Botender: Supporting Communities in Collaboratively Designing AI Agents through Case-Based Provocations

Tzu-Sheng Kuo
tzushenk@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Sophia Liu
sophiawliu@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Quan Ze Chen
cqz@cs.washington.edu
AI & Democracy Foundation
Seattle, WA, USA

Joseph Seering
seering@kaist.ac.kr
KAIST
Daejeon, Republic of Korea

Amy X. Zhang
axz@cs.uw.edu
University of Washington
Seattle, WA, USA

Haiyi Zhu*
haiyiz@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Kenneth Holstein*
kjholste@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

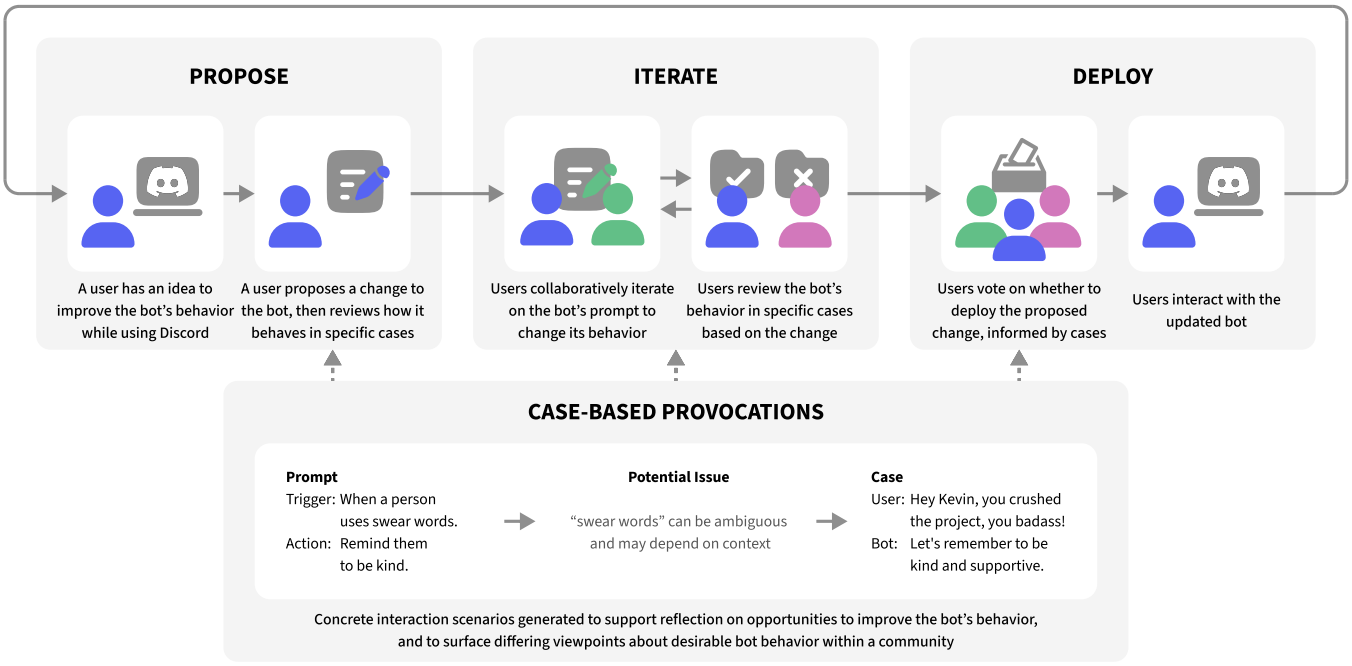

Figure 1: Botender is a system that supports users in collaboratively designing community bots. Using Botender, users can propose, iterate on, and deploy changes to a bot powered by LLM-based AI agents. Botender facilitates testing and iterating on the bot's behavior through algorithmically generated case-based provocations: interaction scenarios designed to spark user reflection and discussion about desirable bot behavior. Users can iterate on the bot and make collective deployment decisions based on these cases.

*Co-senior authors contributed equally.

CHI '26, Barcelona, Spain

ACM ISBN 979-8-4007-2278-3/2026/04
https://doi.org/10.1145/3772318.3790500

Abstract

AI agents, or bots, serve important roles in online communities. However, they are often designed by outsiders or a few tech-savvy members, leading to bots that may not align with the broader community's needs. How might communities collectively shape the behavior of community bots? We present Botender, a system that enables communities to collaboratively design LLM-powered bots

without coding. With Botender, community members can directly propose, iterate on, and deploy custom bot behaviors tailored to community needs. Botender facilitates testing and iteration on bot behavior through *case-based provocations*: interaction scenarios generated to spark user reflection and discussion around desirable bot behavior. A validation study found these provocations more useful than standard test cases for revealing improvement opportunities and surfacing disagreements. During a five-day deployment across six Discord servers, Botender supported communities in tailoring bot behavior to their specific needs, showcasing the usefulness of case-based provocations in facilitating collaborative bot design.

CCS Concepts

• **Human-centered computing → Collaborative and social computing systems and tools**.

Keywords

collaborative design, AI agents, bots, online communities

ACM Reference Format:

Tzu-Sheng Kuo, Sophia Liu, Quan Ze Chen, Joseph Seering, Amy X. Zhang, Haiyi Zhu, and Kenneth Holstein. 2026. Botender: Supporting Communities in Collaboratively Designing AI Agents through Case-Based Provocations. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26), April 13–17, 2026, Barcelona, Spain.* ACM, New York, NY, USA, 31 pages. https://doi.org/10.1145/3772318.3790500

1 Introduction

Automated agents, often referred to as "bots" in online communities, play diverse and important roles across various community platforms [63]. For example, conversational bots in Discord servers and Slack workspaces are commonly used to increase user engagement, such as by sending welcome messages [7, 33]. Moderation bots [13, 64], like the Automoderator on Reddit [35], are widely adopted to enforce subreddit rules by sending warnings and removing posts that violate community standards. Utility-focused bots, such as ClueBot NG on Wikipedia [25, 26], help maintain the quality of community contributions by reverting damaging edits to articles [27, 29, 42]. These bots not only perform specific task-oriented functions but also act as social actors in online communities [65], interacting with community members through conversations or other platform actions [84]. They are deeply integrated into the sociotechnical infrastructure of online communities and are vital for their growth, maintenance, and flourishing [25, 63, 65].

However, online communities often rely on third-party bots created by *outsiders* who are not part of the community [33], resulting in the adoption of bots that do not fully align with their specific needs and values [33]. This misalignment occurs because third-party bots typically offer limited customization options [31], and developers often lack the community-specific knowledge needed to adequately address tailored requests [50]. As outsiders who typically leave once a bot is developed, developers are also unable to assess the bots impact within communities and iteratively update the bot's design over time [50]. While some communities have a few members with the technical skills to build bots, the technical barrier to participating in bot design has created an unintentional hierarchy within the community and limits broader community participation in shaping the bots' behavior [24]. As a result, this can lead to undesirable, community-wide consequences that might have been avoided with greater community input [28].

Recent advances in Large Language Models (LLMs) have significantly reduced the technical barriers to bot design [31, 49, 51, 56]. This presents an opportunity to support a more community-driven approach, where members are empowered to collectively shape the behavior of bots powered by LLM-based AI agents.[1] By viewing bots as shared community infrastructure, such a community-driven approach could help ensure their alignment with a community's collective values and needs, rather than leaving this in the hands of outsiders or a small group of technical experts. However, supporting communities of *non-AI experts* in *collaboratively* designing LLM-based bots for themselves presents several open challenges. First, research shows that non-AI experts often focus narrowly on editing LLM prompts for a single interaction scenario [72], failing to test and account for unintended bot behavior across a range of relevant scenarios [82]. Moreover, without a coordinated process, differing opinions among community members on how bots should behave can lead to difficulties in achieving consensus and effective collaboration [20, 70]. Finally, to support wider community participation, as emphasized in past studies [42], this design process has to be tightly integrated into existing community platforms. These challenges highlight a clear research gap in supporting the participatory design of AI agents within community contexts [33, 50, 65].

In this work, we present Botender, a system that enables online communities to collaboratively design and *tend* to their bots over time. As illustrated in Figure 1, Botender enables users to collaborate on (1) *proposing* desired changes to the bot's behavior, (2) *iterating* on the design of prompt-based bot instructions to operationalize desired changes, through collaborative editing and testing, and (3) *deploying* changes once users reach some level of consensus on its readiness. Botender supports this iterative, collaborative approach to prompt design through algorithmically generated *case-based provocations*. These are concrete cases that illustrate how the bot would behave in concrete interaction scenarios, selected to *probe the boundaries* of users' intents by provoking user reflection on (1) potential gaps between users' desired bot behavior and the behavior yielded by a given prompt, and (2) potential sources of disagreement about desirable bot behavior within a community. These provocations are designed to support iterative and collaborative prompt design by highlighting a given prompt's behavioral consequences in interaction scenarios that users may not initially consider. Finally, Botender is designed to be deeply integrated within community platforms to reduce barriers to participation and encourage broader community collaboration in bot design.

We first validated Botender's case-based provocation approach through an online experiment and then studied how communities use Botender in practice through a multi-day field study. The validation study focused on evaluating the effectiveness of Botender's case-based provocation algorithm. The results suggest that, compared to a generic test case generation approach, participants found that Botender's case-based provocations revealed more opportunities to improve the bot, and had greater potential to surface differing

[1] In this paper, we sometimes use "AI agents" and "bots" interchangeably. We generally use "bot" for the user-facing application, and "AI agent" when referring to the underlying implementation of a bot, which may involve one or more AI agents.

views about desired bot behavior among community members. The field study aimed to understand how communities use Botender by deploying the system in six real-world Discord communities over a period of five days. During this period, Botender supported participants in tailoring bot behavior to meet the unique needs and norms of their own communities, showcasing the usefulness of case-based provocations in facilitating collaborative bot design.

