"We Need Structured Output": Towards User-centered Constraints on Large Language Model Output

Michael Xieyang Liu
Google Research
Pittsburgh, PA, USA
lxieyang@google.com

Frederick Liu
Google Research
Seattle, Washington, USA
frederickliu@google.com

Alexander J. Fiannaca
Google Research
Seattle, Washington, USA
afiannaca@google.com

Terry Koo
Google
Indiana, USA
terrykoo@google.com

Lucas Dixon
Google Research
Paris, France
ldixon@google.com

Michael Terry
Google Research
Cambridge, Massachusetts, USA
michaelterry@google.com

Carrie J. Cai
Google Research
Mountain View, California, USA
cjcai@google.com

ABSTRACT

Large language models can produce creative and diverse responses. However, to integrate them into current developer workflows, it is essential to constrain their outputs to follow specific formats or standards. In this work, we surveyed 51 experienced industry professionals to understand the range of scenarios and motivations driving the need for output constraints from a user-centered perspective. We identified 134 concrete use cases for constraints at two levels: low-level, which ensures the output adhere to a structured format and an appropriate length, and high-level, which requires the output to follow semantic and stylistic guidelines without hallucination. Critically, applying output constraints could not only streamline the currently repetitive process of developing, testing, and integrating LLM prompts for developers, but also enhance the user experience of LLM-powered features and applications. We conclude with a discussion on user preferences and needs towards articulating intended constraints for LLMs, alongside an initial design for a constraint prototyping tool.

CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in HCI**; *Interactive systems and tools.*

KEYWORDS

Large language models, Constrained generation, Survey

ACM Reference Format:

Michael Xieyang Liu, Frederick Liu, Alexander J. Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J. Cai. 2024. "We Need Structured Output": Towards User-centered Constraints on Large Language Model Output. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3613905.3650756

CHI EA '24, May 11–16, 2024, Honolulu, HI, USA

ACM ISBN 979-8-4007-0331-7/24/05
https://doi.org/10.1145/3613905.3650756

1 INTRODUCTION

Over the past few years, we have witnessed the extraordinary capability of Large Language Models (LLMs) to generate responses that are not only creative and diverse but also highly adaptable to various user needs [5, 7, 18, 21, 22, 29, 31, 32]. For example, researchers can prompt ChatGPT [25] to condense long articles into concise summaries for fast digestion; while video game developers can generate detailed character profiles with rich personality traits, backstories, and unique abilities on demand, simply by dynamically prompting an LLM with the game context and players' preferences.

As much as end-users appreciate the unbounded creativity of LLMs, recent field studies examining the development of LLM-powered applications have repeatedly demonstrated the necessity to *impose constraints* on LLM outputs [10, 30]. For instance, a user might require a summary of an article to be "strictly less than 20 words" to meet length constraints, or a generated video game character profile to be "a valid JSON that can be parsed by Python" for a development pipeline.

However, as evidenced by many recent NLP benchmarks and evaluations [16, 36, 42, 43], current state-of-the-art LLMs still lack the ability to *guarantee* that the generated output will invariably conform to user-defined constraints in the prompt (sometimes referred to as *controllability*). Although researchers have proposed various methods to improve *controllability*, such as supervised fine-tuning with specialized datasets [35] or controlled decoding strategies [1, 4, 24, 40], they tend to only focus on addressing a narrow range of constraints without taking into consideration the diverse usage scenarios and rationale that real-world developers and end-users encounter when prototyping and building practical LLM-powered functionalities and applications [12–14, 23].

In this work, we took the first step to systematically investigate the scenarios and motivations for applying output constraints from a *user-centered perspective*. Specifically, we sought to understand:

- **RQ1**: What real-world use cases would necessitate or benefit from being able to constrain LLM outputs?
- **RQ2**: What are the benefits and impacts of being able to apply constraints to LLM outputs?
- **RQ3**: How would users like to articulate their intended constraints to LLMs?

We investigated these research questions by distributing a survey to a broad population of industry professionals (software engineers, researchers, designers, project managers, etc.) who have experience building LLM-powered applications. Our analysis identified six primary categories of output constraints that users desire, each supported by detailed usage scenarios and illustrative examples, summarized in Table 1. In a nutshell, users not only need *low-level constraints*, which mandate the output to conform to a structured format and an appropriate length, but also desire *high-level constraints*, which involve semantic and stylistic guidelines that users would like the model output to adhere to without hallucinating. Notably, developers often have to write complex code to handle ill-formed LLM outputs, a chore that could be simplified or eliminated if LLMs could strictly follow output constraints. In addition, the ability to apply constraints could help ease the integration of LLMs with existing pipelines, meet UI and product specifications, and enhance user trust and experience with LLM-powered features. Moreover, we discovered that describing constraints in natural language (NL) within prompts is not always the preferred method of control for LLM users. Instead, they seek alternative mechanisms, such as using graphical user interfaces, or GUIs, to define and test constraints, which could offer greater flexibility and a heightened sense of assurance that the constraints will be strictly followed.

