

PromptMaker: Prompt-based Prototyping with Large Language Models

Ellen Jiang*
Google Research, PAIR team
USA
ellenj@google.com

Alejandra Molina
Google Research, PAIR team
USA
alemolinata@google.com

Kristen Olson*
Google Research, PAIR team
USA
kolson@google.com

Aaron Donsbach
Google Research, PAIR team
USA
donsbach@google.com

Edwin Toh*
Google Research, PAIR team
USA
edwintoh@google.com

Michael Terry
Google Research, PAIR team
USA
michaelterry@google.com

Carrie J. Cai
Google Research, PAIR team
USA
cjcai@google.com

ABSTRACT

Prototyping is notoriously difficult to do with machine learning (ML), but recent advances in large language models may lower the barriers to people prototyping with ML, through the use of natural language prompts. This case study reports on the real-world experiences of industry professionals (e.g. designers, program managers, front-end developers) prototyping new ML-powered feature ideas via **prompt-based prototyping**. Through interviews with eleven practitioners during a three-week sprint and a workshop, we find that prompt-based prototyping reduced barriers of access by substantially broadening who can prototype with ML, sped up the prototyping process, and grounded communication between collaborators. Yet, it also introduced new challenges, such as the need to reverse-engineer prompt designs, source example data, and debug and evaluate prompt effectiveness. Taken together, this case study provides important implications that lay the groundwork toward a new future of prototyping with ML.

ACM Reference Format:

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 *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI '22 Extended Abstracts)*, April 29-May 5, 2022, New Orleans, LA, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3491101.3503564>

1 INTRODUCTION

Prototyping is a fundamental step in the design process, yet quick proofs of concept are notoriously difficult to achieve with machine

*equal contribution

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

CHI '22 Extended Abstracts, April 29-May 5, 2022, New Orleans, LA, USA

© 2022 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-9156-6/22/04.

<https://doi.org/10.1145/3491101.3503564>

learning [10, 16, 17]. Prototyping with AI can be uniquely challenging, due to the difficulty of: determining a priori what an AI can or cannot do, finding skilled AI technical collaborators, and envisioning AI uses that do not yet exist [17]. Recently, advances in Large Language Models (LLM) like GPT-3 [3] have lowered the barrier to authoring new machine learning functionality on-the-fly by allowing users to feed natural language prompts to an LLM, a practice known as “prompt programming” [1, 3]. For example, given a general purpose LLM, a user could customize it to act like a specialized English-to-French translation engine, by giving the LLM a natural language prompt containing pairs of English and French examples: “English: how are you? French: comment allez-vous? English: goodbye! French: au revoir! English: hello! French: ”. Given this prompt, the LLM is likely to output the French translation: “bonjour!”

While the translation example above serves as a toy example, in practice there are numerous *types* of ML functionality a designer may want to prototype, ranging from innovative new application ideas (e.g., given the weather, generate clothing advice; given favorite ingredients, generate new recipes), to new machine learning models that would form critical components of envisioned applications (e.g., given noisy audio captions, remove dis-fluencies; given a peer review, rewrite it to be more polite). There may also be multiple *reasons* why non-ML practitioners may desire to prototype quick proofs-of-concept of functional ML features, such as for de-risking and testing out the feasibility of half-baked ideas, surfacing edge-cases or realistic AI error modes early on, or grounding communication with ML collaborators via a tangible, functional prototype – currently difficult to do with traditional prototyping techniques alone (e.g. wizard-of-oz [8] and wire-frames) [17]. The relative ease with which LLMs can be customized to produce a wide range of functionality suggests that it may be useful as a flexible *AI design material*, enabling non-ML experts to create new ML functionality through natural language alone. In the context of prototyping, we call this natural language prompting process **prompt-based prototyping**.

This case study reports on the real-world experience of industry teams prototyping new machine-learning-powered concepts and

applications via prompt programming of a state-of-the-art LLM. Through interviews with eight industry professionals (designers, program managers, and content strategists) during a three-week LLM prototyping sprint, as well as a workshop with three front-end engineers who had been developing prompt-based prototypes for several months, we describe how prompt programming transformed the prototyping process by:

- (1) Democratizing who can author with ML (by reducing reliance on ML experts by non-ML experts),
- (2) Substantially reducing prototyping time,
- (3) Increasing early intuition about what may or may not be possible, and
- (4) Acting as a “social glue” between cross-functional collaborators.

Furthermore, we uncover four unique challenges users faced during prompt-based prototyping:

- (1) The strategic know-how required to reverse engineer a design goal into a prompt,
- (2) The effort required to source examples for a prompt,
- (3) The inability to systematically and reliably debug prompts, and
- (4) The difficulty of evaluating whether a prompt is improving.