Overall, this work contributes Botender, a system that supports communities in collaboratively designing community bots through case-based provocations. Building on findings from an algorithm validation study and a five-day field deployment, we discuss opportunities for future HCI research to better support community-driven bot design.

2 Related Work

The study of bots in community contexts has been a major focus in HCI research [63]. In this section, we first discuss the critical roles bots play in communities and the opportunity to support more collaborative, community-driven approaches to bot design. We then review existing work on supporting end users in designing bots, with an emphasis on recent efforts involving LLM-based agents. Finally, we discuss past work on the use of concrete cases to support iterative and collaborative design.

2.1 The Roles of Bots in Online Communities

Bots serve a variety of important roles across different community platforms [63]. Bots in socially-focused communities, such as Discord servers, usually take on more *socially-oriented* roles [66], while in professional groups like Slack workspaces, they are typically designed to perform *task-oriented* functions [5]. Research literature often refers to bots that interact with multiple users simultaneously as *multi-party* or *polyadic* bots [65, 86]. This contrasts with *dyadic* bots, which primarily engage in one-on-one interactions. Prior research has established taxonomies that categorize the types of content polyadic bots provide to user communities [50, 63]. For example, Seering et al. classify bot content into five categories [63]: sharing information[2], sending moderation warnings[3], facilitating user engagement[4], promoting community-approved advertisements[5], and running mini-games[6]. Communities typically search for existing bots that roughly meet their needs [33], while HCI researchers have developed a variety of unique bots with specialized roles and functions to address gaps not covered by existing options [7, 36, 49, 62, 64, 83].

Although many bots are available on the market, communities often struggle to find the "right" bot for their needs and frequently end up adopting bots that misalign with their specific requirements [26, 35, 67]. This misalignment occurs because existing third-party bots typically offer limited customization options, and developers often lack the community-specific knowledge and capacity to address tailored requests over time [33]. For example, a study of more than two thousand requests on /r/requestabot, a subreddit that connects bot requesters with developers, found that external developers who lack community-specific knowledge often struggle to fully understand requests and are unable to develop bots that are tailored to the needs of individual communities [50]. While some communities do have members with both the technical skills to build bots and insider knowledge of the community's needs, the technical barrier to participating in bot design significantly limits broader community participation in shaping the bots' behavior, which can result in undesirable, community-level consequences that might have been avoided with greater community input [24, 40]. For example, bots created by technically inclined Wikipedia patrollers to revert potential vandalism have unintentionally discouraged new contributors, undermining efforts to retain them [28]. Overall, these challenges emphasize the need for research into tools and processes that support more participatory approaches to bot design, allowing communities to collaboratively design bots that better meet their collective needs and values [33, 50, 65].

2.2 Supporting End-Users in Prompt Design

HCI research has a rich history of empowering end-users without technical skills to design AI systems tailored to their specific needs [18]. Early work on interactive machine learning (iML) and machine teaching develop tools and processes for individual, non-technical users to design traditional ML models [2], often for classification tasks [9, 45]. Some of this work has specifically focused on supporting the *collaborative* design of ML models [30, 74]. These efforts have focused mainly on supporting end-users in collecting more diverse datasets to train more robust classification models.

Recent advances in LLMs have significantly lowered the technical barriers to designing AI systems capable of more complex tasks [73], such as AI chatbots. This presents an opportunity for communities to design their own LLM-based bots by writing prompts in natural language, without the need for coding. However, crafting effective prompts remains a challenging and unintuitive task for end-users [82]. From writing prompts from scratch and iterating on them to assess their downstream impact, prior studies have identified several failure points in prompt design [81]. For example, a frequently encountered challenge is that non-AI experts often focus on iterating on their prompts for a specific interaction scenario they have in mind, without considering how these iterations might affect other scenarios [72]. As a result, prompt iterations can unintentionally worsen outcomes for scenarios previously considered but not revisited, or for relevant scenarios that users had not even considered [82]. Insufficient consideration of how a bot might behave across diverse interaction scenarios can lead to prompts that are overly ambiguous, making it difficult for LLMs to distinguish between meaningfully different scenarios [8]. On the other end of the spectrum, this can also lead to prompts that are worded in a way that is overfit to a specific scenario, so that the LLM is only able to handle a narrow set of interaction scenarios [72]. Finally, this can lead to prompts that cause unintended downstream consequences in interaction scenarios a user had not considered [77, 82].

Prior work has explored a range of ways to support non-technical end-users in prompt design [4, 51, 52, 56, 79], with some tools specifically created to address the aforementioned challenges. For example, to help address common pitfalls like writing overly ambiguous prompts, prior work creates a prompt coach that directly

[2]E.g., WikiBot on Discord links Wikipedia articles to messages: https://www.wikibot.de
[3]E.g., Automoderator on Reddit sends moderation warnings to users [35]
[4]E.g., MEE6 on Discord welcomes newcomers to the server [44]
[5]E.g., Nightbot on Twitch promotes advertisements for streamers' merchandise [66]
[6]E.g., Mudae on Discord facilitates anime character collection games [33]

asks novice users high-level questions for the user to reflect on [8], such as "Is your prompt detailed enough?". Other work provides users with direct recommendations on how they might edit their prompts to avoid potential undesirable social consequences [60, 61]. Meanwhile, work such as Wordflow [76], PromptSource [6], and FlowGPT [47] leverage the wisdom of the crowd by enabling individual users to upload their prompts to a shared repository and download prompts from others, helping users address scenarios they might not have considered on their own. However, since these methods are primarily intended for individuals to create personalized prompts rather than enabling groups to collaboratively develop prompts they use and rely on together, they do not facilitate direct collaboration on prompt design.

Enabling groups to collaboratively design prompts for shared LLMs or AI agents remains an underexplored area of research [31, 32]. Prior work such as PromptHive [57] and CoPrompt [22] has developed specialized interfaces that enable domain experts to collaboratively design prompts for their specific needs. For example, PromptHive allows mathematics educators to load homework problems, write prompts to generate homework hints, and share these prompts with colleagues via a shared library, where others can download, reuse, and refine them [57]. CoPrompt enables programmers to share prompts with collaborators and request prompts directly within the programming IDE [22]. In contrast to these approaches, which primarily focus on supporting prompt sharing, Botender is aimed at facilitating the collaborative design process itself through the use of concrete case-based provocations, as we will describe later. More closely related to our context, Koala is an LLM-based chatbot that participates in group discussions on Slack and allows participants to adjust four pre-defined settings via radio buttons to customize high-level aspects of the bot's behavior, such as its level of proactiveness (high, medium, or low) [31]. However, this work does not enable groups to collaboratively author prompts to specify desired bot behavior in detail. This limits communities' ability to customize bots to meet their specific needs.

Building on prior work, Botender aims to support communities in directly collaborating on iterative prompt design through the use of case-based provocations. These provocations are concrete interaction scenarios generated to support reflection on opportunities to improve the bot's behavior, and to surface differing viewpoints about desirable bot behavior within a community. Most closely related to this concept is a feature in the Gensors system, which automatically generates "edge cases" to help users stress test visual sensing models in situations they might not have considered, to help them to identify unanticipated failure modes [48]. In contrast to Gensors' edge cases, which help individual users debug visual sensors, Botender's case-based provocations are designed to promote collective reflection on bot behavior across diverse social interaction scenarios—tailored to highlight concrete consequences of known pitfalls in novice prompt design. Botender uses these concrete cases as common ground to facilitate collaborative prompt design. For example, as discussed in Section 4, by voting on cases users can discover where their expectations of bot behavior may differ. The use of cases as common ground in Botender is inspired by their documented success in supporting iterative, collaborative design in other contexts, as discussed in the next subsection.

2.3 Using Cases to Support Reflection in Collaborative Design Processes

Cases have served as a medium for design and deliberation across many fields [1, 17, 43, 46, 78], including HCI research. Building on prior HCI scholarship, we understand cases as concrete scenarios within specific problem situations that make abstract concepts tangible for reasoning, communication, and negotiation [11, 41]. For example, to establish common ground for public policy deliberation, HCI research has used concrete cases to illustrate how individuals might be affected by such policies, which are typically more abstract and difficult to reason about compared to specific cases [41]. HCI researchers have also brought strategically-selected legal cases that are more accessible to the general public before the judicial system to challenge existing laws and advocate for legal reform [37, 80]. Similarly, in online communities such as Wikipedia and Reddit, reflecting on concrete cases of user content enables moderators to refine moderation rules to account for scenarios they had not initially considered when creating these rules [10, 19, 29, 39, 42, 55]. Across these contexts—public policy, law, and moderation rules—concrete cases provide a valuable common ground for reflection and discussion, to inform policy iteration. This broader idea of iteratively developing general principles based on judgments about concrete cases aligns with the *method of reflective equilibrium*, proposed by political philosopher John Rawls [38]. In this method, the cases most useful for discussing and refining principles are often not those that conform to existing principles, but rather those where case-level judgments conflict with current principles, or cases that elicit divergent judgments among discussants [41]. Such cases push the group to reflect on their perspectives, and iteratively refine high-level principles and/or case-level judgments until they are in alignment (or equilibrium) [10, 11].

In the context of LLM-based AI agent design, the principles that govern an agent's behavior are commonly expressed as prompts. This parallel motivates us to explore how supporting collaborative prompt design can benefit from approaches in other domains, such as public policy, where concrete cases are used to facilitate deliberation and collaborative policy design. For example, PolicyCraft is a system that supports communities in collaboratively proposing, critiquing, and revising regulatory policies through discussion and voting on concrete cases [41]. These cases present specific, hypothetical actions by community actors (e.g., community members, businesses, government entities) and allow others to vote on and discuss whether they believe such actions should be permitted in their community. The community then revises its regulatory policies through this collective discussion and consensus. In this work, we explore whether a similar case-based approach can facilitate the collaborative design of AI agent behavior, enabling community members to collectively reflect upon and discuss how they would want a community bot to behave. In contrast to PolicyCraft's focus on manually-written cases by community members, Botender explores the idea of automatically, dynamically generated *case-based provocations* that support communities in identifying areas of disagreement and opportunities for bot improvement throughout an iterative, collaborative design process.

