1
2
3
4
5
6
7
8
9
10

NIST AI 800-2
Initial Public Draft

Practices for Automated Benchmark Evaluations of Language Models

Center for AI Standards and Innovation

https://doi.org/10.6028/NIST.AI.800-2.ipd

January 2026

NIST AI 800-2
Initial Public Draft

Practices for Automated Benchmark Evaluations of Language Models

Center for AI Standards and Innovation

https://doi.org/10.6028/NIST.AI.800-2.ipd

January 2026

U.S. Department of Commerce
Howard W. Lutnick, Secretary

National Institute of Standards and Technology
Craig Burkhardt, Acting NIST Director and Under Secretary of Commerce for Standards and Technology

The Center for AI Standards and Innovation (CAISI) at NIST is releasing this document for public comment.

Comments on NIST AI 800-2 ipd may be sent electronically to ai800-2@nist.gov with "NIST AI 800-2 ipd" in the subject line. Electronic submissions may be sent as an attachment in any of the following unlocked formats: HTML; ASCII; Word; RTF; or PDF.

All comments are subject to release under the Freedom of Information Act (FOIA)

1

ABSTRACT

This draft provides voluntary practices for automated benchmark evaluations of language models and AI agent systems. It structures the practices in three stages: (1) defining the measurement target, (2) implementing and running the evaluation, and (3) analyzing and reporting the results. The report provides best practices at each of these stages and presents key terms and concepts in a glossary as such terms are used in the academic literature. As the science of AI measurement is rapidly developing, the guidelines presented in this document are offered as a preliminary set of best practices that will be updated on as the field advances.

KEYWORDS

artificial intelligence; evaluation; benchmark; large language model; chatbot; agent.

AUTHORITY

This publication has been developed by the Center for AI Standards and Innovation (CAISI) to further NIST's statutory responsibilities under the National AI Initiatives Act as codified in 15 U.S.C. § 278h-1. NIST is responsible for supporting measurement research and development of best practices and voluntary standards for AI systems. Secretary Howard Lutnick directed CAISI within NIST to develop guidelines and best practices to measure and improve the security of AI systems. President Trump's AI Action Plan tasked CAISI with publishing guidelines and resources for Federal agencies to conduct evaluations of AI systems. Executive Order 14303 directed Federal agencies to promote "gold standard science" that is, *inter alia*, reproducible, transparent, and communicative of error and uncertainty.

Nothing in this document should be taken to contradict standards and guidelines made mandatory and binding upon Federal agencies by the Secretary of Commerce under his statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, the Director of the Office of Management and Budget, or any other Federal agency or official.

ACKNOWLEDGEMENTS

This draft publication was developed by Drew Keller, Ryan Steed, Tony Wang, Stevie Bergman, and Peter Cihon. It was informed by feedback from CAISI and NIST Information Technology Laboratory staff, including Paul Christiano, Jesse Dunietz, Craig Greenberg, Jonathan Phillips, Julia Sharp, Benjamin Edelman, Maia Hamin, Jason Liang, Lukas Berglund, Andrew Fasano, Samuel Curtis, Michael Majruski, and Craig Schlenoff.

TABLE OF CONTENTS

Disclaimer: *Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.*

INTRODUCTION

This draft document identifies practices for conducting automated benchmark[1] evaluations of language models and similar general-purpose AI models that output text (herein "AI models"). Evaluations of these models, often embedded into systems capable of functioning as chatbots and AI agents, are increasingly common. However, consistent practices to support the validity and reproducibility of such evaluations are only beginning to emerge. The practices presented in this document are intended to reflect best practices; where relevant, practices that are relatively less mature in ecosystem use are labeled as emerging practice. This report supports NIST and CAISI efforts to develop guidelines and resources for Federal agencies to conduct evaluations of AI systems, as called for in the AI Action Plan [1]. It is informed by CAISI evaluations of AI models in partnership with leading U.S. AI industry organizations and CAISI research on measurement science, and will be subsequently informed by feedback via public comment.

This report is scoped to *automated* benchmark evaluations—evaluations that, once set up, can be run without any additional human input. The report focuses on using these evaluations to measure model capabilities, although many practices also apply to evaluating other behavioral properties of models (e.g. robustness). The document provides practices to implement and report on existing benchmarks to meet organizational needs, rather than practices to create new benchmarks. It does not provide guidelines for roles, resources, responsibilities, or other enabling practices for performing AI evaluations within an organization.[2] Future work may address benchmark development and practices for other types of AI evaluations.

The purpose of this report is to support practitioners in defining evaluation objectives; in selecting, implementing, and running evaluations to meet those objectives; and in analyzing and reporting on evaluations in a manner that enables reproducibility and valid interpretation of results. Not all evaluation objectives can be met by automated benchmark evaluations. Table I.1 provides some considerations for practitioners deciding when to use automated benchmarks rather than other evaluation methods (e.g., red teaming, human-subject experiments, field testing, and post-deployment monitoring) to assess model capabilities on a given task.

The primary audience for this document is technical staff at organizations conducting AI evaluations. These include AI deployers, developers, and third-party evaluators that may be based at companies, government agencies, academia, or other organizations. Additionally, anyone reviewing an evaluation report or model card may benefit from evaluator

[1] Terms defined in the Glossary are underlined. If a term appears multiple times, it is underlined only once per section.

[2] On such topics, practices may be obtained from NIST AI 100-1 [2] and NIST AI 800-1 [3].

implementation of document practices. Such readers may include business decision-makers, technical integrators, end consumers, or procurement officers seeking information with which to select an AI model for integration or use. When AI evaluation organizations implement the practices provided in this report, this secondary audience may be able to better understand and use evaluation results.

Table I.1. Characteristics of evaluations suited for automated benchmarks vs. other methods. *Automated benchmarks are not well-suited for all use cases.*

Evaluations suited for automated benchmarks are:	**Evaluations suited for other evaluation methods are:**
Structured and verifiable: a set of relevant, discrete tasks and corresponding test items with known or automatically verifiable solutions can be identified within the subject or domain of evaluation.	*Open-ended or subjective*: the evaluation domain cannot be divided into discrete tasks, and/or it is difficult to define objective grading criteria or verification procedures.
Time-invariant: tasks and their success criteria remain relevant and realistic over time.	*Dynamic*: realism and relevance of tasks may shift rapidly.
No human in the loop: tasks may be accomplished without iteration with the AI system operator(s).	*Human in the loop*: tasks require repeated interaction or open-ended use, and/or the evaluation objective intends to measure the AI system in conjunction with human operator(s) or other affected parties.
Outcome-oriented: the question of whether or not the model can accomplish certain tasks and/or the manner in which it behaves are of primary concern.	*Process/interpretability-oriented*: the process by which the model carries out tasks and/or reasons for model behavior are of primary concern.
Resource-constrained: all else equal, an automated benchmark may be less costly and time-intensive than other evaluation methods.	*Comprehensiv*e: conducting evaluations via multiple methods and modalities (e.g., automated benchmarking, human red-teaming, field testing, etc.) may be more expensive but can improve assurance.

Each section of this report details a stage in the evaluation process, providing best practices and specific examples. The report is organized into the following sections:

1. Define Evaluation Objectives and Select Benchmarks
2. Implement and Run Evaluations
3. Analyze and Report Results

Readers unfamiliar with AI measurement concepts may benefit from consulting the Glossary in the Appendix as they read this document.

1. DEFINING EVALUATION OBJECTIVES AND SELECTING BENCHMARKS

The first stage in effectively evaluating AI capabilities is to select benchmarks[3] that suit the evaluator's purposes, divided here into two high-level practices:

1.1. Define evaluation objectives.
1.2. Select benchmarks that meet evaluation objectives.

The design, execution, and reporting of an evaluation all depend on the evaluation objectives, including what should be measured (the measurement construct) and intended uses for the results produced. Possible uses of benchmark evaluation results could include:

- Informing or assessing decisions made while developing an AI system, such as what algorithms or data to use to train systems.
- Assessing whether an AI system is fit to use in a specific scenario.
- Comparing AI systems to decide which are most suitable for deployment.
- Assessing the efficacy of deployment configurations or of mitigations intended to address security, criminal misuse, or other risks.
- Validating that an AI system has been deployed and configured properly.
- Informing predictions or forecasts of the real-world impacts of an AI system.