Taken together, this case study illustrates a foundational shift in who can easily prototype with machine learning, the time required to develop a functional AI prototype, what it means to prototype ML in this new way, and what people need to better support this process, paving the way towards a new future of prototyping with ML.

2 BACKGROUND: PROMPT PROGRAMMING

At the most basic level, a generative language model is designed to continue its input with plausible output (e.g., given a prompt “I went to the”, it might auto-complete this phrase with “the store and bought some apples”). However, when pre-trained on billions of samples from the Internet, recent LLMs like GPT-3 [3] can now adapt at run time to perform new tasks defined by the user. For example, purely using natural language prompts, one could adapt an LLM to translate between languages, write fiction, or generate source code. The design of the prompt to produce a particular type of model output is typically referred to as *prompt programming* [1, 3].

A number of strategies have been developed to design prompts so that they are more likely to produce the desired outcome [14]. The most common strategies can be categorized as few-shot and zero-shot prompting strategies. Few-shot prompts provide sets of examples (e.g., “English: welcome French: bienvenue English: hello French: ”), while zero-shot prompts directly state what ought to happen by priming the model with context, but without any examples (e.g., “The translation of “hello” into French is: ”).

3 CASE STUDIES

This case study reports on the real-world experience of industry professionals prototyping new machine learning-powered features and applications, through prompt programming of an LLM.

We draw our analysis from two groups. The first group participated in a three-week prototyping sprint in August 2021. The purpose of the sprint was to prototype product and feature ideas with large language models. In the first week, participants were introduced to LLMs at a high level, and brainstormed ideas to prototype. In the second and third week, participants were provided with a tool (described below) for prompt-based programming, as well as an online tutorial showing them the basics of prompting. Following a typical design process [4, 9], participants were encouraged to explore multiple ideas initially, and to narrow them down to one or two ideas per team by the start of the third week. Interviews were conducted once at the start of week two, and once at the end of week three, to capture both the early and later phases of prompt-based prototyping. During interviews, we asked participants to describe their existing ML-prototyping processes (if any), how they used the prompt programming tool during the sprint, and how this affected their prototyping practices. The eight participants consisted of designers (6), a content strategist (1), and a program manager (1), working in product teams ranging from conversational agents, geolocation, and games, to R&D and new product innovation.

The second group participated in a one-hour participatory design workshop in March 2021. They consisted of three front-end software engineers who had been prototyping a web-based tool to support creative writers, creating ML-powered features by prompt programming an LLM. During the workshop, participants were asked to reflect on the existing opportunities and challenges of prototyping with LLMs, and to sketch ideas for overcoming those challenges. Both the sprint participants and workshop participants were industry professionals at our institution. Both the sprint interviews and workshop think-alouds were transcribed before conducting thematic analysis.

4 THE PROMPTMAKER TOOL

We developed a user interface, PromptMaker, that enables users to prototype new ML functionality by making and testing natural language prompts. We provided PromptMaker to participants at the start of the sprint, and used it to study the prompt programming process as well as how prompt programming may influence the prototyping process. PromptMaker uses a version [7] of the large model described in [2], which is a 137-billion parameter generative language model that behaves in a similar way to GPT-3 in its ability to follow prompts.

Given a desired ML functionality to prototype (e.g., translate English to French), a prototyper types an initial prompt into the Prompting Panel, either in the freeform mode (Figure 1, part 1a), or in the structured mode (Figure 1, part 1b). The structured mode provides a convenient way to create few-shot prompts: one can add sets of examples (one set per row), and PromptMaker automatically concatenates these examples into a single prompt string to send to the model. After making a prompt, the prototyper then presses the “Run” button in the Testing Panel (Figure 1, part 2) to see how well the model’s top-N outputs match the desired functionality (in this case, the correct French translation). Often, prompts are meant to be *reusable* and should generalize across many different hypothetical user inputs (e.g., a translation prompt should be able to translate any word or phrase provided by a user). Thus, in the freeform mode,