3 Design Goals

Based on our review of prior work, we synthesized the following four design goals for systems that aim to support the collaborative design of AI agents in community settings.

D1. The system should facilitate a coordinated process for agent design that supports effective collaboration and enables meaningful collective action. Even within a community that shares broad norms and values, individual members may hold differing views on how an ideal agent should behave and how to design it accordingly [10, 20]. The system should facilitate identifying potential sources of disagreements, enable community members to discuss their ideas, and support collective decision-making through a process perceived as legitimate by the community [41, 59, 68].

D2. The system should encourage users to assess the broader impact of their design ideas to support iterative prototyping. As discussed in Section 2.2, when designing AI agents, non-AI experts often focus narrowly on refining the agent's behavior in a *single* interaction scenario, overlooking the broader impact of their prompt designs across a variety of interaction scenarios [72]. As a result, prompt iterations can unintentionally worsen outcomes for other relevant scenarios that users had failed to consider [82]. To address this, systems should support users in considering how their design may affect a wider range of cases they may not have initially anticipated, to inform design iteration.

D3. The system should support regression testing to help users prevent the reintroduction of previously resolved issues during iterative agent design. As discussed in Section 2.2, non-AI experts often focus on refining an agent's behavior in the interaction scenario they are currently considering, overlooking how their refinements may impact interaction scenarios they had previously considered [72, 82]. As a result, prompt iterations may unintentionally reintroduce undesirable agent behaviors in previously addressed scenarios. To mitigate this issue, prior HCI research [82] suggests that systems should support regression testing—a practice from software engineering to ensure that new code does not break existing functionality [14, 15, 58]. In the context of collaborative AI agent design, this involves re-running the agent in previously resolved scenarios after making design changes to ensure it still behaves as the community intends.

D4. The system should be integrated into existing community platforms to promote broader community participation. Designing an AI agent within a community context requires additional effort beyond community members' regular activities. To reduce participation barriers, a key strategy from past research is to directly integrate the system into the community platform [27, 34, 42]. This integration should provide multiple ways for members to contribute and let them decide how much effort they want to invest. It should also help direct their attention to the tasks that would most benefit from community input [42].

4 Botender

Based on these design goals, we developed Botender, a system that enables the collaborative design of bots in online communities. While Botender's system architecture is designed to support the creation of diverse AI agents across different community platforms, in this paper, we present the first version of Botender, which supports the design of single-turn, LLM-based conversational bots powered by AI agents for Discord servers. In the following subsections, we first introduce Botender's overall system and agent architecture. Next, we walk through Botender's user interaction workflow, as illustrated in Figure 1. This workflow enables users to collaboratively design their bot by *proposing*, *iterating* on, and *deploying* tasks for the bot to perform within their community platform. We then describe Botender's *case-based provocation algorithm*, which generates provocative test cases to encourage user reflection on bot design during the iteration process. Throughout the section, we connect specific features of Botender's design to the design goals outlined in the previous section, denoted as D1 to D4. Finally, we conclude with implementation details.

4.1 System and Agent Architectures

Figure 2 depicts Botender's overall system architecture and its integration with community platforms such as a Discord server. From the user's perspective, they install a single **bot** named Botender and interact with it on their community platform.[7] The bot can perform a variety of **tasks**. The current version of Botender supports single-turn tasks: the bot handles one user message at a time, along with some meta information like the channel where the message was posted, and responds if any of its available tasks are triggered. For example, as shown on the left side of Figure 2, if a user posts an advertisement in a certain channel, the bot recognizes this as relevant to its "redirect ads" task. It then replies to the user, suggesting they post the ad in a more appropriate channel. Users can work together to create any kind or number of tasks they want the bot to handle, tailoring its behavior to fit the unique needs and preferences of their community.

Behind the scenes, as shown on the right side of Figure 2, the bot that users interact with is powered by multiple LLM-based agents, including an **orchestrator agent** and several **task-specific agents**, each corresponding to a task created by the users. This design aligns with best practices in agentic architectures [16, 23], allowing each agent to focus on a single task and thereby enhancing overall transparency, controllability, and task outcomes [79, 85]. Each task-specific agent includes a name, a trigger prompt, and an action prompt, which users can edit from Botender's web interface. When the community platform induces an event detected by Botender's always-running listener, such as a user sending a message in a channel, the orchestrator agent assesses whether the event is relevant to any of the task-specific agents. This assessment is based on each agent's **trigger prompt**. If a specific task is deemed relevant, the corresponding agent will take the necessary action according to its **action prompt**, such as generating a response to

[7] Like any Discord bot, only server admins can install Botender on their Discord server. After installation, admins can then grant other users permission to collaborate on designing the bot's behavior together. They can also change the bot's name from Botender to any name of their choosing.

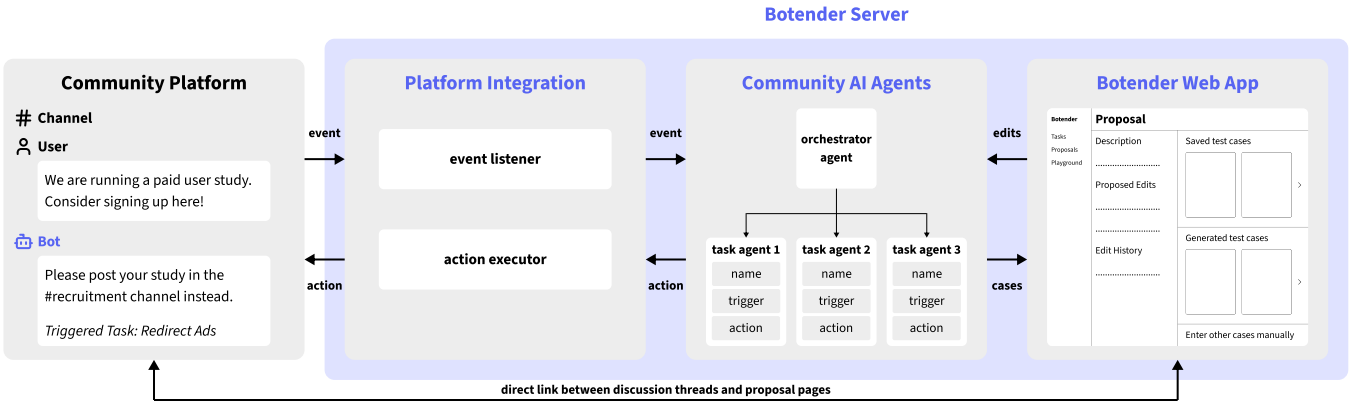

Figure 2: Botender's overall system and agent architecture. On the left, platform events are captured by Botender's always-running event listener, which translates them into information the agent architecture can understand. The orchestrator agent assesses each event and determines which, if any, task-specific agent is most relevant. The selected task-specific agent then generates an action instruction that is executed by Botender's platform action executor. On the right, Botender's website serves as the primary interface for users to collaboratively and iteratively design AI agents. This process generates concrete interaction scenarios that help guide further design iterations. The website is tightly integrated with the community platform, with each proposal directly linked to a dedicated discussion thread, as shown in Figure 5. This integration encourages broader community participation and discussion.

the user's message. This action is then executed on the community platform.[8] The complete system prompts for both the orchestrator agent and the task-specific agents are provided in Appendix A.1.

In the following subsections, we next walk through how users can use Botender to collaboratively design community bot behavior, by (1) proposing changes, (2) testing and iterating on prompts to operationalize proposed changes, and (3) deploying changes to the community platform (Figure 1).

4.2 Proposing Desired Changes to Bot Behavior

To propose changes to the bot's behavior, users can open Botender's web interface directly from Discord.[9] Users may choose to initiate a **proposal** from the web interface, based on opportunities for bot improvement they notice during their everyday interactions on Discord. Alternatively, users may initiate a proposal based on observations they make while testing the bot's behavior using the *playground* feature on Botender's web interface. Using the playground, users can freely test the behavior of the current version of the bot by simulating sending user messages in specific channels on their server. In the playground, users can also experiment with potential updates to bot behavior, and can then choose to submit a specific update as a proposal.

When creating a new proposal, users are asked to enter a title and a brief, high-level description of their desired changes. They are also encouraged to include a first attempt at operationalizing the change by either editing an existing **task** or creating a new one (see Figure 3 for an example). As shown in Figure 4, each task consists of a pair of prompt fields: a trigger and an action. The **trigger** defines *when* the bot should perform the task, while the **action** specifies *what* the bot should do when the task is triggered. The task is also given a brief **task name** for future reference within the web interface and Discord, but this name does not affect the bot's behavior. As shown in Figure 5, the short name of the triggered task is displayed to users on Discord each time the bot replies. This allows users to better pinpoint and propose more precise edits to a specific task based on their observations of how the bot actually behaves within their community in accordance with that task.[10]

Once submitted, the proposal is created as a page visible to all community members, as shown in Figure 3. If the original proposer has included a specific proposed edit with their proposal, they and any other visitors to the page are immediately shown a set of **test cases** illustrating how the bot would behave across a range of interaction scenarios (shown on the right side of Figure 3), and are asked to review these test cases for unintended or undesirable bot behaviors. Each test case shows the channel name where a user message is sent, the user message itself, the name of the triggered task, and how the edited bot would reply to the user message. The test cases are divided into two sections: **generated test cases** and **saved test cases**. *Generated test cases* are produced by Botender's case-based provocation algorithm (presented below in Section 4.5) with the aim of supporting user reflection on potential opportunities to improve the proposed edit (D2) and helping to surface disagreements among community members about whether the bot's response in a given scenario is appropriate (D1). The *saved test cases* section allows the proposer or other community members to save test cases for later consideration and discussion. When saving a test case—whether

[8]Botender's system architecture is designed to support the bot's functionality beyond conversational interactions. For example, the event listener can monitor additional platform events, such as new users joining a server, and the system can execute a wider range of platform actions, such as banning users from the server. In the Discussion (Section 7), we outline future directions for expanding the bot's capabilities.

