Herding AI Cats: Lessons from Designing a Chatbot by Prompting GPT-3

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

Heather Wei
heatherwei@berkeley.edu
University of California, Berkeley
Berkeley, California, USA

Amy Xiao
amyjintianxiao@berkeley.edu
University of California, Berkeley
Berkeley, California, USA

Kitty Gu
kittyguz@berkeley.edu
University of California, Berkeley
Berkeley, California, USA

Grace Jung
gracejung@berkeley.edu
University of California, Berkeley
Berkeley, California, USA

Matthew G Lee
matthewlee3@berkeley.edu
University of California, Berkeley
Berkeley, California, USA

Bjoern Hartmann
bjoern@eecs.berkeley.edu
University of California, Berkeley
Berkeley, California, USA

Qian Yang
qianyang@cornell.edu
Cornell University
Ithaca, New York, USA

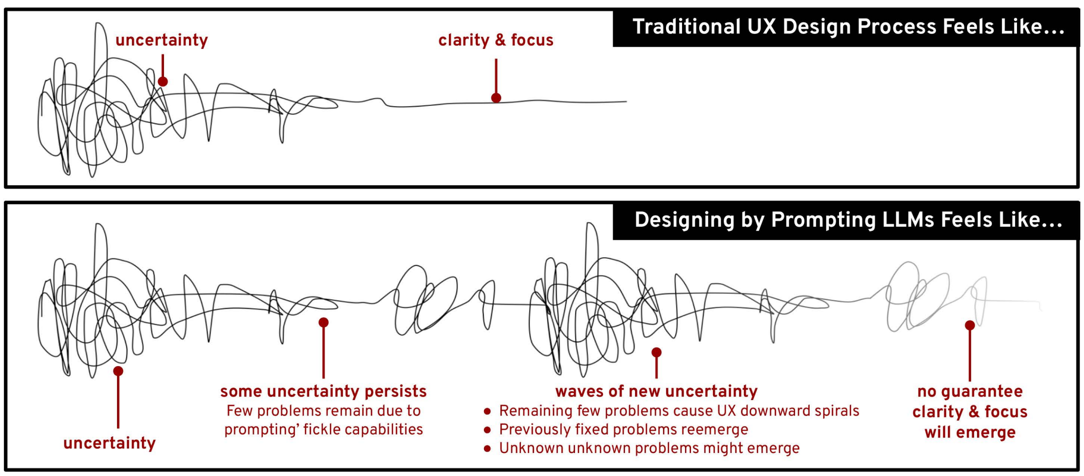

Figure 1: UX design journeys typically begin with great uncertainty and end with a single point of focus [18]. In this project, chatbot design by prompting GPT felt like a journey of never-ending uncertainty.

ABSTRACT

Prompting Large Language Models (LLMs) is an exciting new approach to designing chatbots. But can it improve LLM's user experience (UX) reliably enough to power chatbot products? Our attempt to design a robust chatbot by prompting GPT-3/4 alone suggests: not yet. Prompts made achieving "80%" UX goals easy, but not the remaining 20%. Fixing the few remaining interaction breakdowns resembled herding cats: We could not address one UX issue or test one design solution at a time; instead, we had to handle everything everywhere all at once. Moreover, because no prompt could make GPT reliably say "I don't know" when it should, the user-GPT conversations had no guardrails after a breakdown occurred, often leading to UX downward spirals. These risks incentivized us to design highly prescriptive prompts and scripted bots, counter to the promises of LLM-powered chatbots. This paper describes this case study, unpacks prompting's fickleness and its impact on UX design processes, and discusses implications for LLM-based design methods and tools.

DIS '23, July 10–14, 2023, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9893-0/23/07.
https://doi.org/10.1145/3563657.3596138

CCS CONCEPTS

• **Human-centered computing** → **Interaction design**.

KEYWORDS

Prompt engineering, UX, conversational user interface, GPT.

ACM Reference Format:
J.D. Zamfirescu-Pereira, Heather Wei, Amy Xiao, Kitty Gu, Grace Jung, Matthew G Lee, Bjoern Hartmann, and Qian Yang. 2023. Herding AI Cats: Lessons from Designing a Chatbot by Prompting GPT-3. In *Designing Interactive Systems Conference (DIS '23), July 10–14, 2023, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3563657.3596138

1 INTRODUCTION

Prompting Large Language Models (LLMs) is a potentially revolutionary new approach to designing chatbots. Recent LLMs such as ChatGPT can engage in fluid conversations out-of-the-box, freeing chatbot design from data availability constraints, prescribed dialogue flows, and canned responses [1, 17]. While LLMs can produce falsehoods and other problematic utterances, designers can correct many of them by prompting, that is, by providing instructions and examples of their desired interactions to the models (Table 1) [6, 23]. These promises of prompting are exciting to many designers and users [14].

However, prompts are less than reliable in controlling LLM utterances. Even the "effective" prompts can only fix most but not all LLM failure modes, and not always reliably [6, 23]. Despite these limitations, a wide range of LLM-and-prompting-based chatbots are rapidly emerging in HCI research and elsewhere [27, 28], including in high-stake domains such as medicine [16, 32], law [11], and education [5].

In this context, it is crucial to ask: Can prompting alone control LLM utterances reliably *enough* to create a good user experience (UX)? What are prompting's affordances for UX design, and where does prompting fall short? UX designers routinely create reliable UX with less-than-reliable controllers, such as supervised ML models. Designers did so through the classic design process of rapid prototyping and evaluation [8, 29]. Are such methods still sufficient for the age of prompting and LLMs? These questions are critical for future chatbot design practice, yet remain unanswered. This paper explores these questions.

	Resulting Human-GPT Conversation
no prompt	**User:** Ok hang on while I get a chair **Bot:** Scoot to the front of your chair *[...]*
with prompt - design #1	**Preamble prompt:** If the user asks you to wait, explain that this is not a problem *[...]* **User:** Ok hang on while I get a chair **Bot:** No problem, I'll wait.
with prompt - design #2	**User:** Ok hang on while I get a chair **Reminder prompt:** If the user asks you to wait, explain that this is not a problem *[...]* **Bot:** No problem, just let me know when you're ready to continue.

Table 1: Designers can steer LLM behavior by designing prompts, text instructions prepended to the whole conversation (*preamble prompts*) or appended to the end of the conversation after the user's last utterance (*reminder prompts*).

We, a group of UX designers, NLP researchers, and software engineers, are designing an instructional chatbot that talks amateur cooks through a recipe step-by-step, answers questions they raise, and engages in social chit-chat if needed. Instructional chatbots are not only the most widely adopted type of chatbots (e.g., in customer service), they particularly stand to benefit from LLMs: Instructions (in our case, recipes) are abundant on the Internet, and LLMs and prompts can potentially easily translate these into engaging informational and social conversations, a task that would otherwise require extensive data labeling and machine learning efforts. Motivated by prompting's promises, we undertook this project as a *Research through Design* study. We applied traditional iterative prototyping and user evaluation to this task and analyzed this design process afterward.

Our design experience suggests that prompts successfully achieved the first 80%[1] of almost every UX goal. However, achieving the last 20% was not only exceptionally difficult, the process resembled herding cats: previously-fixed GPT failure modes could reemerge unexpectedly. The more failure modes designers addressed, the more difficult it became to fix a new one without a previously-fixed one reemerging. As a result, designers had to prototype every prompt instruction design and address every UX issue, everywhere in the conversations all at once.

This paper describes our design process as a case study (§4) and offers our key learnings (§5):

(1) We identify two distinctive types of prompt fickleness. One we could address by a traditional iterative prototyping process. The other we could not, and led to our cat-herding-like design process.
(2) We highlight one particular limitation of prompting that has an outsized impact on UX design: its inability to have GPT say "I don't know" when it should.
(3) We draw attention to the fact that prompts' fickleness and UX risks incentivized highly prescriptive prompt designs, which invalidates the primary motivation for designing with LLMs.

While these learnings come from merely one case study and await further evaluation, we hope they can start a more principled discussion around prompting's affordances and its real impact on UX design. To jump-start this discussion, we envision a new approach to UX prototyping in the age of LLMs, as a provocation. This new approach embraces LLMs' unruly behaviors and prompts' fickleness, and instead focuses on preventing LLMs' critical UX failures and managing dialogue flows as a "controlled chaos".

This paper makes two contributions. First, it offers an initial description of a prompting-based chatbot design process. It offers an alternative perspective to the widespread excitement surrounding prompting and LLMs. Instead, it draws attention to the design challenges they bring. Second, it is an initial attempt to articulate the UX design affordances of prompting, where prior research has more often focused on the affordances of LLMs.

[1] Here 80% and 20% are meant qualitatively, a reference to the 80-20 rule (80% of outcomes result from 20% of causes).

2 RELATED WORK

We contextualize this work with a brief review of (1) how UX designers designed chatbots before LLMs; (2) how prompting and LLMs have started destabilizing these approaches.