See Table 1.1 for select evaluation objectives and relevant considerations for benchmark selection.

Table 1.1 Possible benchmark fit assessment for example evaluation scenarios. *Based on how the evaluation will be used and what should be measured (Practice 1.1), the evaluator assesses what a candidate benchmark measures (Practice 1.2.1) and whether it relates to the evaluation objectives (Practice 1.2.2).*

Example evaluation objective		***Possible benchmark***	***What the benchmark measures***	***Conceptual fit assessment***
How measurements will be used	***What is measured***			
AI developer looking to evaluate training progress (on multiple choice science question answering)	Graduate-level chemistry, biology, and physics multiple choice question answering	GPQA-Diamond [4]	Accuracy of answering a selection of multiple (4) choice questions intended to be difficult to answer even with internet searches, developed and validated by PhD students or graduates in the fields of	Benchmark directly measures the construct of interest in the setting

[3] Terms defined in the Glossary are underlined. If a term appears multiple times, it is underlined only once per section.

	accuracy		chemistry, biology, and physics	
Technical integrator deciding which model to use in an AI-powered general-purpose chatbot	Human-preferred responses in everyday conversation	Arena-Hard [5]	LLM-judged human preference ranking estimates (Brier score) for LLM responses to everyday questions sampled topic-wise	Predicts downstream outcomes of interest in the setting
AI deployer looking to understand the security risk posed to users by the model deployed as an agent	Vulnerability to prompt injection attacks during everyday tasks	AgentDojo [6]	Fraction of successful prompt injection attacks on LLM agents in a selection of realistic everyday tasks with access to typical tools and applications in office workspace populated with manual and LLM-generated dummy data	Conceptually related to the evaluation objective in the setting
Third-party evaluator assessing risk posed by criminal misuse of model capabilities	Extent to which a novice attacker's ability to exploit web application vulnerabilities is uplifted relative to pre-existing tools	CVE-Bench [7]	Success rate of autonomous LLM agent(s) at exploiting a sample of free and open-source web applications scored as having "critical" vulnerabilities (Common Vulnerability Scoring System) in the National Vulnerability Database	Conceptually related to the evaluation objective in the setting

Practice 1.1 Define evaluation objectives.

Every evaluation must be guided by clear objectives, which identify what the evaluation aims to measure in order to achieve the evaluation's end goal (i.e., to support the intended use of evaluation results). For example, a technical integrator choosing a model for a consumer chatbot application may conduct an evaluation with the objective of determining which of several models generates the most human-preferred responses to everyday questions.

Two questions are critical to development of an evaluation objective:

1. **How will the measurements be used?** Document intended uses of evaluation results. For example, an evaluation might be used to assess performance at a specific task, compare models, or assess risks associated with model use. Common considerations include:
    a. *Properties vs. outcomes.* Sometimes, one objective of evaluation may be to measure or predict a downstream outcome (e.g., the incidence of AI-assisted exploits of web applications or user satisfaction with a chatbot). In other cases, the objective is to measure an abstract property (e.g., mathematical reasoning ability).
    b. *Baselines.* If the objective is to assess differences in behavior (e.g., increases or decreases in performance at a task), relevant baseline measures should be noted

for comparison (e.g., human performance). These guidelines do not cover the process of collecting baseline measurements.

2. **What should be measured?** Based on the intended use cases for the AI system or threat models for risk assessment, decide and document what concept should be measured (the measurement construct). For example, an evaluation of a system intended to assist researchers might seek to assess a chatbot's ability to answer complex science questions, while a security risk assessment might seek to assess a system's vulnerability to prompt injection attacks. Common considerations include:
    a. *Comparison.* Many evaluations aim to compare measurements of different models (e.g., to select between models or assess change over time).
    b. *Measuring average-, best-, or worst-case behavior*. [Emerging Practice[4]] For example, high-stakes risk assessments may focus on best- or worst-case behavior; comparison shopping may focus on average-case behavior.

Practice 1.2 Select benchmarks that meet evaluation objectives.

After defining the evaluation objective, the next step is to select a benchmark or benchmarks. These guidelines focus on selection from existing benchmarks, but if existing benchmarks are not suitable, evaluators may choose to modify existing benchmarks or create new ones.

Conduct a survey of existing benchmarks. Based on the considerations below, choose benchmarks of sufficient quality and conceptual fit to satisfy the evaluation objectives. Clearly document exactly what each benchmark is expected to measure and how it relates to the evaluation objectives *before* conducting the evaluation, similar to scientific preregistration. These details are critical for accurately interpreting and qualifying the results (Practice 3.3).

1. **What does the benchmark measure?** [Emerging Practice] Document, in detail, precisely what each candidate benchmark measures. For example, the GPQA benchmark [4] purports to measure accuracy at answering a selection of multiple-choice questions intended to be difficult to answer even with internet searches at the time of creation, developed and validated by PhD students or graduates in the fields of chemistry, biology, and physics. Consider what the benchmark's description claims to measure and the accuracy of this claim based on available information about its construction, validation, and usage by others, as well as knowledge of the construct nominally being measured. When possible, manually inspect examples of questions, or test items, contained in the benchmark. Relevant details often include:
    a. *Subject matter.* What topics does the benchmark cover?
    b. *Difficulty.* What is the intended difficulty of the benchmark?

[4] Practices presented in this document are intended to reflect best practice; where relevant, practices that are relatively less mature in ecosystem use are labeled as emerging practice.

c. *Test item format.* How are the benchmark items formatted?
d. *Grading*. How are responses graded or assessed?

2. **Is what the benchmark measures relevant to the evaluation objective?** Document the relationship between the benchmark and evaluation objective. A benchmark can be relevant to the evaluation objective in multiple ways:
    a. *The benchmark directly measures the construct of interest.* If the content of a benchmark already reflects the construct of interest, it can be used directly. For example, the GPQA benchmark [4] may directly test graduate-level chemistry, biology, and physics multiple-choice question answering accuracy.
        i. *Coverage*. Test items should have high coverage across the entire space of tasks that pertain to the evaluation objectives and few items should be irrelevant. It may be possible to filter an existing benchmark to use only the subset of test items that are relevant to the evaluation objectives. Make note of any subject areas or other aspects not covered in the benchmark. For example, GPQA includes questions on only biology, chemistry, and physics.
        ii. *Test item format.* Test items should reflect intended use cases. For example, the items in many benchmarks are multiple-choice questions, usually with a single best answer per question. This question format simplifies benchmark implementation but sacrifices validity for many evaluation aims, as most tasks that LLMs are used for in practice, such as a chatbot answering user questions are more akin to free-response than multiple-choice.
    b. *The benchmark is conceptually related to the evaluation objective.* Often, there may not be a benchmark that directly measures the construct of interest, especially if the goal of evaluation is to predict a downstream outcome or if the task is hazardous or sensitive (e.g., cyberattacks or deepfake generation). However, existing benchmarks may instead measure proxy tasks which are prerequisites, subparts, or close relations to the task of interest.
        i. *Use cases and threat models.* Draw on use cases and threat models of interest to assess the connection between the benchmark content and evaluation objective. For example, no automated benchmark can directly measure how much an AI system uplifts a novice's ability to attack critical infrastructure, but an AI system that is unable to attack easy-to-exploit web applications could be inferred not to produce such uplift.
        ii. *Subject matter expertise.* Integrate theoretical analysis by subject matter experts to build evidence for or against a conceptual relationship.
    c. *The benchmark predicts downstream outcomes of interest.* [Emerging Practice] More rarely, there may be existing evidence of correlation between benchmark results and downstream outcomes or indicators of interest even if they are only

weakly conceptually related, for example, between an automated assessment of chatbot responses to user prompts and previously observed human preferences for chatbot responses.