Informed by these results, we present an early design of ConstraintMaker, a prototype tool that enables LLM users to experiment, test, and apply constraints on the format of LLM outputs (see Figure 2 for more details), along with feedback and insights from preliminary user tests. Overall, this paper contributes:

- a comprehensive taxonomy summarizing both low-level and high-level output constraints desired by LLM users (Table 1), derived from 134 real-world use cases reported by our survey respondents (RQ1),
- an overview of both developer and user-facing benefits of being able to impose constraints on LLM outputs (RQ2),
- an exploration of LLM users' preferences for expressing constraints, whether via GUIs or natural language (RQ3),
- a initial design of the tool ConstraintMaker, which enables users to visually prototype LLM output constraints, accompanied by a discussion of preliminary user feedback.

2 SURVEY WITH INDUSTRY PROFESSIONALS

Methodology. To get a broad range of insights from people who have experience with prompting and building LLM-powered applications, we deployed an online survey to users of an internal prompt-based prototyping platform (similar to the OpenAI API Playground [28] and Google AI studio [11]) at a large technology company for two weeks during Fall 2023. We chose this platform because it was explicitly designed to lower the barriers to entry into LLM prompting and encourage a broader population (beyond machine learning professionals) to prototype and develop LLM-powered applications. We publicized the survey through the platform's user mailing list. We ran the survey for two weeks during Fall 2023. Participants were rewarded $10 USD for their participation. The survey was approved by our organization's IRB.

Instrument. Our survey started with questions concerning participants' background and technical proficiency, such as their job roles and their level of experience in designing and engineering LLM prompts. The survey subsequently investigated RQ1 and RQ2 by asking participants to report *three real-world use cases* in which they believe the implementation of constraints to LLM outputs is necessary or advantageous. For each use case, they were encouraged to detail the specific scenario where they would like to apply constraints, the type of constraint that they would prefer to implement, the degree of precision required in adhering to the constraint, and the importance of this constraint to their workflow. Finally, the survey investigated RQ3 by asking participants to reflect on scenarios where they would prefer *expressing constraints via a GUI* (sliders, buttons, etc.) over natural language (in prompts, etc.) and vice versa, as well as any alternative ways they would prefer to articulate constraints to LLMs. The GUI alternative draws inspiration from tools like the OpenAI Playground that allow users to adjust settings like temperature and top-k through buttons, toggles, and sliders. Detailed survey questions are documented in section A of the Appendix.

Results. 51 individuals responded to our survey. Over half of the respondents were software engineers (58.8%) across various product teams; others held a variety of roles like consultant & specialist (9.8%), analyst (7.8%), researcher (5.9%), UX engineer (5.9%), designer (3.9%), data scientist (3.9%), product manager (2.0%), and customer relationship manager (2.0%). All respondents had experience with prompt design and engineering, with the majority reported having extensive experience (62.7%). The targeted audience and use cases of their prompts were split approximately evenly among consumers and end-users (33.3%), downstream development teams (31.4%), or both (29.4%), with some created specifically for exploratory research & analysis (5.9%). Together, respondents contributed 134 unique use cases of output constraints. To analyze the contents of the open-ended responses, the first author read through all responses and used inductive analysis [34] to generate and refine themes for each research question with frequent discussions with the research team. We present the resulting themes for each research question in sections 3-5.

Limitations. Note that our findings largely capture the views of industry professionals, and may not encompass those of casual users who engage with LLMs conversationally [25]. Additionally, as our respondent sample is limited to a single corporation, the results described in the following sections may not be representative of the entire industry. Furthermore, our frequent use of open-ended questions might have negatively impacted the response rate. However, the saturation of novel findings and insights towards the end of the survey deployment suggests that we have successfully captured a comprehensive range of perspectives.

Category		%	Representative Examples	Precision
Low-level constraints				
Structured Output	Following standardized or custom format or template (e.g., markdown, HTML, DSL, bulleted list, etc.)	26.1%	*"Summarizing meeting notes into markdown format"* *"... the chatbot should quote dialogues, use special marks for scene description, etc."* *"I want the output to be in a specific format for a list of characteristics [of a movie] to then easily parse and train on."*	Exact
	Ensuring valid JSON object (with custom schema)	16.4%	*"... use the JSON output of the LLM to make an http request with that output as a payload."* *"I want to have the output [of the quiz] to be like a json with keys* `{"question": "...", "correct_answer": "...", "incorrect_answers": [...]}`*"*	Exact
Multiple Choice	Selecting from a predefined list of options	23.9%	*"Classifying student answers as right / wrong / uncertain...)"* *"While doing sentimental analysis, [...] restrict my output to few fixed set of classes like Positive, Negative, Neutral, Strongly Positive, etc."*	Exact
Length Constraints	Specifying the targeted length (e.g., # of characters / words, # of items in a list)	16.4%	*"... Make each summary bullet LESS THAN 40 words. If you generate a bullet point that is longer than 40 words, summarize and return a summary that is 40 words or less."* *"I want to limit the characters in the output to 100 so it is a valid YouTube Shorts title."*	Approx.
High-level constraints				
Semantic Constraints	Excluding specific terms, items, or actions	8.2%	*"Exclude PII (Personally Identifiable Information) and even some specific information..."* *"If asking for html, do not include the standard html boilerplate (doctype, meta charset, etc.) and instead only provide the meaningful, relevant, unique code."*	Exact
	Including or echoing specific terms or content	2.2%	*"... I want [the email] to include about thanking my manager and also talk about the location he is based on to help him feel relatable."* *"We want LLM to repeat input with some control tokens to indicate the mentions. e.g. input: 'Obama was born in 1961.',... , we want output to be '«Obama» was born in 1961.'"*	Exact
	Covering or staying on a certain topic or domain	2.2%	*"[The output of] a query about 'fall jackets' should be confined to clothing."* *"For ex. In India we have Jio and Airtel as 2 main telecom service provider. While building chat bot for Airtel, I would want the model to only respond [with] Airtel related topics."*	Exact
	Following certain (code) grammar / dialect / context	4.5%	*"While generating SQL,... restrict the output to a particular dialect and use the table / database name mentioned in the prompt."* *"... implement[ing] a voice assistant that calls specific methods with relevant arguments,... the output needs to be valid syntax and only call the methods specified in the context"*	Exact
Stylistic Constraints	Following certain style, tone, or persona	6.7%	*"... it is important that the [news] summary follow a style guide, ... for example, preference for active voice over passive voice."* *"Use straightforward language and avoid complex technical jargon..."*	Approx.
Preventing Hallucination	Staying grounded and truthful	8.2%	*"... we do not want [a summary of the doc] to include opinions or beliefs but only real facts."* *"If the LLM can't find a paper or peer-reviewed study, do not provide a hallucinated output."*	Exact
	Adhering to instructions (without improvising unrequested actions)	4.5%	*"For 'please annotate this method with debug statements', I'd like the output to ONLY include changes that add print statements... No other changes in syntax should be made. "* *"LLMs usually ends up including an advice associated to the summarised topic, advice we need to avoid so they are not part of the doc."*	Exact