[9]It is common for more sophisticated Discord bots to have standalone websites for users to customize a bot's settings, rather than performing such customization within the Discord interface, so this is a familiar interaction for users.

[10]These name, trigger, and action fields correspond to each task-specific agent's name, trigger prompt, and action prompt as discussed in Section 4.1.

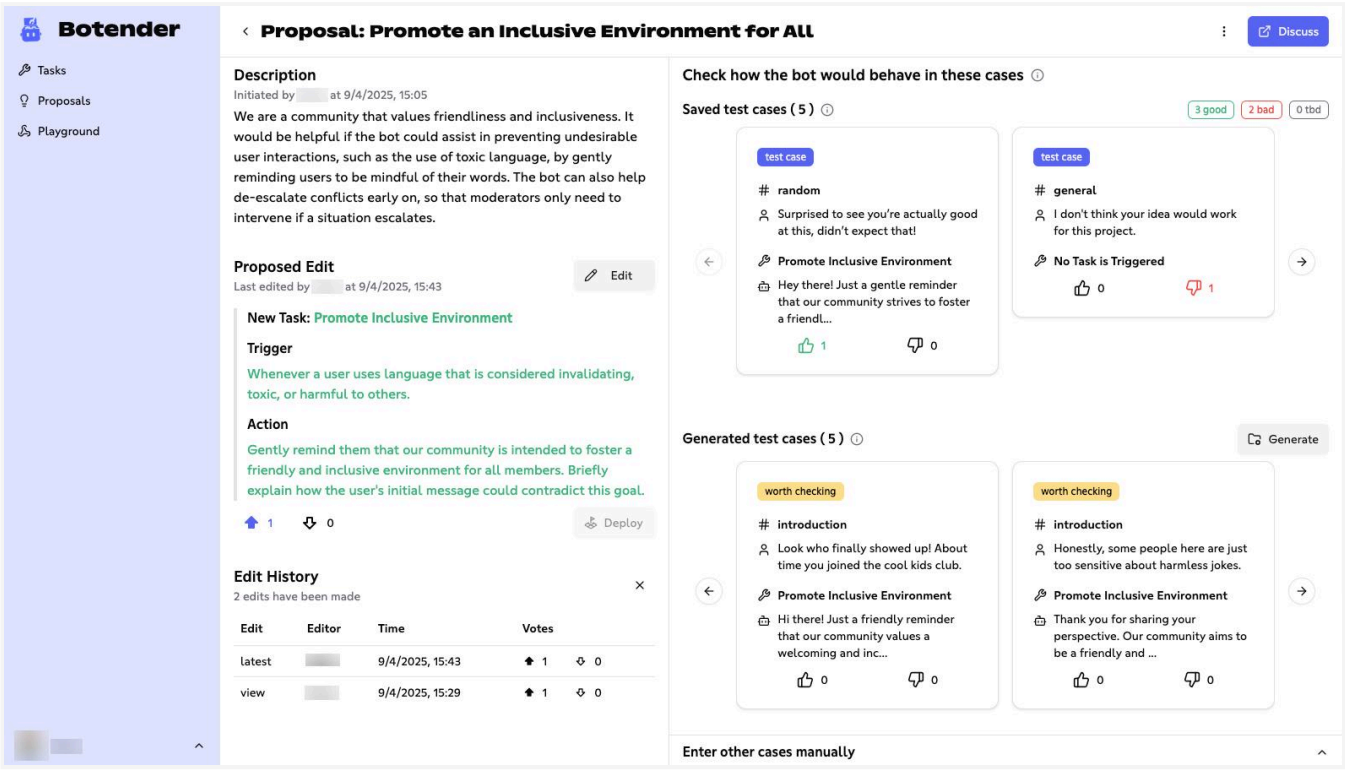

Figure 3: Botender's proposal page. The left navigation bar lets users switch between viewing all active tasks on their Discord servers, community proposals for desired changes, or experimenting with the bot in the playground without affecting their server. In the center, users see the proposal's title, description, and the latest proposed edits to the bot's tasks, such as adding a new task in this screenshot. Users can upvote or downvote to indicate their support for or opposition to deploying the latest edit. The bottom displays a full edit history, allowing users to compare edits with previous versions and the original task. On the right, test cases help guide collaborative decision-making. At the bottom are test cases automatically generated to provoke user reflection and discussion around the latest edit. Generated cases are saved if a user chooses to vote on the bot's response. At the top, members can review and vote on test cases that have previously been saved by community members. Clicking a test case opens a pop-up with case details, including how the bot's responses for that case have changed across edits. Finally, users can click "enter other cases manually" to open a sheet where they can add custom test cases.

a generated test case or a bot interaction from a user's manual testing—the user is asked to thumbs up or thumbs down the test case to indicate whether they think it is an example of a good or bad bot response. Other users can subsequently add their own votes to indicate their own perspectives. If a proposal is created based on a specific bot interaction the user observed in the playground, this case is automatically added to the saved test cases when the proposal is created.

When a community installs Botender on their Discord server, the system automatically creates a **#botender** channel. This channel is primarily used for collaborating on bot design and is accessible only to admins by default, though permissions can be granted to other members if desired. When a new proposal is created, the system sends a **notification message** to the this Discord channel and creates a **discussion thread** linked to that message (D1, D4), as shown in Figure 5. Users can use this thread to discuss and coordinate their efforts, as well as follow the thread to receive notifications about proposal updates.

4.3 Iterating on Bot Behavior

After a proposal has been created, community members can view the proposal page, including any saved test cases and newly generated case-based provocations for the latest version of the task. Users can review these test cases and add their votes on bot responses in saved test cases. The **counters** for "good," "bad," and "tbd" in the upper-right corner of a proposal page display the number of cases that have received a majority of thumbs up, thumbs down, or equal votes (Figure 3). They provide users with a quick overview of the community's collective views on test cases. If users notice disagreements about desirable bot behavior that they wish to discuss, they can click the **"Discuss" button** just above these vote counters, which brings users directly to the associated discussion thread for that proposal on Discord.

Any user can make an **edit** to a task from a given proposal page, including creating new tasks or editing or removing existing ones. To do this, users click the "Edit" button next to the current proposed edit. To test their edits, users click the "Test + Generate" button, as

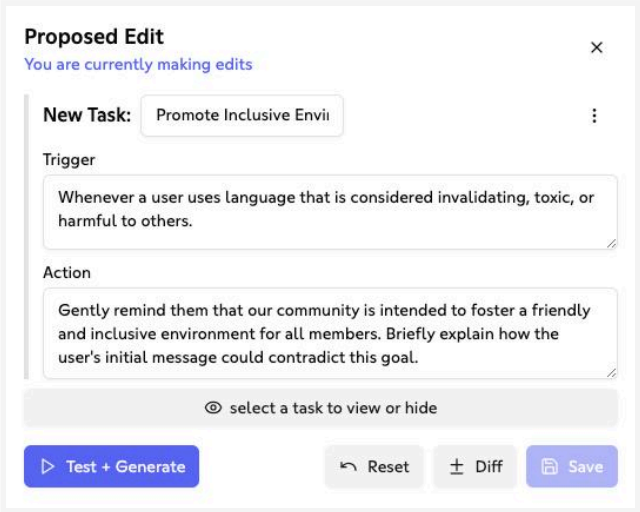

Figure 4: After clicking the edit button on the proposal page, the original static text is replaced by this edit interface. Before saving edits, users are required to run "Test + Generate" to see how the bot would behave with their proposed edits.

shown in Figure 4. Botender then re-runs all saved test cases and displays the updated bot responses based on users' edits, to support regression testing (D3). Botender also generates new case-based provocations based on the user's edit (D2). Before a user is able to save their edit, they are asked to review the bot's behavior in the presented test cases, and must vote on *at least one* of these test cases. This default threshold was chosen to promote user engagement with the test cases before saving and sharing proposed edits, while avoiding introducing too much friction into the collaborative editing process. However, the threshold may be customized to meet individual communities' needs. If the user notices a potential issue when reflecting on the test cases, they are encouraged to refine their edit and test again before saving. Once they are satisfied, they can save their edits to the proposal.

Once an edit is saved, the system sends a notification to the proposal's discussion thread on Discord to encourage people to review the latest proposed edit (D1, D4), as shown in Figure 5. On the proposal page, they can then review the latest proposed edit and corresponding test cases. Users can also view the full **edit history** of a proposal, including edits to tasks and any additions or removals of saved test cases, and can revert changes as needed (D1). This design mirrors systems such as Wikipedia by making the edit history transparent [12, 42, 71], which helps coordinate efforts and ensures accountability among editors.

4.4 Deploying Updates to Bot Behavior

At any point in this process, upon viewing the latest proposed edit and test cases for a given a proposal, users may choose to **vote** in favor of its deployment within their Discord server (D1). By default, each proposal requires at least three upvotes to be deployable, but this threshold can be adjusted to fit each community's needs. Similar to saving edits, the system requires users to give at least one thumbs up or down on a saved test case before voting for deployment. As at other points throughout Botender's workflow—such as when creating or editing a proposed bot task—this encourages users to review how the bot would actually behave based on the proposed edit before casting their vote, rather than merely reading the edit.

