden cost of AI: Carbon footprint and mitigation strategies. Revista de Stiinte Politice Revue des Sciences Politiques 84:9–16, DOI 10.2139/ssrn.5036344
100. Mohamed S, Parvin A, Parra E (2024) Chatting with AI: deciphering developer conversations with ChatGPT. In: Spinellis D, Bacchelli A, Constantinou E (eds) 21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024, ACM, pp 187–191, DOI 10.1145/3643991.3645078
101. Montes CM, Feldt R, Martos CM, Ouhbi S, Premanandan S, Graziotin D (2025) Large language models in thematic analysis: Prompt engineering, evaluation, and guidelines for qualitative software engineering research. CoRR abs/2510.18456, DOI 10.48550/ARXIV.2510.18456, URL https://doi.org/10.48550/arXiv.2510.18456, 2510.18456
102. de Morais Leça M, Valença L, Santos R, de Souza Santos R (2025) Applications and implications of large language models in qualitative analysis: A new frontier for empirical software engineering. In: IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering, WSESE@ICSE 2025, Ottawa, ON, Canada, May 3, 2025, IEEE, pp 36–43, DOI 10.1109/WSESE66602.2025.00013, URL https://doi.org/10.1109/WSESE66602.2025.00013

103. Moumoula MB, Kaboré AK, Klein J, Bissyandé TF (2024) Large language models for cross-language code clone detection. CoRR abs/2408.04430, DOI 10.48550/ARXIV.2408.04430, URL https://doi.org/10.48550/arXiv.2408.04430, 2408.04430
104. Navarro KF, Syriani E, Arawjo I (2026) Reporting and reviewing LLM-integrated systems in HCI: challenges and considerations. In: Oliver N, Shamma DA, Candello H, César P, Lopes P, Bozzon A, Kosch T, Liao VQ, Ma X, Artizzu V, Draxler F, López G, Reinschluessel AV, Tong X, Dugas POT (eds) Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems, CHI 2026, Barcelona, Spain, April 13-17, 2026, ACM, pp 1546:1–1546:18, DOI 10.1145/3772318.3790439, URL https://doi.org/10.1145/3772318.3790439
105. Open Source Initiative (OSI) (2025) Open Source AI Definition 1.0. https://opensource.org/ai/open-source-ai-definition, accessed 2025-08-26
106. 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
107. OpenAI (2025) OpenAI API Reference. https://platform.openai.com/docs/api-reference, accessed 2025-01-13
108. OpenCode Contributors (2025) OpenCode: The open source AI coding agent. https://opencode.ai/, accessed 2026-02-27
109. OpenTelemetry Authors (2026) OpenTelemetry semantic conventions for generative AI. https://opentelemetry.io/docs/specs/semconv/gen-ai/, accessed 2026-05-07
110. Ornelas T, Araújo AA, Araújo J, Araújo M, Trinkenreich B, Kalinowski M (2025) Llm-assisted thematic analysis: Opportunities, limitations, and recommendations. CoRR abs/2511.14528, DOI 10.48550/ARXIV.2511.14528, URL https://doi.org/10.48550/arXiv.2511.14528, 2511.14528
111. Pangakis N, Wolken S, Fasching N (2023) Automated annotation with generative AI requires validation. CoRR abs/2306.00176, DOI 10.48550/ARXIV.2306.00176, URL https://doi.org/10.48550/arXiv.2306.00176, 2306.00176
112. Panickssery A, Bowman SR, Feng S (2024) LLM evaluators recognize and favor their own generations. In: Globersons A, Mackey L, Belgrave D, Fan A, Paquet U, Tomczak JM, Zhang C (eds) Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, URL http://papers.nips.cc/paper_files/paper/2024/hash/7f1f0218e45f5414c79c0679633e47bc-Abstract-Conference.html
113. Papineni K, Roukos S, Ward T, Zhu W (2002) BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, ACL, pp