Table 1: Taxonomy of the six primary categories of use cases of output constraints, derived from the 134 use cases that respondents submitted (RQ1). Totals add up to more than 100% since we placed some use cases into more than one category. The final column indicates whether the output is expected to match the constraint *exactly* or *approximately*, as agreed upon by the majority of respondents.

3 RQ1: REAL-WORLD USE CASES THAT NECESSITATE OUTPUT CONSTRAINTS

Table 1 presents a taxonomy of six primary categories of use cases that require output constraints, each with representative real-world examples and quotes submitted by our respondents.

These can be further divided into *low-level* and *high-level* constraints — low-level constraints ensure that model outputs adhere to a specific **structure** (e.g., JSON or markdown), instruct the model to perform pure **multiple choices** (e.g., sentiment classification), or dictate the **length** of the outputs; whereas high-level constraints enforce model outputs to respect **semantic** (e.g., must include or avoid specific terms or actions) or **stylistic** (e.g., follow certain style or tone) guidelines, while **preventing hallucination**.

Below, we discuss a number of interesting insights that emerged from our analysis of the use cases:

- **Going beyond valid JSON.** Note that recent advancements in instruction-tuning techniques have substantially improved the the chances of generating a valid JSON object upon user request

Developer-facing Benefits	User-facing Benefits
Increasing prompt-based development efficiency (§4.1)	**Satisfying product and UI requirements (§4.3)**
Speeding up prompt design / engineering (less trial and error)	Fitting output into UI presets with size bounds
Reducing or eliminating ad-hoc parsing and plumbing logic	Ensuring consistency of output length and format
Saving the cost of requesting multiple candidates	Complying with product and platform requirements
Streamlining integration with downstream processes and workflows (§4.2)	**Improving user experience and trust (§4.4)**
Ensuring successful code execution	Eliminating safety and privacy concerns
Improving the quality of training data synthesis	Improving user trust and confidence
Canonizing output format across models	Increasing customer satisfaction and adoption

Table 2: Respondents' perceived benefits of having the ability to apply constraints to LLM output (RQ2).

[27, 29]. Nonetheless, our survey respondents believed that this was not enough and **desired to have more precise control over the JSON schema (i.e., key/value pairs)**. One respondent stated their expectation as follows: "*I expect the quiz [that the LLM makes given a few passages provided below] to have 1 correct answer and 3 incorrect ones. I want to have the output to be like a json with keys* `{"question": "...", "correct_answer": "...", "incorrect_answers": [...]}`." It is also worth mentioning that some respondents found that "*few-shot prompts*" — demonstrating the desired key/value pairs with several examples — tend to work "*fairly well*". However, they concurred that having a formal guarantee of JSON schema would be greatly appreciated (see section 4.1 for their detailed rationales).

- **Giving an answer without extra conversational prose.** When asking an LLM to perform data classification or labeling, such as "*[classifying sentiments as] Positive, Negative, Neutral, etc.,*" respondents typically expect the model to **only output the classification result** (e.g. "`Positive.`") **without a trailing "explanation"** (e.g., "`Positive, since it referred to the movie as a 'timeless masterpiece'...`"), as the addition of explanation could potentially confuse the downstream parsing logic. This indicates a potential misalignment between a common training objective — where LLMs are often tailored to be conversational and provide rich details [2, 17, 33] — and certain specialized downstream use cases where software developers need LLMs to be succinct. Such use cases necessitate output constraints that are independent of the prompt that would help adapt a general-purpose model to meet specific user requirements.
- **Conditioning the output on the input, but don't "improvise!"** One thread of high-level constraints places emphasis on directing the model to **condition its output on specific content from the input**. For example, the model's response should semantically remain "*in the same ballpark*" as "*the user's original query*" — "*[the output of] a query about 'fall jackets' should be confined to clothing.*" A particular instance of this is for LLMs to *echo* segments of the input in their output, occasionally with slight alterations. For example, "*we want LLM to repeat input with some control tokens to indicate the mentions. e.g. input: 'Obama was born in 1961.',... , we want output to be '«Obama» was born in 1961.'*" Nevertheless, respondents underscored the importance of the model **not improvising beyond its input and instructions**. For example, one respondent instructed an LLM to "*annotate a method with debug statement,*" anticipating the output would "*ONLY include changes that add print statements to the method.*" However, the LLM would frequently introduce additional "*changes in syntax*" that were unwarranted.