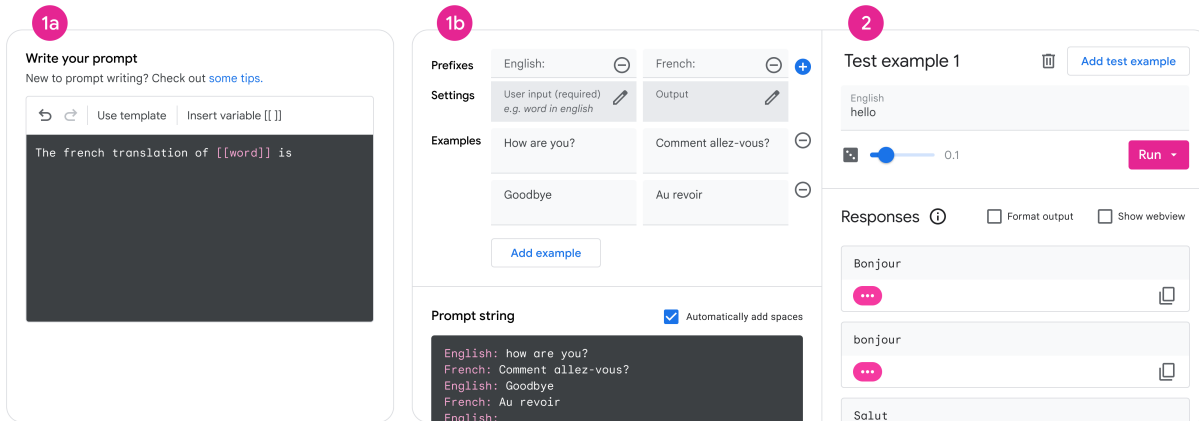


Figure 1: PromptMaker interface. Users prototype new ML functionality by writing a prompt either in freeform mode (1a), or in a structured mode (1b). The structured mode (1b) helps users make few-shot prompts more conveniently, by allowing them to insert one set of examples per row; it automatically concatenates those examples into a raw prompt string. Users can then test the prompt on inputs of their choice (2), view the top-N model outputs, then further iterate on their prompt.

PromptMaker enables prototypers to specify optional variables using a double-bracket notation (e.g., “[[word]]”), which act as placeholders in the prompt. The prototyper can then test their prompt on inputs of their choice (e.g. “hello”) in the Testing Panel. PromptMaker automatically substitutes the bracketed placeholder (“[[word]]”) with the provided test input (e.g., “hello”) when the user presses “Run”. After viewing model results, the prototyper can further iterate on their prompt, add more test examples, or adjust the temperature slider (Figure 1, part 2). The “temperature” slider adjusts the randomness of the generated output. Finally, prototypers can share their prompt with others via a prompt gallery.

5 PARTICIPANTS’ OVERALL PROMPT PROGRAMMING JOURNEY

In this section, we first provide a high-level overview of the prototypes produced in the three-week sprint, then describe the participants’ overall prompt-programming journey.

Participants in the sprint explored a wide range of use cases spanning creativity, recommendations, and conversational agents in different domains. By the end of the sprint, they had converged on eight prototypes, including the following use cases: education (e.g., given a paragraph, generate reading comprehension questions; given a science paragraph, classify which scientific method it is using), games (given characters and roles, generate new characters and roles to create new game rules; given a chess move, generate next chess move), podcasts (given a celebrity, generate a podcast introduction of that person), travel (given a location, generate travel recommendations), conversational personality (given weather-related questions, generate responses with the personality of a character).

Participants’ initial impressions of prompt programming tended to be influenced by the tailored scope of specific demo(s) they had seen. For example, some assumed large language models were primarily meant to be used for chatbot applications, for customizing the personality of bots, or for tasks such as style transfer. While

these are indeed common uses of LLMs, they do not alone capture the key advancement of LLMs, which is the ability to be customized on-the-fly for a wide range of use cases. Some only grasped this unique aspect of LLMs once they had browsed prompts created by others, created a few prompts themselves, or participated in a prompt-making tutorial early in the sprint.

In the early stages of prompt programming, participants oscillated between being astounded by the LLM’s amazing capabilities, to being perplexed by its unpredictability and tendency to go “off the rails.” Participants were particularly impressed with its ability to “learn” to mimic nuanced patterns of speech and adapt them to new topics. For example, one user was surprised that, given some few-shot examples of celebrities and their corresponding *This American Life* podcast introductions, their prompt did a remarkable job of generating a new podcast introduction of a new celebrity (e.g. Michelle Obama), with its output mimicking the typical tropes of a podcast, while simultaneously being germane to a celebrity not provided in the few-shot prompt examples (Michelle Obama).

Conversely, participants were bewildered when their early model prototypes started hallucinating or veering far from their desired functionality. One participant was awestruck that “if I ask if it has a social media account, it will just make something up.” Another said, “After a handful of turns...it would go way off script...and then I would have to rein it back in.” These surprising behaviors prompted users to experience imposter syndrome with respect to their ability to reliably produce prompts (“Maybe there’s someone who can do it better than me”), or wonder if there were unwritten rules that they did not have knowledge of (“I’m not sure how the hardcoded memory works”). Overall, users felt they were in an “uncanny valley of magic, where it’s so magical but it’s not fully magical. You can’t just—in freeform text—describe what you want.”