311–318, DOI 10.3115/1073083.1073135, URL https://aclanthology.org/P02-1040/

114. Pezeshkpour P, Hruschka E (2024) Large language models sensitivity to the order of options in multiple-choice questions. In: Duh K, Gómez-Adorno H, Bethard S (eds) Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024, Association for Computational Linguistics, Findings of ACL, vol NAACL 2024, pp 2006–2017, DOI 10.18653/V1/2024.FINDINGS-NAACL.130, URL https://doi.org/10.18653/v1/2024.findings-naacl.130
115. Pineau J, Vincent-Lamarre P, Sinha K, Larivière V, Beygelzimer A, d'Alché-Buc F, Fox EB, Larochelle H (2021) Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). J Mach Learn Res 22:164:1–164:20, URL https://jmlr.org/papers/v22/20-303.html
116. Rabbi MF, Champa AI, Zibran MF, Islam MR (2024) AI writes, we analyze: The ChatGPT Python code saga. In: Spinellis D, Bacchelli A, Constantinou E (eds) 21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024, ACM, pp 177–181, DOI 10.1145/3643991.3645076
117. Ralph P, Tempero ED (2018) Construct validity in software engineering research and software metrics. In: Rainer A, MacDonell SG, Keung JW (eds) Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering, EASE2018, ACM, pp 13–23, DOI 10.1145/3210459.3210461
118. Ralph P, bin Ali N, Baltes S, Bianculli D, Diaz J, Dittrich Y, Ernst N, Felderer M, Feldt R, Filieri A, de França BBN, Furia CA, Gay G, Gold N, Graziotin D, He P, Hoda R, Juristo N, Kitchenham B, Lenarduzzi V, Martínez J, Melegati J, Mendez D, Menzies T, Molleri J, Pfahl D, Robbes R, Russo D, Saarimäki N, Sarro F, Taibi D, Siegmund J, Spinellis D, Staron M, Stol K, Storey MA, Taibi D, Tamburri D, Torchiano M, Treude C, Turhan B, Wang X, Vegas S (2021) Empirical standards for software engineering research. URL https://arxiv.org/abs/2010.03525, 2010.03525
119. Ralph P, Kuutila M, Arif H, Ayoola B (2024) Teaching software metrology: The science of measurement for software engineering. In: Méndez D, Avgeriou P, Kalinowski M, Ali NB (eds) Handbook on Teaching Empirical Software Engineering, Springer Nature Switzerland, pp 101–154, DOI 10.1007/978-3-031-71769-7_5, URL https://doi.org/10.1007/978-3-031-71769-7_5
120. Raschka S (2026) Components of a coding agent. https://magazine.sebastianraschka.com/p/components-of-a-coding-agent, accessed 2026-06-08
121. Rasheed Z, Waseem M, Ahmad A, Kemell K, Wang X, Nguyen-Duc A, Abrahamsson P (2024) Can large language models serve as data analysts? A multi-agent assisted approach for qualitative data analysis. CoRR abs/2402.01386, DOI 10.48550/ARXIV.2402.01386, URL https:

//doi.org/10.48550/arXiv.2402.01386, 2402.01386
122. Reiss MV (2023) Testing the reliability of ChatGPT for text annotation and classification: A cautionary remark. CoRR abs/2304.11085, DOI 10.48550/ARXIV.2304.11085, URL https://doi.org/10.48550/arXiv.2304.11085, 2304.11085
123. Ren S, Guo D, Lu S, Zhou L, Liu S, Tang D, Sundaresan N, Zhou M, Blanco A, Ma S (2020) CodeBLEU: a method for automatic evaluation of code synthesis. CoRR abs/2009.10297, URL https://arxiv.org/abs/2009.10297, 2009.10297
124. Renze M (2024) The effect of sampling temperature on problem solving in large language models. In: Findings of the Association for Computational Linguistics: EMNLP 2024, pp 7346–7356, DOI 10.18653/v1/2024.findings-emnlp.432
125. Richards J, Wessel M (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, pp 666–671, DOI 10.1109/ICSME58944.2024.00070
126. Robillard MP, Arya DM, Ernst NA, Guo JLC, Lamothe M, Nassif M, Novielli N, Serebrenik A, Steinmacher I, Stol K (2024) Communicating study design trade-offs in software engineering. ACM Trans Softw Eng Methodol 33(5):112:1–112:10, DOI 10.1145/3649598, URL https://doi.org/10.1145/3649598
127. Ronanki K, Berger C, Horkoff J (2023) Investigating ChatGPT's potential to assist in requirements elicitation processes. In: 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, pp 354–361
128. Rozière B, Lachaux M, Chanussot L, Lample G (2020) Unsupervised translation of programming languages. In: Larochelle H, Ranzato M, Hadsell R, Balcan M, Lin H (eds) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, URL https://proceedings.neurips.cc/paper/2020/hash/ed23fbf18c2cd35f8c7f8de44f85c08d-Abstract.html
129. Rozière B, Gehring J, Gloeckle F, Sootla S, Gat I, Tan XE, Adi Y, Liu J, Remez T, Rapin J, Kozhevnikov A, Evtimov I, Bitton J, Bhatt M, Canton-Ferrer C, Grattafiori A, Xiong W, Défossez A, Copet J, Azhar F, Touvron H, Martin L, Usunier N, Scialom T, Synnaeve G (2023) Code Llama: Open foundation models for code. CoRR abs/2308.12950, DOI 10.48550/ARXIV.2308.12950, URL https://doi.org/10.48550/arXiv.2308.12950, 2308.12950
130. Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14(2):131–164, DOI 10.1007/S10664-008-9102-8, URL https://doi.org/10.1007/s10664-008-9102-8
131. Russo D (2024) Navigating the complexity of generative AI adoption in software engineering. ACM Trans Softw Eng Methodol