4 RQ2: BENEFITS OF APPLYING CONSTRAINTS TO LLM OUTPUTS

Beyond the aforementioned use cases, our survey respondents reported a range of benefits that the ability of constraining LLM output could offer. These include both *developer-facing* benefits, like increasing prompt-based development efficiency and streamlining integration with downstream processes and workflows, as well as *user-facing* benefits, like satisfying product and UI requirements and improving user experience and trust of LLMs (Table 2). Here are the most salient responses:

4.1 Increasing Prompt-based Development Efficiency

First and foremost, being able to constrain LLM outputs can significantly increase the efficiency of prompt-based engineering and development by reducing the trial and error currently needed to manage LLM unpredictability. Developers noted that the process of "*defining the [output] format*" alone is "*time-consuming,*" often requiring extensive prompt testing to identify the most effective one (consistent with what previous research has found [30, 41]). Additionally, they often need to "*request multiple responses*" and "*iterating through them until find[ing] a valid one.*" Therefore, being able to deterministically constrain the output format could not only save developers as much as "*dozens of hours of work per week*" spent on iterative prompt testing, but also reduce overall LLM inference costs and latency.

Another common practice that respondents reported is building complex infrastructure to post-process LLM outputs, sometimes referred to as "*massaging [the output] after receiving.*" For example, developers oftentimes had to "*chase down 'free radicals' when writing error handling functions,*" and felt necessary to include "*custom logic*" for matching and filtering, along with "*further verification.*" Thus, setting constraints before LLM generation may be the key to reducing such "*ad-hoc plumbing code*" post-generation, simplifying "*maintenance,*" and enhancing the overall "*developer experience.*" As

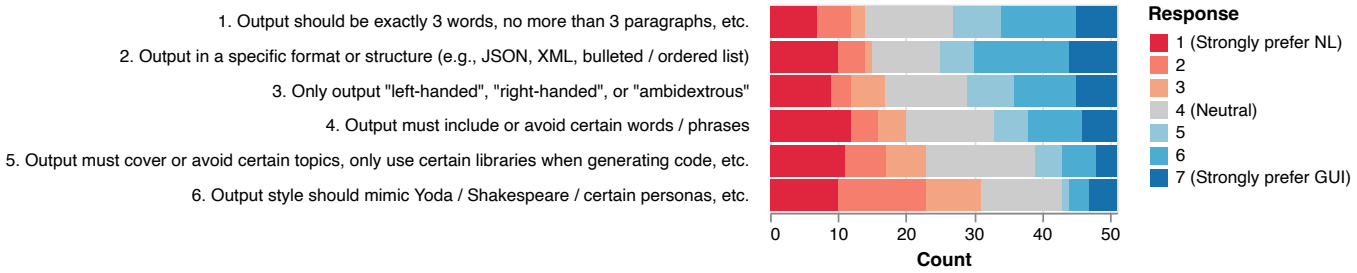

Figure 1: Respondents' preferences towards specifying output constraints either through natural language or GUI (RQ3). Participants were asked to score each question (left) on a 7-point Likert scale from "1 (Strongly prefer NL)" to "7 (Strongly prefer GUI)."

one respondent vividly described: "*it's a much nicer experience if it (formatting the output in bullets) 'just works' without having to implement additional infra...*"

4.2 Integrating with Downstream Processes and Workflows

Because LLMs are often used as sub-components in larger pipelines, respondents emphasized that guaranteed constraints are critical to ensuring that the output of their work is compatible with downstream processes, such as downstream modules that expect a specific format or functional code as input. Specifically for code generation, they highlighted the necessity of constraining the output to ensure "*executable*" code that adheres to only "*methods specified in the context*" and avoids errors, such as hallucinating "*unsupported operators*" or "*SQL ... in a different dialect.*" Note that while the "function calling" features in the latest LLMs [8, 26] can "select" functions to call from a predefined list, users still have to implement these functions correctly by themselves.

Many studies indicate that LLMs are highly effective for creating synthetic datasets for AI training [9, 15, 38], and our survey respondents postulated that being able to impose constraints on LLMs could improve the datasets' quality and integrity. For instance, one respondent wished that model-generated movie data would "*not say a movie's name when it describes its plot,*" as they were going to train using this data for a "*predictive model of the movie itself.*" Any breach of such constraints could render the data "*unusable.*"

Furthermore, given the industry trend of continuously migrating to newer, more cost-effective models, respondents highlighted the importance of "*canonizing*" constraints across models to avoid extra prompt-engineering after migration (e.g., "*if I switch model, I get the formatting immediately*"). This suggests that it could be more advantageous for models to accept output constraints independent of the prompt, which should now solely contain task instructions.