By the end of the sprint, participants felt that they had developed a more nuanced, calibrated understanding of the LLM’s strengths and weaknesses through interactive play. One participant described their early experiences in prompt-making as a roller coaster of

elation and disappointment, eventually resolving in tempered optimism: *“I’ve kinda gone through a couple phases... I remember when GPT-3 came out and getting all hyped about it, then going ‘nah this thing is just gonna produce gobbly gook.’ This sprint re-energized me about it. Just getting in and tinkering with it and seeing what it can do. I can sort of see the light at the end of the tunnel more.”*

6 HOW PROMPT PROGRAMMING AFFECTED AI PROTOTYPING

6.1 Improving the AI Prototyping Process

6.1.1 Reducing prototyping time and reliance on others. By far the most substantial benefit noted by participants was their empowered ability to author and originate new uses cases of ML, and to prototype ML functionality much more quickly than they otherwise would. Previously, designers would reach an inevitable roadblock once they needed to transition their static prototypes into functional, model-backed prototypes: *“You have a pen and paper idea, you’ll play with it using Sheets or Slides or pen paper scissors, do a WoZ [Wizard-of-Oz] exercise. As soon as you have to write code, the process slows down, someone has to write a backend and server, everyone needs to have a Linux box.”* With prompt programming, users could quickly transform half-baked ideas into rough, functional prototypes, without needing to know how to code, find a custom model, or gather any training data. Hence, participants felt that prompt programming enabled them to fail faster and test ideas substantially more quickly, before investing more time and resources: *“You want to fail fast and then when you got something good, then start investing in more code and more UX.”*

As may be inferred, part of the time savings can be attributed to a reduced reliance on others. In the early ideation phases, participants typically needed to interact with engineers and teammates with more expertise in machine learning, even to get a basic sense of model capabilities and limits: *“It was sort of a process of interrogation – not of the model itself, but of the people who had the ability to understand it or created it.”* Once they have an initial idea, designers bemoan the time and effort typically required to find machine learning collaborators and convince them to build out their ideas: *“Spending a long time trying to find the ML researchers... We don’t have a lot of ML engineers on our team, or any. Being able to... do some experiments ourselves without having to bother a bunch of SWEs (software engineers) is really useful.”*

Even when they could find like-minded collaborators, participants noted it usually takes them much longer to communicate the idea to someone else, compared to building it themselves in Prompt-Maker: *“I was able to sketch something in 20 minutes... I didn’t have to write a big a long doc... like a deck or a document...”* For many, this increase in self agency significantly sped up their prototyping process: *“Maybe four times as fast!”*

6.1.2 Building an early intuition for interaction feasibility. Prompt programming enabled users to directly probe and interact with their envisioned ML functionality, thus allowing them to quickly test out the feasibility of a half-baked idea. Previously, the process of thinking through possible human-AI interactions often involved manually brainstorming hypothetical user inputs and model outputs: *“As somebody who’s not an engineer, prototyping for me is*

usually just me sitting down and writing a bunch of sample dialogues.” To develop an intuition for model capabilities prior to one being fully implemented, participants noted that they would try to find interactive machine learning demos online geared towards non-ML audiences (e.g., Teachable Machine [6]), or self-educate by reading popular blog posts. However, they found it difficult to gain an intuition for what might be possible, or what realistic AI quirks or errors might arise, through these indirect methods: *“It was very indirect... You have to really intellectualize how it’s working rather than get an intuition of how it’s working.”*

Conversely, participants felt it was a world of difference when they could directly author and interact with raw ML material: *“Being able to play around with [the model] directly and... get a sense of what things are easy and what things are hard... just so much more useful than a wishful thinking sample dialog.”* By gaining a hands-on intuition for potential model capabilities and limitations, users felt they were able to prototype the “physics” or feasibility of their half-baked ideas: *“I could say, ‘This isn’t completely outside of the rules of physics. This might work, you know? This is close enough to something that you will believe it’s worth investing [in].”* Another participant echoed this sentiment: *“I have a much better sense of when I ask it to generate something, whether something interesting will happen or whether it will be a mess... helps to develop a sense of what kinds of things are worth trying out.”*

To discover the possibilities, one user made prompts for *multiple* envisioned applications, each of which overshot or undershot the model’s capabilities until one hit a sweet spot. First, he attempted to make a “scientific method classifier,” which the model could not do reliably. He then made a word definition generator, which seemed too easy for the model. Finally, he made a prompt that generated new game characters. With this final prompt, he discovered some potential limits (*“It’s not inventing new rules... just reflections of rules that exist”*) and some remarkable capabilities (*“if it understands a thing, it can translate that thing onto other words”*), culminating in a personal sense of the model’s strengths and weaknesses (*“Certain kinds of meaning it’s good at, but higher abstractions of meaning it’s bad at”*).