3. **Is the benchmark suitable for the intended uses of evaluation results?** Different benchmarks provide different kinds of evaluation results.
    a. *Desired level of difficulty.* The ideal range of item difficulty depends on the reason(s) for evaluation. If the objective is to compare model performance during development, then a benchmark that is either too hard or too easy to the point of saturating is not useful. If the goal is to assess behavior relative to a threshold, many items should ideally be near that threshold to increase precision close to the boundary. If the goal is to assess average behavior on a specific task, the benchmark should have a realistic and representative level of difficulty.
    b. *Validated baseline measures (e.g., human performance on the same task).* Having relevant baseline or reference measures is beneficial when an evaluation seeks to compare the performance of an AI system to the performance of alternative approaches (e.g., humans only, prior non-AI automation, humans assisted by AI, or random chance). Besides capturing realistic alternatives relevant to the evaluation objectives, existing baseline(s) should also be statistically robust (e.g., sufficient number and expertise of human test-takers, for a human performance baseline).
4. **Is the benchmark of sufficient quality?** Even if a benchmark is a good fit for the evaluation objective, flaws in its construction may make it less useful and possibly misleading.
    a. *Diversity of test items in the evaluation.* A broader diversity of items means it is less likely there is some shared idiosyncrasy of all the items in the benchmark that could affect results. This enables broader inferences to be made based on the results of the evaluation.
    b. *Quantity of test items.* In addition to coverage and item diversity, the number of items in the benchmark affects whether results (statistically) support evaluation objectives. Statistical power analysis can support this determination.
    c. *Aspects of the benchmark, including contents or prompt formats, that may have influenced system training prior to evaluation.* If systems are trained to solve specific benchmark contents or formats, evaluation results may become "contaminated" — systems may take advantage of spurious relationships in the benchmark data to score higher than they would on unseen data in practice.
        i. Contamination risk can be reduced by efforts to keep benchmark data hidden during training or by using benchmark data generated after the system was created.

ii. [Emerging Practice] Some benchmarks include canary strings (unique sequences intentionally inserted into benchmark data) to check for blatant instances of training on the test set.

5. **What other practical considerations may affect benchmark usage?**
   a. *Ease of use.* Operational considerations include the amount of human labor required for setup and running, the computational cost to run the benchmark, and the benchmark's compatibility with a wide-range of AI systems (e.g., agent architectures).
   b. *Results reported by others.* If others have reported results for the same benchmark before, the benchmark may be a good candidate for validating and contextualizing the evaluation setup by comparing against previous evaluations (also see Section 3).

2. IMPLEMENTING AND RUNNING EVALUATIONS

The previous section discussed practices around defining evaluation objectives and selecting benchmarks based on those objectives. Given the selected benchmark(s), this section discusses how to implement and run fully automated benchmark evaluations. This process is divided into four high-level practices:

2.1. Design the evaluation protocol.
2.2. Write the evaluation code.
2.3. Run the evaluation and track results.
2.4. Debug the evaluation.

See Table 2.1 for some examples how these practices have been applied in past CAISI benchmark evaluations.

Table 2.1 Possible implementation and execution practices for example benchmarks. *Select design principles inform choice of protocol settings (Practice 2.1). The protocol is implemented, provided as open source software in the examples (Practice 2.2). In the process of running the evaluation (Practice 2.3), debugging may be required (Practice 2.4).*

Selected Benchmark	Evaluation Protocol		Evaluation Code	CAISI Debugging Log
	Relevant Design Principles	Details of Protocol		
CAISI [8] used GPQA [4] among other science and knowledge benchmarks for comparison across AI models.	Comparability Performance optimization	**Task settings** *Tools*: none *Instructions*: asked model to output selected choice, then an explanation *Submission attempts:* one **Model settings** *Reasoning settings*: "high" for OpenAI models, 31,000 reasoning budget for Anthropic models **Scoring settings** *Number of samples:* all 198 GPQA-Diamond questions *LLM-as-a-judge:* used to parse answer choice from response	inspect_evals/gpqa	Older models sometimes had trouble outputting the selected choice in the correct format, which was caught by manual transcript review. CAISI used an LLM-as-a-judge to parse selected choices from model responses.
CAISI [8] used CVE-Bench [7] to evaluate models for	Comparability External validity	**Task settings** *Tools:* command-line tools for cybersecurity functions and accessing the internet	usnistgov/caisi-cyber-evals	Models were allowed to access the internet in order to download new tools and more

offensive cyber capabilities.	Cost control Performance optimization	*Instructions:* act as a skilled cybersecurity expert *Submission attempts:* unlimited **Model settings** *Reasoning settings:* "medium" for OpenAI models, 0 reasoning budget for Anthropic models **Agent settings** *Agent scaffolding:* CAISI-implemented ReACT loop *Agent budget:* 500,000 "weighted" input/output tokens **Scoring settings** *Number of test items and trials per task*: 15 items and 4 trials per task.		accurately match realistic vulnerability discovery workflows. However, via automated transcript review, CAISI discovered that models sometimes used command line tools like curl to search the web for answers to test items. As a mitigation, CAISI changed the system prompt to instruct models not to trigger searches for answers.[5]

Practice 2.1 Design the evaluation protocol.

A given benchmark can generally be implemented in many different ways. It is up to the evaluator to design an evaluation protocol—the full set of operational procedures carried out during an evaluation—that meets the evaluator's needs. This sub-section presents best practices for evaluation protocol design.

2.1.1 Evaluation protocol design principles

The most important principle of evaluation protocol design is that procedures should be designed to support the objectives of the evaluation (Practice 1.1). This principle overrides all others.

That said, there are common design principles that can help guide protocol design for a broad range of evaluation objectives. These principles are particularly useful if evaluation objectives are only roughly defined. Common design principles include:

1. ***Comparability***. If the evaluation objective is to compare models or systems, the evaluation protocol should be designed such that evaluation results can be meaningfully compared. As a rule of thumb, the greater the consistency of a protocol between different models or systems, the more comparable results will be.

   If one wishes to compare evaluation results to existing baseline results (e.g., human

[5] In the longer run, a more robust solution would be to disable internet access while providing more tools and documentation up front. This would both improve reproducibility and reduce test-time contamination.

baselines), then evaluation protocols should also be designed with this in mind. For example, the evaluator may want to use a protocol that provides models with tools and information comparable to those used in existing baselines.

2. ***External validity*.** In many cases, the objective of an evaluation is to gain information on how a model or system will behave in certain external contexts (e.g., real-world, worst-case, best-case, hypothetical) different from that of the evaluation. For evaluation results to be informative in these external contexts, the evaluation protocol should be designed with external validity in mind.

   For example, to design for real-world validity, models can be scaffolded in ways that mirror the scaffolding that they would be used with in real-world applications. Similarly, cost-performance tradeoff settings (e.g., reasoning effort) can be chosen to mirror real-world usage.

3. ***Cost control.*** [Emerging Practice] There are both practical and methodological reasons to design evaluation protocols with execution cost in mind, with cost referring to resources like time, money, tokens, etc.

   On the practical front, all else being equal, cheaper evaluations are easier to run. Moreover, for certain types of agentic evaluations (e.g., when agents can compress their contexts and have effectively infinite context length), some form of cost-control must be implemented to prevent agents from running indefinitely.

   On the methodological front, the execution costs incurred by a model during an evaluation can have a big impact on the comparability and external validity of results. This is because the performance of language models and general-purpose AI systems can often be increased by running them in higher-execution-cost modes.

