Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences

Shreya Shankar
UC Berkeley
Berkeley, California, USA
shreyashankar@berkeley.edu

J.D. Zamfirescu-Pereira
UC Berkeley
Berkeley, California, USA
zamfi@berkeley.edu

Björn Hartmann
UC Berkeley
Berkeley, California, USA
bjoern@eecs.berkeley.edu

Aditya G. Parameswaran
UC Berkeley
Berkeley, California, USA
adityagp@berkeley.edu

Ian Arawjo
Université de Montréal
Montréal, Québec, Canada
ian.arawjo@umontreal.ca

ABSTRACT

Due to the cumbersome nature of human evaluation and limitations of code-based evaluation, Large Language Models (LLMs) are increasingly being used to assist humans in evaluating LLM outputs. Yet LLM-generated evaluators simply inherit all the problems of the LLMs they evaluate, requiring further human validation. We present a mixed-initiative approach to "validate the validators"—aligning LLM-generated evaluation functions (be it prompts or code) with human requirements. Our interface, EvalGen, provides automated assistance to users in generating evaluation criteria and implementing assertions. While generating candidate implementations (Python functions, LLM grader prompts), EvalGen asks humans to grade a subset of LLM outputs; this feedback is used to select implementations that better align with user grades. A qualitative study finds overall support for EvalGen but underscores the subjectivity and iterative nature of alignment. In particular, we identify a phenomenon we dub *criteria drift*: users need criteria to grade outputs, but grading outputs helps users define criteria. What is more, some criteria appear *dependent* on the specific LLM outputs observed (rather than independent and definable *a priori*), raising serious questions for approaches that assume the independence of evaluation from observation of model outputs. We present our interface and implementation details, a comparison of our algorithm with a baseline approach, and implications for the design of future LLM evaluation assistants.

CCS CONCEPTS

• **Human-centered computing → Interactive systems and tools**; • **Computing methodologies → Natural language processing**.

KEYWORDS

language models, auditing, evaluation, interfaces, prompt engineering, active learning

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0628-8/24/10
https://doi.org/10.1145/3654777.3676450

ACM Reference Format:
Shreya Shankar, J.D. Zamfirescu-Pereira, Björn Hartmann, Aditya G. Parameswaran, and Ian Arawjo. 2024. Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3654777.3676450

1 INTRODUCTION

Large Language Models (LLMs) make mistakes—they hallucinate, ignore instructions, and generate invalid or uncalibrated outputs [26]. But validating the behavior of LLMs is challenging. In response, researchers and industry developers have created tools for prompt engineering and auditing that help people with testing outputs more systematically [1, 17, 24, 25, 28, 37, 44, 55]. Such approaches require *metrics*, i.e., functions that automatically score LLM outputs, each typically an assertion with *true* or *false* values. These metrics increasingly include calls to "evaluator" LLMs, e.g., [1, 28, 55, 61], that act as "judges," grading outputs on qualities hard to articulate in code; for instance, the "conciseness" of an output.

While LLM-based validators are commonly used in practice and can be effective, crafting these validators—both code-based and LLM-based—so that they align well with user preferences remains challenging. Finding the right prompt for LLM-based assertions is difficult, e.g., they are unintuitively sensitive to seemingly minor changes in wording or structure [46], as is crafting code-based assertions, such as choosing the appropriate regex. This process can be time-consuming and is not well-supported by current tools. How can users reap the efficiency benefits of LLM-assisted evaluation of LLM outputs, while ensuring alignment with their specific preferences? How can we help users craft and validate effective validators?

In this paper, we propose a mixed-initiative approach, EvalGen, to address this automated-evaluation alignment problem in the context of prompt engineering. Our approach streamlines the selection of metrics under practical constraints of user effort and latency. Specifically, an LLM suggests criteria in natural language, based on user context (e.g., the prompt under test), that the user can modify. An LLM then generates a pool of candidate assertions for each criterion—either code or LLM grader prompts that output "true" or "false." While the user waits for the LLM to generate candidates, they are asked to grade outputs with a simple "good" (thumbs-up) or "bad" (thumbs-down) voting scheme. These grades then guide the automatic selection of assertions that optimize for alignment with user preferences. After assertion selection, a final report card reveals the alignment between the chosen assertions and the user's grades. Our approach generalizes beyond the particulars of our specific design, and could be extended to, for instance, update metric implementations with feedback from human preferences, or query the user for finer-grained individual grades.

EvalGen is embedded inside an existing open-source interface for prompt engineering and auditing, ChainForge [1]. Our alignment algorithm adapts SPADE [48], a fully-automated algorithm for generating Python assertions from the revision history of a prompt. We performed an offline verification of our human-guided alignment algorithm versus SPADE, then ran a qualitative user study with nine (9) industry practitioners who use LLMs in production contexts. Since our participants were industry practitioners and thus possibly dealing with NDA-protected data, we offered a task adapted from a real LLM pipeline prompt. Our study design did not impose restrictions on how participants used EvalGen, and users could choose whether to ask the tool to suggest criteria, enter criteria manually, or grade a few LLM outputs first before proceeding to the criteria specification screen.

Our study finds overall support for EvalGen, with one important caveat. We observed a "catch-22" situation: to grade outputs, people need to externalize and define their evaluation criteria; however, the process of grading outputs helps them to define those very criteria. We dub this phenomenon *criteria drift*, implying that *it is impossible to completely determine evaluation criteria prior to human judging of LLM outputs*. Even when participants graded first, we observed that they still refined their criteria upon further grading, even going back to change previous grades. Thus, our findings suggest that users need evaluation assistants to support rapid iteration over criteria and implementations *simultaneously*. Since criteria are *dependent* upon LLM outputs (and not independent from them), this raises questions about how to contend with criteria drift in the context of other "drifts"—e.g., model drift [5], prompt edits, or upstream changes in a chain. Our findings also *(i)* underscore the necessity of *mixed-initiative* approaches to the alignment of LLM-assisted evaluations that also embrace messiness and iteration, and *(ii)* raise broader questions about what "alignment with user preferences" means for evaluation assistants.

We first position our work (Sec. 2) and present EvalGen's design (Sec. 3) and implementation details (Sec. 4). We then present two evaluations: an off-line evaluation of our approach (Sec. 5), and a qualitative study with developers (Sec. 6 & 7). Finally, we suggest implications for future work (Sec. 8).

2 MOTIVATION AND RELATED WORK

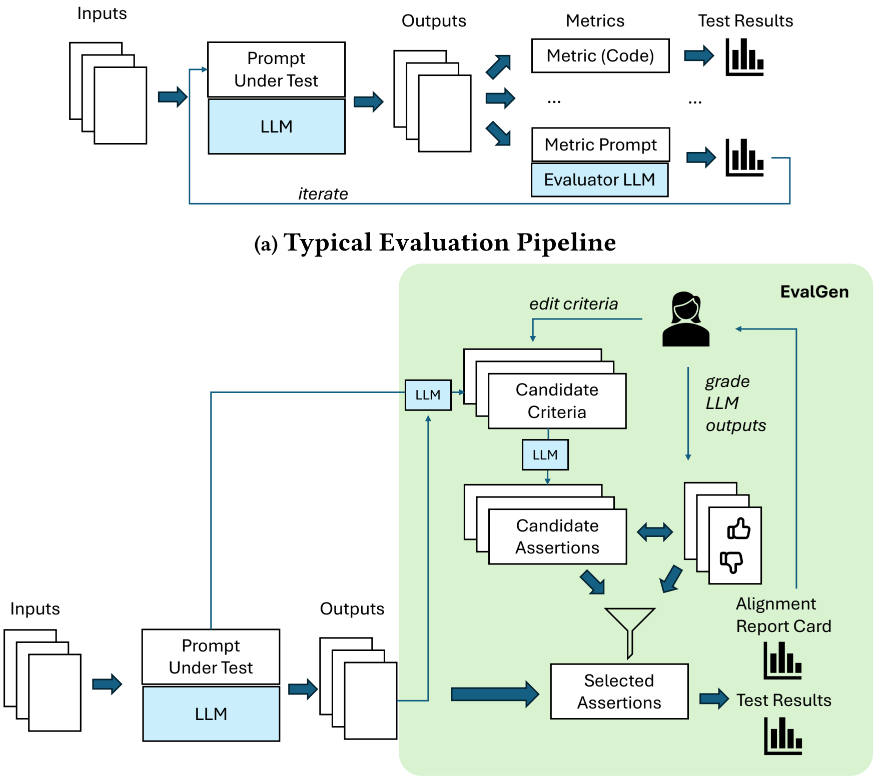
In response to the popularity of black-boxed LLMs like ChatGPT, prompt engineering (PE) has emerged as a new practice and research area. Alongside PE is the auditing of model behavior in practices such as "red-teaming," used to identify harmful outputs in internal teams to tweak LLM behavior, usually prior to release [33, p.17]. These tasks have spurred the advent of new tools for "LLM operations" (hereafter called LLMOps) and new terminology such as "prompt template", "chain of thought", "agents", and "chains."

Automating Evaluations of Prompts. When evaluating LLM behavior, users typically send off hundreds or thousands of queries to models. As users reach the limits of manual evaluation, users set up automated evaluation pipelines (Figure 1a) in code or with other LLMs. Here we use the term LLM-based evaluators; other work uses terms such as "LLM-as-a-judge" [61] or "co-audit" [19][1]. Public PE tools like promptfoo [55] and ChainForge [1] allow users to write their own evaluation metrics to score LLM response quality, and support both code-based and LLM-based evaluators. For instance, in promptfoo, users can write a rubric in a config file to specify how an LLM should evaluate responses, and may use pre-created grader prompt templates or customize them; an example is the assertion "the response is not apologetic." Prototypes such as EvalLM [28] and PromptsRoyale [43] also support LLM evaluators, oftentimes exclusively, to help users compare between two prompts. Of PE tools, only EvalLM offers a way to help users calculate the alignment of LLM evaluators with their expectations, but this feature is mentioned only in the design section of the paper and is absent from the user study. At best, users of PE tools inspect LLM-generated evaluator outputs manually to double-check; at worst, the tool hides individual scores entirely. Regardless of aligning metric implementations with user preferences, even identifying *what* metrics to evaluate for custom tasks remains challenging for LLM practitioners [40]. While many evaluation tools require users to declare metrics they care about, some prior work [48] and EvalGen employ LLMs to propose custom metrics based on prompts in the user's LLM pipelines.