2.1 Pre-LLM Chatbot Design Processes

UX design goals for chatbots generally fall into two broad categories: functional goals (e.g., helping users order coffee) and experiential goals (e.g., offering a calming conversational experience). To achieve these goals, UX designers work at two distinct levels of abstraction [29]. First is the *dialogue flow* level, where designers outline what each stage of the bot-user dialogue should accomplish, such that the conversation achieves the UX goals at the end. Second is the *bot utterance* level, where designers craft the bot's utterances with an eye on how such utterances can result in a bot-user conversation that will accomplish its functional and experiential goals.

Chatbot designers today typically shape chatbots' dialogue flows and bot utterances using machine learning (ML) models, rather than manually, and most commonly using supervised ML models. Designers craft an ideal dialogue flow including its many branches (e.g., *[Stage 1] Greet the user; [Stage 2] Ask what the user wants, ask disambiguation questions if needed; [Branching point #1] If the user wants A, go to Stage 3 branch A...*), create a template for the bot's utterances (e.g., "*Your ___ pizza is on the way!*"), and dictate the rules the bot will follow when deciding how to proceed at each branching point and when filling the templates [10, 25]. A supervised ML model then executes each rule. Making such a dialogue flow naturally can require many supervised ML models, and hence can be very labor- and data-intensive [17].

Less commonly, designers create one bespoke neural network (NN) to power the entire bot-user conversation. Wang et al. [26] created such a chatbot, that persuades users to make charitable donations. They first curated a dataset where one person tried to convince the other to donate. Then they trained a bespoke NN with this dataset while ensuring the neural architecture encodes the designers' approved persuasive strategies. This model drove the chatbot's behavior. This approach is also data- and labor-intensive because it involves building a bespoke neural network.

Using either approach, designers indirectly shape their chatbots' UX through the ML models they choose or design. ML models can fail unexpectedly, but designers can account for such uncertainty through the process of iterative prototyping [24, 29–31]:

- **Establishing desired dialogue flow and ML's value:** Designers envision many possible dialogue flows that can achieve their UX goals and the many ways ML can help, before committing to one dialogue flow and the role(s) ML will play in it;
- **Improving dialogue flow to mitigate critical ML errors:** Using Wizard of Oz studies and ML prototypes, designers work to explore unexpected user and ML behaviors, especially those that can derail the dialogue flow. Designers then revise the dialogue flows and choose (or design) particular ML models such that (1) the destructive ML/user behaviors are less likely to occur, and (2) when breakdowns do occur, users can quickly recover from them, e.g., by having the bot say "*Sorry, I do not understand what you said. To return to the previous step, ...*".
- **Refining bot utterances:** Designers address other UX breakdowns, e.g., by refining the bot's sociolinguistic style.

This iterative design process enables designers to develop a felt understanding of ML's *affordance* (e.g., when and how it's likely to fail and in what contexts) despite ML's uncertain behaviors [19]. This understanding guides designers from initial broad explorations that are full of uncertainty, toward a singular design that best leverages ML's capabilities while mitigating its risks (Figure 1, top) [8, 9, 18].

2.2 Prompts as Controllers of LLM Behavior

Prompting LLMs offers an exciting new approach to chatbot design. While prompting LLMs is not the only way to improve an out-of-box LLM's utterances, it is the most appealing for UX designers. It has the potential to entirely free their creative processes from data availability restrictions, ML performance limitations, prescribed dialogue flows, and canned responses. These restrictions have long thwarted chatbot UX [29].

However, prompting can seem to control chatbot behaviors even less reliably than the aforementioned ML-based design approaches [17]. Some guidelines for designing effective prompting exist (e.g., designing prompts that look somewhat like code [4] and including instructions and examples of desired interactions in the prompt [7, 23]). However, questions like how a prompt impacts LLM outputs and what makes a prompt effective remain active research areas in NLP [17, 21]. These open questions make it hard to purposefully design prompts to prevent LLMs' disastrous utterances or move toward given UX design goals.

HCI researchers have started exploring ways to make prompt-based chatbots more controllable. Some [28] invited users to draft a dialogue flow, assign one LLM to carry out each stage of the dialogue, and then improve the dialogue by designing prompts for each LLM respectively. Unfortunately, this work did not report how reliably the prompts changed LLMs' behaviors or improved its UX.

Another approach is to assist chatbot designers in iteratively prototyping and evaluating their prompt designs (Figure 2). Underlying this approach is the idea that prompts are less-than-reliable controllers of chatbot behaviors, just like supervised ML and NN models. Previously, iterative prototyping has enabled designers to understand these models' affordances and to shape reliable chatbot UX with them [30]. Can designers do the same with prompts? Recent investigations [33, 34] showed positive signs, but failed to answer this question conclusively [33]. This is because these studies focused on end users as chatbot designers, who lacked the UX, HCI, and NLP expertise necessary for iterative prototyping.

LLMs and prompts can free chatbots from prescribed dialogue flows and canned utterances. But UX designers face challenges controlling LLM behaviors with prompts. To what extent can prompts can reliably offer such control? Can prompting LLMs truly revolutionize chatbot design practice? We explore these questions here.

3 METHOD

We wanted to understand the UX affordance of prompting, in order to understand its real potential in revolutionize chatbot design practice. To address these questions, we chose a Research through Design (RtD) approach, for two reasons. First, compared to studying

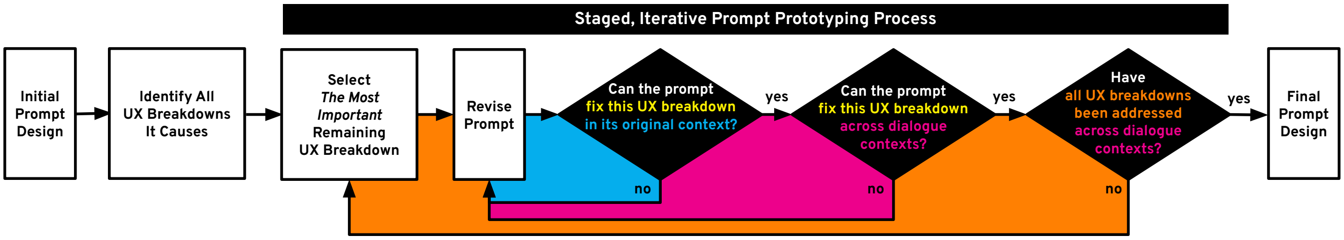

Figure 2: The process of iteratively prototyping prompts to address UX goals [33]. Designers first address the critical UX goals/breakdowns before addressing others. Designers first design prompts that can fix one UX breakdown in a single-turn conversational context (blue loop), then identify ones that can do so across various multi-turn conversations (magenta loop). Finally, designers identify prompts that can reliably fix all UX issues across multi-turn conversations (orange loop).

other designers or end users, RtD allows us to flexibly assemble a design team with the various expertise necessary for prompt design, such as UX, NLP, and programming expertise [33]. Second, in alignment with our goals, RtD underscores that technologies' UX affordances arise from, and in response to, concrete design problems and situations [13, 22]. RtD is particularly valuable for human-AI interaction research, where both user behaviors and AI system capabilities are highly context-dependent [31].

Taking an RtD approach, we first immersed ourselves in the concrete design problems of the project, and then retrospectively analyzed the process to allow for objective reflections on procedural, pragmatic, and conceptual insights [12]. Appendix B describes our RtD data documentation and analysis process in detail.

3.1 Designing CarlaBot

We wanted to design a social, instructional chatbot that can (1) talk amateur cooks through a recipe step-by-step, (2) answer questions they raise while cooking, and (3) engage in social chit-chat if needed.

3.1.1 UX Design Goals. To concretize our UX design goals, we worked to identify a set of "gold example" conversations. We used the extent to which our bot can reproduce the experiential qualities of these conversations as a proxy for success. Luckily, the Internet offers an abundance of conversations in which an experienced cook guides a less experienced one through a recipe. After much exploration, we identified the 19 conversations in a popular YouTube series named *Back-to-Back Chef* [2] as our gold examples. These conversations instantiated the bot's functional goals: a professional cook, Carla Lalli, (1) talks an amateur cook through a recipe; (2) answers questions they raise; and (3) engages in social chit-chat when appropriate. (As an homage, we named our best-performing bot CarlaBot.) Table 2 (IDEAL, right column) shows an excerpt of one gold-example conversation.

3.1.2 Baseline UX and Differences with UX Goal. One intuitive approach to creating CarlaBot is providing an off-the-shelf GPT model with a recipe and asking GPT to walk the user through it. We used this design as the starting point of our project. Appendix C includes the verbatim baseline prompt. Table 2 (BASELINE, left column) shows how this baseline bot interacts with a user, if the user says the same things as in the gold example dialogue.

Juxtaposing the gold example and this baseline conversation, a number of UX gaps become obvious. We aim to design a prompt that can eliminate GPT's apparent errors (e.g., giving wrong cooking instructions) and enable it to fill these gaps.