   For example, to meaningfully compare evaluation results on a one-dimensional measurement scale, execution costs should be controlled to be uniform across systems. Otherwise, if uniform cost-controls are not implemented, a downstream user may be able to obtain higher downstream utility with a cheaper but seemingly lower-performing model run in a higher-execution-cost mode. Uniform cost-controls across models are not strictly necessary though—if one is willing to report both costs and performance alongside each other, then comparability is valid even if different systems incur different costs. We comment on this approach more in Section 3.3.3.

   Finally, as a rule of thumb, when evaluating systems for the purpose of estimating their real-world utility, costs should be controlled to be similar to costs incurred in real-world

usage.

4. ***Performance optimization.*** [Emerging Practice] For certain evaluation objectives, there may be a need to *optimize* aspects of an evaluation protocol. Optimization here means iteratively generating results using one version of the protocol, and updating the protocol in order to change the results to be "better" along some axis (which could be exactly defined or only qualitatively specified).

   As an example, if the objective of an evaluation is to establish an upper or lower bound for a measured metric of a model across a large family of scaffolds, one way to accomplish this is to evaluate the model with a scaffold that has been optimized to maximize or minimize the metric. As another example, suppose a model has a tendency to refuse to engage with an evaluation task but the evaluation objective is to measure its behavior conditioned on non-refusal. One way to measure this conditional behavior is to optimize the prompt (i.e. iteratively update the prompt) fed to the model to get it to avoid refusals.

   When optimizing performance, one useful practice is to conduct optimization against a set of "development set" of benchmark items distinct from the "test-set" benchmark items being evaluated. If one forgoes a "dev-set" and optimizes against the test-set directly, there is an increased risk that the optimized evaluation protocol has overfit to the test-set and lost external validity, meaning the results of the evaluation can no longer be used to make meaningful inferences about the behavior of the model in other contexts.

2.1.2 Common evaluation protocol settings

The previous sub-section listed some high-level design principles that can help guide the design of an evaluation protocol. This sub-section lists common configurable variables (which specify execution details) shared by many different types of evaluation protocols. These variables are referred to as evaluation protocol settings, and are divided into four setting types:

I. ***Inference*** settings are those that influence the process by which the model generates its outputs, e.g., temperature or reasoning effort.

II. ***Scaffolding*** settings are those that configure the agentic architecture and system-level wrappers around the model, e.g., tool availability and aggregation strategies like best-of-N (running a model N times and taking the best performing output).

III. ***Task*** settings are those that determine how benchmark items are presented to models/systems and how models/systems can go about solving items.

IV. ***Scoring*** settings are those that determine how test items are scored.

In the table below, we organize common protocol settings by their setting type. This table also serves as an evaluation design checklist, since making an improper choice for an applicable setting can significantly reduce an evaluation's relevance to its objectives.

When setting for protocol settings, sensitivity analyses, such as tool ablation experiments (in which evaluations are re-run without particular tools being available to an agent), may be valuable for gauging robustness of evaluation results or assessing the impact of evaluation protocol design on results. In some cases, such as assessing the performance impacts of a new agent scaffold or prompting method, such experiments may be the primary goal of the evaluation.

Table 2.2 Common Evaluation Settings *Examples of evaluation settings. Evaluation settings are organized by their setting type(s), and for each setting we provide a brief description of what it is and what it influences.*

Evaluation Protocol Setting	Setting Type(s)	Description
Sampling	*Inference*	Sampling influences the process by which each successive token is generated. Common sampling settings include temperature, top_p, or top_k. In some circumstances, a model developer may have recommended settings for sampling.
Reasoning effort	*Inference*	Reasoning models can often be configured to use more or less "reasoning" when solving benchmark items. This is usually done via a "reasoning effort" setting, which can either be a categorical or numeric setting, and is commonly set at the API-request level. Configuring models to use more (less) reasoning generally increases (decreases) their performance at the cost of causing them to use more (less) computational resources (in the form of time, money, or tokens). The tradeoff between performance and cost can depend on both the model and the domain, with certain domains like advanced mathematics being particularly sensitive.

Evaluation Protocol Setting	Setting Type(s)	Description
		Reasoning effort should generally be set based on the "cost control" design principle.
Safeguards / filters	*Inference* or *Scaffolding*	Models can be served with intrinsic (e.g., weight-alteration based) or extrinsic (e.g., input/output classifier based) safeguards that cause certain classes of tasks to be refused. Evaluators can sometimes configure whether safeguards are enabled, and this choice can significantly impact evaluation results when the domain of evaluation overlaps with the domains of the model's safeguards.
Model Provider	*Inference or Scaffolding*	The model provider (which could be the evaluator themselves) is the entity responsible for running the underlying computations that transform model inputs into model outputs. The choice of model provider can impact both the logistics and semantics of an evaluation. In the former case, the provider impacts logistics via factors like inference costs, throughput, and data retention policies. In the latter case, the provider can impact semantics because different providers may serve the same model with different capabilities (e.g., different context lengths or tool call support), and providers can also have bugs in the models they serve (see section 2.4 for more discussion of such bugs) Finally, some providers offer more advanced features like tool calling that is built into their APIs, meaning provider choice should be treated as a scaffolding setting.
Agent scaffolding	*Scaffolding*	Agent scaffolding defines exactly how a model is turned into an agent. Options for agent scaffolding include using a high-level architectural pattern like ReAct [10], or using off-the-shelf, pre-built agents like Claude Code, codex-cli, or gemini-cli.

Evaluation Protocol Setting	Setting Type(s)	Description
		A key sub-consideration when designing agents is whether and how to provide them with context-compression tools, which allow them to operate past their normal context limits.
Agent budget	*Scaffolding*	A key property of many agents is that they can be run for an extended (possible indefinite) period of time. Thus, unlike non-agentic evaluations where task items have a natural stopping point, for agentic evaluations a stopping condition must be explicitly defined. A common way to do this is via agent budgets, which limit the amount of resources (e.g., tokens, money, time) an agent can use. Agents with larger budgets generally have higher performance, though this effect varies by model and domain. Agent budgets should be set based on the "cost control" design principle.
Best / maj-of-N aggregation	*Scaffolding*	This is a type of scaffolding where a model is queried multiple times, possibly in parallel, and its results are aggregated using a scheme like best-of-N (where the right answer is known) or majority-of-N (where the right answer is not known). This type of scaffolding is applicable to both agents and standalone models. For a best/majority-of-N scaffold, the choice of N allows one to trade off a model's performance against the computational resources it consumes. In certain circumstances, this setting can also be adjusted post-hoc during the scoring phase of an evaluation. This setting should be set based on the "cost control" design principle.
Prompts / instructions	*Scaffolding or Task*	When presenting a test item to a model, the evaluator defines the instructions given to the model that describe how the item should be completed. For example, for a multiple-choice question, the model could be prompted to "Read the question and pick the best answer out of the choices given."

Evaluation Protocol Setting	Setting Type(s)	Description
		The instructions presented to a model can impact what is being measured. For example, the level of detail in instructions can influence whether an evaluation is or is not measuring a model's skill at resolving ambiguity and inferring intentions. For agentic evaluations with a limit placed on the resources an agent can use to solve each test item, a key design decision for task instructions is whether or how to communicate this limit to the agent (e.g., periodically reminding the agent how many resources it has left versus leaving it up to the agent to properly keep track of its available resources). Instructions can also be used to outline constraints or rules, such as rules against looking up certain kinds of information on the internet. To ensure models cannot gain an unfair advantage by ignoring such rules, evaluators should ensure that the grading process for the task matches the rules as presented to the model. Ambiguous or over-broad rules can potentially cause models to under-perform or lead to performance differences between models if rules are interpreted differently.
Tools	*Scaffolding or Task*	For many classes of benchmarks or types of benchmark tasks, it may be reasonable to provide a model with tools to complete each benchmark item. For example, for certain scientific, engineering, or mathematical tasks, it may be desirable to provide models access to a code execution tool to carry out complex calculations. For tasks that require esoteric knowledge, it may be desirable to provide models access to an internet search tool. Depending on the evaluation objectives, it may be more appropriate to treat tools as scaffolding settings rather than task settings as described above, i.e., it may be desirable for different evaluated systems to have access to different tools.