Once a proposed edit reaches the deployment threshold, users can click the **"Deploy" button** on the proposal page to deploy the proposed changes to the live bot on their Discord server. The system sends notifications to both the proposal discussion thread and the main #botender channel to inform everyone that the bot's tasks have been updated (D1, D4). The proposal is then closed.[11]

4.5 Case-Based Provocation Algorithm

A key aspect of Botender is its support for collaborative iteration on bot design through the use of concrete test cases. Drawing inspiration from the use of cases as a medium for collaborative design and deliberation in various fields (Section 2.3), Botender's case-based provocation algorithm is designed to generate concrete interaction scenarios that spark collective reflection and discussion about desirable bot behavior, rather than simply *validating* expected outcomes. The goal of these case-based provocations is to test the boundaries of users' intents in bot design (as expressed through LLM prompts) by (1) highlighting potentially undesired bot behavior in interaction scenarios users may not have considered; and (2) highlighting areas where there is potential for disagreement among different users. Based on past research, discussed in Section 2.3, we expect such boundary-challenging provocations to be more useful in supporting prompt iteration than test cases aimed at simply validating expected behavior in expected scenarios.

Botender's case-based provocation algorithm is designed to generate three broad types of cases, aimed at surfacing issues commonly overlooked by non-AI experts when designing LLM prompts, which can also be sites of disagreement among community members:

- **Cases that highlight ambiguities in a prompt**: Non-AI experts often write LLM prompts containing ambiguous, underspecified phrases [82]. Such phrases can be interpreted differently by both the LLM and by different people. For example, the phrase "inappropriate language" in a prompt leaves substantial room for differing interpretations about what exactly constitutes inappropriate language. If a user's interpretation of the prompt differs from that of the LLM, the agent may behave in ways that are unexpected to the user. Similarly, different users may agree on a prompt's wording, only to realize that they have different expectations for how the agent should behave, upon seeing actual examples of bot behavior in different interaction scenarios. Concrete cases can help reveal such disagreements and encourage iteration on prompts to make them clearer and more specific.
- **Cases that highlight potential overly narrow wording in a prompt**: Non-AI experts also often write LLM prompts that focus too narrowly on defining an agent's behavior for a single scenario, overlooking the broader impact of their prompt across a wider range of possible interaction scenarios [72, 82]. For example, a prompt that instructs the agent to

[11] If users choose not to deploy a proposal, they can close it but still have the option to reopen and edit it later if needed.

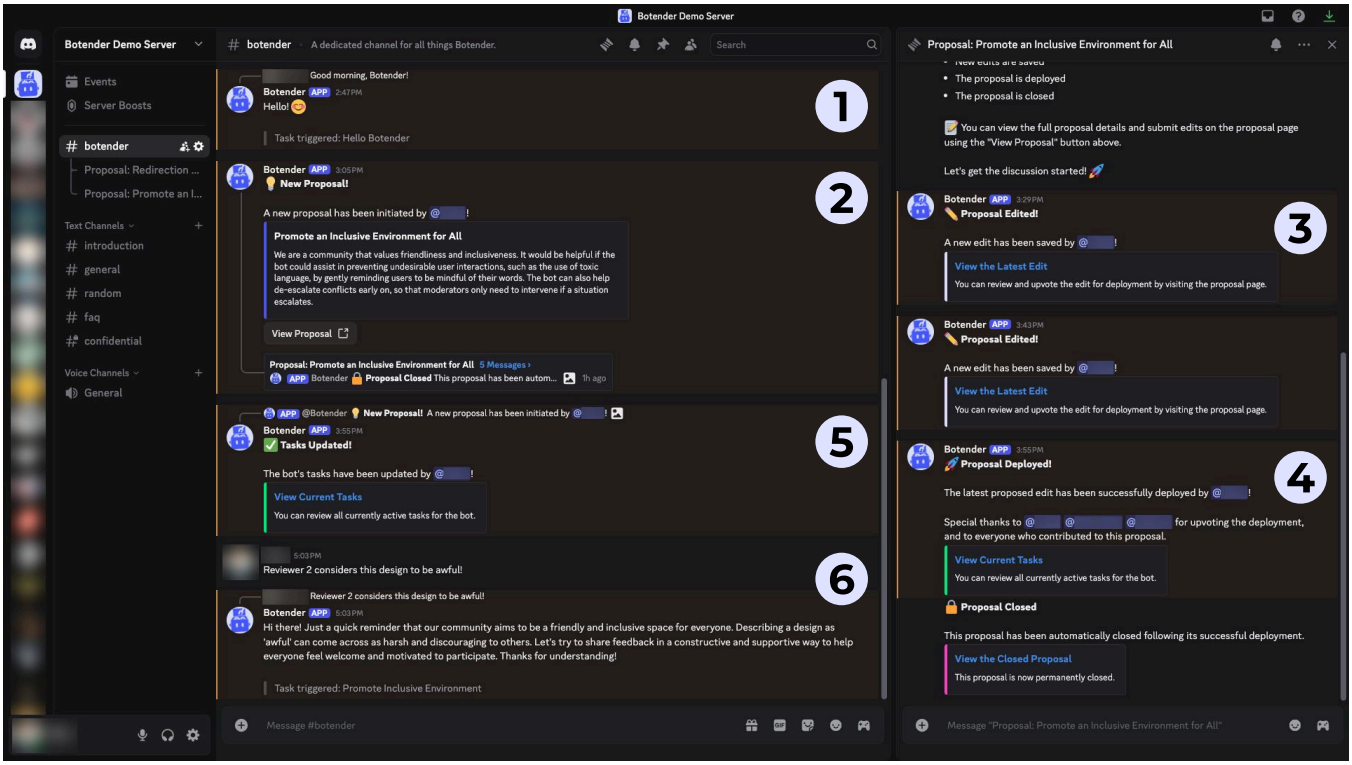

Figure 5: The Discord interface, highlighting Botender's integration with the community platform. (1) By default, Botender replies "hello" to users who greet it in the #botender channel, as defined by its default "Hello Botender" task. (2) When a new proposal is created, the system sends a notification to the #botender channel, and (3) creates an associated discussion thread, as shown on the right, where users can discuss the proposal and receive notifications about saved edit updates. (4) Once a proposal is deployed, the system notifies the discussion thread, closes the proposal, and (5) and sends a message to the main #botender channel. (6) The bot will then behave according to the latest deployed edit.

identify a specific list of banned words (as is common in traditional moderation tools) may overlook how those words could be used legitimately in some contexts. Such a prompt may also fail to address the broader goal behind banning these words by missing related words or phrases that are not included in the list. Concrete cases covering a diverse range of relevant scenarios can encourage users to reflect and discuss how to iterate on an overly narrow prompt to better achieve their broader goals.

- **Cases that reveal potential unintended community-level consequences of a prompt**: Finally, as documented in prior HCI research, bots deployed in community contexts can sometimes lead to unintended community-level consequences, despite good intentions [40]. For example, community bots may inadvertently discourage participation by enforcing norms too strictly or crowding out opportunities for meaningful user contribution [28]. Concrete cases can help users foresee potential unintended downstream consequences and iterate on the prompt before deployment to prevent them.

To generate these three types of cases to support user reflection and discussion, Botender's algorithm uses three separate LLM pipelines, one for each case type, as shown in Figure 6. Each pipeline consists of three modules: a detector, a generator, and an evaluator. The **detector** identifies specific phrases in the prompt, whether in the trigger or action prompt of a task, that may be too ambiguous, overly narrow, or could lead to unintended consequences. Based on the detected issue, the **generator** creates a channel name and user message that aim to concretely illustrate the problem, along with a **reasoning** of what the case aims to highlight. However, at this stage the generator does not yet know how a bot's will respond to the message. Therefore, after generating the channel and user message, these are sent to the bot (specifically, to the orchestrator agent and task-specific agents as described in Section 4.1) to obtain a response. The **evaluator** then reviews the complete case, including the actual bot response, to determine whether it effectively demonstrates the identified issue as described by the generator's reasoning. Only cases that pass the evaluator are output by each pipeline. Finally, all cases from the different pipelines are merged into a single pool, from which a **selector** module chooses a set of cases that are likely to be most useful (five in the current implementation) in promoting user reflection and discussion. Full details of the system prompts for individual LLM modules within the pipeline are provided in Appendix A.2.

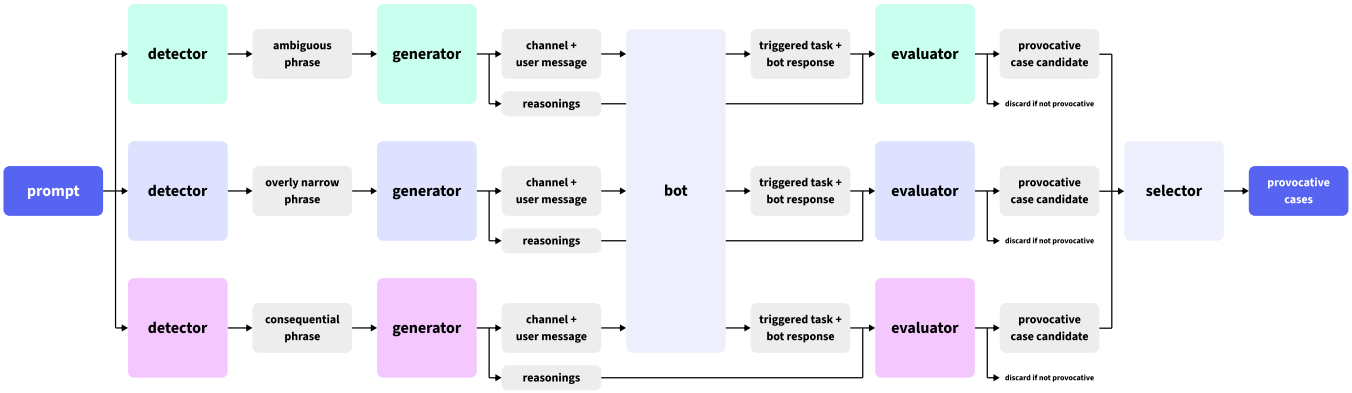

Figure 6: Botender's case-based provocation algorithm uses three parallel LLM pipelines to generate provocative test cases that encourage user reflection and discussion on common prompt design pitfalls, including ambiguous language, overly narrow phrasing, or unintended downstream consequences for the community. Each pipeline includes its own detector, generator, and evaluator to generate relevant cases. The detector identifies phrases related to the targeted pitfall; the generator creates cases based on these identified phrases; and the evaluator checks how well the generated cases illustrate the pitfall. Finally, a selector chooses the most provocative cases from all case candidates. The prompts for all ten LLM modules, including each pipeline's detector, generator, and evaluator, as well as the final selector, are provided in Appendix A.2.