33(5):135:1–135:50, DOI 10.1145/3652154, URL https://doi.org/10.1145/3652154

132. Sallou J, Durieux T, Panichella A (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, NIER@ICSE 2024, Lisbon, Portugal, April 14-20, 2024, ACM, pp 102–106, DOI 10.1145/3639476.3639764, URL https://doi.org/10.1145/3639476.3639764
133. Santos A, Vegas S, Oivo M, Juristo N (2021) A procedure and guidelines for analyzing groups of software engineering replications. IEEE Trans Software Eng 47(9):1742–1763, DOI 10.1109/TSE.2019.2935720
134. Schäfer M, Nadi S, Eghbali A, Tip F (2024) An empirical evaluation of using large language models for automated unit test generation. IEEE Trans Software Eng 50(1):85–105, DOI 10.1109/TSE.2023.3334955
135. Schilke O, Reimann M (2025) The transparency dilemma: How ai disclosure erodes trust. Organizational Behavior and Human Decision Processes 188:104405, DOI 10.1016/j.obhdp.2025.104405, URL https://www.sciencedirect.com/science/article/pii/S0749597825000172
136. Schneider K, Fotrousi F, Wohlrab R (2025) A reference model for empirically comparing llms with humans. In: 47th IEEE/ACM International Conference on Software Engineering: Software Engineering in Society, ICSE-SEIS 2025, Ottawa, ON, Canada, April 27 - May 3, 2025, IEEE, pp 130–134, DOI 10.1109/ICSE-SEIS66351.2025.00018, URL https://doi.org/10.1109/ICSE-SEIS66351.2025.00018
137. Schröder S, Morgenroth T, Kuhl U, Vaquet V, Paaßen B (2025) Large language models do not simulate human psychology. CoRR abs/2508.06950, DOI 10.48550/ARXIV.2508.06950, URL https://doi.org/10.48550/arXiv.2508.06950, 2508.06950
138. Schroeder K, Wood-Doughty Z (2024) Can you trust LLM judgments? reliability of LLM-as-a-judge. CoRR abs/2412.12509, DOI 10.48550/ARXIV.2412.12509, URL https://doi.org/10.48550/arXiv.2412.12509, 2412.12509
139. Schulhoff S, Ilie M, Balepur N, Kahadze K, Liu A, Si C, Li Y, Gupta A, Han H, Schulhoff S, Dulepet PS, Vidyadhara S, Ki D, Agrawal S, Pham C, Kroiz GC, Li F, Tao H, Srivastava A, Costa HD, Gupta S, Rogers ML, Goncearenco I, Sarli G, Galynker I, Peskoff D, Carpuat M, White J, Anadkat S, Hoyle AM, Resnik P (2024) The prompt report: A systematic survey of prompting techniques. CoRR abs/2406.06608, DOI 10.48550/ARXIV.2406.06608, URL https://doi.org/10.48550/arXiv.2406.06608, 2406.06608
140. Schulz KF, Altman DG, Moher D (2010) CONSORT 2010 statement: updated guidelines for reporting parallel group randomised trials. BMJ 340:c332, DOI 10.1136/bmj.c332
141. Schwartz R, Dodge J, Smith NA, Etzioni O (2020) Green AI. Communications of the ACM 63(12):54–63, DOI 10.1145/3381831