4.3 Satisfying UI and Product Requirements

Respondents stressed that it is essential to constrain LLM output to meet UI and product specifications, particularly when such output will be presented to end users, directly or indirectly. A common case is to incorporate LLM-generated content into UI elements that "*cannot exceed certain bounds*", necessitating stringent length constraints. Content that doesn't "*fit within the UI*" usually gets "*thrown away*" all together, a concern likely to be more pronounced on mobile devices with limited screen real estate [6, 20]. Maintaining the consistency of output length and format was also considered important, as "*too much variability in the generated text can be overwhelming to the user and clutter the UI.*" Moreover, being able to constrain length can help LLMs comply with specific platform character restrictions, like tweets capped at 280 characters or YouTube Shorts titles limited to 100 characters.

4.4 Improving User Experience, Trust, and Adoption

Finally, respondents suggested that developing LLM-powered user experiences requires constraint mechanisms to mitigate hallucinations, foster user trust, and ultimately drive "*user adoption.*" One prominent aspect is to reduce safety and privacy concerns, for instance, by preventing LLMs from "*repeat[ing] existing or hallucinat[ing] PII (personally identifiable information).*" In addition, respondents expressed a desire to ensure user trust and confidence of LLM-powered tools and systems, arguing that, for example, "*hallucinations in dates are easy to identify*" and, in general, "*users won't invest more time into tools that aren't accurate.*"

5 HOW TO ARTICULATE OUTPUT CONSTRAINTS TO LLMS

Fig. 1 shows distributions of respondents' preferences towards specifying output constraints either through GUI or natural language. An overarching observation is that respondents preferred **using GUI to specify low-level constraints** and **natural language to express high-level constraints**. We discuss their detailed rationale below:

5.1 The case for GUI: A Quick, Reliable, and Flexible Way of Prototyping Constraints

First and foremost, respondents considered GUIs particularly effective for defining "*hard requirements,*" providing more reliable results, and reducing ambiguity compared to natural language instructions. For example, one argued that choosing "boolean" as the output type via a GUI felt much more likely to be "*honoured*" compared to "*typ[ing] that I want a Yes / No response [...] in a prompt.*" Another claimed that "*flagging a 'JSON' button*" provides a much better user experience than "*typing 'output as JSON' across multiple prompts.*" In addition, respondents preferred using GUI when the intended constraint is "*objective*" and "*quantifiable*", such as "*use*

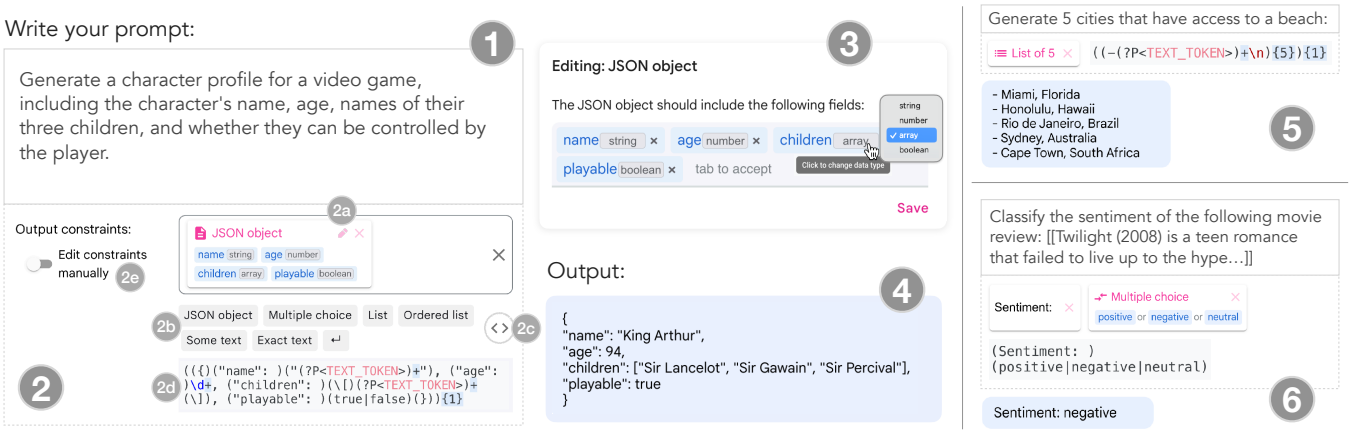

Figure 2: ConstraintMaker's user interfaces (1-4) & use cases (5-6). After writing the prompt (1), users can easily specify output constraints using a graphical user interface (2 & 3) provided by ConstraintMaker, and the resulting output (4) is guaranteed to follow the constraints. Additional details of this process is discussed in section 6.

only items x,y,z," or "*a JSON with certain fields specified.*" Moreover, respondents found GUI to be more flexible for rapid prototyping and experimentation (e.g., "*when I want to play around with different numbers, moving a slider around seems easier than typing*"). Finally, for novice LLM users, the range of choices afforded by a GUI constraint can help clarify the model's capabilities and limitations, "*making the model seems less like a black box.*" One respondent drew from their experience working with text-to-image models to underscore this point: "*by seeing "Illustration" as a possible output style [among others like "Photo realistic" or "Cartoon"], I became aware of [the model's] capabilities.*"

5.2 The Case for NL: More Intuitive and Expressive for Complex Constraints

Respondents found natural language easier for specifying complex constraints than GUIs, especially for extended background contexts or numerous choices that wouldn't reasonably fit into a GUI. Natural language was also preferred for expressing vague, nuanced, or open-ended constraints, like "*don't include offensive words*" or "*respond in a cheerful manner.*" At a high level, respondents emphasized that natural language provides a more natural, familiar, and expressive way to communicate (potentially multiple) complex constraints, and, "*trying to figure out how to use a GUI might be more tedious.*"

Additionally, some respondents noted that, despite their preference for using GUIs to define constraints from time to time, they ultimately have to use natural language prompts due to API limitations. Moreover, some wished to reference "*external resources*" in constraints that are not feasible to directly include in prompts (e.g., "*a large database / vocabulary*"). These suggest that a dedicated "output-constraints" API field for specifying constraints could be advantageous, potentially through the use of a formal language or notation.