6.2 Reducing the Social Friction of AI Prototyping

Participants described the complex social processes typically involved in prototyping with AI. The previous section already described one potential source of friction: finding machine learning experts to help understand what is possible, or to help build prototypes. This section describes other ways that prompt-based prototyping may affect social processes.

6.2.1 Prototypes as boundary objects to ground communication between collaborators. Participants felt that their prompt-based prototypes served as mechanisms for *grounding* communication between cross-functional collaborators with a common, tangible object. In this light, the prototypes act as *boundary objects* [11, 12, 15], providing a common frame of reference for collaborators. Whereas it can be hard to see eye-to-eye on an abstract idea, jointly interacting with the same functional prompt helped users understand each others’ perspectives and make progress faster: *“I was actually referencing something, but he [an engineering collaborator] could*

then use [it], and so it was a common material that we were both using [despite] different abilities... I think that was really important in terms of increasing that velocity.” Without this grounding mechanism, “I would have had to detail out what my intention was far more deeply because we weren’t referencing the same material.” One user contrasted the more arduous, intellectual process of communicating ideas through documents, to the more direct, instinctive process of communicating through prototypes: “That was one of the more time consuming processes in a way because you had to think about how to communicate what you want rather than a material understanding of what you have.”

6.2.2 Reducing risk imposed on collaborators. Critically, participants felt that because prompt-making helped them *de-risk* their own ideas, it also allowed them to reduce the burden they would otherwise impose on *collaborators*, who would typically need to invest time upfront without guaranteed rewards. For example, a participant felt that, by demonstrating an idea was “not outside of the laws of nature in terms of how the [model] may operate,” they felt like they were imposing less risk on engineering collaborators: “It didn’t feel like I was putting as much risk on [my collaborator] as a result of doing that. It kind of... derisks the pitch... Being able to really really quickly ground yourself without risking anybody else’s time or attention, that was great.”

6.2.3 Creating transferable objects between collaborators. Interestingly, a few participants reported that the modularity and reusability of a prompt allowed their team to quickly build on each other’s progress. For example, some teammates each individually created few-shot examples in a spreadsheet, then merged them together. Another user discovered that structuring a prompt with prefixes not only helped the prompt perform more reliably (e.g., using “Place:” and “Description:” in this prompt: “Place: Space Needle in Seattle... Description: Seattle Center was built for the 1962 World’s Fair”), but also enabled teammates to easily borrow and modify that template to create new character styles: “I realized wait a minute I could just give [collaborators] a dump of the format... one wrote Tony soprano, another wrote Hipster guy. That was awesome!... It was quicker for them to rewrite what I already had [written] for the base Seattle response.”

7 CHALLENGES OF PROMPT-BASED PROTOTYPING

7.1 Reverse-Engineering Design Goals into Prompts

Although prompting can be done entirely in natural language, users still faced an immediate hurdle of determining how to “reverse-engineer” their desired functionality into an effective prompt. With *zero-shot* prompts, participants quickly noticed that different paraphrases of the same prompt could have vastly different levels of effectiveness towards producing the desired effect. For example, while writing a zero-shot prompt to generate example sentences given a vocabulary word, one participant was surprised to find that the prompt “The sentence that uses the word [[word]] would be: ” worked well, but a different phrasing “Use the word [[word]] in a sentence: ” did not work well. These differences are likely due

to certain prompts matching patterns in the LLM’s training data better than other prompts.

Given this, beginners were particularly surprised when their initial zero-shot prompts did not behave as intended. Since zero-shot prompting has the ostensible look-and-feel of other familiar interactions (e.g., sentence auto-completion, or talking to a chatbot), participants initially treated the LLM like an agent that can “understand” a request like a fellow human collaborator (e.g. “Use the word ‘eccentric’ in a sentence”). Instead, they needed to work *backwards* from a desired goal to a zero-shot prompt design (e.g. by considering “how is existing text on the internet phrased, to show example vocabulary words used in a sentence?”); in effect, they needed to alter their mental model by treating the LLM as an entity that produces the most likely output given the input and its training data, as opposed to a chatbot or human collaborator. Participants found this process of working *backwards* somewhat counter-intuitive, and often relied on acquiring tips and tricks through word-of-mouth, such as “name-dropping” keywords that are likely to be correlated with the desired results (e.g., some learned that “tldr:” can help prime the prompt to perform summarization).

Eventually, some participants gravitated more towards few-shot prompts, because it gave them a more systematic way of reverse-engineering their goal into pairs of inputs and outputs (compared to zero-shot prompts, whose syntax felt more open-ended). Still, it took some effort to transform a problem into input and output pairs: “How do I get it to do the thing I want it to do? Made me think about the two sides, what the role [input] and the description [output] should be.”