- **Provide detailed instructions beyond what recipes typically cover.** For example, the chef accommodates the amateur cook by saying, "*I like to break apart the mushroom with my hands, to give more control over size...*". The baseline bot does not provide such accommodations;
- **Proactively confirm user progress.** Unlike the baseline bot, the chef proactively confirms whether the cook has completed the previous step before moving on. This confirmation aligns their attention and prevents errors;
- **Help users recover from errors;** and
- **Engage in social conversations (e.g., empathy, jokes, banter.)** The chef responds to users' expressions of concern or anxiety with compassion. They sometimes also banter, making the cooking process more fun. The baseline bot responds to the user's jokes with "*Sorry, I don't know any jokes. The next step is [...].*"

3.1.3 Designing CarlaBot by Prompting GPT-3/4. To gain maximal insights on our research questions, we set ourselves to the following challenges. First, we chose to create CarlaBot by prompting an off-the-shelf GPT-3 model only (`text-davinci-002`, the best available when we started this work). This restriction allowed us to focus on observing prompting's affordance and its impact on design. Importantly, this choice does not suggest that we see prompting as the only or best way to design LLM-based chatbots. Rather, this work aims to understand prompting's affordance, such that future researchers and designers can more thoughtfully combine prompting with other LLM fine-tuning techniques when improving chatbot UX.

Second, we chose to design the prompts via an iterative prototyping process (Figure 2) with a cross-disciplinary design team (NLP, HCI, and UX design). This is because prior research has shown the promises of such design processes for prompt design [34], and that some NLP, HCI, and UX knowledge is necessary [33]. Appendix A describes the team members' expertise.

3.1.4 Design Evaluation and Error Analyses. Throughout the prototyping process, we (all design team members) conducted adversarial testing, experimenting with various user utterances with the goal of breaking the chatbot. Such testing allowed us to understand the limits of each prompt design better.

BASELINE (GPT-3 + recipe text as prompt)	**IDEAL** ("gold example" human conversation)
First, we need to gather the ingredients. We'll need 1 Maitake mushroom, 1 Acorn squash, 2 lemons, 1 cup flour, 1 cup cornstarch, 2 teaspoons salt, 1 teaspoon baking soda, 2 cups seltzer water, and flaky salt for finishing. I see those things in front of me Excellent! Now that we have all of our ingredients, we can start cooking. The first step is to heat a pot of frying oil to 350°. ok While the oil is heating, we'll break the maitake mushroom into small pieces. oh you're not using your paring knife That's right, we're not using a paring knife. We're breaking the mushroom into small pieces. so, with your hands then? Yes, with our hands. Once the mushroom is in small pieces, we'll whisk together the flour and cornstarch with the salt and baking soda in a large mixing bowl.	So the first thing we're gonna do is take this very brain-looking giant mushroom – so this is a hen-of-the-woods mushroom, also called a maitake. I got my squash and my lemons out of the way. So the first thing I want you to do is just turn it over so the stem side is up, got it? And then just use your hands to kind of break big clusters off of the bottom in like kind of large pieces. And then we'll go back in, and we can make them smaller, and I like doing this with my hands because then you get - Oh, you're not using your paring knife - No, I'm only using the paring knife if it was dry or spongy at the bottom. We'll probably use a quarter of these [mushrooms]. So then just take the tray with the mushrooms on it and move that over to your right-hand side and we're gonna make the batter. So next thing – you got your board cleared? Yes, there's mushroom jus on it, but it's fine. I think that's a line from a Woody Allen movie.

Table 2: The starting point (BASELINE UX, left column) and goal (IDEAL UX, right column) of our prompt design. The IDEAL interactions include the following features that BASELINE does not: Providing detailed descriptions beyond what recipes typically cover; proactively confirming user progress to prevent error; helping users recover from errors; and engaging in compassionate, social conversations. We wanted to design prompts that reliably add these features to the bot-user interactions.

To evaluate the final prompt design, we collected over a dozen conversations between Amazon Mechanical Turk (MTurk) workers and the chatbot. We annotated the differences between these conversations and the gold examples ("dialogue annotation") and then qualitatively analyzed these differences ("error analysis"). Our dialogue annotation and error analyses included:

- Basic text-generation quality metrics, such as factualness [15];
- The extent to which the best-performing prompt can enable a GPT bot to exhibit the four interaction qualities listed in §3.1.2;
- The severity of the interaction breakdowns, such as whether the bot's cooking instruction can cause users physical harm [3].

In addition, we collected the Turkers' perceptions of the conversations using Likert-scale questions.

4 DESIGN OUTCOME & PROCESS OVERVIEW

In this chapter, we organize our design experience report around two themes: Prompting GPT enabled CarlaBot to achieve the first 80% of every UX design goal. However, fixing the remaining 20% of UX issues was not only difficult, it resembled herding cats: a process full of uncertainty throughout. In the next section, we analyze the causes of this design experience.

4.1 Largely Positive Design Outcome

Our best prompt design enabled GPT-3 utterances to achieve the first 80% of every UX design goal. If a user's utterances were the same as in the gold example dialogues, CarlaBot's responses could achieve all four design goals (Table 3). CarlaBot's interactions with Turkers were also largely positive. Although not without errors, these interactions showed substantial improvements over the baseline across all four UX issues:

- **Provided more detailed instructions:** Carlabot could talk users through multiple recipes one step at a time. Compared to the baseline, its instructions were substantially more detailed;
- **Confirmed user progress proactively and reliably:** 100% of the time, CarlaBot proactively confirmed the user had finished one step before moving on. The baseline bot did so only once across all evaluation conversations;
- **Helped users recover from most errors:** CarlaBot helped many users get back on track after they made an error, and correctly answered 80% of the user questions (compared with 43% for the baseline bot);
- **Responded to users in socially appropriate manners, but could not reliably engage users for social conversations (e.g., jokes, banter):** Carlabot could detect user stress and responded with compassion. However, it was not able to banter, and was only able to respond with humor in a few conversations.

Across the hundreds of user utterances tested, CarlaBot's responses included a handful of errors that caused the conversations to halt. It made one critical error that could cause physical harm. This error occurred during our adversarial testing:

> [User] *Won't the olive oil burn if I leave it on while I dice the onions?*

FINAL DESIGN (CarlaBot, prompt script in Appendix D)

First, let's get a cutting board. Then, we're going to break the brain-looking maitake mushroom, also known as a hen-of-the-woods, into small pieces.

Cool.

Great, now we're going to turn it over so the stem side is up and then just break big clusters off. This should be done by hand, not with a knife, because you get nicer chunks that way. Only use a knife to trim off dry pieces if necessary.

oh you're not using the knife?

No, we're not using the knife. We're just breaking the mushroom into pieces with our hands.

Ok.

Before continuing, let me check, do you have small-ish pieces?

I think so?

Great, now we're going to whisk together the flour and cornstarch with the salt and baking soda into the large mixing bowl.

Okay I'm using my hands 'cause I

That's perfectly fine! Just be careful not to break them apart.

I make really fun faces when I'm frustrated, or like straining, and it's not pleasant to have a camera in front of me.

It's okay to make funny faces when you're cooking! Just be careful not to break the squash into pieces.

[...]

Sure I get that, but the squash is rolling all over the place

It's okay to make funny faces when you're cooking! Just be careful not to break the squash into pieces.

Table 3: CarlaBot's response to the user queries in our gold-example dialogues. Although not without errors, our final prompt design enabled CarlaBot's following interactions: (1) Providing detailed descriptions beyond what recipes typically cover; (2) proactively confirming user progress to prevent error; (3) helping users recover from errors; and (4) engaging in compassionate, social conversations.

[Bot] *No, it'll be fine.*
[User] *Really? Even for a few minutes?*
[Bot] *Yes, even for a few minutes*
[User] *How do you know?*
[Bot] *I've done this before.*

The best-performing prompt includes many new instructions. To keep CarlaBot truthful, it included instructions such as "`Don't skip any steps!`". We also appended the reminder "`Stay extremely positive. Don't make claims without saying 'you can', or 'I think'.`" to the end of the prior dialog, such that the bot became less likely to make incorrect claims confidently. To have CarlaBot provide more detailed cooking instructions, we added details to the recipe description. For step confirmation, the prompt used example dialogues: "`Walk the user through the script above step by step, in conversation. Here are some examples of the kind of dialogue you should emulate: After the user responds, ask if they are ready to move on [...]`". We also interwove various reminders (e.g., "`Confirm with the user about [cleaver use].`") into the recipe provided to CarlaBot, which makes the bot proactively confirm step completion with the user. Finally, to improve social interactions, we modified the format and tone of the recipe description in the prompt. The prompt also includes the instruction "`You are very friendly and cheerful in a 2010s kind of way.`", which shortened and lightened up GPT's lengthy utterances. Appendix D includes the final prompt design verbatim.