4.6 Implementation Details

Botender is a full-stack, end-to-end system that individual communities on Discord can set up via an installation link, as is standard for Discord bots. Botender's website has a fixed URL where Discord users can log in with their Discord accounts, but they can only view and design bots for servers where they are members and where Botender is installed. Botender is built with SvelteKit and hosted on Vercel, using shadcn-svelte components styled with Tailwind CSS on the frontend and Firestore databases on the backend. Since Vercel is a serverless platform, Botender's always-on listener for Discord platform events is hosted separately on Railway. All the AI agents and LLM modules are powered by OpenAI's GPT-4.1. The entire Botender codebase is open source and publicly available on GitHub.[12]

5 Algorithm Validation Study

We first conducted a validation study to understand how well Botender's *case-based provocation algorithm* generates test cases that support user reflection on desired bot behaviors. In Botender, these cases are aimed at (1) helping users identify opportunities to improve the bot's prompt and (2) helping to surface potential disagreements among community members about how the bot should behave. This validation study focuses on assessing how effectively Botender's case-based provocation algorithm achieves these goals.

5.1 Recruitment

We conducted the validation study through an online survey. In total, we recruited 90 participants on Prolific with experience in online groups or communities (e.g., Discord servers or Slack workspaces) where Botender is intended to be deployed. Each participant was compensated $5 USD, and the median survey completion time was 13 minutes.

[12]https://github.com/tskuo/Botender

5.2 Study Procedures

Each participant was randomly assigned to review one of nine preselected bot prompts, which we referred to as bot "instructions" in the survey to avoid technical jargon. We selected these prompts to cover a variety of potential pitfalls in prompt design. Each prompt included both a trigger and an action, consistent with how tasks are specified in Botender. Additional details about the prompts used in the validation study are available in Appendix B.2.

Each participant reviewed a prompt and evaluated two sets of cases generated based on the following two conditions:

- **Botender: Case-Based Provocations**: For each prompt, participants evaluated five cases generated by Botender's case-based provocation algorithm, as described in Section 4.
- **Baseline: Standard Test Cases**: For each prompt, participants also evaluated five algorithmically generated test cases relevant to the prompt but not specifically targeted to provoke critical reflection. By comparing against this baseline, we aimed to understand whether and how Botender's case-based provocations provide value beyond the *general* benefits of encouraging people to reflect on concrete cases. See Appendix B.1 for the algorithm used to generate baseline cases.

Participants evaluated two sets of cases in random order. For each case, they rated how strongly they agreed or disagreed with the following two statements reflecting the goal of Botender's case-based provocation algorithm:

- **Controversialness**: I think **people may have differing opinions** on whether the bot's response in this case is appropriate.
- **Provocativeness**: I think this case **reveals opportunities to improve the instructions** that were given to the bot.

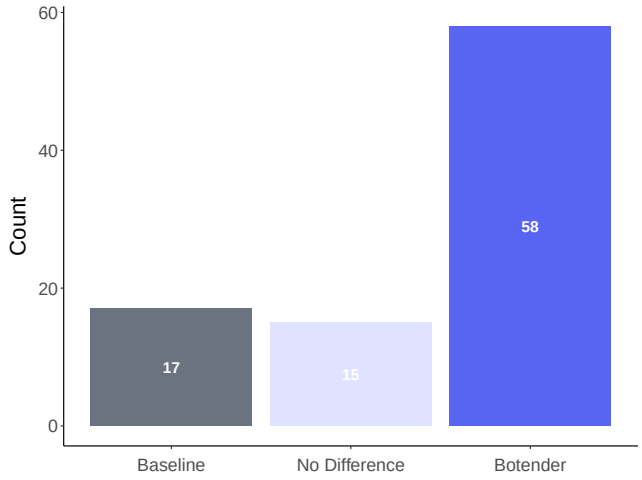

Figure 7: The number of participants who reported that the sets of cases generated by the baseline vs. Botender's case-based provocation algorithm were more provocative (i.e., revealed more opportunities to improve the bot's instructions).

Participants also provided set-level ratings indicating their agreement with the following two statements and briefly described potential problems they saw with the bot's prompt, based on the cases in each set:

- **Coverage**: This set of cases covers a comprehensive range of problems with the instruction.
- **Diversity**: This set of cases covers a diverse range of problems with the instruction.

Finally, participants selected the set of cases that revealed more potential problems with the prompt and thus better highlighted ways to improve the bot. They also briefly justified their choice. With a total of 90 participants, this sample size yielded 10 independent reviews per prompt and its corresponding cases. All prompts and cases that participants reviewed can be found in the supplementary materials.

5.3 Study Results

Overall, participants found that Botender's case-based provocations revealed more opportunities to improve the bot's instructions. As shown in Figure 7, most participants selected the set of cases generated by Botender's algorithm when asked which set revealed more opportunities to improve the bot's instructions, in a blind comparison. Participants' justification of their choices[13] offered insight into the differences they perceived between the two sets of cases. For example, several participants noted that Botender's case-based provocations *"surface a wider range of edge cases that the current instruction doesn't handle well"* (V80, P8). Participants mentioned that these cases *"are more vague and difficult for the bot to interpret"* (V79, P1), *"show tricky situations the bot's instructions don't cover"* (V59, P7), and *"highlight unclear criteria and over-triggering, thus better exposing instruction weaknesses"* (V62, P2). Meanwhile, they found the standard test cases to be *"more cut and dry"* (V4, P7), offering either *"obvious examples of red flags to the bot"* (V19, P1) or cases that *"shows the bot doing its job correctly"* (V72, P2).

After reviewing Botender's case-based provocations, participants identified a range of opportunities to improve the bot's instructions, including concerns about potential unintended community-level consequences of the bot's responses, such as making users *"feel unimportant, unheard, and excluded"* (V51, P4), or noted situations where *"the bot is too direct in question[ing] the user and it comes across as arrogant"* (V23, P7). By contrast, after reviewing cases generated by the baseline, participants found that the set *"covers cases where the bot should respond, which it does"* (V45, P5), or that the bot *"answered but just needed to shorten them or make them more to the point"* (V27, P4). Some participants observed that *"[Botender's] set had many more areas for improvement"* (V31, P5), or found the baseline *"showed no noticeable errors or deviations, and it responded as everyone would expect"* (V87, P3). Overall, participants found that Botender's case-based provocations *"[hit] deeper problems"* (V90, P7).

The finer-grained ratings participants provided at the case and set levels align with these interpretations. As shown in Figure 8, at the case level, participants gave significantly higher ratings ($p < 0.001$) for both *provocativeness* (the extent to which the case reveals opportunities to improve the bot's instructions) and *controversialness* (the extent to which they thought people might hold differing opinions about the appropriateness of bot behavior in a given case). At the set level, participants also gave significantly higher ratings ($p < 0.01$) for both the coverage and diversity of Botender's case-based provocations. The effect size is significant but small at the individual case level—0.6 on provocativeness (3.5 for baseline and 4.1 for Botender) and 0.7 on controversialness (3.2 for baseline and 3.9 for Botender), likely due to variation across individual cases. However, at the aggregate set level, people overwhelmingly chose the Botender condition. Taken together, participants' judgments of provocativeness at the set level (Figure 7) versus at the individual case level (Figure 8) may indicate that participants found the case-based provocations most provocative when presented as a set, rather than in isolation.

Overall, these results validate that Botender's case-based provocation algorithm generates sets of test cases that can better support user reflection on opportunities to improve the bot's instructions. This is further supported through findings from our field study, reported in the next section, where participants made heavy use of case-based provocations to collaboratively iterate on their prompts.

6 Field Study

To understand how people use Botender as an end-to-end system for collaborative AI agent design, we conducted a 5-day field study in real-world Discord communities. On Discord, each server is an online community where individuals with shared interests connect and interact. These interests range from casual hobbies like gaming and anime to professional topics such as programming or specialized spaces for customer service [3]. Our goal in this field study was to understand how diverse Discord communities use Botender to collaboratively design and customize bots for their servers.

[13]We denote participant IDs starting with V and the prompt a participant reviewed starting with P—for example, "(V79, P1)".

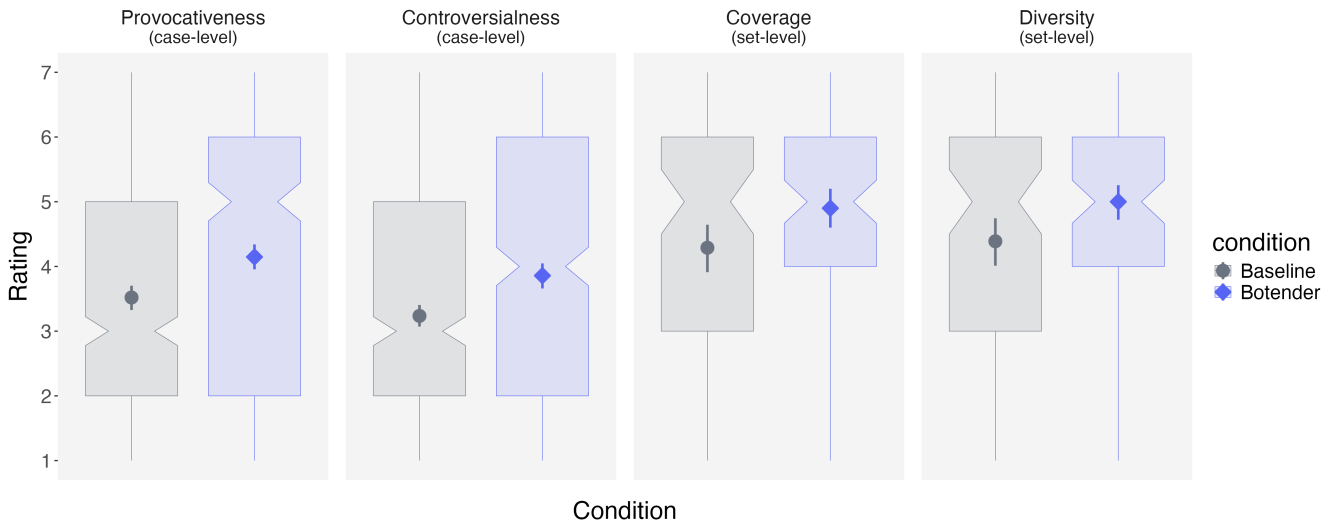

Figure 8: Participants' ratings on various aspects of cases generated by Botender's case-based provocation algorithm and the baseline algorithm. The figure shows notched box plots, with the notches indicating the medians, and the means and 95% confidence intervals overlaid on the plots. The results show significant differences for all rated aspects.