7.2 Example-Sourcing

One benefit of LLMs is that users can quickly customize model behavior by providing a few examples of input-output pairs (as a few-shot prompt string). Despite these benefits, participants noted the challenges involved with finding or generating these examples, a process we dub *example-sourcing*.

While a few participants searched the web or extracted examples from their own content, the vast majority created their own made-up examples on-the-fly to use in their prompts. Among the rare few who sourced examples from an existing corpus, it often took additional effort to clean the data: “I happen to have this collection of Star Trek scripts. Unfortunately they’re kind of messy and they haven’t been formatted... a bunch of line breaks and extra stuff in there.” In essence, while prompt programming lowers the barrier for non-experts to produce new AI prototypes, they may still need to engage in more traditional machine learning activities, such as creating and curating a dataset for training the model (albeit, a much smaller dataset than is typical for deep neural networks). However, in contrast to typical machine learning, the source of the examples is often the users themselves.

Participants noted that example-sourcing was upper-bounded by the limits of their own creativity and mental energy required to conjure up examples. To extend beyond their own limits, one user recruited a collaborator to broaden their set of examples, and commented on the level of human intelligence required to produce high-quality AI-output: “I recruited a creative writer to help out... it was interesting to see the kind of creative stuff he put in and how his

voice would shine through...” Rather than sourcing examples from other colleagues, another participant sourced additional examples from the model itself by picking good results from initial model output, adding them to the few-shot prompt, running the model, and then repeating this process: *“I would generate some and... just scroll through scroll through scroll through until I found some gold, and put that back in and go from there... I’m kinda using [it] for coming up with... sequences of words that I would not have come up with on my own.”* However, even this process could be painstaking if it requires sifting through dozens of examples before coming across a good one: *“It required a lot of digging through examples.”*

7.3 Debugging and Control

While prompt-making, participants found it particularly difficult to debug their prompts, partly due to the non-deterministic nature of model output. Because generative models produce different outputs on different runs, it is difficult for users to decipher whether it was a change they made that produced an error, or whether it was due to random luck of the draw: *“makes it hard for you to understand which lever you pulled correctly.”* As a result, users formed hypotheses and folk theories on what might have produced the changes observed. Because prompts often contain multiple components and characteristics (keywords, phrasing, prefixes, length), users could not tell which specific aspect of their prompt most influenced the results: *“So is my specifying one, two, three sentences actually making a difference or is it the fact that my summary was one sentence?”* Some even wondered if there was a system error: *“I don’t know what changed. Maybe there was an upgrade or something.”*

A common problem users encountered was the model either *over-generalizing*, by straying too far from the examples provided, or *under-generalizing*, by fixating too much on the examples provided. To combat under-generalizing, some tried to eliminate overly-influential examples, or shuffled examples to decrease the influence of the final example: *“I would find it starting to fixate on certain examples, so I would have to eliminate those examples... I might have tried reordering things.”* Under-generalization was particularly problematic when users wanted the prompt to generate new topics different from the few-shot examples, but following a style similar to the examples. For example, one participant used touristic descriptions of Seattle in some few-shot examples, with the intent of having the model generate touristic descriptions of other cities, but could not find a way to prevent the model from continuing to create Seattle-related descriptions. To combat under-generalization, users tried to diversify their prompt examples for broader coverage: *“I put a second example, so that when I generated output it would have more variety.”* Overall, this need to debug prompts and modify “training” examples (even among non-engineers) suggests that prompt-based prototyping practices may require practices commonly found in software engineering and machine learning.

Beyond debugging, participants also had trouble *controlling* and expressing logic that they wanted their prototype to follow. One user wanted to create low-level rules like *“only output a single sentence.”* Another needed to express high-level logic, such as enforcing the prototype to follow canonical social scripts (e.g., well-known steps to making a restaurant reservation), or enforcing the model to behave differently in different settings (e.g., phone vs. car vs. tablet).

They desired to define this sort of “business logic” into prompts, but could not find an immediate way to do so: *“How do you provide real-world pragmatic information...appropriate guardrails that is required for a specific domain or business?”*

7.4 Evaluating Prompt Effectiveness

Participants also attributed the difficulty of debugging to the challenge of *evaluating* whether their prompts were improving. Those newer to prompt-making struggled with the non-deterministic nature of the model, whereas more experienced prompt-makers found it difficult to systematically evaluate and track improvements over time.

To address the non-determinism of the model, a common strategy employed to evaluate changes to the prompt was to re-run the model multiple times to see if the observed behavior was consistent. Another strategy was to test the prompt on “extreme caricatures,” or on inputs that the user has deep familiarity with, to test against a known groundtruth: *“I would always pick things that I’m obviously familiar with.”* Yet another strategy was to first test the prompt at a low temperature setting to confirm that results were reasonable, then increase the temperature to see if model output yielded interesting results as opposed to completely irrelevant results: *“If you crank it up a little bit, you see ‘is it going to give me something interesting...? Or is it giving me something so far off from what I want that it’s no longer helpful?’”* In other words, they were evaluating whether ostensibly reasonable output at low temperatures meant that the model truly matched their intended concept, or whether it may have been due to noise or other confounds.