Over-trust and Over-generalization of LLM Behavior. That tools provide little assistance to validate evaluator quality is alarming, considering that other research shows people tend to over-rely and over-trust AI systems [4, 29, 32, 53]. For instance, in one high-profile incident, researchers from MIT posted a pre-print on arXiv claiming that GPT-4 could ace the MIT EECS exam. Within hours, work by Chowdhuri et al. debunked the study [6], citing problems arising from over-reliance on GPT-4 to grade itself. Other work has found further reasons to be cautious: LLMs asked to choose the best response from a set can be consistently biased by set ordering [31, 54]; and LLMs can be highly sensitive to seemingly innocuous formatting changes [46].

A related problem to over-reliance is over-generalization. Zamfirescu et al. [60] found that users unfamiliar with PE tend to over-generalize from single failures (causing them to throw out potentially good prompts), rather than having a holistic view of the overall performance of a prompt or chain. This was despite the fact that the interface had support for systematic testing. Similarly, Arawjo et al. [1] found that even people familiar with LLMs (developers, academics in ML) struggled to scale up their evaluations, appearing to over-generalize from a limited number of outputs even after an automated evaluation pipeline was setup. The authors identified three modes of PE on open-domain tasks, with the second, "limited evaluation," characterized as users "prototyping an evaluation" [1], and suggested that future work focus on supported users in prototyping evaluation pipelines. Over-generalization is common in traditional ML, too–Kocielnik et al. [30] found that AI

[1] An analogous problem exists in software engineering as well: developing a set of assertions, often in the form of a set of unit tests or regression tests, that give developers confidence that their code is correct and that code changes do not (re)introduce bugs.



(a) **Typical Evaluation Pipeline**

(b) **The EvalGen Evaluation Pipeline**

Figure 1: EvalGen's approach to assisting users in aligning evaluations. Users iterate through the process of refining criteria and grading. Note that LLM pipeline inputs and outputs are provided by our larger system, and outside the scope of this paper.

systems that showcase subsets of errors, like false positives or false negatives, that have the same accuracy, can lead to vastly different perceptions of accuracy.

Approaches to Aligning LLMs. The HCI community has extensively studied interactive machine learning (iML). In iML, users iteratively develop models by selecting training examples, labeling data, and evaluating model performance [11]. Interfaces that facilitate seamless transitions between these activities result in fewer errors and outputs that better match users' expectations [41, 49]. Some iML interfaces even use ML to assist users, for example, in scaling up labeling, reducing overall user effort required [9]. When using iML concepts for developing LLM pipelines, we must acknowledge a key challenge with LLMs: they often work with little to no specific training data [40]. Users may simply prototype with inputs they imagine the LLM would see, hoping the prompt generalizes.

In the ML and NLP communities, researchers have explored many ways to align LLMs—and their evaluations—to specific user tasks. Many approaches rely on custom model training or fine-tuning [7], but all strategies heavily rely on humans to identify examples of desirable and undesirable outputs. For instance, Liu et al. [35] demonstrated using annotated LLM outputs—judged on criteria like consistency and relevance—as "few-shot examples" for calibrating LLM-based evaluators. Beyond classical summarization and NLP tasks, in response to the ad-hoc tedium of PE [59], academics and developers are building automated prompt optimization tools, maximizing some user-defined metric on a labeled set of examples. For instance, given some metrics and prompts, Khattab et al. [27] automatically run variations of inserted few-shot examples and LLM-generated rephrasings to optimize the prompt. Other work urges users to write assertions to guide outputs with a mix of code and natural language suggestions [45, 51], but writing these assertions is left up to developers, which is often time-consuming and error-prone. A broader point is that research in LLMOps optimization tends to come from the domains of NLP and ML, where authors generally validate tool performance against benchmark datasets with pre-defined metrics, leaving open the question of how well they perform in the wild on idiosyncratic user tasks, e.g. EvoPrompt, PromptBreeder, and AutoCalibrate [13, 21, 35]. It thus remains unclear how to support developers in their prototyping of evaluations, with the problem becoming even more pressing as the popularity of prompt optimization increases.

Overall, this work reveals that users need more support for (a) *prototyping* evaluations and (b) *validating* evaluators of LLM outputs. It also reveals that auditing LLM outputs is far from easy, with humans prone to the dual biases of over-generalization and over-reliance. One recent LLM-assisted approach, SPADE [48], makes headway on these issues, helping developers generate Python assertion functions for LLM outputs from prompt history. Here we leverage a similar algorithmic approach to SPADE, but embed it inside an LLM-assisted user interface for evaluator prototyping, EvalGen, that also assists with criteria generation, measuring alignment with human preferences, and visualizing results.

3 EVALGEN DESIGN

In designing EvalGen, our goal was (1) to investigate how to assist developers in creating evaluators to grade LLM outputs, and (2) to help them "validate the validators" through both automated assistance and transparency around how aligned each evaluator is with their expectations. As we covered in Section 2, emerging practices in prompt engineering, LLM auditing, and prompt optimization involve the writing of evaluation functions (metrics) to automate grading. These functions may be code- or LLM-based. Based on this context, we set out to design an LLM-powered evaluation assistant that provided developers control over metric criteria, evaluator type (code or LLM), and implementation (i.e., function) generation and selection processes, without asking them to come up with criteria or write code or grader prompts themselves.[2]

3.1 EvalGen Workflow

We implemented EvalGen in an existing open-source system for prompt engineering, ChainForge [1], which handles querying multiple LLMs with parametrized prompts, running code- and LLM-based evaluators, plotting scores, and chaining. In ChainForge, users write LLM pipelines by creating nodes of various types to represent their dataflow, such as an "input" node feeding into a "prompt" node. We discuss here only our extension, chiefly a pop-up screen that helps the user define, implement, and validate evaluation functions. We also implemented a new node, Multi-Eval, that allows users to include multiple evaluators in a single node and run all evaluators on the outputs of the pipeline's previous node. Finally, we made improvements to plotting per-criteria scores in the Table View of the LLM output inspector, which can be accessed via the Multi-Eval node. Fig. 1b provides a high-level overview of the EvalGen architecture compared to the typical LLM output evaluation pipeline; we discuss implementation details in Sec. 4.

Figure 2 depicts the workflow of in the context of the EvalGen interface, excluding returning to the main workflow with selected implementations and using the Table View to inspect scores. EvalGen assists a developer in engineering an evaluation of LLM outputs for a single prompt template. First, EvalGen is accessed as a button on a "Multi-Eval" node we added to ChainForge, which is attached to a Prompt Node (**Fig. 2a**). A Wizard opens, depicting three options (**Fig. 2b**): Infer, Manual, and Grade First. A description of EvalGen (not shown) appears above the options. Clicking Infer or Manual leads to the Pick Criteria screen (**Fig. 2c**); clicking Grade First leads to the Grading screen (**Fig. 2d**) and asks users to grade at least five outputs, before continuing to the Pick Criteria screen.

The Pick Criteria interface is depicted in **Fig. 2c**. An LLM has generated criteria suggestions in natural language (Sections 4.1 and 4.2), along with a toggle to prefer a Python code-based or LLM-based evaluator. The user can edit all parts—including the titles or descriptions and type of evaluator—or add new criteria not suggested by the LLM. They can also delete criteria or deselect criteria as needed. Our design choice to use binary criteria (true/false outputs) for validators reflects common industry practice, as seen in tools such as LangChain [12], PromptFoo [55], and Guardrails [20]. Pressing "Implement It" passes the criteria to a second LLM that generates candidate implementations.