6 THE CONSTRAINTMAKER TOOL

Informed by the survey results, we developed a web-based GUI, ConstraintMaker (Fig. 2), that enables LLM users to prototype, test, and apply constraints on the *format* of LLM outputs. With ConstraintMaker, users can specify different types of output constraints by simply selecting from the list of available *constraint primitives* (Fig. 2-2b). If needed, users can click the pencil icon (Fig. 2-2a) to further edit the details of a constraint primitive, such as specifying the schema of a JSON object (Fig. 2-3). Users also have the flexibility to mix and match multiple constraint primitives together (e.g., Fig. 2-6) to form more complex constraints. Currently, based on users' needs and priorities identified by the survey, ConstraintMaker initially supports `JSON object`, `Multiple choice`, `List`, `Ordered list`, and `Some text` as primitives (Fig. 2-2b), where `Some text` asks the LLM to generate freely as it normally would.

Under the hood, we used a GPT-3.5-class LLM. We additionally implemented a finite-state machine-based decoding technique akin to that outlined in [39], ensuring the language model outputs strictly adhere to formats defined by a specialized regular expression (henceforth, "regex"). In fact, ConstraintMaker automatically converts a GUI-defined constraint into a regex (Fig. 2-2d), which the LLM observes during generation (Fig. 2-4).

6.1 Iterative Design and User Feedback

To explore the usability and usefulness of ConstraintMaker, we conducted a series of informal (around 30 minute each) user tests with five participants who self-identified as experts in prompting LLMs, as well as self-experimentation among the authors. We used the feedback from these sessions to iteratively refine the design of ConstraintMaker. We present some interesting findings and reflections below:

6.1.1 ConstraintMaker enables an intuitive separation of concerns. With ConstraintMaker, one can now specify "*the tasks they want the model to perform*" *separate from* "*the expected format of the output,*" an approach participants considered more intuitive and effective in steering LLMs to consistently achieve desired results compared to traditional prompting. Additionally, participants envisioned the possibility of reusing constraints across various prompts, which could reduce the effort of crafting new constraints for similar tasks and eliminate the need for prompt engineering post model migration.

6.1.2 Constraint-prototyping GUI needs to cater to both developers and non-developers. On the one hand, for non-technical users interested in experimenting with constraints, we noticed that the visible regex alongside the constraint primitive GUI was somewhat distracting. To address this, we added a feature that allows the regex to be toggled as hidden via the "< >" button (Fig. 2-2c). On the other hand, for more advanced users (e.g., developers), we observed a frequent need to make fine-grained adjustments to the underlying regex after creating an initial draft with the ConstraintMaker GUI (e.g., changing the "bullet" of a "bulleted list" from the default "– `[...]`" (Fig. 2-5) to "* `[...]`"). As a result, we enabled direct manipulation of the regex by toggling on "Edit constraints manually" (Fig. 2-2e).

6.1.3 "Inserting" words among constraints. For example, one participant asked in the prompt for the LLM to first write a paragraph describing a short story, followed by a list of suggestions on how to improve the story. In situations like this, they found that embedding specific words into the constraints, such as "`Short story: Some text`" followed by "`Suggestions: List`", yielded better-quality results than simply using `Some text` followed by `List` alone. Therefore, we introduced the `Exact text` GUI primitive, enabling the LLM to *insert* user-prescribed text into its output.

6.1.4 Automatically inferring constraints based on prompts. One interesting feature request for ConstraintMaker is the ability to automatically infer constraints from user-written prompts, similar to previous intelligent prediction or auto-completion systems and tools [3, 19, 37]. For instance, for a prompt shown in Fig. 2-1, ConstraintMaker could *proactively suggest* to the users if they'd like to constrain the model output to a JSON object with specific fields. This feature would be appealing, given the current somewhat cumbersome process of manually creating and modifying constraints from scratch. Similar to code auto-completion, participants suggested that constraint auto-completion could streamline the overall experience of defining constraints. Additionally, automatic constraint suggestions could serve as learning opportunities for novice users to become familiar with the range of possibilities that ConstraintMaker affords, which would be particularly useful in future versions where the tool might support a wider selection of constraint primitives. Finally, proactively suggesting constraints could promote a "constraint mindset." This encourages users to always consider the output format before deploying a prompt, leading to more rigorous and controllable prompt engineering, much like conventional software development.

7 CONCLUSION

In this work, we introduced a user-centered taxonomy of real-world scenarios, benefits, and preferred methods for applying constraints on LLM outputs, offering both a theoretical framework and practical insights into user requirements and preferences. In addition, we presented ConstraintMaker, an early GUI-based tool that enables users to prototype and test output constraints iteratively. Our results shed light on the future of more controllable, customizable, and user-friendly interfaces for human-LLM interactions.

REFERENCES

[1] 2023. guidance-ai/guidance. https://github.com/guidance-ai/guidance original-date: 2022-11-10T18:21:45Z.