More experienced prompt-makers indicated that a key barrier to evaluating prompts was the mental load of skimming and evaluating large volumes of text: *“It’s a pain to read all of these things and hold it all in your mind.”* Currently, this process is somewhat haphazard: *“I just kind of eyeball it and think it’s a better answer but it’s super judgmental.”* To more systematically evaluate their prompts, users wished they could assign a “quality score” to model outputs, and see how that score changes over iterations. They also wished this score could be automated, but the scoring they suggested (e.g., number of pronouns, parts of speech) tended to be highly specific to their use case (e.g., creative writing), rather than a metric that could be universally applied. Other suggestions included building canonical “test sets” for their prompt, to test the prompt on different types of input and see where it breaks down: *“helps you narrow down which data is on the manifold of cases that don’t work.”* For example, a “paraphrase” prompt could be tested on words, sentences, and paragraphs. A related idea was to test the same test set on multiple variations of the same prompt, to determine which prompt works best. Beyond evaluating the prompt itself, some designers desired to evaluate their proofs-of-concept within more authentic environments, such as plugging prompts into visual prototyping tools or existing web apps.

8 DISCUSSION

8.1 Considering How LLMs Alter the Need for ML Expertise in Prototyping

Currently, AI expertise may be concentrated within a few individuals, such as AI engineers and researchers. This concentration of expertise increases barriers of access to non-experts who must rely on others, both to determine whether an idea is feasible, and to implement ideas via functional prototypes. LLMs have the potential to alter these dynamics by empowering non-experts to directly create functioning prototypes, gain the intuition necessary to conceive of promising new ML use cases, and use those working proofs-of-concept to ground communication with collaborators. Hence, we may find that future AI-based features and products can be more rapidly iterated on and de-risked earlier in the product cycle, since more people have the ability to test out ideas, and in less time.

While prompt programming lowers the barrier to prototyping new AI ideas, it may occasionally deter practitioners from pursuing designs that could eventually work with alternative ML methods (e.g., training a specialized model, prompt-tuning [13], etc.) or a better prompt design. In other words, the inability to produce an effective prompt does not necessarily mean that a model cannot eventually be produced to solve the intended problem. Conversely, successes on a few example inputs may not generalize to behavior in authentic environments, thereby limiting the prototype's utility in understanding larger issues related to creating responsible AI. At the same time, we also observed in our study that, relative to the status quo, prompt programming gave users a much more intuitive way of developing calibrated expectations about generative models in general, and to develop an awareness of risks and edge cases they may not otherwise have considered. Given these benefits and trade-offs, it may be worthwhile to provide onboarding documentation [5] to sensitize prompt programmers to the fact that there is a limit to which conclusions can be drawn from prompt-based prototyping. In addition, these onboarding materials could provide go-to strategies for effective prompt design, so that users are less likely to be deterred by naive implementations of ineffective prompts.

8.2 Considering Changes to AI Prototyping Practices

Given a shift in *who* can prototype with ML, we may also witness a parallel shift in *what* non-ML practitioners do when prototyping in the future. Currently, designers engage in activities such as Wizard-of-Oz [8], static mocks, or manual dialogues to imagine what interactions may look like. With LLMs, their roles could expand beyond consideration of the primary design concept, to considering other issues such as how to handle AI failure modes (by using concrete model output to guide that thought process). In other words, they may now prototype more “deeply.” They may also begin to borrow practices from traditional software development and ML, such as debugging the prompt, finding examples to “train” the prompt, testing the prompt, etc. Overall, prompt-based prototyping introduces a new hybrid between UX design and ML modeling: It introduces common ML practices into the traditional

design process, while imbuing AI development with more user-centered perspectives through broadening participation to more diverse professions.

8.3 Integration With Existing Prototyping Tools

Finally, users desired to plug their prompts into existing prototyping tools, so that they can bring their prototypes to life within authentic product environments (e.g., a web application) or visual mockups (e.g., in Figma). While PromptMaker currently allows users to export their prompt as an endpoint to plug into web applications, future work could consider no-code solutions, such as allowing a prompt to be easily imported through a plugin into existing prototyping tools like Figma. For example, a designer could drag and drop a text box that is backed by an LLM prompt in a design tool. Integrating prompt-based prototypes into authentic environments could enable designers and developers to get earlier *in situ* feedback from users reacting to ideas presented within their envisioned context of use.