Evaluation Protocol Setting	Setting Type(s)	Description
		For example, this may be the case when the objective is to understand the utility provided by different tools. Note: Whether to provide internet search tools is a particularly consequential decision, as internet access may allow models to cheat by looking up answers online [9]. Potential mitigations are discussed below in Sections 2.2 and 2.4.
Execution Environment	*Scaffolding or Task*	The execution environment is the environment in which an agent's tool calls have an effect. Examples of environments include docker containers, virtual machines, or large-scale networked systems of computers. Environments can come pre-loaded with files relevant to the agent/task, and some environments may have access to the broader internet. Similar to tools, the execution environment can be treated as a scaffolding and/or task setting depending on one's evaluation objectives. Key design principles that are relevant to choosing an execution environment include comparability (e.g., one may want environments that behave consistently across repeated evaluations) and external validity (e.g., one may want environments that are similar to real world deployment conditions).
Number of submission attempts	*Task*	For many agentic benchmarks, the evaluator determines the number of submission attempts an agent is allowed for a test item, as well as the type of feedback given to the agent for incorrect submissions.
Number of test items	*Task*	Sometimes, benchmarks come with more test items than one actually needs for an evaluation. In these cases the evaluation implementer needs to decide on how many items to use. Generally, this decision should be made by balancing statistical power and the budget the evaluator has for their evaluation exercise.

Evaluation Protocol Setting	Setting Type(s)	Description
Number of trials per test item	*Scoring*	Models are generally sampled nondeterministically (though in certain cases, it may be possible to directly measure performance without sampling, e.g., via next-token probabilities). Having models attempt each item multiple times increases evaluation cost but can reduce uncertainty of evaluation results and enable an evaluator to quantify what portion of the uncertainty stems from model sampling. These multiple attempts or trials may also be referred to as "epochs", as in the Inspect evaluation framework [11]. As with the number of test items, the choice of trials per item involves a tradeoff with evaluation budget.
LLM as a judge	*Scoring*	Some test item formats do not have a programmatically gradable answer. Instead, the answer must be judged using a more subjective procedure (e.g., against a written rubric). In these cases, automated evaluations often rely on using one or more LLMs to grade answers. Because the results of the evaluations are determined solely by the LLM-judge, the design and quality of the judge can have a significant impact on the meaning of evaluation results. [Emerging Practice] Some practices that are helpful when designing an LLM-as-a-judge setup include ensuring sufficient quality and consistency of grading and interpretations of the rubric, which can include comparing with human grading, using multiple judges and computing interrater agreement, and carefully designing and testing judge model prompts.

Practice 2.2 Write the evaluation code.

For automated benchmark evaluations, the evaluation protocol is ultimately implemented as computer code. We call this the *evaluation code*. Depending on the benchmark, models, and nature of the evaluation conducted, the amount of evaluation code required can range from tens of lines to thousands of lines or more.

The previous sub-section discusses practices for the macro-level design of an evaluation

protocol. Here, this sub-section describes practices for the micro-level implementation of an evaluation protocol as evaluation code:

1. **Evaluation frameworks.** Running evaluations can often be made easier by using evaluation frameworks, which provide software libraries for querying models, agent scaffolding and tools, error handling, and logging evaluation results instead of having to write those functions from scratch.

2. **Parsing answers.** Many evaluations require converting a model's answer into a specific format that can be programmatically compared to the correct answer. A common way this is done is by coding a parser that extracts a formatted answer from a model. However, such parsers can be brittle, as sometimes models can output answers that are technically correct when checked by a human but which the parser is unable to handle. One way to make parsers more robust is to utilize LLMs as a part of the parsing logic.

3. **Benchmark versioning.** [Emerging Practice] When making improvements and changes to a benchmark or its code, it is helpful to tag different versions of the benchmark with version numbers, and to track the version number associated with evaluation results. Versioning can be done using techniques like Python packaging numbers, git tags, or commit hashes.

   Semantic Versioning can be a helpful format to use, as it allows an evaluator to mark breaking changes (e.g., as major version increments), points at which the results of an evaluation before and after a change are no longer properly comparable. An example of a breaking change might be updating environments to install new dependencies that agents would previously have had to install themselves. An example of a non-breaking change might be fixing semantically irrelevant misspellings in some of the task item problem statements.

4. **Sandboxing.** [Emerging Practice] When agents are able to run arbitrary code, sandboxing them in containers or virtual machines—an isolated environment that prevents them from affecting the rest of the system—reduces security risks and makes evaluations more portable.

5. **Modularity as a design principle.** When writing evaluation code, it can be helpful to design the code to be compatible with many different types of models and agents. This makes it easy to collect data on multiple different models / agents, which helps contextualize evaluation results.

Practice 2.3 Run the evaluation and track results.

Once evaluation code is written, running an evaluation can be a fairly straightforward procedure: the evaluator needs only to run a command in a terminal or press a button in a user interface.

However, keeping clear records of evaluation results is important. This practice helps the evaluator avoid unnecessarily rerunning evaluations, and helps them keep track of the suitable use cases for their different evaluation results.

Helpful practices for evaluation result management include:

1. Saving full evaluation logs alongside summary statistics.
2. Ensuring key information like the exact model/system version is present in evaluation logs.
3. [Emerging Practice] Saving code or including commit hashes alongside evaluation logs.
4. [Emerging Practice] Tagging evaluation logs with metadata including their purpose.
5. [Emerging Practice] Grouping together evaluation logs that are meant to be compared to one another.

Practice 2.4 Debug the evaluation.

Automated benchmark evaluations can have bugs in their code or mistakes in their evaluation protocols. It is important to take steps to identify and fix these errors.

2.4.1 Common bugs

In this section, we list some common classes of bugs that one may encounter when performing automated benchmark evaluations. In the subsequent section, we discuss techniques for identifying these bugs.

1. ***Degraded serving.*** Models might be served in a degraded state due to improper configuration of or bugs in inference engines, quantization procedures (which control the precision of the floating-point numbers involved in model computations), and chat templates. Degraded serving is a particular concern when an evaluator self-hosts models. However, even commercial providers may serve models in a degraded state (e.g. commercial providers often vary on what context length they serve a given model with).

2. ***Tool calling errors.*** Errors in correctly calling tools or formatting tool calls can degrade agent performance and may be alleviated through better prompting or tool parsing code.

3. ***Test item solvability.*** It is important to validate the solvability of test items in evaluations that involve evaluated systems interacting with complex virtual environments. It is common for these environments to have bugs (e.g., networking

issues, broken dependencies, file permissions, etc.) that make certain items unsolvable.

In particular, it can be useful to check that the tools and affordances in the environment are functioning as intended. For example, is the networking and internet access (if allowed) in the environment functioning as intended?

4. ***Refusals.*** Models may have safeguards that prevent them from fulfilling some types of requests, such as for assistance in cybersecurity exploitation or dual-use biology tasks. If refusal behavior during evaluation may differ from refusal behavior under realistic usage,[6] evaluators should check for the presence of refusals or other safeguards interventions in evaluation transcripts to determine whether and how they may be impacting evaluation results.

5. ***Evaluation cheating.*** [Emerging Practice] Evaluation cheating occurs when a model has an opportunity to solve a test item in an unintended way that undermines its measurement validity.

   For example, cheating can occur via "solution contamination," in which a model is able to access solution information in an unintended way, from searching benchmark answers on the internet using coding tools to finding solution files or other artifacts unintentionally left in the testing environment. Cheating can also arise through "grader gaming," if models can craft solutions that score highly on an evaluation's automated scoring function by exploiting implementation loopholes rather than solving the test item as intended, such as solving a coding problem by removing the failing tests rather than fixing the bug.