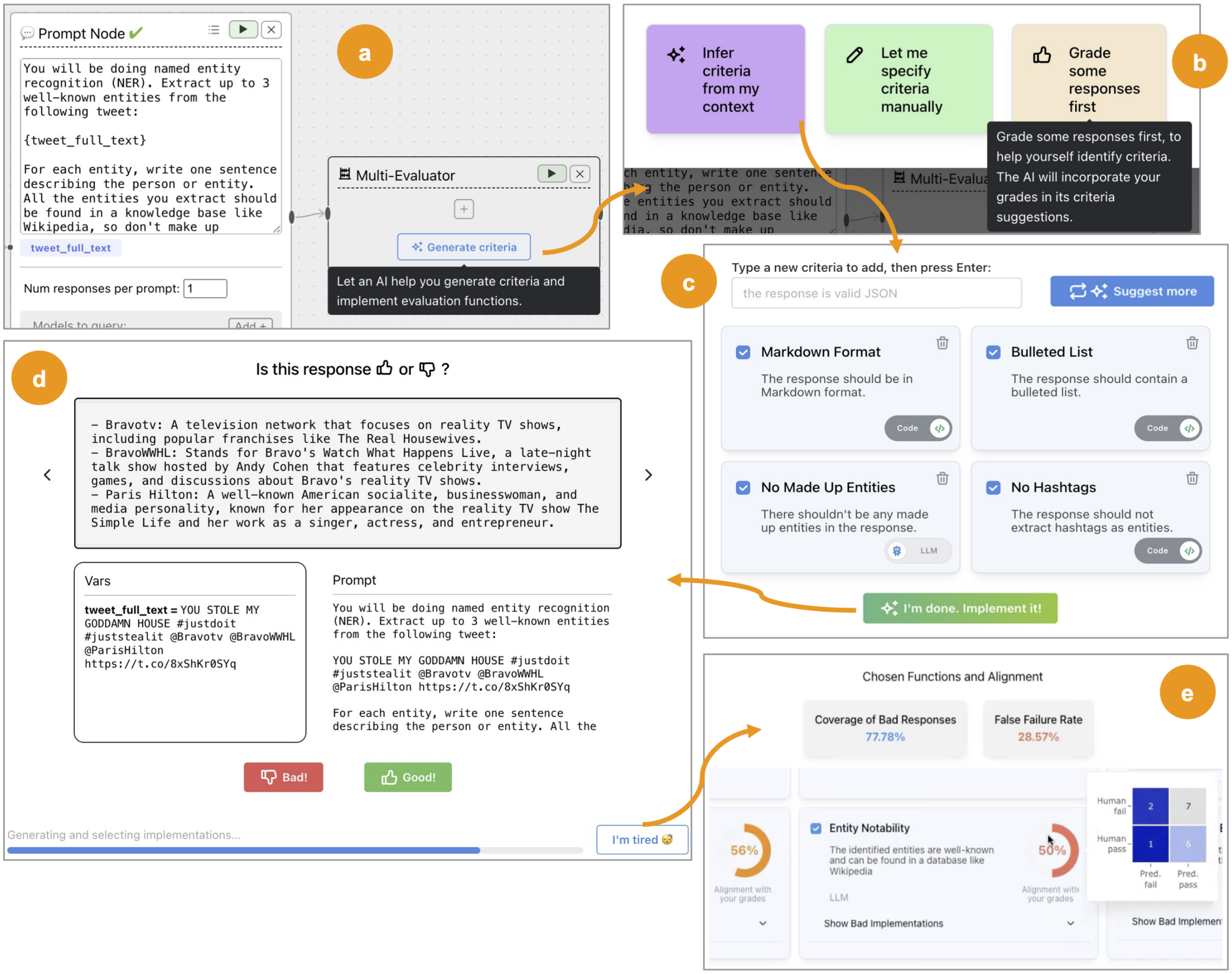
While implementations are generated and executed on LLM outputs, users are asked to grade outputs. EvalGen uses these grades to pick implementations that align best with their preferences. **Fig. 2d** depicts the Grading screen. A single LLM response is presented to the user, centered in focus in the grader window. The context of the prompt and any input variables (vars) is also present. The user grades outputs via the Good and Bad buttons. Since it may be time-consuming to ask the developer to grade on a per-criterion basis, for the grader interface we decided on the simplicity of thumbs-up/down scoring. Such scoring is a noisy yet informative signal of output quality—if a response is given a thumbs-up, it is assumed to pass all criteria, and so if a candidate assertion fails on that response, the candidate is down-ranked in the pool (details in Section 4.1). Importantly, to address the limitations of binary feedback, users can additionally provide natural language feedback on outputs they grade as bad. Users may also click arrows to navigate through outputs (e.g., if they want to revise a prior grade).[3]

Finally, after the user is done grading and all candidate implementations are generated, executed, and filtered for alignment with grades, a Report Card screen appears with feedback on per-criteria and aggregate measures of alignment with user grades (**Fig. 2e**). Hovering over per-criteria metrics shows a confusion matrix of how aligned that particular criterion is to the human grades, while the aggregate metrics show the coverage and false failure rate (see Section 5) of the selected subset of EvalGen-generated assertions. The user then returns to the main ChainForge interface (not shown), where the selected implementations are available in a "Multi-Eval" node, titled by criteria. The user can edit or add more criteria, inspect and visualize evaluation results (Fig. 3), etc.; however, this is outside the scope of our design discussion.

Our design reflects trade-offs between developer effort and robust human verification of LLM-generated metrics. The human cannot *completely* validate an LLM-based evaluator: the point of LLM evaluators is to reduce the effort required by the developer, who would otherwise have to grade outputs manually. The only way to fully align an LLM evaluator would be to ask the user to label all outputs; obviously, this defeats the purpose. Asking the developer to *grade some outputs using some time they would have spent waiting anyway*, is the key idea behind our design.

[2] To clarify our terminology throughout the paper: an evaluator is, broadly, some entity that assesses LLM output quality, while an assertion is the specific implementation generated by EvalGen that performs the evaluator role.

[3] We initially used a progress bar for grading a preset number of outputs. However, calling LLMs and executing assertions take an indeterminate amount of time: suggesting an "end point" to user grading may lose valuable information when the user still has to wait for generations to return. The user may also find grading enjoyable or important. As such, we did not seek to limit user grading. However, we kept this progress bar in the Grade First screen (accessed via **Fig.2b**).



```
You will be doing named entity
recognition (NER). Extract up to 3
well-known entities from the
following tweet:

{tweet_full_text}

For each entity, write one sentence
describing the person or entity.
All the entities you extract should
be found in a knowledge base like
Wikipedia, so don't make up
```

Num responses per prompt: 1

Let an AI help you generate criteria and implement evaluation functions.

Grade some responses first, to help yourself identify criteria. The AI will incorporate your grades in its criteria suggestions.

Type a new criteria to add, then press Enter: the response is valid JSON

☑ Markdown Format The response should be in Markdown format. Code

☑ Bulleted List The response should contain a bulleted list. Code

☑ No Made Up Entities There shouldn't be any made up entities in the response. LLM

☑ No Hashtags The response should not extract hashtags as entities. Code

Is this response 👍 or 👎 ?

```
- Bravotv: A television network that focuses on reality TV shows,
including popular franchises like The Real Housewives.
- BravoWWHL: Stands for Bravo's Watch What Happens Live, a late-night
talk show hosted by Andy Cohen that features celebrity interviews,
games, and discussions about Bravo's reality TV shows.
- Paris Hilton: A well-known American socialite, businesswoman, and
media personality, known for her appearance on the reality TV show The
Simple Life and her work as a singer, actress, and entrepreneur.
```

Vars

tweet_full_text = YOU STOLE MY GODDAMN HOUSE #justdoit #juststealit @Bravotv @BravoWWHL @ParisHilton https://t.co/8xShKr0SYq

Prompt

```
You will be doing named entity recognition
(NER). Extract up to 3 well-known entities
from the following tweet:

YOU STOLE MY GODDAMN HOUSE #justdoit
#juststealit @Bravotv @BravoWWHL
@ParisHilton https://t.co/8xShKr0SYq

For each entity, write one sentence
describing the person or entity. All the
```

Generating and selecting implementations...

Chosen Functions and Alignment

Coverage of Bad Responses 77.78%

False Failure Rate 28.57%

☑ Entity Notability The identified entities are well-known and can be found in a database like Wikipedia. LLM

Figure 2: The workflow of our EvalGen prototype, from (a) a Prompt Node attached to an empty Multi-Eval Node, showing a Generate Criteria button; (b) the pop-up EvalGen Wizard with three options, Infer, Manual, and Grade First; (c) the Pick Criteria screen, allowing users to describe criteria in natural language and toggle Code or LLM implementations; (d) the Grade screen, with the LLM output (top), input variables (left), and prompt (right), Good and Bad grade buttons, and an "I'm Tired" button (bottom-right) to finish; and finally (e) the Report Card screen, showing the alignment of each criteria and across criteria. Hovering over the alignment shows a confusion matrix. Note that some descriptions and elements have been clipped for space.

4 IMPLEMENTATION

4.1 System Architecture

Like prior work on evaluator assistants [28, 48], our solution decomposes evaluations into *criteria* and *assertions* (boolean functions that implement the criteria by evaluating outputs). We employ LLMs in generating criteria, based on the prompt [48], and in generating various candidate implementations of each criterion [28, 48]. As users grade, we rank candidate assertions that implement each criterion based on their alignment with user grades (see Section 4.2 for how we define alignment). At a high level, alignment is a combination of the assertion's *coverage*, or ability to catch erroneous outputs that the user also thinks are bad, and its *false failure rate*, i.e., how often are failures flagged incorrectly, a measure of its ability to not erroneously fail outputs that the user thinks are good.

EvalGen's architecture differs from prior work in two main components: first, EvalGen solicits grades from the user on a sample of LLM outputs—requiring some policy to sample LLM outputs to grade. Second, in contrast to SPADE [48], which operates offline and solves an integer linear program to generate the optimal assertion set, EvalGen employs an online (i.e., streaming) system architecture to progressively optimize for the most aligned assertion set. Our system, as depicted in Figure 1b, is structured into three components:

Criteria Suggestion. We use GPT-4 to propose various binary evaluation criteria in natural language. Developers can select from these suggestions or add their own criteria, choosing whether each should be evaluated with a purely code-based function or a function that involves calls to another LLM.

Candidate Assertion Synthesis and Execution. Based on the selected criteria, we use GPT-4 to asynchronously generate one or more candidate assertions as code or a grader prompt. For each criterion, we issue one call to GPT-4 to generate multiple candidate assertions (within JSON markers) in a streaming fashion. Every time we detect the end of a marker in any GPT-4 response, we parse the candidate assertion and submit it to EvalGen's executor, which will run it on LLM pipeline outputs.

Grading Sampler. This component samples LLM pipeline outputs for the user to give binary feedback on (thumbs up/down). When the user grades an LLM output, we update internal estimates of alignment for each candidate assertion, and we sample the next output for the user to grade.

Once the user does not want to grade LLM outputs anymore, or is finished grading all outputs, for each criterion, we select the candidate assertion with the highest alignment with the user's grades. The user can provide a threshold for the false failure rate (as defined in Section 5) such that EvalGen only selects assertions that do not exceed this threshold.