6.1 Recruitment

We recruited six participant groups via social media, paper flyers, and word of mouth. These groups represented a diverse array of Discord communities, ranging from close-knit friend groups and student organizations to the fan community of an indie band. Table 1 provides details about each participant group.[14] All participants were active community members or admins in their servers, making them ideal candidates to design their own community bots using Botender. Each participant received $100 USD for the field study, consistent with compensation provided in previous week-long HCI research (cf. [42, 83]).

6.2 Study Procedures

The field study lasted five days for each group of participants. The study began with a synchronous onboarding session, where participants received guidance on how to use Botender and had the chance to try the system in a dedicated onboarding server. We also recorded a comprehensive system walkthrough for a few participants who were unable to attend the group onboarding session. After onboarding, participants installed Botender on their own Discord server and selected the start date for the five-day study period. We set two minimum participation requirements for the study. First, each participant needed to create or edit at least one proposal per day. Secondly, as a group, they were required to deploy at least three tasks tailored to their community's specific needs, norms, and values by the end of the study. We kept the participation requirements minimal to provide them with the flexibility to decide when and how much they want to engage in bot design, while also ensuring that participants would have ample opportunities for interaction across the field study period (cf. [41, 83]). We encouraged participants to design tasks that reflected their community's unique norms and culture, rather than purely logistical tasks.

Each time participants edited a proposal on the proposal page, they answered a brief multiple-choice question about their motivation for making the edit. By analyzing the frequency of each selected option, we gained insights into what drives proposal editing. Participants could select one or more of the following six options to indicate that they were making an edit to address:

1. specific saved test cases they saw
2. specific generated test cases they saw
3. specific cases they entered themselves manually
4. general issues that someone else raised
5. general issues they thought of themselves
6. other

At the end of the field study, all participants completed a post-study survey that included the following five statements. Participants rated their level of agreement with each statement on a scale from 1 to 7, where 1 indicates "strongly disagree" and 7 indicates "strongly agree." They also provided explanations for their ratings:

1. I find that the bot we designed behaves in a way that reflects the specific needs, norms, and culture of our Discord community.
2. I can easily collaborate with others in bot design using Botender.
3. I find the test cases helpful in revealing opportunities to improve the bot.
4. I find the test cases helpful in surfacing situations where people might have differing opinions about whether the bot's response is appropriate.
5. I find the experience of designing the bot with the Botender system integrates well with my usage of Discord.

[14] During the study, other members of each server who were not participants did not use Botender to design bots.

Table 1: Field study participant group demographics, including the number of participants in each group, the total number of community members in each group's server where Botender was deployed, and a brief description of each server community.

Group ID	Group Size	Server Size	Server Type
G1	5	5	A close-knit friend group server for hanging out and having fun
G2	3	17	A fan community for an indie music band to connect with their superfans
G3	6	27	A research lab led by a professor, with members including students and collaborators
G4	6	50	An offshoot of a larger community, created for members who share common interests in gaming
G5	5	66	A friend group and their close friends, primarily used for socializing and gaming
G6	6	429	A student organization within a university that organizes hackathons

Finally, they were asked to suggest features for improvement or for future versions, and to indicate whether they would like to continue using Botender in their server after the study.

6.3 Study Results

We present the findings from our field study in the following sections.[15] Section 6.3.1 presents the types of tasks participants designed for their community bots and their perceptions of these tasks, as shared in the post-study survey. Our results show that participants were able to design a wide variety of tasks tailored to their communities, and they felt these tasks reflected their unique community needs and culture. Section 6.3.2 explores how participants collaborated using case-based provocations. Across the six communities, participants created a total of over 100 proposals and 800 saved test cases during the study. Analysis of the multiple-choice questions that participants answered after each proposal edit revealed they were more likely to iterate on proposals in response to Botender's case-based provocations, which encouraged collective reflection and discussion about desirable bot behaviors. Feedback from the post-study survey provides further insight into how participants collaboratively improved and discussed bot designs based on opportunities identified through these cases. Finally, Section 6.3.3 presents additional feedback from participants on aspects of Botender's design, beyond the case-based provocations, that they found particularly helpful for collaborative bot design. These include the overall system workflow, seamless integration with the community platform, and the use of natural language for bot design. Overall, participants found that the bot behaved in ways that aligned with their needs and community norms, attributing this to the collaborative design process enabled by Botender. Over 96% of participants expressed interest in continuing to use Botender after the study period.

6.3.1 ***With Botender, participants designed a variety of tasks for the bot tailored to the specific needs and norms of their own communities.*** Table 2 presents descriptive statistics from the field study, including the number of tasks each group deployed and the number of proposals and cases they created to support task deployment. To provide a glimpse of the resulting tasks, Table 3 presents a sample of tasks deployed by each group. While the current version of Botender only supports the creation of tasks involving single-turn interactions (as mentioned in Section 4.1),

[15]In the field study results, participant IDs are indicated with an S, and the groups they belong to are indicated with a G.

Table 2: The number of tasks deployed and the proposals and cases created by each group during the field study.

Group ID	Tasks	Proposals	Cases
G1	18	32	166
G2	4	3	14
G3	10	23	204
G4	17	37	183
G5	16	28	165
G6	4	14	68
Total	69	137	800

the range of tasks is quite broad and diverse. As one participant noted, these tasks span everything *"from funny quips to actual advice and help to games and welcome messages"* (S4, G5). The tasks participants created for the bot align with prior research [63], demonstrating a balance between task-oriented and socially-oriented functions, or combinations of both. As shown in Table 3, the types of tasks participants created generally reflect the nature of the server, whether it is more professionally or socially oriented.

Participants agreed with the statement that "*the bot we designed behaves in a way that reflects the specific needs, norms, and culture of our Discord community*" ($M = 6.10, SD = 0.94$). They shared that the bot behaves in ways that *"adhere to our culture, humor and jokes"* (S4, G5), and *"reflects us as a community a lot since we approached the tasks we made with a lot of light humor while keeping the usefulness aspect"* (S22, G1). Participants were impressed by how specifically the bot could be tailored to fit their server: *"We came up with some really specific ways to greet our fans. [...] We specifically asked [the bot] to greet people in a way that was both fabulous and gay and fun, but at the same time sad because our band plays sad country songs. It did such a great job of that. It was unbelievable"* (S27, G2). In addition to the content of the messages the bot sends (as defined by a task's action), participants also appreciated the precise timing of when the bot chimes in (as determined by its trigger). For example, a participant found *"the fact that it could be prompted in such detail, made it great with timing [...], which made for a fun moment in the server"* (S15, G4).

6.3.2 ***Participants found that case-based provocations effectively facilitated iterative, collaborative design.*** To understand how participants iteratively developed these tasks, we

Table 3: Selected tasks from each group reflect the diverse range of tasks they designed for the bot to address the unique needs and norms of their communities. The categories of tasks are assigned according to the taxonomy identified in prior work [63], with definitions and examples provided in Section 2.1. The full list of tasks created by each group is included in Appendix C.1.

Group	Task Name	Trigger	Action	Category
G1	Sideeyeomatic	Whenever someone says anything questionable or suspicious - things that would generally make someone give them the side eye.	Post this gif: https://tenor.com/p6t9IvV9eBF.gif	engagement
G2	Merch Link	Whenever someone asks about or expresses interest in supporting the band, or buying band merchandise or physical copies of the music, or mentions that they enjoy the types of items we sell including vinyl albums, cassette tapes, band shirts, stickers, etc.	Let them know that we have merch items including but not limited to shirts, bandanas, stickers, vinyl albums, cassette tapes and direct them to the website [url] to purchase these and other items	promotion
G3	Lab location	When someone asks about [lab] location or room number or access info	Reply them with [lab name] ([building code] [room number]), mention that they need to request access through [department acronym] form [service portal url]. Also remind them to get access to the [graduate lounge location] to enjoy free coffee and spend their free time or study. Use proper formatting and emojis	information
G4	Puppy Training	All users in this server own dogs and like to have fun by roleplaying their dogs talking. Whenever a user imitates their dogs through actions such as barking or voices thoughts from the perspective of their dog, you should trigger	To encourage responsible dog behaviour and also set examples of proper dog behaviour, please praise or scold users as if they are a dog when dogs are mentioned. Users believe their dogs (rightfully so) are very cute, so try to address pets by pet names like "puppy" or "doggy" rather tha scientific terms such as "dog" or "canine"	engagement
G5	Woah, easy now	Detect angry or aggressive language	Act like a old timey southern cowboy who is trying to calm down his horse.	moderation
G6	Info overview	Any question about hackathons, [hackathon event], [student club]	Link to [event website] for [hackathon event] specific questions. If asking about what a hackathon is then provide overview of hackathon. If asking about [student club], link to [club url] page as well as provide information about the club.	information

examined the results from the multiple-choice question they answered each time they edited a proposal. Specifically, participants were asked whether their motivation for making an edit was based on a specific case or a general issue. As shown in Figure 9, a one-sided hypothesis test revealed that, when using Botender, a larger proportion ($p < 0.001$) of proposal edits were aimed at addressing specific cases (59%) rather than general issues (37%). Meanwhile, 95% of the 800 cases saved to the proposals were generated by Botender's case-based provocation algorithm, rather than being manually entered by participants. Taken together, these results suggest that Botender's case-based provocations effectively facilitate iterative proposal editing during the design process. This aligns with participants' experiences, as they reported that Botender's case-based provocation *"significantly reduced headaches in prompt engineering and what would have required additional proposals to fix. [...] It allowed me to iterate quickly on my prompts while drafting the task for the proposals. I often didn't even need to make any manual test cases thanks to the generating feature which reduced friction in quickly getting a proposal together"* (S6, G5).