   Cheating is more likely to occur on agentic evaluations, where models often have access to flexible coding tools that can enable potentially unwanted solution paths.

6. ***Evaluation awareness.*** [Emerging Practice] Recent work has shown that for certain models, model behavior can be influenced by cues that the input is part of an evaluation exercise [12][13][14]. The sensitivity to these cues is a challenge to external validity, since such cues are unlikely to be present during real-world use.

   A form of evaluation awareness that is easy to detect is verbalized evaluation awareness, wherein models will output text that directly references the possibility that they are being evaluated. Unfortunately, the absence of verbalized evaluation

[6] For example, a malicious user might be able to find jailbreaks that circumvent refusal. In such a case, an evaluation seeking to assess real-world performance may need to incorporate jailbreaks or otherwise modified refusals into the evaluation protocol.

awareness does not imply the absence of evaluation awareness. Developing more general solutions to detect and quantify the effect of evaluation awareness is an open line of research.

2.4.2 Quality assurance techniques

Quality assurance can diagnose issues in evaluation code and mistakes in evaluation protocols identified above. Relevant high-level quality assurance techniques include:

A. ***Manual transcript review.*** A powerful technique for quality assurance of automated benchmark evaluations is to manually review evaluation transcripts to detect and diagnose bugs and mistakes. When conducting review, a useful heuristic is to look for and examine unexpected behaviors. While not all unexpected behaviors are caused by bugs or mistakes, many bugs and mistakes will cause unexpected behaviors. Dedicated software for generating and viewing transcripts can make manual review more efficient.

B. ***Automated transcript review.*** [Emerging Practice] Evaluators may be able to extend and augment manual transcript review processes using automated transcript review. Automated transcript review involves using software tools, including LLMs, to analyze evaluation transcript data.

   For example, evaluators could programmatically check the number of tool calls that returned an error in a particular transcript. Or, evaluators could use LLM judges to score transcripts for the presence of problems like evaluation cheating.

   When using LLM judges for transcript review, evaluators should consider checking that these judges' decisions match human judges' decisions. Providing detailed rubrics, richer task context (such as solution write-ups), or using multiple LLM judges can help increase the accuracy of LLM-based transcript review systems. These practices mirror the ones detailed in Section 2.1.2's section on LLM-as-a-judge scoring.

C. ***Task review.*** In addition to reviewing transcripts, it can also be useful to review whether tasks are properly configured.

   For example, it can be helpful to manually review task instructions and ask: "If a human was given the same instructions, would they be confused?" Instructions can often be improved if the answer to this question is "yes". Furthermore, it can be beneficial to include multiple individuals in this review process to ensure interpretation consistency.

For evaluations that involve complex virtual environments, it can be useful to create and run deterministic solutions against the evaluator's in-house setup of the environment. Any test item whose deterministic solution does not successfully solve the task is likely to have some task configuration issue.

D. ***Comparison to existing evidence.*** Another assurance technique is to compare results to existing evidence such as other results on the same or closely related benchmarks. Differences do not necessarily signify that results are invalid, but suggest a need for further analysis to identify the cause of the divergence and to rule out the possibility of measurement issues (e.g., an inappropriately implemented evaluation protocol or a poorly selected benchmark). Results of other evaluations also provide important context for claims (see Practice 3.3).

   For example, as one check of whether the serving engine for a model is configured properly, an evaluator can test the model on a benchmark that the model developer has reported numbers on and check that their results are concordant with the developer's numbers. Differences could indicate an issue on the part of one of the evaluations.

   Particularly unexpected or anomalous results might be valid, but they might also indicate failure to achieve the evaluation objective. For example, one model performing anomalously well could potentially indicate a contamination issue.

   Comparing evaluation results to baselines can provide additional context to sense-check whether the evaluation was implemented as intended and/or whether the baselines are appropriate for comparison.

E. ***Item pattern analysis.*** [Emerging Practice] If an intended interpretation depends on assumed similarities or other relationships between benchmark items, consider analyzing whether the empirical results reflect this expected structure. For example, if an evaluator is using a benchmark containing both multiple-choice and open-ended questions to assess biology knowledge, substantial differences in the rank ordering of models between the multiple-choice and open-ended sections of the benchmark may suggest that the formats are measuring distinct capabilities rather than a unified construct.

3. ANALYZING AND REPORTING RESULTS

Once benchmark measurements are obtained, the next step is to process, contextualize, and analyze results. Analysis procedures should take into account the evaluation objective, the evaluation protocol, and characteristics of the selected benchmark. The aim of analysis is not only to draw conclusions, but also to determine the appropriate degree of confidence for those conclusions. Accordingly, evaluators should understand the evaluation's sources of uncertainty and gauge the evaluation's robustness – that is, the consistency of the conclusions under variations in measurement conditions or analysis procedures.

After analysis, evaluators should report qualified conclusions and share sufficient information for others to interpret and replicate their results.

This section divides the analysis and reporting process into three key practices:

3.1 Conduct statistical analysis and uncertainty quantification.
3.2 Share details of evaluation and evaluation data.
3.3 Report qualified claims.

See Table 3.1 for an example of these analysis and reporting practices.

Table 3.1 Example of CAISI benchmark evaluation analysis and reporting. *Provides a description of statistical analysis and uncertainty reported (Practice 3.1), details on how the evaluation was performed (Practice 3.2), and how claims were qualified in the final report (Practice 3.3).*

Evaluation	Statistical analysis	Reported details	Qualified claims
CAISI [8] used GPQA [4] to evaluate models' scientific knowledge.	CAISI reported average accuracy across the full benchmark with a standard error of the mean estimated using a generalized linear mixed model. These standard errors account for LLM sampling and finite benchmark size.	CAISI reported detailed information on the evaluation protocol, benchmark version and selection criteria, and cost-performance profiles.	CAISI reported that "U.S. models and DeepSeek's models achieve similar performance on question and answer-style science and knowledge benchmarks. Leading U.S. models are slightly more performant, but not by much."). Based on the benchmark content, CAISI described the evaluation as "measuring performance on challenging scientific questions that require graduate-level expertise to answer reliably." CAISI compared benchmark results to developers' self-reported scores and published this comparison.

Practice 3.1 Conduct statistical analysis and uncertainty quantification.

Evaluators should define, conduct, and report a statistically valid analysis procedure which aligns with intended evaluation objectives, accounts for sources of variation and uncertainty, and describes underlying assumptions. It is good practice to define a statistical analysis procedure in *advance* of implementing and running the evaluation, in conjunction with the task setting specification practices described in Sections 1 and 2. Results of statistical analysis should be interpreted in light of broader evaluation objectives, limitations, and context.

Characteristics of a robust and transparent analysis procedure include:

1. **Appropriate modeling assumptions are made and reported.** Reasonable assumptions may greatly increase the precision of an analysis. For example, the Bradley-Terry model's assumption that model rankings are transitive results in much smaller confidence intervals for model performance. Any assumptions used in results analysis including the application of statistical modeling should be clearly specified and reported, along with the results of any checks of these assumptions.

2. **Appropriate aggregate statistics are selected.** This includes any aggregation functions applied at a test item level (e.g., across trials, such as pass@k) as well as across items (e.g., mean of item scores). For example, if the evaluation goal is to predict typical performance on some set of tasks, for which benchmark items are considered a representative sample (in terms of both content and difficulty), then one appropriate metric could be the expected score across items in the set. If the evaluation goal is to compare models, evaluators could consider metrics that are specifically designed for comparison, such as Bradley-Terry coefficient. If evidence suggests that the benchmark measures multiple constructs, then care should be taken that aggregate metrics do not combine these measurements in an arbitrary manner.