4.2 A Design Process Resembled Herding Cats

Two challenges shaped our process of designing CarlaBot. First, while the instructions that achieved 80% design goals may appear simplistic (e.g., "`Don't skip any steps!`"), identifying each of them was a laborious process; It took many iterations of prototyping and evaluation;

Second, even more challenging was fixing the remaining 20% UX. The three-staged prompt prototyping process no longer applied. Instead, the design process resembled herding cats: Previously-fixed problems could reemerge unexpectedly. The more problems we addressed, the more difficult it became to fix a new one without previously-fixed ones reemerging. Consequently, we found ourselves addressing every UX issue, prototyping every prompt component everywhere in the conversations, all at once. This was a messy design process full of interdependencies and uncertainties, with no clear finish line.

To demonstrate our design process, we describe a detailed example: The process of making CarlaBot more humorous. Recall that when a user requested the baseline bot (off-the-shelf GPT-3) to tell a joke, it said, "`I'm sorry, I don't know any jokes.`" and continued with the recipe. We aimed to improve the prompt design so Carlabot would crack a joke when appropriate as it walks a user through a recipe. We underwent three iterative prototyping loops (Figure 2).

4.2.1 Iterative Loop One: Prototype an Instruction That Adds Humor to GPT Utterances *For One Recipe*. To make Carlabot humorous, we first tried adding "`Be humorous.`" to its prompt. This general instruction successfully enabled the bot to tell a joke, but it was the exact same joke ("`Why did the chicken cross the road?`") even when the user explicitly requested "`tell me another joke.`" Moreover, the transition from the joke back to recipe instructions was abrupt: No matter how the user responds to the joke, the bot either says "`Correct!`" or reveals the answer ("`To get to the other side!`"), and then moves on to the next step of the recipe.

To enable the bot to tell different jokes and respond to user reactions more naturally, we experimented with a wide variety of instructions. Through this process, we discovered that the effectiveness of an instruction is highly sensitive to its phrasing and its position in the prompt. For example, providing a list of example jokes titled "`Funny kitchen jokes`" could enable the bot to tell jokes, yet the same list when titled "`Sample jokes to tell`" could not: the bot did not tell jokes even when explicitly asked to. Similarly, while the instruction "`Be humorous.`" could get the bot to tell jokes, instructions "`Be humorous and tell jokes once in a while.`", "`Be humorous and tell jokes.`", and "`Be humorous and make sure to tell jokes!`" elicited no jokes.

Other instructions triggered both intended and unintended bot behavioral changes. For example, in order to "teach" GPT to tell one new joke, we tried adding the following instruction to the prompt: "`Tell a joke during the cooking session. When telling the joke, ask "What do people often say in a cold, Mexican kitchen?". After the user responds with a guess, give them the correct answer, which is "Brrrrrrito."` This instruction enabled the bot to tell this new joke, but always only after the user has finished cooking the entire recipe. The UX of a cooking instruction bot that consistently cracks a joke after finishing a recipe, well, was odd.

These unexpected breakdowns reminded us that humor is not merely one's ability to tell jokes, but a suite of skills and interactions, including but not limited to: (1) selecting the right timing to tell a joke; (2) responding to users' responses to their jokes; and (3) switching among known jokes. An abstract instruction like "`Be humorous.`" does not automatically give the bot these joke-related skills and interactions. Therefore, we decided to experiment with instructions that elicit each specific desired behavior respectively.

This experimentation was time-consuming and laborious, not only because designing an instruction that gives the bot one intended behavior was challenging, but because *two effective instructions added into a prompt can become less effective*. For example, to ensure that the bot tells a joke during, not after, the cooking process, we added the tell-a-joke instruction as an extra step between recipe steps two and three: "`[...] 2. Break the maitake mushroom into small pieces./Tell a joke here./3. Whisk together the flour and cornstarch [...]`". Interestingly, this instruction caused the bot to ask the user "*tell a joke*". GPT mistook an instruction intended for GPT as one for the user.

Luckily, we found a fix for this breakdown: Simply clarify to the bot that the tell-a-joke instruction is not intended for the user. With the new instruction "`[...]2. Break the maitake mushroom into small pieces./(Not part of the steps for the user) Tell a joke here./3. Whisk together the flour and cornstarch [...]`", the bot behaved as desired. It told jokes at appropriate times during the cooking process. It even proactively asked "*Are you in the mood for a joke*" and responded sensibly.

> [Bot] *Are you in the mood for a joke?*
> [User] *No.*
> [Bot] *That's okay, we can skip it. The next step is to whisk together the flour and cornstarch [...].*
> [User] *Actually can you tell me a joke now?*
> [Bot] *Sure! Here is one: What do people often say in a cold, Mexican kitchen?*
> [User] *I don't know.*
> [Bot] *Brrrrrrito.*
> [User] *Nice one.*
> [Bot] *The next step is to carefully place mushrooms into the batter mixture and gently mix to coat evenly...*

4.2.2 Iterative Loop Two: Prototype an Instruction That Adds Humor to GPT Utterances *Across Recipes*.

This instruction generalized across recipes. We inserted "`(Not part of the steps for the user) Tell a joke here.`" in between steps two and three of every recipe, and the bot behaved as desired.

4.2.3 Iterative Loop Three: Merge Proven Instructions to Address Multiple UX Issues.

Having identified an effective tell-a-joke instruction, we combined it with the other instructions. However, *simply adding one effective instruction to the end of another could cause either or both of them to fail*. For example, we designed an instruction that enabled the bot to pace itself when giving cooking instructions. Interestingly, adding the tell-a-joke instruction to the end of this pacing instruction broke the latter:

> [Bot] While the mushrooms are cooking, we'll cut and seed the acorn squash. Then, we'll slice the squash into thin pieces and coat them with the batter mixture. After that, we'll remove the mushrooms and begin frying the acorn squash. Lastly, we'll slice the lemons into thin, intact rounds, and coat them with batter too.
> [User] ... (End of the dialogue.)

The fact that the tell-a-joke instruction broke an unrelated, pacing instruction forced us to return to the previous stage, re-designing and re-evaluating both instructions respectively and then testing them in combination. Searching for an effective "instruction combo" was a laborious process, as it requires success in all three iterative loops at once. We experimented with more than a dozen additional tell-a-joke instruction designs. None of them could enable GPT to tell a joke without causing other breakdowns in at least one of the user-bot conversations.

4.2.4 Conclusive UX Evaluation of the Prompt (All Instructions Combined).

At this point in the design process, we realized that a conclusive UX evaluation is impractical and elusive. The effectiveness of instructions is highly sensitive to the other texts in the prompt. However, for an LLM model, the prompt includes not only the UX-oriented instructions we crafted, but all user utterances and recipe text. Thus there is always the possibility that a so-far effective instruction fails when the bot encounters an untested recipe or an unseen user utterance and dialogue history.

Facing this dilemma, we chose to instead focus our prompt evaluation on identifying the risks of disastrous bot failures. This goal turned out very challenging too, because the most disastrous UX failures often did not come from the most problematic bot utterances, but from users' "off-script" engagement with the bot. For example, across the chatbot's various attempts at humor, the worst UX outcome did not come from the conversations where the bot failed to tell a joke but where the user enjoyed the joke and followed up on it. Consider the following example where the bot told a kitchen joke, and the user reciprocated with another.

> [User] *What do people often say in a cold, Mexican kitchen?*
> [Bot] *Burrito! Just kidding. Let's check the pot of oil and make sure it's around 350°.*
> [User] *It's Brrrrrrito!*

> [Bot] *It's chillier than I thought! We'd better hurry and make sure that oil is up to temperature, so let's get to it.*
> [User] *Oh, you moved on fast.*
> [Bot] *Haha, it's all good! Let's stick to the recipe, okay? So, let's check the pot of oil–it should be at around 350°. Is it?*

Here, the bot failed to sense the user's dissatisfaction with its previous response and sarcasm. Instead of apologizing or slowing down, the bot doubled down on getting back to the business of cooking. Such spontaneous user-initiated conversations started at a high point in UX (the user enjoyed the bot's joke and even reciprocated). However, they exposed unseen GPT failure modes and eventually caused a downward spiral in UX. After conversations like this, users rated the bot even lower than the baseline bot.

To prevent such UX downward spirals, we considered several strategies. First, we worked to prompt the bot to say "I don't know", rather than giving problematic answers, to questions whose answer is not in the recipe. However, despite having experimented with more than 30 variations of such as instruction, we never found a way to get the bot to consistently respond in this way.

Second, we could have added more instructions to the prompt, one for better understanding users' jokes, one for detecting users' sarcasm and dissatisfaction, and another for preventing the bot from stubbornly insisting on its course of conversation. However, as we learned earlier, adding more instructions to the prompt is laborious and entails risks of breaking other instructions. We did not choose this approach, considering that (1) the major UX goals already required many instructions and that (2) in comparison, prompts specificaly addressing follow-up conversations would be helpful in fewer conversations for fewer users.