4.2 Selecting Assertions & Eliciting Grades

EvalGen maintains dynamic estimates for the following:

Selectivity of Candidate Assertions and Confidence Scores for Potentially Poor Outputs. The selectivity is the probability that an assertion will classify an LLM output as passing and is adjusted each time EvalGen executes a candidate assertion on an LLM output. We also maintain a confidence score for each output, which estimates the likelihood that an LLM output is of low quality, without having been explicitly evaluated by the user. The scores are dependent on assertion selectivity and are revised whenever EvalGen evaluates a new assertion against an LLM output, or when a user grades an LLM output directly.

Assertion Alignment. For each criterion, we select the candidate assertion with the highest alignment score. We adopt notation from Shankar et al. [48] in defining alignment. Formally, let E be a set of LLM pipeline input-output pairs and $f : E \longrightarrow 0, 1$ represent an assertion. Let y be a binary vector, where $y_i \in \{0, 1\}$ represents whether the user thinks an LLM output e_i is bad (0 is bad, 1 is good). Suppose $F = f_1, f_2, \ldots, f_j$ is a set of j assertions. The coverage and false failure rate (FFR) of F are represented by the following equations:

$$\text{Coverage}\,(F) = \frac{\sum_i \mathbb{I}\left[y_i = 0 \wedge (\exists f \in F, f\,(e_i) = 0)\right]}{\sum_i \mathbb{I}\left[y_i = 0\right]}$$

$$\text{FFR}\,(F) = \frac{\sum_i \mathbb{I}\left[y_i = 1 \wedge (\exists f \in F, f\,(e_i) = 0)\right]}{\sum_i \mathbb{I}\left[y_i = 1\right]}$$

In both definitions, $\mathbb{I}$ is the indicator function. Intuitively, coverage represents the set's true negative rate, while false failure rate represents the set's false negative rate. An aligned set of assertions would have a high coverage and low false failure rate. We define the alignment of F as the harmonic mean of coverage and the inverse of FFR:

$$\text{Alignment}\,(F) = 2 \times \frac{\text{Coverage}\,(F) \times (1 - \text{FFR}\,(F))}{\text{Coverage}\,(F) + (1 - \text{FFR}\,(F))}$$

Note that alignment is the F1 score; however, we are concerned with the precision and recall of failures (i.e., when $f = 0$, not when $f = 1$), and we are concerned with a set (i.e., when any assertion returns 0). See Appendix A for a complete description of assertion selectivity and how it impacts confidence scores; how EvalGen uses confidence scores to sample grades from the user; and how EvalGen determines the resulting assertion set based on alignment.

5 ALGORITHM EVALUATION

Before proceeding to our user study, we conducted an offline evaluation of EvalGen's selection algorithm. This evaluation served as a sanity check to ensure the quality of our technical implementation, verifying that any findings in the subsequent user study would not be the result of significant implementation flaws. Our experiment aimed to understand how *soliciting human input* at the criteria suggestion stage impacts the size (number of assertions) and alignment of the resulting assertion set. We compared to a baseline, SPADE [48], a fully automated system that generates criteria and candidate assertions and chooses the minimal assertion set that meet coverage and false failure rate constraints.

5.1 Evaluation Setup

We developed two LLM pipelines based on real-world datasets. The *medical* pipeline operates on a dataset of 84 unstructured text transcripts from doctor-patient calls [57], aiming to extract specific information (e.g., symptoms, medication) without revealing any personally identifiable information (PII). This task requires assertions to ensure compliance with privacy laws. The *product* pipeline involved crafting SEO-friendly descriptions for 100 Amazon products and their reviews [22]. We selected this task because it mirrors actual LLM applications (there are a number of startups using AI to write SEO-optimized product descriptions), and it benefits from assertions: for example, even if there are negative reviews, the descriptions should not say negative things about the products, which would adversely affect the products' sales potential. Our prompts are presented in Appendix B. For both prompts, the placeholder variables (i.e., `transcript` and `document`) represent the input context to inject at pipeline runtime.

We used OpenAI's GPT-3.5-Turbo to generate outputs. Two of the paper authors manually graded all LLM outputs to establish ground-truth labels. The medical and product pipelines had 68% and 51% good outputs, respectively. Common issues included the presence of personal information in the medical pipeline outputs and bad reviews or lengthy content in the product pipeline outputs.

5.2 Impact of Human Input in the Criteria Generation Step

There are two differences between SPADE and EvalGen in how they generate assertion sets. The first difference is that EvalGen asks the user to add, edit, or remove criteria before generating different candidate assertions, whereas SPADE does not solicit any input from the user about the criteria. The second difference is in the selection of the assertions themselves: given user-confirmed criteria and a sample of grades provided in a UI, EvalGen picks the most aligned assertion per criterion that meets some false failure rate threshold. Meanwhile, SPADE solves an optimization problem to select a minimal assertion set that meet a false failure rate threshold and cover all SPADE-generated criteria.

5.2.1 Evaluation Procedure. We ran SPADE on both pipelines with all labeled outputs, initially setting a 10% false failure rate (FFR) threshold. The product pipeline required adjusting to 40% FFR to find a viable assertion set. This illustrates the challenge of balancing coverage with false failures, underscoring the need for evaluator systems to effectively make these trade-offs.

Subsequently, we ran EvalGen for both pipelines with the same thresholds. For the medical pipeline, we defined three evaluation criteria: word count, presence of the six targeted keys, and absence of PII, with the first two implemented via code-based assertions and the last via an LLM evaluator. The product pipeline criteria included absence of negative reviews, absence of links, adherence to markdown format, and word count limitation, with only the first criterion requiring LLM implementation. To create the aligned assertion sets, we provided EvalGen with 16 graded outputs per pipeline instead of all graded outputs (which would have been between 80 and 100 per pipeline)—given the impracticality of expecting users to extensively grade in a single session.

5.2.2 Results. Our results in Table 1 show that EvalGen, by incorporating human judgment during criteria selection, achieved equal or better alignment than SPADE with fewer assertions for both pipelines. In the medical pipeline, SPADE added unnecessary assertions (e.g., one for a neutral tone), while EvalGen maintained a more focused set. In the product pipeline, EvalGen's assertion set was less than half the size of SPADE's, with increased coverage (73% vs. 49%). For the product pipeline, some of SPADE's assertions were unrealistic, like a Python function designed to flag specific negative phrases such as "never order" and "disappointed" in the output. In contrast, EvalGen returned a more pragmatic assertion for this criterion—an LLM-based validator to ensure the product descriptions remained entirely positive.

	Medical Pipeline		Product Pipeline	
Metric	**EvalGen**	**SPADE**	**EvalGen**	**SPADE**
Dataset Size	84	84	100	100
# Bad Outputs	27	27	49	49
# Assertions	**3**	5	**4**	9
Coverage	0.33	0.33	**0.73**	0.49
FFR	0.10	0.10	0.39	0.39
Alignment (%)	48.29	48.29	**66.46**	54.35

Table 1: Comparison of EvalGen and SPADE Across Pipelines. With user input at the criteria stage, EvalGen achieves the same or greater alignment with fewer functions.

6 USER STUDY DESIGN

To understand how developers might use EvalGen to build evaluators for LLM pipelines, we conducted a qualitative study with nine industry practitioners experienced in LLMs. This approach allowed for detailed feedback on our validator alignment workflow.

Recruitment and Participants. We recruited nine industry practitioners via a Twitter post, calling for anyone interested in solving the problem of "who validates the validators," selecting the first nine respondents with experience in coding and building LLM pipelines for companies or products. Participants included software engineers, ML scientists, startup executives, and independent consultants. While nine might seem small, studies suggest that as few as five participants can provide significant usability insights [2, 38]. We focused on LLM-experienced developers for their ability to compare EvalGen to existing workflows.

Procedure. Studies were conducted over Zoom, beginning with a brief background discussion. We introduced participants to our ChainForge LLM pipeline for named entity recognition (NER) on tweets, using GPT-3.5-Turbo. The prompt was: *You will be doing named entity recognition (NER). Extract up to 3 well-known entities from the following tweet: {tweet_full_text} For each entity, write one sentence describing the person or entity. All the entities you extract should be found in a knowledge base like Wikipedia, so don't make up entities. Return your answer as a bulleted Markdown list, where each bullet is formatted as `- entity: description`. Do not extract hashtags as entities.* We chose this task for its real-world relevance, concise input format, and existing popularity within the research community [16, 34, 52]. Participants could modify the task or prompt if they wanted.

After explaining EvalGen's functionality, participants were given remote control access and up to 40 minutes to explore the tool while thinking aloud. We communicated that we were mainly interested in observing their process of creating assertions, not interacting with other features of ChainForge such as comparing different LLM APIs. If the participant had any questions about the interface, we answered them. Post-exploration, we conducted a 10-minute open-ended interview, asking about EvalGen's assertion generation approach and perceived alignment with their grades. Participants rated alignment on a 7-point Likert scale. The entire study lasted 45-75 minutes. Our study was approved by our institutional review board (IRB), and participants generously volunteered their time.

Analysis. We asked participants to think aloud while using the tool, while we took notes on their thoughts and any visible emotions (e.g., delight when EvalGen suggested a criterion they struggled to externalize, or frustration when they could not find a good assertion for a criterion). We employed open and axial coding [14] to identify common themes across the video call transcripts and notes for each participant. Initially, we coded individual sentences of interest for each participant, then grouped these into broader themes on a per-participant basis in a second pass of coding. Finally, we consolidated these themes across all participants, reorganizing them as needed.