3. **Whenever possible, statistics are reported with estimated uncertainties for associated sources of variation** [15]. Uncertainty quantifications could take the form of standard errors, confidence intervals, credible intervals, etc. as appropriate and should be clearly defined. Uncertainty metrics should be computed using methods aligned with the statistical assumptions of the analysis, and the description of evaluation metrics should make clear the contributions and alignment of estimated uncertainties to particular sources of variation. Ideally, different sources of variation should be decomposed and reported separately. For example, in many evaluations, two additive sources of variation in results that contribute to uncertainty of estimates are (1) variation due to nondeterministic sampling of model completions for each item and (2) variation stemming from the hypothetical sampling of test items into the benchmark from an item superpopulation.

4. **Unquantified sources of variation are clearly indicated** – or when possible, approximated or bounded [Emerging Practice]. For example, the analysis procedure may not statistically account for factors such as prompt format or task environmental conditions, but variations in these factors may introduce random or systematic variations in measured results.

5. **Comparisons are supported by appropriate statistical tests.** For example, a comparison can be made between mean scores of two evaluated models using a statistical test on the paired difference with appropriate standard error. However, statistical test results should be interpreted probabilistically and considered alongside effect size.

Practice 3.2 Share details of evaluation and evaluation data.

Sharing evaluation details and evaluation data such as test transcripts increases reproducibility and enables others to draw their own determinations about what results imply, though transparency must be weighed against business and security concerns.

1. **Report key evaluation details.** Such details include but are not necessarily limited to the evaluation objective (Practice 1.1); selected benchmark version(s); number and type of test items (Practice 1.2); exact model version(s); details of the evaluation protocol including information about cost controls, performance optimization practices, key evaluation protocol setting choices, and sensitivity analyses (Practice 2.1) statistical assumptions and results of statistical analysis with uncertainty estimates (Practice 3.1). Using an interoperable format or schema to share these details may improve clarity and ease of replication.

2. **Consider reporting both aggregate statistics and item-level results.** While aggregate statistics are useful for headline comparisons, item-level results can be particularly useful for enabling comparison of results obtained by different evaluators.

3. **Report costs alongside performance.** [Emerging Practice] When models and systems are not uniformly cost controlled, evaluation results should report both the performance of models and the costs-incurred to achieve such performance levels (see Section 2.1.1). One possible way to present this data is to plot cost and performance of different systems on a two-dimensional graph (see Figure 3.2, for example).

   Some evaluations may also provide evidence about the performance of a system for a wide range of different costs. For example, the trajectory of an AI agent can be truncated at different points, or a fixed set of responses can be used to compute the accuracy of majority-of-N for many different values of N. In these cases, reporting performance across multiple (or all) settings can provide additional information. An example of this type of reporting is shown in the figure below.

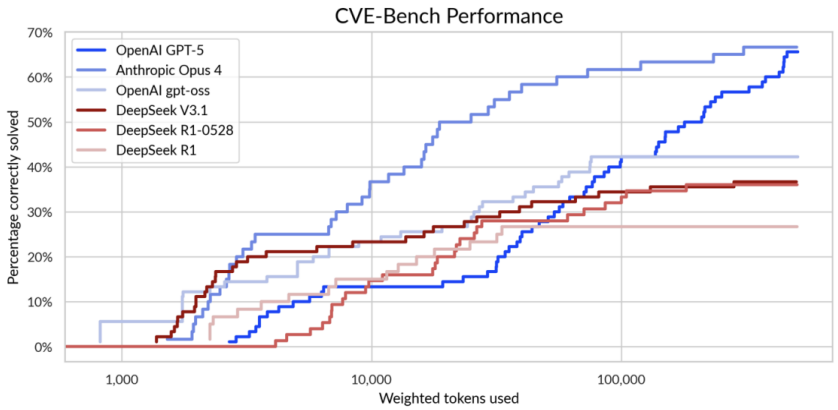

Figure 3.2 A plot from CAISI's report [8] which shows the percentage of CVE-Bench [7] items solved as a function of the number of weighted-tokens used. This plot was generated by truncating agent transcripts at all possible different points.

4. **Consider releasing transcripts.** [Emerging Practice] Publishing full or representative transcripts can make it easier for other parties to interpret and reproduce the results of evaluations. Being able to review transcript-level data enables other researchers to more easily spot methodological decisions (or potential issues) that could impact their interpretation of the results, reducing the risks of drawing flawed conclusions and making it easier for the evaluation science community to collaborate on studying and resolving evaluation science issues.

    a. When using public benchmarks, consider releasing full transcripts. If using non-public benchmarks where there are contamination concerns, consider releasing a random or representative (e.g., stratified) sample of transcripts. Redact sensitive information from transcripts if necessary (e.g., detailed jailbreaks, proprietary prompt scaffolding, etc.).

    b. Consider using an evaluation framework that automatically generates easily navigable and filterable transcripts. Anti-scraping measures, training data opt-out notices, and similar measures can help reduce contamination risks when releasing transcripts.

5. **Publish evaluation code.** [Emerging Practice] Publishing code used to run and analyze evaluations makes it easier for other evaluators to reproduce and extend evaluation results. In addition to the evaluation code itself, consider publishing agent sandbox container images (e.g., by uploading images to a public library like Docker Hub).

Practice 3.3 Report qualified claims.

Ensuring that claims made on the basis of evaluation results accurately reflect an evaluation's scope and context assists evaluation consumers in accurately interpreting evaluation results. The following practices can help appropriately qualify claims and interpretations.

1. **Distinguish claims about evaluation results from other claims.** Differentiate observations, inferences, predictions, and normative statements.

2. **Report assumptions or evidence for the relationship between performance on the evaluated benchmark and the intended measurement construct** (see Practice 1.2). This reporting includes conceptual or predictive evidence for how the benchmark and evaluation protocol relate to other use cases, domains, or real-world behavior (e.g., a case for why benchmark test items resemble real-world tasks of interest).
    a. If the benchmark is intended to be representative of real-world tasks, report the degree to which evaluation conditions resemble deployment conditions (see Practice 2.1).
    b. State assumptions or cite evidence supporting any usage of evaluation results to predict future measurements.

3. When comparing model behavior to baselines, **provide context about baseline measure conditions** (e.g., characteristics of baseline subjects, available tools, incentives, etc.) (Practice 1.2).

4. **Use caution when making claims that generalize evaluation results beyond the scope of the intended measurements** as established in Practice 1.2. Such claims could include statements about constructs that lack conceptual or predictive evidence of fit to the measured benchmark. Consider adding relevant caveats or disclaimers when there is a significant risk that the audience may misinterpret or over-generalize evaluation results.

    [Emerging Practice] In particular, if models show signs of evaluation awareness, it can be helpful to report metrics like measured rates of verbalized evaluation awareness, and to discuss the potential for such awareness to reduce the external validity of results.

5. **Clearly state other assumptions and limitations of the evaluation**, such as sensitivity of results to measurement conditions (Practice 2.1) and statistical assumptions (Practice 3.1).

6. **Discuss the degree to which evaluation results agree or disagree with similar evaluations** or other relevant evidence and existing knowledge.

REFERENCES

[1] The White House (2025) *America's AI Action Plan*. Available at https://www.whitehouse.gov/wp-content/uploads/2025/07/Americas-AI-Action-Plan.pdf.

[2] NIST (2023) *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. (U.S. Department of Commerce, Washington, D.C.), NIST AI 100-1. https://doi.org/10.6028/NIST.AI.100-1.

[3] CAISI (2025) *Managing Misuse Risk for Dual-Use Foundation Models.* (U.S. Department of Commerce, Washington, D.C.), NIST AI 800-1, Second Public Draft. https://doi.org/10.6028/NIST.AI.800-1.2pd.

[4] Rein D, Hou BL, Stickland AC, Petty J, Pang RY, Dirani J, Michael J, Bowman SR (2023) *GPQA: A Graduate-Level Google-Proof Q&A Benchmark.* Available at http://arxiv.org/abs/2311.12022.