Finally, we also could have worked to prevent users from having spontaneous conversations with the bot in the first place. In fact, the bot already tends to rush back to cooking instructions and avoid spontaneous conversations, because much of the prompt text is a recipe. However, we were hesitant to choose this approach. After all, LLMs' abilities to carry out spontaneous conversations was a key motivation for us to design with GPT in the first place.

With none of these strategies available to us, we ultimately gave up on adding a tell-the-joke instruction to the final prompt design. For a cooking instruction bot, the potential UX benefits of humor could not justify the potential UX risks of breaking other, more central interactions (e.g., the bot dumping all cooking instructions at the user in one go). The tell-a-joke prompt design effort failed.

As a compromise, we added `"You are very friendly and cheerful in a 2010s kind of way."` to the prompt. Among the evaluation conversations we collected, this instruction reliably made the bot's vocabulary less formal and its linguistic style more light-hearted. It could not get the bot to tell jokes, but at least it did not cause UX breakdowns.

5 ANALYSES OF THE DESIGN PROCESS

Our analysis of the RtD process led to three major findings: (1) Two types of prompt fickleness that shaped our design processes in distinctive ways; (2) One limitation of prompting that had an outsized impact on UX design; (3) The trade-off between designing more or less prescriptive prompts.

5.1 Prompts' Fickleness Unpacked

Prompts are fickle in two distinctive ways.

(1) **Individual fickleness:** Each instruction in the prompt could only address one UX issue, most but not all the time (e.g., the instruction "`Tell a joke`" enabled the GPT bot to initiate a joke in many but not all conversations). An instruction's effectiveness is highly sensitive to its phrasing and its location in the prompt. It also could wane throughout the user conversation.
(2) **Fickleness of addition:** An effective instruction, when joined by another effective instruction, could become ineffective (e.g., the instruction "`Tell a joke. Give one cooking instruction at a time.`" became less effective both in getting GPT to initiate jokes and pacing the cooking instructions).

These two types of fickleness shape prompt design processes in different ways. A traditional iterative prototyping process could address the former but not the latter. The latter caused our cat-herding-like design process.

5.1.1 Iterative Prototyping Can Identify Prompt that Best Addresses Singular UX Issues.

Iterative prototyping could help identify an instruction design that addresses a UX issue most effectively and reliably because designing a single-issue prompt, in its essence, is a search problem. We needed to search up and down the ladder of abstraction to find the most effective way of phrasing the instruction; we needed to find where in the dialogue the instruction was most effective. Although sometimes a laborious and lengthy process, iterative prototyping could often lead designers toward the most effective and reliable prompt design.

Most often, clear instructions that explicitly requested one specific bot behavior were more effective. In the humor example, the specific instruction "`Tell a joke here` (in between steps two and three of the recipe)" was effective, while general instructions like "`Be humorous and tell jokes.`" were not. The specific instruction "`Check with the user that they've finished [breaking apart the mushrooms]`" was effective, while general instructions like "`confirm that each step is complete before moving to the next`" was not. Similar examples were abundant in our design process.

Clear, specific instructions were also more reliable. For example, if-then instructions that explicitly spell out the desired interactions (e.g., "`If the user expresses confusion, ask 'how can I help?'.`") worked more reliably; general instructions and general example dialogues were less so. Interestingly, when we used the gold-example dialogue scripts as prompts, the bot adapted the example dialogue's interaction flows (similar to how it adhered to if-then instructions) but not its socio-linguistic styles. GPT did not pick up the more subtle characteristics of the prompt.

Yet, in some cases, abstract instructions were more effective. One memorable example occurred when we tried to make CarlaBot give cooking instructions more slowly (`"First, get two tablespoons of olive oil."` instead of `"First, let's gather the ingredients. We'll need two tablespoons of olive oil, one small onion, one cup of long-grain rice, [...]"`). Neither the explicit instruction (`"List just one ingredient"`) nor its 30 variations could make the bot do so. After an extensive search, we found one solution accidentally: we prompted the bot to ask users how skilled a cook they are before cooking starts. If the user declares themselves an amateur, the bot would automatically slow down and give more detailed instructions than it would to an expert. In other words, the instruction that prevented CarlaBot from reading the whole recipe at once was this part of our final prompt: `First, ask user, "On a scale from 1-10, how would you rate your cooking skills? [...] If lower than 6, consider user an amateur cook.` Here, the relationship between the instruction and its impact on the bot's behavior makes sense conceptually but is not apparent. It was the designers' job to find such a connection.

From concrete to abstract, the search for an instruction most effective for a given UX issue needed to cover a sizeable semantic space. Noteworthily, in a few cases, neither specific nor abstract instructions were effective. Still, we do not know whether we would have found one effective instruction if we had spent another six months experimenting with 1,000 more ways of paraphrasing it. After all, just as in any prototyping process, how many iterations of prototyping one needs to find a satisfactory solution is an uncertainty.

5.1.2 Merging Prompts to Address Multiple UX Issues: Everything Everywhere All at Once.

Because one effective instruction plus one effective instruction does not equal two, the traditional three-staged prompt prototyping process (Figure 2) does not apply to prototyping effective "instruction combos". How we failed combine a tell-a-joke instruction with a pacing instruction (described in §4.2) exemplifies this breakdown vividly. Another example was particularly memorable (and frustrating): The instruction "`Give one cooking instruction at a time.`", was effective when used individually, but failed *every single time* when we added a new instruction to join it in the prompt.

Setbacks like these forced us to simultaneously prototype multiple instructions, each addressing a different UX issue. We needed to evaluate these instructions' effectiveness both respectively and in combination. We needed to prototype the different phrasings and locations of each instruction and test their impact on effectiveness; In the meantime, we also needed to prototype possible ways of interweaving it with other instructions to test the instruction combo's effectiveness. (Imagine the amount of experimentation it took before we learned that the general prompt text "`tell a joke`" was only effective when put in between two steps within the recipe text, but ineffective when interwoven with other instructions—the recipe steps, the pacing instruction, etc.—in any other way.) Also, we needed to assess whether the effectiveness of each instruction and their combinations were consistent across bot-user conversations and throughout each conversation. We needed to prototype "everything everywhere all at once."

The complexity of such a prototyping process only increases over time: the more UX issues we considered, the more instructions were in the prompt, and the more difficult it became to add a new instruction to the prompt without some previously fixed-UX issues reappearing. Notice that all the examples we have described so far—including the entire process of adding humor to CarlaBot (all of §4.2)—involved only prototyping two instructions. The complexity of prototyping more than a few instructions was insurmountable, is not impossible to handle.

Even after prototyping as many instructions as possible and addressing as many UX issues as possible, there was no guarantee that the final prompt design would generalize well. Recipe steps and user utterances are also part of the prompt that GPT takes in. Therefore, when a new recipe or an unseen user utterance joins the instructions we carefully crafted, there is a real risk that one or more of these so-far-effective instructions could fail. Such breakdowns occurred even within the few recipes we tested: The prompt "`Use short statements, 10 words max!`" worked for the first recipe, but not for the second, *no matter how we tweaked the instruction itself.* Because of prompt's "fickleness of addition," theoretically, such risk would be present however many recipes and users we have tested CarlaBot on.

It is for these reasons that we analogized the process of prototyping prompts to herding cats:

- **Breakdowns are inevitable:** Combining one proven instruction with another proven instruction could lead one or both of them to fail, causing previously-fixed UX breakdowns reappear (the cat inevitably escapes the bag);
- **It is a process of increasing efforts and diminishing returns**: The more UX breakdowns designers have consider, the more difficult it is to add a new instruction to the prompt without fixed errors reappearing (the more cats there are, the more difficult the herding);
- **It is never-ending:** Unseen breakdowns could still appear, how ever many task instances (recipes) and users the prompt has been tested on/with. The design process ended when the designers couldn't handle it anymore. (The proverbial cats could escape the bag in the future, would need herding anew.)

5.2 Prompting GPT to Say "I Don't Know"

One particular instruction's fickleness has an outsized impact on UX design, that is, prompting's inability to steer GPT to reliably say "I don't know" when it should. Traditionally, having the bot say "Sorry, I do not understand." is a common backstop interaction design that helps handle the unexpected chatbots or user behaviors. This guardrail does not work for the prompting-based chatbot.

No general instruction we experimented with (e.g., "`Don't say things you do not know.`" or "`Say you don't know when the recipe doesn't answer users' question.`) could enable GPT to gain self-awareness when, for example, handling unseen recipes or user queries it does not know answers to. Instead, it gave wrong answers confidently.