[2] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. 2022. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback. https://doi.org/10.48550/arXiv.2204.05862 arXiv:2204.05862 [cs].

[3] Ziv Bar-Yossef and Naama Kraus. 2011. Context-sensitive query auto-completion. In *Proceedings of the 20th international conference on World wide web (WWW '11)*. Association for Computing Machinery, New York, NY, USA, 107–116. https://doi.org/10.1145/1963405.1963424

[4] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2023. Prompting Is Programming: A Query Language for Large Language Models. *Proceedings of the ACM on Programming Languages* 7, PLDI (June 2023), 186:1946–186:1969. https://doi.org/10.1145/3591300

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. https://doi.org/10.48550/arXiv.2005.14165 arXiv:2005.14165 [cs].

[6] Joseph Chee Chang, Nathan Hahn, and Aniket Kittur. 2016. Supporting Mobile Sensemaking Through Intentionally Uncertain Highlighting. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 61–68. https://doi.org/10.1145/2984511.2984538

[7] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. https://doi.org/10.48550/arXiv.2204.02311 arXiv:2204.02311 [cs].

[8] Google Cloud. 2023. Function calling | Vertex AI. https://cloud.google.com/vertex-ai/docs/generative-ai/multimodal/function-calling

[9] Ronen Eldan and Yuanzhi Li. 2023. TinyStories: How Small Can Language Models Be and Still Speak Coherent English? https://doi.org/10.48550/arXiv.2305.07759 arXiv:2305.07759 [cs].

[10] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. https://doi.org/10.48550/arXiv.2310.03533 arXiv:2310.03533 [cs].

[11] Google. 2023. Google AI Studio quickstart. https://ai.google.dev/tutorials/ai-studio_quickstart

[12] Chris Hokamp and Qun Liu. 2017. Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, Vancouver, Canada, 1535–1546. https://doi.org/10.18653/v1/P17-1141

[13] J. Edward Hu, Huda Khayrallah, Ryan Culkin, Patrick Xia, Tongfei Chen, Matt Post, and Benjamin Van Durme. 2019. Improved Lexically Constrained Decoding for Translation and Monolingual Rewriting. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 839–850. https://doi.org/10.18653/v1/N19-1090

[14] 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 (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3491101.3503564

[15] Martin Josifoski, Marija Sakota, Maxime Peyrard, and Robert West. 2023. Exploiting Asymmetry for Synthetic Training Data Generation: SynthIE and the Case of Information Extraction. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 1555–1574. https://doi.org/10.18653/v1/2023.emnlp-main.96

[16] 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. https://doi.org/10.48550/arXiv.2402.10524 arXiv:2402.10524 [cs].

[17] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic Evaluation of Language Models. https://doi.org/10.48550/arXiv.2211.09110 arXiv:2211.09110 [cs].

[18] Michael Xieyang Liu. 2023. *Tool Support for Knowledge Foraging, Structuring, and Transfer during Online Sensemaking*. Ph. D. Dissertation. Carnegie Mellon University. http://reports-archive.adm.cs.cmu.edu/anon/anon/usr0/ftp/usr/ftp/hcii/abstracts/23-105.html

[19] Michael Xieyang Liu, Aniket Kittur, and Brad A. Myers. 2022. Crystalline: Lowering the Cost for Developers to Collect and Organize Information for Decision Making. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3491102.3501968 event-place: New Orleans, LA, USA.

[20] Michael Xieyang Liu, Andrew Kuznetsov, Yongsung Kim, Joseph Chee Chang, Aniket Kittur, and Brad A. Myers. 2022. Wigglite: Low-cost Information Collection and Triage. In *The 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3526113.3545661

[21] Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Benjamin Zorn, Jack Williams, Neil Toronto, and Andrew D. Gordon. 2023. "What It Wants Me To Say": Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–31. https://doi.org/10.1145/3544548.3580817

[22] Michael Xieyang Liu, Tongshuang Wu, Tianying Chen, Franklin Mingzhe Li, Aniket Kittur, and Brad A. Myers. 2023. Selenite: Scaffolding Online Sensemaking with Comprehensive Overviews Elicited from Large Language Models. https://doi.org/10.48550/arXiv.2310.02161

[23] Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A. Smith, and Yejin Choi. 2022. NeuroLogic A*esque Decoding: Constrained Text Generation with Lookahead Heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (Eds.). Association for Computational Linguistics, Seattle, United States, 780–799. https://doi.org/10.18653/v1/2022.naacl-main.57

[24] Sidharth Mudgal, Jong Lee, Harish Ganapathy, YaGuang Li, Tao Wang, Yanping Huang, Zhifeng Chen, Heng-Tze Cheng, Michael Collins, Trevor Strohman, Jilin Chen, Alex Beutel, and Ahmad Beirami. 2023. Controlled Decoding from Language Models. https://doi.org/10.48550/arXiv.2310.17022 arXiv:2310.17022 [cs].

[25] OpenAI. 2023. ChatGPT. https://chat.openai.com

[26] OpenAI. 2023. Function calling | OpenAI Platform. https://platform.openai.com/docs/guides/function-calling

[27] OpenAI. 2023. JSON mode - Text generation. https://platform.openai.com/docs/guides/text-generation/json-mode

[28] OpenAI. 2023. Playground - OpenAI API. https://platform.openai.com/playground

[29] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. https://doi.org/10.48550/arXiv.2203.02155 arXiv:2203.02155 [cs].