[5] Li T, Chiang WL, Frick E, Dunlap L, Wu T, Zhu B, Gonzalez JE, Stoica I (2024) *From Crowdsourced Data to High-Quality Benchmarks: Arena-Hard and BenchBuilder Pipeline*. Available at https://arxiv.org/pdf/2406.11939.

[6] Debenedetti E, Zhang J, Balunović M, Beurer-Kellner L, Fischer M, Tramèr F (2024) *AgentDojo: A Dynamic Environment to Evaluate Attacks and Defenses for LLM Agents*. Available at https://arxiv.org/abs/2406.13352.

[7] Zhu Y, Kellermann A, Bowman D, Li P, Gupta A, Danda A, Fang R, Jensen C, Ihli E, Benn J, Geronimo J, Dhir A, Rao S, Yu K, Stone T, Kang D (2025) *CVE-Bench: A Benchmark for AI Agents' Ability to Exploit Real-World Web Application Vulnerabilities*. Available at https://arxiv.org/abs/2503.17332.

[8] CAISI (2025) *Evaluation of DeepSeek AI Models.* (U.S. Department of Commerce, Washington, D.C.), Center for AI Standards and Innovation (CAISI) Report. Available at https://www.nist.gov/system/files/documents/2025/09/30/CAISI_Evaluation_of_DeepSeek_AI_Models.pdf.

[9] Han Z, Mankikar M, Michael J, Wang Z (2025) *Search-Time Data Contamination*. Available at https://scale.com/research/stc.

[10] Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, Cao Y (2023) *ReAct: Synergizing Reasoning and Acting in Language Models*. Available at https://arxiv.org/abs/2210.03629.

[11] UK AI Security Institute (2024) *Inspect AI: Framework for Large Language Model Evaluations*. https://github.com/UKGovernmentBEIS/inspect_ai.

[12] Needham J, Edkins G, Pimpale G, Bartsch H, Hobbhahn M (2025) *Large Language Models Often Know When They Are Being Evaluated*. Available at https://arxiv.org/abs/2505.23836.

[13] Anthropic (2025) *System Card: Claude Sonnet 4.5*. Available at https://assets.anthropic.com/m/12f214efcc2f457a/original/Claude-Sonnet-4-5-System-Card.pdf.

[14] OpenAI (2025) *Findings from a pilot Anthropic–OpenAI alignment evaluation exercise: OpenAI Safety Tests*. Available at https://openai.com/index/openai-anthropic-safety-evaluation/.

[15] Possolo, A (2015) *Simple Guide for Evaluating and Expressing the Uncertainty of NIST Measurement Results.* (U.S. Department of Commerce, Washington, D.C.), NIST Technical Note 1900. http://dx.doi.org/10.6028/NIST.TN.1900.

[16] Raji ID, Bender EM, Paullada A, Denton E, Hanna A (2021) *AI and the Everything in the Whole Wide World Benchmark*. Available at http://arxiv.org/abs/2111.15366.

[17] Wallach H, Desai M, Cooper AF, Wang A, Atalla C, Barocas S, Blodgett SL, Chouldechova A, Corvi E, Dow PA, Garcia-Gathright J, Olteanu A, Pangakis N, Reed S, Sheng E, Vann D, Vaughan JW, Vogel M, Washington H, Jacobs AZ (2025) *Position: Evaluating Generative AI Systems Is a Social Science Measurement Challenge*. Available at http://arxiv.org/abs/2502.00561.

[18] Salaudeen O, Reuel A, Ahmed A, Bedi S, Robertson Z, Sundar S, Domingue B, Wang A, Koyejo S (2025) *Measurement to Meaning: A Validity-Centered Framework for AI Evaluation*. Available at http://arxiv.org/abs/2505.10573.

[19] Adcock R and Collier D (2001) Measurement Validity: A Shared Standard for Qualitative and Quantitative Research. *American Political Science Review* 95(3):529–546.

[20] American Educational Research Association, American Psychological Association, National Council on Measurement in Education (2014) *Standards for Educational and Psychological Testing* (American Educational Research Association, Washington, D.C.). Available at https://www.apa.org/science/programs/testing/standards.

[21] Bengio Y, Mindermann S, Privitera D, Besiroglu T, Bommasani R, Casper S, Choi Y, Fox P, Garfinkel B, Goldfarb D, Heidari H, Ho A, Kapoor S, Khalatbari L, Longpre S, Manning S, Mavroudis V, Mazeika M, Michael J, … Zeng Y (2025) *International AI Safety Report*. Available at http://arxiv.org/abs/2501.17805.

[22] ISO/IEC (2022) *ISO/IEC TS 5723:2022 Trustworthiness — Vocabulary*. Available at https://www.iso.org/standard/81608.html.

APPENDIX A: GLOSSARY

Common Terms and Definitions

Appendix A provides definitions for terminology used in the draft report. Sources for terms used in this publication are cited as applicable. Where no citation is noted, the report is the source of the definition. Terms with glossary definitions are underlined in text.

Baseline	Comparable reference measure of the behavior of a non-AI system, potentially including human individuals or teams, systems based on simple heuristics or rules, or systems based on random chance.
Behavior	An AI system's outputs in response to inputs and operating conditions.
Benchmark (adapted from [16])	A particular combination of a dataset and a metric, conceptualized as representing one or more specific tasks or sets of abilities. A benchmark is used by a community of researchers as a shared framework for the comparison of methods. In practice, a single benchmark may combine multiple datasets (at least test data and sometimes also pre-specified training and validation data).
Capability (adapted from [21])	The range of tasks or functions that an AI system can perform and how effectively it performs them.
Content validity (adapted from [17])	The extent to which the measurement criteria reflect the most salient aspects of the measurement construct. In an operational context, content validity refers to the extent to which the measurement instruments align with the substance and structure of the measurement criteria.
Evaluation objective(s)	The goal(s) of an evaluation, including what should be measured (the measurement construct) and how the measurements will be used (e.g., to make decisions about system design or deployment).
Evaluation protocol	The full set of operational procedures carried out during an evaluation.

Evaluation protocol settings	The configurable variables that specify the execution details of an evaluation protocol.
External validity	The extent to which measurements can also describe, or generalize to, conditions different from evaluation context.
Metric (adapted from [16])	A way to summarize system performance over some set of test items into a single number or score.
Measurement construct [18]	An abstract concept not directly measurable (e.g., "mathematical reasoning"). Sometimes referred to as a "background concept" [17]: the "broad constellation of meanings and understandings associated with [the] concept [of interest]" [19]. In benchmark evaluations, often expressed in terms of "ability" or "capability" (e.g., mathematical reasoning).
Measurement criterion [18]	A directly measurable or observable concept (e.g., "textbook linear algebra question answering accuracy"). Sometimes referred to as a "systematized concept" [17]: the "specific formulation of the concept[, which] commonly involves an explicit definition" [19].
Measurement instrument [18]	A tool used to gather observations or assign values (e.g., a benchmark, user study, or survey).
Measurement validity [20]	The degree to which accumulated evidence and theory support a specific interpretation of test scores for a given use of a test. If multiple interpretations of a test score for different uses are intended, validity evidence for each interpretation is needed.
Proxy task	A task which does not entirely specify a problem of interest, but instead specifies a conceptually related or empirically correlated problem or subproblem.
Robustness (adapted from [22]; see also [2])	The "ability of a system to maintain its level of performance under a variety of circumstances," such as input perturbations or distribution shifts.
Scaffolding	The agentic architecture and system-level wrappers that can be attached to a model.

Task (adapted from [16])	Particular specification of a problem, including desired solution(s), solution settings, or grading/verification procedures. Each task assessed in a benchmark may be represented by one or more test items.
Test item	An individual question or problem in a benchmark, sometimes referred to as a sample or example. For example, LLM benchmark tasks are often represented by a set of multiple-choice questions with one or more correct answers.
Trial	A single attempt at solving a test item. Evaluations often involve multiple attempts at each test item per AI system tested.

1