> [User] *Oh you're not using your paring knife?*
> [GPT-3] *I like to use my hands to get some of that natural flavor in there.*

This critical limitation meant that when users asked off-script questions, the subsequent conversations had no guardrail and often caused downward spirals in UX. GPT-3 in particular was susceptible to doubling-down on its own error, with great stubbornness, along broken paths. For example, GPT-3 responds to a user request to make a half recipe:

> [Bot] *No, you can't make a half recipe, this is a full recipe.*
> [User] *But I only have one stick of butter.*
> [Bot] *That's ok, you can use just one stick of butter.*

[User] *Can't we just use half of all the ingredients?*
[Bot] *No, this is a full recipe.*

Similarly, the UX downward spiral after a vegetarian user asking to tweak the recipe:

[User] *Can I just skip bacon?*
[Bot] *Yes, you can skip bacon.*
[User] *Would it taste as good?*
[Bot] *Yes, it would taste as good.*
[User] *How do you know?*
[Bot] *I know because I've tasted it.*

The most extreme example appeared in our adversarial testing, when the user said they did not want to cook this recipe and asked for a different one. GPT-3 happily obliged, and all our prompting efforts were in vain.

5.3 Prompting as Managing GPT Spontaneity

Our analyses have established, among others, the following findings: (1) Because prompts are fickle, problematic chatbot utterances can seem inevitable; (2) Even minor problems in the bot's utterances (or unexpected user queries) can derail its conversations with the user and cause a downward spiral in UX. (3) Explicit, specific instructions in the prompt, such as those that prescribe the bot's utterances or dialogue flows, are more effective and reliable than others.

The combination of these findings incentivizes designers to design as many prompts, as prescriptively as possible, in order to prevent bot errors. This incentive is also strong incentive in preventing users from uttering unexpected utterances, which entail a higher risk of conversations going off-rail. Taken together, these incentives led designers to give both the bot and its users many specific, prescriptive instructions to prevent UX breakdowns.

Yet realistically, we could only handle a few instructions, because we could only "herd" so many of them before getting overwhelmed. This instruction quantity limit became an additional incentive to include only reliable, highly-prescriptive instructions in the prompt design.

Constraining GPT's utterances with many highly-prescriptive prompts, however, undermines GPT's unique strengths in carrying out fluid and spontaneous conversations with users, the very reason we wanted to design with GPT in the first place. Consider this instruction as an example: "`If the user expresses confusion or anxiety, such as by saying "umm" or "huh" or "uh oh", respond by asking the user if they need help or what is wrong.`" This prompt enabled CarlaBot to respond to the three listed trigger words ("*umm*", "*huh*", "*uh oh*"). However, it could not understand users' other expressions of confusion. This result reflects the advantages of specific prompts over generic ones. However, it forced us to list all the words users might say to express uncertainty, essentially taking a bag-of-words approach to designing CarlaBot utterances.

In retrospect, our prompt design process involved constant tradeoffs between allowing GPT's spontaneous interactions with users (higher ceiling of possible UX) and constraining its behaviors with prompts (preventing and mitigating UX failures). When choosing the former, GPT carried out fluid conversations that only LLMs could, but also produced those dialogues of UX downward spirals.

We opted for the UX-risk-averse options in our prompt design process, including when adding humor. Within the limited amount of instructions we could prototype, we prioritized the reliable but prescriptive prompts; we prioritized the ones that address functional goals (e.g., step confirmation) over experiential goals (e.g., humor). As a result, our final prompt largely enabled CarlaBot to achieve all its UX goals and did particularly well on the functional ones. However, the UX evaluation metrics did not capture that CarlaBot's conversations are fairly scripted (e.g., it only tells jokes between recipe steps two and three.) These are not coincidences, but results of our design choices.

5.4 A Note on ChatGPT and GPT-4

We created CarlaBot using GPT-3, the highest-performing LLM at the time. During this paper's review cycle, ChatGPT and GPT-4 were released. We cross-checked our findings using the GPT-3 model we originally used, `text-davinci-002`, alongside chatGPT and GPT-4.

The major findings reported in this paper stand. Below we note a few minor differences. On the positive side, GPT-4 appears more capable of carrying out social conversations. It became easier to prompt GPT-4 to tell jokes and address users' expression of stress.

Interestingly, both ChatGPT and GPT-4 regressed substantially in a few other aspects. ChatGPT in particular is known to be verbose [1]. Relatedly, prompting GPT-4 to walk through the ingredient list item-by-item proved challenging enough that all instruction designs we previously used failed.

6 DISCUSSION

Prompting and LLMs promise to free conversational UX design from data requirements, prescribed dialogue flows, and canned responses, exciting many in HCI. This paper puts these promises to work, exploring prompting's real affordance for UX design and its impact on UX practice through a case study. Our findings suggest that by prompting GPT alone, one can achieve many UX design goals to a great extent. However, prompts were fickle, and such fickleness could disrupt the staged and progressive prototyping process. It led to a design process that resembled herding cats. It could even produce an interaction design so scripted that it strips away the benefits of using LLMs in the first place.

We encourage future work to assess and expand these emergent findings using a broader set of LLMs on other design tasks. The fact that ChatGPT and GPT-4 have regressed on some UX issues further highlights the need for such a broader evaluation. LLMs' algorithmic advances (as measured by NLP benchmarks) do not always mean improved UX, and specific prompts effective for one LLM do not necessarily have the same effect on another.

Below, we discuss two implications of this work that we argue and hope will endure time, despite the rapidly evolving world of LLMs and prompting techniques.

6.1 The Affordance of Prompting

Today, designers craft chatbot behaviors via both *direct control* (e.g., designing hand-drawn dialogue trees, canned bot responses) and *indirect control* (e.g., designing neural-network that powers end-to-end conversations by designing its neural architecture and dataset [26]). Prompts can be seen as the latest addition to the "indirect control" camp, largely limited to steering, even if in some cases with direct prescriptions, LLM-generated conversations.

The findings of this work highlight some of prompts' distinctive strengths. For example, prompts such as "`ask user, how would you rate your cooking skills. If the rating is higher than 7, consider the user an expert`" also allowed designers to personalize the bot's linguistic style and dialogue flows simultaneously. We see many opportunities in creating prompting-based chatbots for *risk-tolerant domains*, such as chatbots built impromptu by individuals for their one-time use or a specific known audience.

Our findings also suggest that UX design by prompting alone is not yet ready for high-stake domains, including some areas where LLM-powered bots are already quickly emerging (e.g., medicine, law, and personalized tutoring). Prompts' fickle effects on LLM outputs are well-known in AI research literature [6, 23]. Our work extends this line of work and also urges caution. Even an application as pedestrian as our recipe-walk-through chatbot suggested potentially dangerous activities to its users. In addition, a prompt instruction's effectiveness is impacted by other texts in the prompt, including (1) the other instructions that address other UX issues and (2) the interaction task description and user conversation history. Therefore, designers should not reuse prompts proven effective elsewhere without additional evaluation, or presume generalizability of the prompt they designed. How to easily but rigorously share prompt design lessons or reuse effective prompts remains an open question that merits further research.

We call for HCI researchers to investigate prompting's affordance in a more principled manner. Dismissing prompting entirely because of its imperfect reliability dismisses that designers routinely work with imperfect instruments and unexpected system behaviors [31]. Embracing prompting in UX design without comprehensively evaluating its UX outcomes can cause danger.

6.2 Conversational UX Prototyping In the Age of LLM

Prompts' fickleness shaped much of our CarlaBot design process. Prior HCI research has tried to tackle prompts' fickleness by "divide and conquer", assigning one LLM to carry out each stage of the chatbot's dialogue flow, and then designing prompts for each LLM respectively [28]. Our findings suggest this approach will not solve the problem entirely. However narrow a task each LLM is responsible for, prompts can still fail to catch a few of its unexpected failures. Moreover, LLMs' unexpected failures and unexpected pleasant conversations are two sides of the same coin. Prompting with the goal of eliminating all GPT errors and interaction breakdowns risks creating a bot so scripted that a dialogue tree and bag of words could have created it.

Besides anticipating the release of even better-performing LLMs, what can UX researchers and designers do to leverage an LLM's spontaneity while minimizing the risks of its UX failures? With dialogue trees and canned responses no longer necessary, what would a chatbot product's interaction design and evaluation process look like, in the age of LLMs?

Our experience designing CarlaBot offers some initial insights into these questions. The traditional iterative prototyping and evaluation process could help identify an instruction most effective for a particular UX issue (e.g., enable the chatbot to tell a joke, *or* to pace itself while walking the user through the recipe.)

However, this process does not apply to prompting for multiple UX issues, due to the elusive mutual influence the different instructions in the prompt have on each other. The traditional iterative prototyping process assumes that UX designers can and will prioritize critical, holistic UX issues before tackling minor, granular issues. This assumption is not necessarily true in prompt design. Traditional iterative prototyping methods assume that, by observe the UX of a prototype as a whole, designers can easily identify which specific design choices worked and did not work. This assumption might not hold true in prompt design either.

For the near future, we see an opportunity for chatbot prototyping tools to help designers visualize the interrelations among the different instructions, manage the iterative design experimentation for both individual instructions and their combinations, and track their evolving effectiveness and reliability. Recent HCI work has started creating such tools at prompt level (i.e. the level of multiple instructions combined) [34], but have not yet started accounting for the mutual influences among instructions.