142. Sclar M, Choi Y, Tsvetkov Y, Suhr A (2024) Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. In: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, OpenReview.net, URL https://openreview.net/forum?id=RIu5lyNXjT
143. Shull F, Singer J, Sjøberg DIK (eds) (2008) Guide to Advanced Empirical Software Engineering. Springer, DOI 10.1007/978-1-84800-044-5
144. Silva A, Monperrus M (2025) Repairbench: Leaderboard of frontier models for program repair. In: IEEE/ACM International Workshop on Large Language Models for Code, LLM4Code@ICSE 2025, Ottawa, ON, Canada, May 3, 2025, IEEE, pp 9–16, DOI 10.1109/LLM4CODE66737.2025.00006, URL https://doi.org/10.1109/LLM4Code66737.2025.00006
145. Sjøberg DIK, Bergersen GR (2023) Construct validity in software engineering. IEEE Trans Software Eng 49(3):1374–1396, DOI 10.1109/TSE.2022.3176725
146. Song Y, Wang G, Li S, Lin BY (2025) The good, the bad, and the greedy: Evaluation of LLMs should not ignore non-determinism. In: Chiruzzo L, Ritter A, Wang L (eds) Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Association for Computational Linguistics, pp 4195–4206, DOI 10.18653/V1/2025.NAACL-LONG.211, URL https://doi.org/10.18653/v1/2025.naacl-long.211
147. Sonnekalb T, Gruner B, Brust C, Mäder P (2022) Generalizability of code clone detection on CodeBERT. In: 37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, ACM, pp 143:1–143:3, DOI 10.1145/3551349.3561165
148. Stack Overflow (2025) 2025 developer survey. https://survey.stackoverflow.co/2025/, 49,009 respondents from 166 countries, May–June 2025
149. Strubell E, Ganesh A, McCallum A (2019) Energy and policy considerations for deep learning in NLP. In: Korhonen A, Traum DR, Màrquez L (eds) Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, Association for Computational Linguistics, pp 3645–3650, DOI 10.18653/V1/P19-1355, URL https://doi.org/10.18653/v1/p19-1355
150. Sullivan GM, Sargeant J (2011) Qualities of qualitative research: part I. J Grad Med Educ 3(4):449–452
151. Sumers TR, Yao S, Narasimhan K, Griffiths TL (2024) Cognitive architectures for language agents. Trans Mach Learn Res 2024, URL https://openreview.net/forum?id=1i6ZCvflQJ
152. Takerngsaksiri W, Pasuksmit J, Thongtanunam P, Tantithamthavorn C, Zhang R, Jiang F, Li J, Cook E, Chen K, Wu M (2025) Human-in-

the-loop software development agents. In: 47th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, SEIP@ICSE 2025, Ottawa, ON, Canada, April 27 - May 3, 2025, IEEE, pp 342–352, DOI 10.1109/ICSE-SEIP66354.2025.00036, URL https://doi.org/10.1109/ICSE-SEIP66354.2025.00036

153. Tam TYC, Sivarajkumar S, Kapoor S, Stolyar AV, Polanska K, McCarthy KR, Osterhoudt H, Wu X, Visweswaran S, Fu S, Mathur P, Cacciamani GE, Sun C, Peng Y, Wang Y (2024) A framework for human evaluation of large language models in healthcare derived from literature review. npj Digit Medicine 7(1), DOI 10.1038/S41746-024-01258-7, URL https://doi.org/10.1038/s41746-024-01258-7
154. Verdecchia R, Engström E, Lago P, Runeson P, Song Q (2023) Threats to validity in software engineering research: A critical reflection. Inf Softw Technol 164:107329, DOI 10.1016/J.INFSOF.2023.107329, URL https://doi.org/10.1016/j.infsof.2023.107329
155. Wagner S, Barón MM, Falessi D, Baltes S (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, pp 24–27, DOI 10.1109/WSESE66602.2025.00011
156. Wang A, Morgenstern J, Dickerson JP (2025) Large language models that replace human participants can harmfully misportray and flatten identity groups. Nat Mac Intell 7(3):400–411, DOI 10.1038/S42256-025-00986-Z, URL https://doi.org/10.1038/s42256-025-00986-z
157. Wang C, Huang K, Zhang J, Feng Y, Zhang L, Liu Y, Peng X (2024) How and why llms use deprecated apis in code completion? an empirical study. CoRR abs/2406.09834, DOI 10.48550/ARXIV.2406.09834, URL https://doi.org/10.48550/arXiv.2406.09834, 2406.09834
158. Wang F, Arora C, Liu Y, Huang K, Tantithamthavorn C, Aleti A, Sambathkumar D, Lo D (2025) Multi-modal requirements data-based acceptance criteria generation using llms. In: 40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16-20, 2025, IEEE, pp 3334–3345, DOI 10.1109/ASE63991.2025.00275, URL https://doi.org/10.1109/ASE63991.2025.00275
159. Wang S, Liu Y, Xu Y, Zhu C, Zeng M (2021) Want to reduce labeling cost? GPT-3 can help. In: Moens M, Huang X, Specia L, Yih SW (eds) Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021, Association for Computational Linguistics, Findings of ACL, vol EMNLP 2021, pp 4195–4205, DOI 10.18653/V1/2021.FINDINGS-EMNLP.354, URL https://doi.org/10.18653/v1/2021.findings-emnlp.354
160. Wang X, Kim H, Rahman S, Mitra K, Miao Z (2024) Human-LLM collaborative annotation through effective verification of LLM labels. In: Mueller FF, Kyburz P, Williamson JR, Sas C, Wilson ML, Dugas POT, Shklovski I (eds) Proceedings of the CHI Conference on Hu-