7 USER STUDY FINDINGS

Our user study revealed several insights into participants' experiences with EvalGen and their evaluation processes:

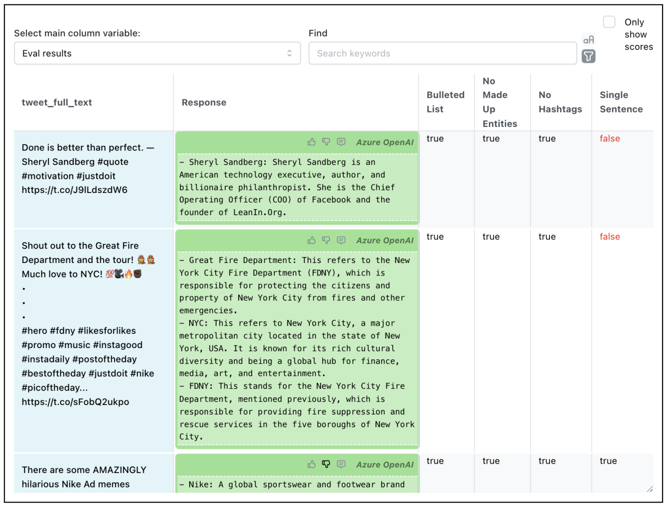
- Participants felt that EvalGen was a great starting point for assertions, and wanted to—and could—exercise control over EvalGen's assistance.
- Participants encountered difficulties in aligning assertions with their preferences, primarily due to two challenges in grading: *(i)* some criteria proved difficult for human evaluation (e.g., adherence to a target word count), and *(ii)* we observed a "criteria drift" phenomenon, wherein criteria evolved as participants graded more LLM outputs, affecting both the definitions of existing criteria and the overall set of criteria.
- Participants' perceptions of alignment and their needs varied based on the evaluator type (i.e., code-based vs. LLM-based).

We elaborate on these findings below, first describing the typical participant workflow and highlighting areas where participants sought to exercise control, then discussing the challenges they faced in aligning assertions with their preferences.

7.1 Typical Participant Workflow

All participants ($n = 9$) engaged with the provided task, which involved a prompt template for Named Entity Recognition (NER) on a dataset of 100 tweets, as described in Section 6. Three participants modified the prompt, with one opting to change the task from NER to sentiment analysis. After finalizing their prompt choice, participants generally followed this sequence of activities:

(1) **Eyeballing LLM outputs:** Participants reviewed the table of 100 LLM outputs to assess their overall quality and reasonableness.
(2) **Starting EvalGen:** Participants started EvalGen, which presented three options (Figure 2b): auto-generate criteria, write criteria manually, or grade outputs before generating criteria. Six participants chose auto-generation, one wrote a criterion before auto-generating, and two opted to grade first (P4, P9).
(3) **Grading outputs:** Those who graded first evaluated between 5 and 10 outputs, assigning 2 to 4 "thumbs-down" grades, before proceeding to auto-generate criteria.
(4) **Refining criteria:** Upon receiving EvalGen's suggestions, participants removed some and added one or two of their own criteria. They typically maintained EvalGen's suggested evaluation type (code-based or LLM-based), even in rare instances where the suggestion seemed suboptimal (e.g., checking word count with an LLM API call, rather than code).
(5) **Grading more outputs:** Participants graded outputs while EvalGen generated and evaluated candidate assertions. Grading duration varied, with some continuing for up to 10 minutes and others stopping after 10 grades.
(6) **Understanding alignment on *graded* outputs:** Participants examined the "Report Card" screen, inspecting the resulting assertion set. All but one participant (P9) readily understood the concepts of coverage and false failure rate. They reviewed various assertion implementations, including those misaligned with their grades.
(7) **Eyeballing alignment on *ungraded* outputs:** Returning to the main interface, participants executed all assertions on the full set of 100 LLM outputs and examined the results table (Figure 3).



Select main column variable:
Eval results

Find
Search keywords

□ Only show scores

tweet_full_text	Response	Bulleted List	No Made Up Entities	No Hashtags	Single Sentence
Done is better than perfect. — Sheryl Sandberg #quote #motivation #justdoit https://t.co/J9lLdszdW6	Azure OpenAI – Sheryl Sandberg: Sheryl Sandberg is an American technology executive, author, and billionaire philanthropist. She is the Chief Operating Officer (COO) of Facebook and the founder of LeanIn.Org.	true	true	true	false
Shout out to the Great Fire Department and the tour! 👩‍🚒👩‍🚒 Much love to NYC! 💯🎥🔥✊🏿 . . . #hero #fdny #likesforlikes #promo #music #instagood #instadaily #postoftheday #bestoftheday #justdoit #nike #picoftheday... https://t.co/sFobQ2ukpo	Azure OpenAI – Great Fire Department: This refers to the New York City Fire Department (FDNY), which is responsible for protecting the citizens and property of New York City from fires and other emergencies. – NYC: This refers to New York City, a major metropolitan city located in the state of New York, USA. It is known for its rich cultural diversity and being a global hub for finance, media, art, and entertainment. – FDNY: This stands for the New York City Fire Department, mentioned previously, which is responsible for providing fire suppression and rescue services in the five boroughs of New York City.	true	true	true	false
There are some AMAZINGLY hilarious Nike Ad memes	Azure OpenAI – Nike: A global sportswear and footwear brand	true	true	true	true

Figure 3: The Table View, showing inputs, LLM outputs, and evaluation results per criteria for the NER task (Sec. 6).

(8) **Iterating on criteria:** Three participants revisited the EvalGen wizard to further refine their criteria and assertions. Over half expressed interest in additional iteration, given more time.

7.2 EvalGen Provides a Refinable Starting Point for Assertions

Participants appreciated EvalGen's assistance in generating an initial set of assertions, valuing the ability to exert control when necessary. P8 summarized this sentiment: "This is how I would want a workflow to assist me in evals—basically I want the AI to do 80% of it, and there can be escape hatches if the AI fails." Eight out of nine found grading while waiting for EvalGen to generate and execute assertions to be a good use of time, though P7 suggested showing what EvalGen was doing with the grades.

7.2.1 *LLM-generated criteria alleviates writer's block.*

Eight out of nine participants were pleasantly surprised by the suggested criteria. P4 said, "I get writer's block when thinking about what assertions to write, so this is great." Participants who graded before selecting criteria (P4 and P9) found this useful, with P9 realizing it was "the correct thing to do" despite initially choosing it as the "option that required the least thinking."

7.2.2 *Users want to continue refining assertions after reviewing alignment on graded results.*

In the Report Card screen, participants appreciated seeing multiple assertion implementations and alignment scores. Some wanted to grade more responses to improve alignment (P1, P7). Participants liked seeing coverage and false failure rates but desired per-criterion alignment information (P5, P6, P7, P8).

Participants *really* liked viewing the table of assertion results on all LLM outputs, with all participants expressing interest upon first viewing it (Figure 3). In this table, rows represent LLM outputs, and columns represent assertions. P2 said that this table "earns trust"; P5 said that it was "cool to see all the results on examples

P1	P2	P3	P4	P5	P6	P7	P8	P9
6	5	3	4	5	3	1	2	5

Table 2: Ratings (1-7, 7 best) for the statement, "*I felt like the assertions aligned with my grades.*" Responses were mixed.

[they] didn't grade." Some participants, like P6, wanted automatic recomputation and visualization of changes:

> *One thing I would find cool is if there is a way to easily see how changes to my prompt impact the overall [coverage and false failure rate] scores. Just very quickly being able to visualize how [my prompt edit] changes the classifications on a bunch of [LLM outputs.]*

This suggests a desire for visualizations that compare original and revised prompts, akin to LLM Comparator [25] and EvalLM [28].

7.3 Alignment is Iterative, Criteria- and Implementation-specific

Perceptions of the tools' support of alignment were polarized across participants, as shown in Table 2. Participants would say utterances with a questioning tone, like "I guess" and "sure," when grading, indicating their uncertainty (P2, P5, P7, P8, P9). Looking more closely at their interactions, we observed a catch-22 situation: participants needed to externalize criteria in order to grade outputs, but they also needed to grade outputs—providing feedback on why bad outputs were bad—in order to externalize criteria. Here, we explore their challenges.

7.3.1 Criteria drift. Grading outputs spurred changes or refinements in participants' criteria, which we refer to as criteria drift. We observed two types of drift. First, participants wanted to *add new criteria* when they observed new "types" of bad LLM outputs (P2, P5, P6, P8, P9). In the EvalGen interface, they could not go back and add new criteria; they had to wait for all candidate assertions to finish executing, move past the report card screen, and start a new EvalGen process. Second, as participants graded more outputs, we found that they *reinterpret existing criteria to better fit the LLM's behavior* (P2, P5, P6, P8, P9). For example, P2 and P8 had a "proper noun" criterion, which was supposed to assess that "the entities extracted were proper nouns." At first, they rated as bad any LLM outputs that contained *any* entity that was not a proper noun. But, after observing that responses had varying numbers of proper nouns, both wanted to change their criteria such that *most* of the entities were proper nouns, rather than all. P9 appreciated the ability to provide feedback on unsatisfactory outputs before finalizing the initial criteria set, as this process helped them articulate their criteria. Twice, P8 mentioned that they gave a bad grade not because they believed the output was bad, but because they wanted to be consistent with previous grades—good labeling practice, perhaps, but not good for alignment. P7 noticed that some outputs included hashtags from the original tweet inputs (e.g., #justdoit), while in other cases, the outputs did not use the hashtag symbol but still mentioned the entities referred to by the hashtags (e.g., "Nike" instead of #Nike). This inconsistency led P7 to rethink the criteria definition for including hashtags; specifically, whether it was acceptable for the LLM to replicate entities from hashtags, provided the hash (#) was removed. P5 expressed the same uncertainty. "I think it's hard to know until you see it," P7 said.

7.3.2 Users prefer to adjust their grading approach based on the difficulty of evaluating a criterion. Overall, participants generally liked the process of grading LLM responses and feeling like the grades were useful, but they wanted to prioritize grading criteria they felt *needed their alignment*—especially for LLM-based assertions (P3, P5, P7, P8). For example, P3 expressed that they would trust the assertions more if the EvalGen process allowed them to set different false failure rates per criteria (since LLMs might be worse at evaluating some criteria), instead of one global false failure rate constraint for the entire assertion set:

> *There are criteria where you can be okay with failing, and then there are other criteria where you are like, 'this must absolutely pass'... [T]here's a [spectrum] of failure as opposed to: it just passes or fails.*

Relatedly, some participants expressed that they didn't trust their grades because they *themselves* couldn't evaluate some criteria as well as an automated solution (P2, P5, P6, P7, P8; we discuss further in Section 7.4). A criterion like word count is hard for humans to assess but easy for a good Python function to evaluate. P8 desired to grade for only one criteria, reasoning that it might improve efficiency ("I generally want to be in the loop for these tests...but I want to put myself in the loop in a way that is efficient.").