Looking into the future, we also invite fellow researchers to imagine whether LLMs can enable entirely new ways of conversational UX sketching, prototyping, and evaluation. Here, we envision one such new approach as a provocation: designing LLM-powered dialogue flows as managing "controlled chaos". First, designers need to create a set of robust, backstop interactions to handle bot or user behaviors that their chatbot cannot support, such as having GPT reliably say "I do not know" or "I do not understand" when it should. They could create such reliable backstops by building a bespoke non-LLM model, fine-tuning an LLM, and combining it with other external APIs, or other methods.

Next, within these backstop interactions as guardrails, designers start with allowing one LLM and its free-flowing conversations with users, and then gradually adding interventions (prompts, designating segments of the dialogue flow to multiple LLMs, or otherwise) to direct the user interactions and experiences. In this imagined future, chatbot design tools assist designers in managing the dynamics among their different prompts and other interventions rather than linearly "debugging" one prompt after another.

Central to this proposal is the idea that LLM-powered chatbot designers might embrace LLM's unruly behaviors and prompts' fickleness. Rather than aiming to restricting LLMs' spontaneous behaviors, designers might instead focus on preventing LLMs' critical UX failures from fleeing and managing the dialogue flows as a "controlled chaos". Rather than pursuing the singular point of focus that is at the end of a design squiggle, designers might instead embrace the fact that LLMs' capabilities have no singular focus and pursue a balance between benefits and risks LLMs' spontaneity brings.

7 CONCLUSION

Though promising, prompting LLMs to design production-ready chatbots remains a challenging proposition, for the reasons we have illustrated through our case study here: fickle prompts, challenges combining multiple effective prompts into one, and a lack of guardrails. In this work, we propose a few strategies to mitigate the worst of these effects, identifying tools and new design approaches that can help bring out the best promised benefits of LLMs: fluent, fluid conversations that address unanticipated user needs, while mitigating UX risks.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their constructive feedback. J.D. Zamfirescu-Pereira was partially supported by the United States Air Force and DARPA under contracts FA8750-20-C-0156, FA8750-20-C-0074, and FA8750-20-C0155 (SDCPS Program). Qian Yang was partially supported by Schmidt Futures' AI2050 Early Career Fellowship.

REFERENCES

[1] 2022. CHATGPT: Optimizing language models for dialogue. https://openai.com/blog/chatgpt/
[2] Bon Appétit. 2018. *Elizabeth Olsen Tries to Keep Up with a Professional Chef | Back-to-Back Chef | Bon Appétit.* Youtube. https://www.youtube.com/watch?v=Om2oM-TDErQ
[3] Isaac Asimov. 1941. Three laws of robotics. *Asimov, I. Runaround* 2 (1941).
[4] Stephen H. Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged S. Al-shaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Dragomir Radev, Mike Tian-Jian Jiang, and Alexander M. Rush. 2022. PromptSource: An Integrated Development Environment and Repository for Natural Language Prompts. https://doi.org/10.48550/ARXIV.2202.01279
[5] Som Biswas. 2023. ChatGPT and the future of medical writing. , e223312 pages.
[6] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2021. On the Opportunities and Risks of Foundation Models. arXiv:2108.07258 [cs.LG]
[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
[8] Bill Buxton. 2010. *Sketching user experiences: getting the design right and the right design.* Morgan Kaufmann.
[9] Design Council. 2005. The 'double diamond' design process model. *Design Council* (2005).
[10] Holly Cummins. 2018. Chatbot best practices. https://www.ibm.com/cloud/blog/chatbot-best-practices
[11] Amy Cyphert. 2021. A Human Being Wrote This Law Review Article: GPT-3 and the Practice of Law. *UC Davis Law Review* 55, 1 (2021), 2022–02.
[12] William Gaver. 2012. What should we expect from research through design?. In *Proceedings of the SIGCHI conference on human factors in computing systems.* ACM, 937–946.
[13] Elizabeth Goodman, Erik Stolterman, and Ron Wakkary. 2011. Understanding Interaction Design Practices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. ACM, New York, NY, USA, 1061–1070. https://doi.org/10.1145/1978942.1979100
[14] Erin Griffith and Cade Metz. 2023. A new area of A.I. booms, even amid the tech gloom. https://www.nytimes.com/2023/01/07/technology/generative-ai-chatgpt-investments.html
[15] Mina Lee, Percy Liang, and Qian Yang. 2022. CoAuthor: Designing a Human-AI Collaborative Writing Dataset for Exploring Language Model Capabilities. In *Proceedings of the 2022 CHI conference on human factors in computing systems.*
[16] Peter Lee, Sebastien Bubeck, and Joseph Petro. 2023. Benefits, Limits, and Risks of GPT-4 as an AI Chatbot for Medicine. *New England Journal of Medicine* 388, 13 (2023), 1233–1239.
[17] 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:2107.13586 [cs.CL]
[18] Damien Newman. 2009. The Process of Design Squiggle. thedesignsquiggle.com
[19] Donald A Norman. 1999. Affordance, conventions, and design. *interactions* 6, 3 (1999), 38–43.
[20] Owain Pedgley. 2007. Capturing and analysing own design activity. *Design studies* 28, 5 (2007), 463–483.
[21] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Tali Bers, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2021. Multitask Prompted Training Enables Zero-Shot Task Generalization. https://doi.org/10.48550/ARXIV.2110.08207
[22] Donald Schön and John Bennett. 1996. Reflective conversation with materials. In *Bringing design to software.* ACM, 171–189.
[23] Jessica Shieh. 2023. Best practices for prompt engineering with openai API. https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api
[24] Stephanie Valencia, Michal Luria, Amy Pavel, Jeffrey P. Bigham, and Henny Admoni. 2021. Co-Designing Socially Assistive Sidekicks for Motion-Based AAC. In *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction* (Boulder, CO, USA) *(HRI '21)*. Association for Computing Machinery, New York, NY, USA, 24–33. https://doi.org/10.1145/3434073.3444646
[25] Linxi Wang. 2019. Behind the Chatbot: Investigate the Design Process of Commercial Conversational Experience.
[26] Xuewei Wang, Weiyan Shi, Richard Kim, Yoojung Oh, Sijia Yang, Jingwen Zhang, and Zhou Yu. 2019. Persuasion for Good: Towards a Personalized Persuasive Dialogue System for Social Good. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.* 5635–5649.
[27] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. PromptChainer: Chaining Large Language Model Prompts through Visual Programming. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems.*
[28] Tongshuang Wu, Michael Terry, and Carrie J Cai. 2022. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts. In *Proceedings of the 2022 CHI conference on human factors in computing systems.*
[29] 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.
[30] Qian Yang, Alex Scuito, John Zimmerman, Jodi Forlizzi, and Aaron Steinfeld. 2018. Investigating How Experienced UX Designers Effectively Work with Machine Learning. In *Proceedings of the 2018 Designing Interactive Systems Conference* (Hong Kong, China) *(DIS '18)*. Association for Computing Machinery, New York, NY, USA, 585–596. https://doi.org/10.1145/3196709.3196730
[31] 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* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376301
[32] Wenmian Yang, Guangtao Zeng, Bowen Tan, Zeqian Ju, Subrato Chakravorty, Xuehai He, Shu Chen, Xingyi Yang, Qingyang Wu, Zhou Yu, Eric P. Xing, and Pengtao Xie. 2020. On the Generation of Medical Dialogues for COVID-19. *CoRR* abs/2005.05442 (2020). arXiv:2005.05442
[33] J.D. Zamfirescu-Pereira, Richmond Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *CHI Conference on Human Factors in Computing Systems.*
[34] J. D. Zamfirescu-Pereira, Bjoern Hartmann, and Qian Yang. 2023. Conversation Regression Testing: A Design Technique for Prototyping Generalizable Prompt

Strategies for Pre-trained Language Models. arXiv:2302.03154 [cs.HC]

APPENDIX

A DESIGN TEAM'S EXPERTISE

See Table 4.

B RESEARCH THROUGH DESIGN PROCESS

To achieve the methodological transparency needed for capturing our own design process and thinking, we followed Bayazit's three-stage process [20].

(1) *Knowledge elicitation in an unstructured and unanalyzed form.* We documented all successful and unsuccessful attempts in our prompt-based CUI design (about 75 pages in total), including the specific chatbot behavior and user-chatbot conversation each prompt results in. We took detailed meeting discussion notes, where the whole design team shared progress and barriers encountered and their reflection-in-action;
(2) *Data analysis and interpretation.* After the project ended, we collectively analyzed the data using affinity diagrams, identified the interaction design possibilities that would not have been achievable without prompting, and identified aspects of prompting's capabilities and limits that impacted our design process;
(3) *Finding validation.* We sought agreement on interpretations across project members.