9 CONCLUSION

This case study provides important implications toward a new future of prototyping with ML: prompt-based prototyping substantially reduced prototyping barriers, while also introducing new prompt design challenges. While this research studied the use of PromptMaker in a large industry setting, there are numerous settings in which ML prototyping time is currently a bottleneck, and for which non-ML expert participation is crucial. We hope future work will build on this case study to lower the barriers to ML prototyping in other environments, as well as for the general public.

10 APPENDIX

10.1 Acknowledgments

We thank Gabe Clapper, Jess Holbrook, and Matt Jones for making the sprint possible, Jimbo Wilson and Tongshuang Wu for feedback on PromptMaker, and Lucas Dixon and Merrie Morris for feedback on the paper. We thank all participants for their thoughtful feedback.

10.2 Author contributions

Ellen Jiang and Edwin Toh led the engineering effort on PromptMaker and helped with paper writing. Kristen Olson conducted formative interviews, conducted and managed user studies and transcription, and helped with paper writing. Alejandra Molina created UX and visual design for PromptMaker, helped with the workshop and sprint tutorial, and created visuals for the paper. Aaron Donsbach facilitated the case study workshop and the prompting tutorial for the sprint, and provided insight on prompting strategies. Mike Terry proposed the initial study design, conducted user studies, gave high-level scientific advice, and contributed to paper writing. Carrie Cai guided the research, conducted user studies and analysis, gave high-level scientific advice, and wrote much of the paper.

REFERENCES

- [1] [n. d.]. GPT-3 Creative Fiction. <https://www.gwern.net/GPT-3>. Accessed: 2021-03-30.

- [2] Daniel Adiwardana, Minh-Thang Luong, David R. So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, and Quoc V. Le. 2020. Towards a Human-like Open-Domain Chatbot. arXiv:2001.09977 [cs.CL]
- [3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Matusz 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. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. <https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf>
- [4] Bill Buxton. 2010. *Sketching user experiences: getting the design right and the right design*. Morgan kaufmann.
- [5] Carrie J. Cai, Samantha Winter, David Steiner, Lauren Wilcox, and Michael Terry. 2019. "Hello AI": Uncovering the Onboarding Needs of Medical Practitioners for Human-AI Collaborative Decision-Making. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 104 (Nov. 2019), 24 pages. <https://doi.org/10.1145/3359206>
- [6] Michelle Carney, Barron Webster, Irene Alvarado, Kyle Phillips, Noura Howell, Jordan Griffith, Jonas Jongejan, Amit Pitaru, and Alexander Chen. 2020. Teachable machine: Approachable Web-based tool for exploring machine learning classification. In *Extended abstracts of the 2020 CHI conference on human factors in computing systems*. 1–8.
- [7] Eli Collins and Zoubin Ghahramani. 2021. LaMDA: our breakthrough conversation technology. <https://blog.google/technology/ai/lamda/> Accessed: 2021-07-14.
- [8] Nils Dahlbäck, Arne Jönsson, and Lars Ahrenberg. 1993. Wizard of Oz studies—why and how. *Knowledge-based systems* 6, 4 (1993), 258–266.
- [9] Steven P Dow, Alana Glassco, Jonathan Kass, Melissa Schwarz, Daniel L Schwartz, and Scott R Klemmer. 2010. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM Transactions on Computer-Human Interaction (TOCHI)* 17, 4 (2010), 1–24.
- [10] Matthew K Hong, Adam Fourney, Derek DeBellis, and Saleema Amershi. 2021. Planning for Natural Language Failures with the AI Playbook. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–11.
- [11] Bonnie E John, Len Bass, Rick Kazman, and Eugene Chen. 2004. Identifying gaps between HCI, software engineering, and design, and boundary objects to bridge them. In *CHI'04 extended abstracts on Human factors in computing systems*. 1723–1724.
- [12] Charlotte P Lee. 2007. Boundary negotiating artifacts: Unbinding the routine of boundary objects and embracing chaos in collaborative work. *Computer Supported Cooperative Work (CSCW)* 16, 3 (2007), 307–339.
- [13] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691* (2021).
- [14] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586* (2021).
- [15] Susan Leigh Star. 1989. The structure of ill-structured solutions: Boundary objects and heterogeneous distributed problem solving. In *Distributed artificial intelligence*. Elsevier, 37–54.
- [16] Qian Yang, Justin Cranshaw, Saleema Amershi, Shamsi T Iqbal, and Jaime Teevan. 2019. Sketching nlp: A case study of exploring the right things to design with language intelligence. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.
- [17] Qian Yang, Aaron Steinfeld, Carolyn Rosé, and John Zimmerman. 2020. Re-examining whether, why, and how human-AI interaction is uniquely difficult to design. In *Proceedings of the 2020 chi conference on human factors in computing systems*. 1–13.