7.3.3 What constitutes "alignment" is subjective, especially when converting natural language criteria to code-based assertions. For code-based assertions, EvalGen's interpretation of the criterion (i.e., GPT-4's interpretation) did not match what the participants expected. As such, no matter how many grades the participant gave, all candidate assertions were similarly misaligned. Participants who observed this were confused why their grades seemingly had little impact on some of the chosen assertions (P3, P5, P7, P9). For example, while all participants had a criterion to enforce that there were no entities with hashtags in the output, some participants interpreted this as any hashtags representing entities should not be extracted as entities: e.g., if the output included the hashtag #Nike, P5 did not want Nike to be extracted at all. On the other hand, P9 wanted Nike to be extracted as an entity, but they did not want the LLM output to include the hash (#). Both P5 and P9 got the same code-based assertion for the criterion, which simply checked for the presence of the hash character in the output—this assertion did not align with P5's grades, but did with P9's. This particular misalignment can also be viewed as an instance of criteria drift, as described in Section 7.3.1, since P5 was only able to refine the criterion after grading several LLM outputs. For another criterion, P5 felt that EvalGen could not find a good assertion that aligned with their grades, but also said that they were "lost at what would be a good implementation."

Overall, alignment is not merely a matter of performance, i.e., the idea that "a better LLM would do better". Misalignment sometimes occurred due to tacit criteria that participants held which was not explicitly explained in natural language, e.g., one participant preferred the entity "Nike" over "Nike Shoes" being extracted by the LLM pipeline, while another participant was satisfied with "Nike

Shoes" as the extracted entity. Like prior work has found for cross-LLM comparison [1], this tacit understanding of criteria could be highly subjective and contradictory across participants.

7.4 Alignment Needs and Preferences Differ for Code vs. LLM Evaluators

All participants appreciated EvalGen's ability to generate both code-based and LLM-based assertions, expressing the need for both types. P4 and P6 liked the ability to correct EvalGen's suggested type. However, participants wanted different approaches for construction and iteration.

7.4.1 Users like having control over the evaluation type. Participants had clear preferences for when to use code-based versus LLM-based assertions. They generally preferred code for formatting checks, count-based checks, and specific phrase inclusion/exclusion. LLM-based assertions were favored for "fuzzy" criteria (P6, P8) or when external knowledge was required. P2 opted for LLM-based assertions when they couldn't immediately think of a Python function for the criteria. Interestingly, P8 sometimes preferred LLM-based assertions due to their relative forgiveness with unexpected output qualities or formats.

7.4.2 Users want to directly verify code-based implementations. For code-based assertions, many participants (P2, P5, P7) wanted to see and select the best Python function themselves, rather than relying on EvalGen's selection process. P5 stated, "When something can be solved using Python code, I do have an envisioned [implementation] in mind that I can easily verify. Just showing [me] the [code] will be quicker." P7 suggested iterating on code-based assertions by providing feedback to make them more or less "fancy," though they acknowledged that this depends on code complexity.

7.4.3 LLM-based assertions are harder to trust. While participants found EvalGen's suggested code-based assertions to be more obviously misaligned than the LLM-based assertions (Section 7.3.3), they also found LLM-based assertions harder to trust. This was primarily because they could edit code-based assertions more easily (P2, P5, P6, P8, P9). Some participants (P3, P6, P8) were skeptical about using LLM-based assertions in production pipelines. P8 expressed, "I cannot begin to think about how LLMs as validators [in production] can work, I'm very skeptical." P9 raised concerns about maintaining evaluations over time, asking, "How do I maintain my evals over time; do I have to rerun this entire process?" One suggestion was an interface that continually realigns LLM-based assertions, possibly involving end-users of the LLM pipeline. Several participants (P3, P4, P5, P7) wanted EvalGen to use their grades and feedback in prompting LLMs for candidate assertions. Some (P3, P5) even suggested including labeled LLM outputs in the prompts for LLM-based assertions. This suggests an optimization loop akin to ConstitutionMaker and DSPy [27, 42], but for *assertion generation and validation*, rather than prompt optimization.

8 DISCUSSION

8.1 Implications of Criteria Drift

The practice of benchmarks in ML and NLP presume a world of well-defined criteria (and well-labeled data) on which to judge LLM outputs. For instance, AutoCalibrate is a method to calibrate LLM evaluators with human preferences that requires large expert-labelled datasets with settled (i.e., established upfront) criteria [35]. However, in practice, we found that developers rapidly iterate over criteria, and furthermore that cognitively engaging with LLM outputs helps them to refine their criteria. This suggests criteria refinement and grading should happen *in tandem* in interactive settings, and poses challenges to alignment methods that presume settled, expert labels. Future system designs should support these requirements. A system might adjust criteria dynamically as the user grades and gives feedback. Per-criteria grading and including examples of both good and bad LLM outputs within LLM-based evaluator prompts could be beneficial. However, whenever outputs change, we may need to ask users to re-grade or re-think their criteria. The dependence of criteria on LLM outputs implies that users cannot build evaluations independently from prompting LLMs—and assessing the outputs.

Our criteria drift finding echoes prior work in educational settings, where instructors often update grading rubrics to reflect common errors [50]. Evaluation assistants might pursue crowdsourcing methods to determine accurate grades for LLM outputs (e.g., majority voting, self assessments) [8, 10, 39]. Another challenge is extending adapted criteria to grade both ungraded and future unseen LLM outputs. How do evaluation assistants consistently sample grades for outputs that reflect the overall distribution of LLM pipeline successes and failures?

The reader might wonder when criteria "settle." Perhaps there was simply not enough time in our study, and had participants graded for an hour or two, they might have solidified their criteria, and criteria drift goes away. There is reason to believe that this situation does not change with more time—as we saw, the criteria participants refined changed *to adapt to the behavior of the LLM outputs being evaluated*—a *dependent,* rather than independent assessment of quality. In the real world, similar situations exist where criteria are never "fully settled" as more inputs come in—consider the court of law. One of our participants remarked that people "know a bad output when they see it." Their adage reflects a U.S. Supreme Court Justice's famous opinion in a 1964 court case about obscene content [18]. As in that remark, the decisions of human validators seem at first glance "to be based on a non-rational, intuitive gut reaction, instead of reasoned analysis; it seems to be utterly subjective and personal" [18, p.1025]. However, as the law scholar Gewirtz argues, perhaps subjectivity is *not* necessarily a sign of irrationality (contrasting with some imagined future AI that is entirely objective and rational, entirely "aligned" or "better" than humans). On the contrary: "There are good reasons to accept the imperfect in a judge. We should encourage judges to believe and say: This is the best I can do now; it doesn't solve all the problems, but it's a start, and I'll keep thinking" [18, p.1027]. This raises a deeper epistemic question for evaluation assistants—is "alignment" an actualizable goal? To what extent does our common terminology and assumptions—e.g., that there is a "ground truth" set of labels we merely need to elicit—fail us? Is validating the validators only ever a work-in-progress?

8.2 Operationalizing Assertions

Participants expressed the desire to deploy their assertions in production, either in the critical path of the LLM pipeline or in a more passive deployment. Assertions have different operational requirements, and as criteria change over time, evaluation assistants should adapt assertion sets.

Our study confirmed the necessity for both code-based and LLM-based assertions, with participants feeling each type required distinct treatment. Code-based assertions, often useful for sanity checks, can be used in the critical path of the LLM pipeline. In fact, a number of LLMOps tools exist for users to implement such code-based guardrails [20, 23, 45]. However, our participants highlighted the challenge of finding the *right* implementation for a given assertion—which can depend on the characteristics of LLM outputs. For example, determining the acceptable length of a response might vary significantly based on the observed output distribution.

Some participants wanted their collaborators to grade outputs in EvalGen. When allowing multiple users to collaborate on grading, evaluation assistants must consider inter-rater reliability and handle disagreements. Interfaces similar to creating a "pull request" for a new assertion and workflows akin to continuous integration/continuous deployment (CI/CD) could be beneficial for team collaboration.

8.3 Future Work and Limitations

The potential of evaluation assistants extends beyond EvalGen's supported binary judgments. For instance, rather than setting rigid thresholds for criteria like word count, it might be more useful to monitor variations across different LLM outputs. Like in traditional ML monitoring [47], there is still imprecision in what counts as "bad," given the distribution of outputs. Tracking finer-grained information in evaluations, beyond simple true/false conditions, can aid in debugging issues within LLM pipelines once a user knows the output is bad. Additionally, evaluation assistants can facilitate end-to-end alignment in chains of multiple LLM calls[1, 12, 15, 56] or compound AI systems [58], addressing the challenges that arise in these more complex setups. Moreover, users increasingly want their prompts to automatically improve based on assertion results. Some frameworks already experiment with using feedback from assertions or user grades to refine prompts [36, 42, 51]. Incorporating this into an evaluation assistant could foster a co-evolutionary environment where prompts, assertions, and evaluative mechanisms are continuously refined in a unified interface.

Our study has two main limitations: our offline evaluation focused on only two pipelines, and our qualitative study involved a small sample of experienced LLM developers. Moreover, participants did not have enough time to iterate many times in EvalGen, and our setup did not cover the deployment phase of LLM workflows. Future work might explore best practices and pitfalls of evaluations in the broader LLMOps lifecycle.