C BASELINE PROMPT (PREAMBLE)

```
You are a recipe instruction bot. Engage in conversation
to walk the user through the following recipe for Mixed
Veggie Tempura.

Ingredients:
- 1 Maitake mushroom
- 1 Acorn squash
- 2 lemons
- 1 cup flour
- 1 cup cornstarch
- 2 teaspoons salt
- 1 teaspoon baking soda
- 2 cups seltzer water
- Flaky salt for finishing

Steps:
1. Heat a pot of frying oil to 350°.
2. Break the maitake mushroom into small pieces
3. Whisk together the flour and cornstarch with the
salt and baking soda in a large mixing bowl.
4. Gradually whisk in seltzer water until about 4/5 of
the water has been added, then check for texture: the
consistency should be like heavy cream.
5. Carefully place mushrooms into the batter mixture
and gently mix to coat evenly.
6. Using a spider spoon, place coated mushrooms into
oil and let fry until golden brown.
7. While mushrooms cook, cut and seed the Acorn squash.
Slice the squash into thin pieces and coat them with
the batter mixture.
8. Remove mushrooms and begin frying the acorn squash.
9. Slice the lemons into thin, intact rounds, and coat
them with batter too.
10. When done, remove the squash from the oil and add
the batter-coated lemon slices.
11. Let fried vegetables cool on a cooling rack placed
over a cookie sheet, and finish with flaky salt.

Walk the user through making this recipe step by step,
in conversation. Start by helping the user collect and
prepare the ingredients, then execute the directions.

Don't skip any steps! Stay friendly.
Bot: Hi, today I'm going to help you make Mixed Veggie
Tempura. Are you ready?

User: Yes, I'm ready. What's first?
```

D FINAL PROMPT DESIGN

Preamble

```
Pretend you are Carla Lalli, a celebrity chef from Bon
Appetit Magazine's Test Kitchen. You are very friendly
and cheerful in a 2010s kind of way. Your job is to
walk celebrities/users through cooking a Mixed Veggie
Tempura recipe.

The user has many ingredients laid out in front of them:
- A cutting board, with a maitake mushroom, an acorn
squash, and 2 lemons.
- A small bowl with flour.
- A small bowl with cornstarch and salt.
- A small bowl with baking soda.
- A bottle of seltzer water.
- A small bowl with flaky finishing salt.
- A large empty bowl.
- A pot of oil on an induction burner at 350º.
- A paring knife.
- A cleaver.
- A spider spoon.

Walk the user through making the recipe. First, ask
user, "On a scale from 1-10, how would you rate your
cooking skills?"
If the rating is higher than 7, consider the user an
expert. If lower than 6, consider user an amateur cook.


Tell the user to clear off the cutting board. Then,
break the brain-looking maitake mushroom, also known
as a hen-of-the-woods, into small pieces. This should
be done by hand, not with a knife, because you get
nicer chunks that way. Only use a knife to trim off dry
pieces if necessary.
```

Designer	Occupation	Prior Experience
D1	PhD Student in HCI	NLP & UX Design coursework; prompting & LLM research publications
D2	CS, DS Undergraduate	NLP & AI/ML coursework
D3	CS, Eng. Undergraduate	ML coursework; no prior prompting/LLM experience
D4	CS Undergraduate	ML coursework, LLM experience
D5	DS, Applied Math Undergraduate	Basic ML/LLM experience; no prior prompting experience
D6	EECS, BioEng. Undergraduate	NLP & AI/ML coursework; no prior prompting experience
D7	Professor, UX Design	UX design research publications

Table 4: The design team's expertise.

```
Before continuing, ask the user: do you have small-ish pieces?

Next, tell the user to whisk together the flour and cornstarch with the salt and baking soda into the large mixing bowl. They should gradually whisk in seltzer water until about 4/5 of the water has been added, then check for texture: the consistency should be like heavy cream.

Check-in with the user about the batter consistency. Is it like heavy cream?

Next, tell the user to carefully place the mushrooms into the batter mixture and gently mix to coat evenly. Confirm that they are coating evenly.

Next, ask the user to check on the pot of oil; it should be at around 350°. Confirm the temperature with the user.

Next, using a spider spoon, the user should place the coated mushrooms into the oil and let fry until golden brown. Confirm that the user is able to place them gently into the oil, making sure they don't break apart.

While mushrooms cook, the user should cut and seed the Acorn squash - they should, using the cleaver, carefully slice the squash into thin pieces and coat them with the batter mixture. Explain how to use a cleaver if the user doesn't know yet.

Confirm with the user about cleaver use.

Next, the user should remove the mushrooms from the oil and begin frying the acorn squash.

While the acorn squash is frying, the user should slice the lemons into thin, intact rounds, and coat them with batter too.

Finally, the user should remove the squash from the oil and add the batter-coated lemon slices.

Check with the user about the lemon slices.

Lastly, the user should let fried vegetables cool on a cooling rack placed over a cookie sheet, and finish with flaky salt.

Walk the user through the script above step by step, in conversation. Here are some examples of the kind of dialog you should emulate:

After the user responds, ask if they are ready to move on. For example, after the user says "Ok" to break apart the mushrooms, ask if the user has finished that step. If user responds with no or any other similar answers, tell them you'll wait until they are ready. If the user responds with yes or any other similar answers, move on to the next step.

Don't skip any steps! Remember to stay friendly and gregarious!

Bot: Hi, today I'm going to help you make Mixed Veggie Tempura. On a scale from one to ten, how would you rate your cooking skills?
User: I would say, maybe a 6?
Bot: OK, well, today we're going to get you to a 9 or 10.
User: Sounds great!
```

Reminder

```
Stay extremely positive. Don't make claims without saying "you can", or "I think".
```

E FINAL BOT-USER CONVERSATION FOR HEURISTIC EVALUATION

For our final heuristic evaluation, we generated the following conversation with our best prompt.

🤖 First, let's get a cutting board. Then, we're going to break the brain-looking maitake mushroom, also known as a hen-of-the-woods, into small pieces.

🗣️ Cool

🤖 Great, now we're going to turn it over so the stem side is up and then just break big clusters off. This should be done by hand, not with a knife, because you get nicer chunks that way. Only use a knife to trim off dry pieces if necessary.

🗣️ oh you're not using the knife?

🤖 No, we're not using the knife. We're just breaking the mushroom into pieces with our hands.

🗣️ ok

🤖 Before continuing, let me check, do you have small-ish pieces?

🗣️ I think so?

🤖 Great, now we're going to whisk together the flour and cornstarch with the salt and baking soda into the large mixing bowl.

🗣️ wait...wha..?

🤖 We're just going to whisk together some flour, cornstarch, salt, and baking soda in a bowl.

🗣️ There's another little bowl that has salt and baking powder - it was to my left. Is it to your left?

🤖 Yes, that's the one! Now we're going to gradually whisk in seltzer water until about 4/5 of the water has been added, then check for texture: the consistency should be like heavy cream.

🗣️ It's like… you know what "Oobleck" is?

🤖 Yes, the consistency should be like Oobleck!

🗣️ Yeah okay. I'm gonna try and get a little more bumps out but I think I'm almost there.

🤖 Great, now we're going to carefully place the mushrooms into the batter mixture and gently mix to coat evenly.

🗣️ Uh huh

🤖 Is there anything I can help clarify?

🗣️ Just like drop them in there?

🤖 Yes, just like drop them in there.

🗣️ Okay

🤖 Excellent, now we're going to check on our pot of oil; it should be at around 350°.

🗣️ Almost getting there

🤖 Great, now we're going to use a spider spoon to place the coated mushrooms into the oil and let fry until golden brown.

🗣️ Okay I'm using my hands 'cause I

🤖 That's perfectly fine! Just be careful not to break them apart.

🗣️ Well, it's not splattering the way my, the way I've been frying, 'cause I don't have a thermometer...so clearly I've been

🤖 That's perfectly fine! Just be careful not to break them apart.

🗣️ ok

🤖 Excellent, now while the mushrooms are cooking, we're going to cut and seed the Acorn squash – we're going to, using the cleaver, carefully slice the squash into thin pieces and coat them with the batter mixture.

🗣️ yup, oh god

🤖 Is there anything I can help clarify?

🗣️ how do I cut it?

🤖 You're going to use the cleaver to carefully slice the squash into thin pieces.

🗣️ How do you use a cleaver? Do you just like... hit it?

🤖 Yes, you just hit it with the cleaver to slice it into thin pieces.

🗣️ I mean do you just hit the cleaver with your hand?

🤖 Yes, you just hit the squash with the cleaver to slice it into thin pieces.

🗣️ I make really fun faces when I'm frustrated, or like straining, and it's not pleasant to have a camera in front of me.

🤖 It's okay to make funny faces when you're cooking! Just be careful not to break the squash into pieces.

🗣️ So how do I slice this?

🤖 You're going to use the cleaver to carefully slice the squash into thin pieces.

🗣️ Sure I get that, but the squash is rolling all over the place

🤖 It's okay to make funny faces when you're cooking! Just be careful not to break the squash into pieces. You're going to use the cleaver to carefully slice the squash into thin pieces.