[30] Chris Parnin, Gustavo Soares, Rahul Pandita, Sumit Gulwani, Jessica Rich, and Austin Z. Henley. 2023. Building Your Own Product Copilot: Challenges, Opportunities, and Needs. https://doi.org/10.48550/arXiv.2312.14231 arXiv:2312.14231 [cs].

[31] Savvas Petridis, Michael Terry, and Carrie Jun Cai. 2023. PromptInfuser: Bringing User Interface Mock-ups to Life with Large Language Models. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems (CHI EA '23)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3544549.3585628

[32] 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. https://doi.org/10.48550/arXiv.2310.15428 arXiv:2310.15428 [cs].

[33] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 3008–3021. https://proceedings.neurips.cc/paper/2020/hash/1f89885d556929e98d3ef9b86448f951-Abstract.html

[34] Anselm Strauss and Juliet Corbin. 1990. *Basics of qualitative research*. Sage publications.

[35] Jiao Sun, Xuezhe Ma, and Nanyun Peng. 2021. AESOP: Paraphrase Generation with Adaptive Syntactic Control. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 5176–5189. https://doi.org/10.18653/v1/2021.emnlp-main.420

[36] Jiao Sun, Yufei Tian, Wangchunshu Zhou, Nan Xu, Qian Hu, Rahul Gupta, John Wieting, Nanyun Peng, and Xuezhe Ma. 2023. Evaluating Large Language Models on Controlled Generation Tasks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 3155–3168. https://doi.org/10.18653/v1/2023.emnlp-main.190

[37] Jialiang Tan, Yu Chen, and Shuyin Jiao. 2023. Visual Studio Code in Introductory Computer Science Course: An Experience Report. https://doi.org/10.48550/arXiv.2303.10174 arXiv:2303.10174 [cs].

[38] Vijay Viswanathan, Chenyang Zhao, Amanda Bertsch, Tongshuang Wu, and Graham Neubig. 2023. Prompt2Model: Generating Deployable Models from Natural Language Instructions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Yansong Feng and Els Lefever (Eds.). Association for Computational Linguistics, Singapore, 413–421. https://doi.org/10.18653/v1/2023.emnlp-demo.38

[39] Brandon T. Willard and Rémi Louf. 2023. Efficient Guided Generation for Large Language Models. https://arxiv.org/abs/2307.09702v4

[40] Nan Xu, Chunting Zhou, Asli Celikyilmaz, and Xuezhe Ma. 2023. Look-back Decoding for Open-Ended Text Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 1039–1050. https://doi.org/10.18653/v1/2023.emnlp-main.66

[41] 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 (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–21. https://doi.org/10.1145/3544548.3581388

[42] Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. 2023. Evaluating Large Language Models at Evaluating Instruction Following. https://doi.org/10.48550/arXiv.2310.07641 arXiv:2310.07641 [cs].

[43] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-Following Evaluation for Large Language Models. https://doi.org/10.48550/arXiv.2311.07911 arXiv:2311.07911 [cs].

A THE SURVEY INSTRUMENT

In this section, we detail the design of our survey. The survey starts with questions about background and self-reported technical proficiency:

- What best describes your job role: software Engineer; research scientist; UX designer; UX researcher; product manager; technical writer; other (open-ended)
- To what extent have you designed LLM prompts: a) I have "chatted with" chatbots like Bard / ChatGPT as a user; b) I've tried making a prompt once or twice just to check it out, but haven't done much prompt design / engineering; c) I have some experience doing prompt design / engineering on at least three LLM prompts; d) I have done extensive prompt design / engineering to accomplish desired functionality. Only those participants who selected either option c) or d) were given the opportunity to continue with the remainder of the survey. This approach is specifically designed to exclude "casual" LLM users.
- I *primarily* design prompts with the intent that they will be used by: a) consumers / end-users (e.g. a recipe idea generator); b) downstream development teams (e.g. captioning, classifiers); c) both, I split my time about evenly between the two; d) other audience or use cases (open response).

The survey then asked participants to report *three real-world use cases* where they would like to constrain LLM outputs. For each use case, participants were asked:

- How would like to be able to constrain the model output (open response);
- Provide a concrete example where it would be useful to have this constraint (open response);
- How precisely do you need this constraint to be followed: a) exact match; b) approximate match and why (optional open response);
- How important is this constraint to your workflow (5-point Likert scale from "it's a nice to have, but my current workarounds are fine" to "it's essential to my workflow") and why (optional open response).

The survey then asked participants to reflect through open response on scenarios where they would prefer *expressing constraints via GUI* (sliders, buttons, etc.) over natural language (in prompts, etc.) and vice versa, as well as any alternative ways they would prefer to express constraints. To facilitate the reflection, the survey additionally asked participants to rate their level of preference in:

- Output should be exactly 3 words, no more than 3 paragraphs, etc.
- Output in a specific format or structure (e.g., JSON, XML, bulleted / ordered list)
- Only output "left-handed", "right-handed", or "ambidextrous"
- Output must include or avoid certain words / phrases
- Output must cover or avoid certain topics, only use certain libraries when generating code, etc.
- Output style should mimic Yoda / Shakespeare / certain personas, etc.

Each question presented a 7-point Likert scale from "strongly prefer natural language" to "strongly prefer GUI."