9 CONCLUSION

This work presented EvalGen, a mixed-initative approach to aligning LLM-generated evaluation functions with human preferences. EvalGen assists users in developing both criteria for acceptable LLM outputs and functions to check these standards, ensuring evaluations reflect the users' own grading standards. In a qualitative study with 9 expert users, we observed a pattern we call *criteria drift*, where users refine their evaluation standards as they grade more LLM outputs. Recognizing the mutual dependency of criteria on LLM outputs highlights new directions for designing future evaluation assistants.

ACKNOWLEDGMENTS

We acknowledge the support from grants DGE2243822, IIS-2129008, IIS-1940759, IIS-1940757 and IIS-1955394 awarded by the National Science Foundation, an NDSEG Fellowship, funds from the State of California, funds from the Alfred P. Sloan Foundation, as well as EPIC lab sponsors: GResearch, Adobe, Microsoft, Google, and Sigma Computing. The authors would also like to thank Helen Weixu Chen for their software engineering help in preparing EvalGen for deployment.

REFERENCES

[1] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena Glassman. 2023. ChainForge: A Visual Toolkit for Prompt Engineering and LLM Hypothesis Testing. *arXiv preprint arXiv:2309.09128* (2023).

[2] Hugh Beyer and Karen Holtzblatt. 1998. Contextual inquiry. *Defining customer-centered systems* 31 (1998).

[3] Pierre Boyeau, Anastasios N Angelopoulos, Nir Yosef, Jitendra Malik, and Michael I Jordan. 2024. AutoEval Done Right: Using Synthetic Data for Model Evaluation. *arXiv preprint arXiv:2403.07008* (2024).

[4] Zana Buçinca, Maja Barbara Malaya, and Krzysztof Z Gajos. 2021. To trust or to think: cognitive forcing functions can reduce overreliance on AI in AI-assisted decision-making. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–21.

[5] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. How is ChatGPT's behavior changing over time? *arXiv preprint arXiv:2307.09009* (2023).

[6] Raunak Chowdhuri, Neil Deshmukh, and David Koplow. 2023. No, GPT4 can't ace MIT. https://flower-nutria-41d.notion.site/No-GPT4-can-t-ace-MIT-b27e6796ab5a48368127a98216c76864

[7] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems* 30 (2017).

[8] Aida Mostafazadeh Davani, Mark Díaz, and Vinodkumar Prabhakaran. 2022. Dealing with disagreements: Looking beyond the majority vote in subjective annotations. *Transactions of the Association for Computational Linguistics* 10 (2022), 92–110.

[9] Michael Desmond, Michelle Brachman, Evelyn Duesterwald, Casey Dugan, Narendra Nath Joshi, Qian Pan, and Carolina Spina. 2022. AI Assisted Data Labeling with Interactive Auto Label. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 13161–13163.

[10] Steven Dow, Anand Kulkarni, Scott Klemmer, and Björn Hartmann. 2012. Shepherding the crowd yields better work. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. 1013–1022.

[11] John J Dudley and Per Ola Kristensson. 2018. A review of user interface design for interactive machine learning. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 8, 2 (2018), 1–37.

[12] Harrison Chase et al. 2023. LangChain. https://pypi.org/project/langchain/.

[13] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797* (2023).

[14] Uwe Flick. 2013. *The SAGE handbook of qualitative data analysis*. Sage.

[15] FlowiseAI, Inc. 2023. FlowiseAI Build LLMs Apps Easily. flowiseai.com.

[16] Michel Naim Gerguis, Cherif Salama, and M Watheq El-Kharashi. 2016. ASU: An Experimental Study on Applying Deep Learning in Twitter Named Entity Recognition.. In *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*. 188–196.

[17] Katy Ilonka Gero, Chelse Swoopes, Ziwei Gu, Jonathan K Kummerfeld, and Elena L Glassman. 2024. Supporting Sensemaking of Large Language Model Outputs at Scale. *arXiv preprint arXiv:2401.13726* (2024).

[18] Paul Gewirtz. 1996. On "I know it when I see it". *The Yale Law Journal* 105, 4 (1996), 1023.

[19] Andrew D. Gordon, Carina Negreanu, José Cambronero, Rasika Chakravarthy, Ian Drosos, Hao Fang, Bhaskar Mitra, Hannah Richardson, Advait Sarkar, Stephanie

Simmons, Jack Williams, and Ben Zorn. 2023. Co-audit: tools to help humans double-check AI-generated content. arXiv:2310.01297 [cs.HC]

[20] Guardrails 2023. Guardrails AI. https://github.com/guardrails-ai/guardrails.

[21] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532* (2023).

[22] Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. 2024. Bridging Language and Items for Retrieval and Recommendation. *arXiv preprint arXiv:2403.03952* (2024).

[23] Instructor 2023. Instructor, Generating Structure from LLMs. https://jxnl.github.io/instructor/.

[24] Ellen Jiang, Kristen Olson, Edwin Toh, Alejandra Molina, Aaron Donsbach, Michael Terry, and Carrie J Cai. 2022. PromptMaker: Prompt-based Prototyping with Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 35, 8 pages. https://doi.org/10.1145/3491101.3503564

[25] Minsuk Kahng, Ian Tenney, Mahima Pushkarna, Michael Xieyang Liu, James Wexler, Emily Reif, Krystal Kallarackal, Minsuk Chang, Michael Terry, and Lucas Dixon. 2024. LLM Comparator: Visual Analytics for Side-by-Side Evaluation of Large Language Models. arXiv:2402.10524 [cs.HC]

[26] Adam Tauman Kalai and Santosh S Vempala. 2023. Calibrated language models must hallucinate. *arXiv preprint arXiv:2311.14648* (2023).

[27] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714* (2023).

[28] Tae Soo Kim, Yoonjoo Lee, Jamin Shin, Young-Ho Kim, and Juho Kim. 2023. Evallm: Interactive evaluation of large language model prompts on user-defined criteria. *arXiv preprint arXiv:2309.13633* (2023).

[29] Agnes M Kloft, Robin Welsch, Thomas Kosch, and Steeven Villa. 2023. " AI enhances our performance, I have no doubt this one will do the same": The Placebo effect is robust to negative descriptions of AI. *arXiv preprint arXiv:2309.16606* (2023).

[30] Rafal Kocielnik, Saleema Amershi, and Paul N. Bennett. 2019. Will You Accept an Imperfect AI? Exploring Designs for Adjusting End-user Expectations of AI Systems. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3290605.3300641

[31] Zongjie Li, Chaozheng Wang, Pingchuan Ma, Daoyuan Wu, Shuai Wang, Cuiyun Gao, and Yang Liu. 2023. Split and merge: Aligning position biases in large language model based evaluators. *arXiv preprint arXiv:2310.01432* (2023).

[32] Q. Vera Liao and S. Shyam Sundar. 2022. Designing for Responsible Trust in AI Systems: A Communication Perspective. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 1257–1268. https://doi.org/10.1145/3531146.3533182

[33] Q Vera Liao and Jennifer Wortman Vaughan. 2023. AI transparency in the age of LLMs: A human-centered research roadmap. *arXiv preprint arXiv:2306.01941* (2023).

[34] Xiaohua Liu, Furu Wei, Shaodian Zhang, and Ming Zhou. 2013. Named entity recognition for tweets. *ACM Transactions on Intelligent Systems and Technology (TIST)* 4, 1 (2013), 1–15.

[35] Yuxuan Liu, Tianchi Yang, Shaohan Huang, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. 2023. Calibrating LLM-based evaluator. *arXiv preprint arXiv:2309.13308* (2023).

[36] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2024).

[37] Aditi Mishra, Utkarsh Soni, Anjana Arunkumar, Jinbin Huang, Bum Chul Kwon, and Chris Bryan. 2023. PromptAid: Prompt Exploration, Perturbation, Testing and Iteration using Visual Analytics for Large Language Models. *arXiv preprint arXiv:2304.01964* (2023).

[38] Jakob Nielsen. 1994. *Usability engineering*. Morgan Kaufmann.

[39] Aditya G Parameswaran, Shreya Shankar, Parth Asawa, Naman Jain, and Yujie Wang. 2023. Revisiting prompt engineering via declarative crowdsourcing. *CIDR* (2023).

[40] Chris Parnin, Gustavo Soares, Rahul Pandita, Sumit Gulwani, Jessica Rich, and Austin Z Henley. 2023. Building Your Own Product Copilot: Challenges, Opportunities, and Needs. *arXiv preprint arXiv:2312.14231* (2023).

[41] Kayur Patel, Naomi Bancroft, Steven M. Drucker, James Fogarty, Amy J. Ko, and James Landay. 2010. Gestalt: integrated support for implementation and analysis in machine learning. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology* (New York, New York, USA) *(UIST '10)*. Association for Computing Machinery, New York, NY, USA, 37–46. https://doi.org/10.1145/1866029.1866038

[42] Savvas Petridis, Ben Wedin, James Wexler, Aaron Donsbach, Mahima Pushkarna, Nitesh Goyal, Carrie J. Cai, and Michael Terry. 2023. ConstitutionMaker: Interactively Critiquing Large Language Models by Converting Feedback into Principles. arXiv:2310.15428 [cs.HC]

[43] PromptsRoyale. 2023. PromptsRoyale. https://www.promptsroyale.com.

[44] Charvi Rastogi, Marco Tulio Ribeiro, Nicholas King, and Saleema Amershi. 2023. Supporting Human-AI Collaboration in Auditing LLMs with LLMs. *arXiv preprint arXiv:2304.09991* (2023).

[45] Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. *arXiv preprint arXiv:2310.10501* (2023).

[46] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2023. Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting. *arXiv preprint arXiv:2310.11324* (2023).

[47] Shreya Shankar, Labib Fawaz, Karl Gyllstrom, and Aditya Parameswaran. 2023. Automatic and Precise Data Validation for Machine Learning. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2198–2207.