man Factors in Computing Systems, CHI 2024, ACM, pp 303:1–303:21, DOI 10.1145/3613904.3641960
161. Widder DG, Whittaker M, West SM (2024) Why 'open'ai systems are actually closed, and why this matters. Nature 635(8040):827–833
162. Wiesinger J, Marlow P, Vuskovic V (2025) Agents. Google Whitepaper, https://www.kaggle.com/whitepaper-agents, accessed 2026-06-08
163. Willison S (2026) Opus system prompt. https://simonwillison.net/2026/Apr/18/opus-system-prompt/, accessed 2026-04-28
164. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2024) Experimentation in Software Engineering, Second Edition. Springer, DOI 10.1007/978-3-662-69306-3
165. Xia Y, Shao H, Deng X (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, DOI 10.1145/3665348.3665391
166. Xiao T, Treude C, Hata H, Matsumoto K (2024) DevGPT: Studying developer-ChatGPT conversations. In: Spinellis D, Bacchelli A, Constantinou E (eds) 21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024, ACM, pp 227–230, DOI 10.1145/3643991.3648400
167. Xu C, Guan S, Greene D, Kechadi MT (2024) Benchmark data contamination of large language models: A survey. CoRR abs/2406.04244, DOI 10.48550/ARXIV.2406.04244, URL https://doi.org/10.48550/arXiv.2406.04244, 2406.04244
168. Xu R, Sun Y, Ren M, Guo S, Pan R, Lin H, Sun L, Han X (2024) AI for social science and social science of AI: A survey. Inf Process Manag 61(2):103665, DOI 10.1016/J.IPM.2024.103665, URL https://doi.org/10.1016/j.ipm.2024.103665
169. Yan L, Hwang A, Wu Z, Head A (2024) Ivie: Lightweight anchored explanations of just-generated code. In: Mueller FF, Kyburz P, Williamson JR, Sas C, Wilson ML, Dugas POT, Shklovski I (eds) Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, ACM, pp 140:1–140:15, DOI 10.1145/3613904.3642239
170. Yang G, Zhou Y, Chen X, Zhang X, Han T, Chen T (2023) Exploitgen: Template-augmented exploit code generation based on CodeBERT. J Syst Softw 197:111577, DOI 10.1016/J.JSS.2022.111577, URL https://doi.org/10.1016/j.jss.2022.111577
171. Yang Y, Shin J, Choi B, Park M, Ju D, Lee C, Chun S, Kang D, Kim J, Yoon S (2025) From code foundation models to agents and applications: A comprehensive survey and practical guide to code intelligence. arXiv preprint arXiv:250211827
172. Yuan J, Li H, Ding X, Xie W, Li YJ, Zhao W, Wan K, Shi J, Hu X, Liu Z (2025) Understanding and mitigating numerical sources of nondeterminism in LLM inference. In: Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Sys-

tems 2025, NeurIPS 2025, URL https://openreview.net/forum?id=Q3qAsZAEZw

173. Zhang J, Yu S, Chong D, Sicilia A, Tomz MR, Manning CD, Shi W (2025) Verbalized sampling: How to mitigate mode collapse and unlock LLM diversity. CoRR abs/2510.01171, URL https://arxiv.org/abs/2510.01171, 2510.01171
174. Zhao C, Habule M, Zhang W (2025) Large language models (LLMs) as research subjects: Status, opportunities and challenges. New Ideas in Psychology 79:101167, DOI 10.1016/j.newideapsych.2025.101167, URL https://www.sciencedirect.com/science/article/pii/S0732118X25000236
175. Zhou R, Chen L, Yu K (2024) Is LLM a reliable reviewer? A comprehensive evaluation of LLM on automatic paper reviewing tasks. In: Calzolari N, Kan M, Hoste V, Lenci A, Sakti S, Xue N (eds) Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy, ELRA and ICCL, pp 9340–9351, URL https://aclanthology.org/2024.lrec-main.816
176. Zhu Y, Zhang P, ul Haq E, Hui P, Tyson G (2023) Can ChatGPT reproduce human-generated labels? A study of social computing tasks. CoRR abs/2304.10145, DOI 10.48550/ARXIV.2304.10145, URL https://doi.org/10.48550/arXiv.2304.10145, 2304.10145