[48] Shreya Shankar, Haotian Li, Parth Asawa, Madelon Hulsebos, Yiming Lin, JD Zamfirescu-Pereira, Harrison Chase, Will Fu-Hinthorn, Aditya G Parameswaran, and Eugene Wu. 2024. SPADE: Synthesizing Assertions for Large Language Model Pipelines. *arXiv preprint arXiv:2401.03038* (2024).

[49] Patrice Simard, David Chickering, Aparna Lakshmiratan, Denis Charles, Léon Bottou, Carlos Garcia Jurado Suarez, David Grangier, Saleema Amershi, Johan Verwey, and Jina Suh. 2014. Ice: enabling non-experts to build models interactively for large-scale lopsided problems. *arXiv preprint arXiv:1409.4814* (2014).

[50] Arjun Singh, Sergey Karayev, Kevin Gutowski, and Pieter Abbeel. 2017. Gradescope: a fast, flexible, and fair system for scalable assessment of handwritten work. In *Proceedings of the fourth (2017) acm conference on learning@ scale*. 81–88.

[51] Arnav Singhvi, Manish Shetty, Shangyin Tan, Christopher Potts, Koushik Sen, Matei Zaharia, and Omar Khattab. 2023. DSPy Assertions: Computational Constraints for Self-Refining Language Model Pipelines. *arXiv preprint arXiv:2312.13382* (2023).

[52] Chanchal Suman, Saichethan Miriyala Reddy, Sriparna Saha, and Pushpak Bhattacharyya. 2021. Why pay more? A simple and efficient named entity recognition system for tweets. *Expert Systems with Applications* 167 (2021), 114101.

[53] Helena Vasconcelos, Matthew Jörke, Madeleine Grunde-McLaughlin, Tobias Gerstenberg, Michael S Bernstein, and Ranjay Krishna. 2023. Explanations can reduce overreliance on ai systems during decision-making. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW1 (2023), 1–38.

[54] Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023. Large Language Models are not Fair Evaluators. arXiv:2305.17926 [cs.CL]

[55] Ian Webster. 2023. promptfoo: Test your prompts. https://www.promptfoo.dev/.

[56] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. PromptChainer: Chaining Large Language Model Prompts through Visual Programming. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 359, 10 pages. https://doi.org/10.1145/3491101.3519729

[57] Wen-wai Yim, Yujuan Fu, Asma Ben Abacha, Neal Snider, Thomas Lin, and Meliha Yetisgen. 2023. Aci-bench: a novel ambient clinical intelligence dataset for benchmarking automatic visit note generation. *Scientific Data* 10, 1 (2023), 586.

[58] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. 2024. The Shift from Models to Compound AI Systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/.

[59] JD Zamfirescu-Pereira, Heather Wei, Amy Xiao, Kitty Gu, Grace Jung, Matthew G Lee, Bjoern Hartmann, and Qian Yang. 2023. Herding AI cats: Lessons from designing a chatbot by prompting GPT-3. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*. 2206–2220.

[60] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 437, 21 pages. https://doi.org/10.1145/3544548.3581388

[61] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems* 36 (2024).

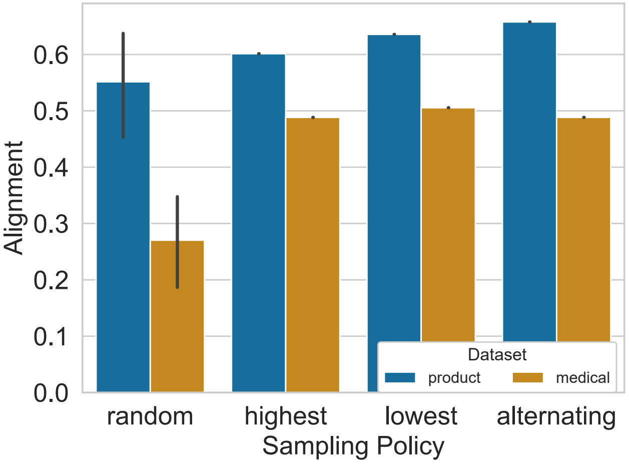


Figure 4: Alignments for assertion sets that result from different policies to sample grades from the user. Each policy was tested across 10 trials, with each involving a sample of 16 LLM outputs. Randomly sampling LLM outputs for grading introduces significant variance in alignment across the entire dataset.

A ALGORITHMS FOR SELECTING ASSERTIONS & ELICITING GRADES

A.1 Assertion Selectivity and Impact on LLM Output Quality Confidence

One way to establish confidence in whether an LLM output is problematic is to assess the selectivity, or pass rate, of assertions that fail it. Intuitively, assertions that frequently fail outputs (low selectivity) provide limited insight into output quality. For example, an assertion that trivially fails every output offers no discernment and has a selectivity of 0.

EvalGen leverages selectivity estimates of assertions to assign a confidence score to each LLM output, indicating the likelihood it is of poor quality. The rationale is straightforward: an output is more likely to be problematic if failed by assertions known for their high selectivity. Concretely, for a set of assertions F where each assertion $f \in F$ returns 1 for a pass and 0 for a fail, we calculate the confidence score for an LLM output e as follows:

$$\sigma(e) = \sum_{f \in F} \text{selectivity}(f) \times f(e)$$

The score σ is always non-negative. A score of 0 means no assertions have failed the output, indicating a higher likelihood of quality, while lower scores, resulting from failures by non-selective assertions, point to uncertainty or potential issues with the output.

A.2 Sampling Grades

Given that users may not want to grade so many outputs in the EvalGen interface, choosing which outputs for users to grade is crucial for aligning the system's evaluations with user expectations. Randomly selecting outputs without considering their predicted quality can lead to misalignment, especially if the selected samples aren't representative of the entire dataset. Prior work also underscores the importance of soliciting a representative graded sample of LLM outputs [3, 48].

Given σ scores as previously defined, we consider a number of strategies to sample outputs for grading:

- **Random:** Sample outputs at random (uniformly)
- **Highest:** Sample the outputs with the highest σ. This approach focuses on potentially problematic content.
- **Lowest:** Sample the outputs with the lowest σ, prioritizing outputs that don't fail any assertions or fail low-selectivity assertions.
- **Alternating:** Alternate between high and low σ, aiming for a diverse sample with both bad and good outputs.

In Section 5, we test these strategies against a random baseline on two different LLM pipelines. We employ an alternating sampling policy for the EvalGen user studies. We do not claim to have the best sampling policy; we chose an alternating policy with the hope that it would solicit a balanced sample of good and bad grades.

One may wonder why we do not list a policy that ranks the outputs by score and samples the middle for grading. While this might seem akin to seeking out uncertain cases—as is common in active learning—our scores represent the likelihood of outputs being poor. They do not differentiate between good and bad per se. Therefore, outputs with low scores may still vary widely in quality, reflecting our system's uncertainty.

A.3 Evaluation of Sampling Policy

We described in Appendix A.2 four options we considered to sample LLM outputs: random, highest, lowest, and alternating. Here, we compare EvalGen's sampling policy, *alternating*, to these other three baselines. For this experiment, we sampled 16 outputs to grade, but in practice the user can grade more or fewer. Using the same LLM pipelines as described in Section 5.1, to assess sampling variance, we conducted 10 trials for each of the four sampling policies—where, for each trial, we kept the same set of candidate assertion functions.

The findings, shown in Figure 4, reveal that the random sampling policy exhibits a large variance in alignment. This inconsistency could lead to user frustration, particularly if the effort spent in grading outputs results in assertion sets with unpredictable relevance to their specified criteria. The alternative sampling strategies, which weight the probability for an LLM output to be graded by the selectivity (i.e., pass rate) of assertions that fail it, consistently yielded higher alignment scores across the entire datasets than the random policy. Notably, while the alternating policy didn't consistently outperform, our results suggest that any non-random policy implemented in EvalGen may achieve satisfactory outcomes.

In this offline study, as shown in Figure 4, there's no variation in the outcomes of the non-random policies because they are deterministic. However, in real-world use, EvalGen updates its predictions as it receives new information, so there could be some differences in results over time. Initially, when users start grading outputs in EvalGen, they might effectively be grading random outputs for the first one or two outputs, as the σ scores update and stabilize.

B TASK PROMPTS

We prepared two prompts and corresponding datasets for tasks to present users (5.1). Both pipelines were adapted from prior work [22, 57] and correspond to medical record processing and a product description writing, respectively. The medical pipeline prompt is as follows:

```
You are extracting insights from some medical records. The records
contain a medical note and a dialogue between a doctor and a patient.
You need to extract values for the following: Chief complaint,
History of present illness, Physical examination, Symptoms
experienced by the patient, New medications prescribed or changed,
including dosages (N/A if not provided), and Follow-up instructions
(N/A if not provided). Your answer should not include any personal
identifiable information (PII) such as name, age, gender, or ID. Use
"the patient" instead of their name, for example. Return your answer
as a bullet list, where each bullet is formatted like ‘chief
complaint: xx.‘ If there is no value for the key, the value should be
‘N/A‘. Keep your response around 150 words (you may have to summarize
some extracted values to stay within the word limit).

{transcript}
```

And the product pipeline prompt is as follows:

```
You are an expert copywriter. You need to write an e-commerce product
description based on the product details and customer reviews. Your
description should be SEO-optimized. It should use an active voice
and include the product’s features, benefits, unique selling points
without overpromising, and a call to action for the buyer. Benefits
describe how product features will work for the buyer, addressing
exactly how the product will improve their lives. Clearly distinguish
between features (e.g., lightweight, USB-chargeable) and benefits
(e.g., convenience, nutritious drinks on-the-go). Don’t mention
weaknesses of the product or use generic or repetitive language.
Don’t make up review text or quotes. Don’t include any links. Don’t
cite the reviews too heavily. Divide your description into readable
chunks divided by relevant subheadings. Keep your description around
200 words, no more than 300, in Markdown format.

{document}
```