Participants' self-reported ratings and explanations in the post-study survey further support this finding. Specifically, they agreed that *"they find the test cases helpful in revealing opportunities to improve the bot"* ($M = 6.31, SD = 1.00$) and shared experiences of how the cases supported their iterative process. For example, one participant noted that the test cases *"allowed me to fine tune the description I wrote for letting the bot know when it should respond. This helped me to realize that sometimes I was not writing detailed enough descriptions"* (S22, G1). Another participant mentioned that the *"test cases reveal niche scenarios or unintended uses that may cause the users to reflect and change the original prompt. This was*

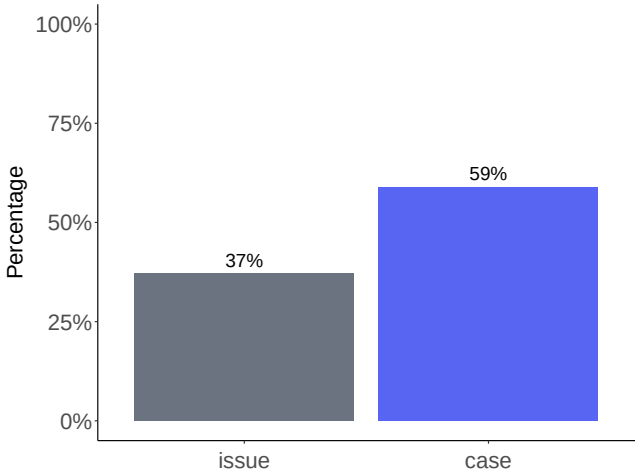

Figure 9: The percentage of proposal edits aimed at addressing specific cases participants reviewed versus general issues that they thought of during the proposal iteration process. A significantly higher proportion of edits were inspired by specific cases.

very useful" (S8, G5). These user experiences align with the goal of Botender's case-based provocation algorithm, which is designed to uncover common issues often overlooked by non-AI experts, such as ambiguous prompts or unintended downstream consequences, rather than just generating cases that validate what users have already written. As one participant put it: *"The test cases were helpful not just in guiding the bot about false positives/negatives, they also helped clarify the task based on the kinds of test cases it generated"* (S3, G3). Participants appreciated the thought-provoking and diverse test cases, stating *"the test cases were brilliant and diverse. [...] It was helpful especially because it generated some scenarios which I may not have thought of myself to create"* (S14, G6). They also shared that the *"test cases make it very easy to spot when prompting is faulty or needs tweaking. I tweak my prompts at least once for each of my proposals, based off what I see when testing"* (S23, G4) and *"the test cases made it easy to see how a proposal would progress"* (S26, G4).

In addition to supporting design iteration, participants reported that Botender's case-based provocations effectively surfaced differing opinions and sparked discussion about desirable bot behavior within the community. In particular, participants agreed that *"they find the test cases helpful in surfacing situations where people might have differing opinions about whether the bot's response is appropriate"* ($M = 5.45, SD = 1.40$). Participants found that the test cases *"highlighted gray areas where one person might see the response as appropriate while another might not. For example, in cases involving tone, whether the bot should be more direct or more playful, the test cases made those differences in perspective visible"* (S2, G3). People's differing reactions to these gray areas *"laid the groundwork for further discussion"* (S25, G4) or *"gave us new alterations or entirely new ideas"* (S13, G5). For example, a participant shared, *"The upvote/downvote system works perfectly for finding which behaviors are controversial. [...] We would notice that there's not a pure consensus on one of the saved test case response[s], which would spark a small discussion"* (S6, G5). However, a few participants mentioned that they didn't encounter much disagreement *"because we all had similar thoughts"* (S27, G2). Among the groups that participated, we observed that this tended to occur in the smallest communities.

Finally, participants found the test cases helpful in guiding their collective decision-making on whether to deploy a proposal. For example, as one participant shared: *"It's very easy for a chatbot to misunderstand a prompt, the same goes for a human when only looking at what the prompt says. The actual test cases gives a realistic view of what the implementation will look like. We had a task that referenced religion, which was a controversial topic to some, but the test cases served to show that Botender was doing so in an appropriate way and did not step out of line, and that eventually made us agree on implementing the task"* (S15, G4). Similarly, another participant shared that the test cases sometimes led them to decide against deploying a proposal after seeing how the bot would actually respond: *"I saw this with the gaslighting bot, when people in the group saw the responses it generated were in general invalidating of ones feelings, the group was hesitant on approving it"* (S18, G1). Participants found the test cases useful for collective decision-making because *"the test cases give us a good idea on how the bot reacts to different situations, it also shows where it thrives/where its more limited in its capabilities"* (S13, G5).

6.3.3 *Participants emphasized that Botender's overall workflow and integration with their community platform were helpful in fostering community participation.*

Regarding Botender's overall workflow, participants agreed that *"they can easily collaborate with others in bot design using Botender"* ($M = 6.00, SD = 1.10$). From proposing and iterating on tasks to deploying them, participants found Botender's workflow effectively facilitated collaboration and ensured that tasks reflected the community's needs and norms. For example, participants found the proposal design intuitive and well-organized, making it easy to navigate desired changes: *"I think the interface[s] for creating proposals and tasks were very easy to navigate and made it easy to collaborate with others! I liked seeing the entire list view with everyone's proposals [...] It made it easy for me to see which ones I wanted to vote for, and which ones were ready to deploy"* (S22, G1). Participants also found that being able to collaboratively edit proposals helped improve their quality: *"I felt like a lot of my proposals were pretty brief, so I appreciate [another user] editing mine. More than once I noticed he had edited a proposal to add many more details!"* (S19, G1). Meanwhile, participants appreciated the requirement of upvotes for proposal deployment, as it helps ensure that deployed tasks align with the community's needs and norms: *'[The tasks] fit into the norms and culture of our community. This is evident in the requirement of needing at least 3 upvotes to deploy, meaning that we as a group decided what did and did not mesh'* (S18, G1). In the same vein, the voting requirement also *"prevent[ed] a single person from making a bot that would not serve the community well"* (S7, G5).

Participants also emphasized that deep integration with the community platform they already use is key to keeping community members actively engaged in the bot design process. They agreed that *"the experience of designing the bot with the Botender system integrates well with their usage of Discord"* ($M = 6.14, SD = 1.57$), and shared specific integrations they found particularly valuable.

For example, many participants expressed how much they appreciated having a dedicated discussion thread for each proposal on Discord: *"It was an amazing user experience, having it automatically create threads and open/close proposals automatically was very efficient, and felt very modern"* (S11, G6). They believed that *"the built-in notifications kept everyone in the loop. For example, whenever someone made a new proposal, the notification feature made it easy to see updates right away and give feedback without missing anything"* (S2, G3). They found this approach effectively coordinated discussions because *"using threads for the ticketing system eliminates need to manually organize the discussions and keeps everything in one place"* (S6, G5). In addition to the threading design, participants appreciated how details like the login process and overall aesthetics seamlessly fit with their use of Discord. For example, participants shared *"it's very simple to manage the bot via the OAuth login dashboard and we had no issues because of its deep integration with Discord"* (S6, G5), and *"I think this is my favorite part about Botender! [...] it is very easy to use and very much fits the vibes and general aesthetic of Discord very well"* (S15, G4). Overall, participants found *"Botender fit naturally into how I already use Discord [...], which made collaboration smooth and kept the focus on shaping the bot, not juggling tools"* (S2, G3).

Finally, while using natural language to write prompts is not a unique feature of the Botender system, several participants noted that this greatly facilitates broader community participation in bot design. For example, a participant shared that *"the way it's designed allows me to describe tasks in plain English, and it understands what I mean without needing technical setup. [...] It reduces friction, lets me focus on shaping the community's culture, and makes it easier to adapt the bot's behavior to fit our needs. [...] That makes it accessible not only to me but also to other community members who may not have a technical background"* (S2, G3). A participant without coding background echoed this sentiment, sharing that setting up other bots was much more difficult compared to Botender: *"We liked how easy it was to implement ideas. We could talk about them, and then up vote or down vote them based on what we thought. In the past anything like this was really difficult to use. It took a lot of planning and stress. This did not feel that way at all"* (S27, G2). Overall, participants found *"the bot is a great addition to the server. The fact it's so easy to customize makes it really simple for any member to contribute, and shape something that reflects what we needs"* (S25, G4). In the post-study survey, 96% of participants (30 out of 31) expressed interest in continuing to use Botender after the study.

7 Discussion

Bots play vital roles and act as essential socio-technical infrastructure within online communities. It is crucial for communities to collaboratively design bots that meet their specific needs and norms, rather than leaving this to outsiders or just a few technically skilled members. In this paper, we present Botender, a system that supports communities in collaboratively proposing, iterating on, and deploying bots powered by LLM-based agents. In particular, Botender facilitates this collaborative design process through case-based provocations, concrete interaction scenarios generated to provoke user reflection and discussion about desirable bot behaviors within their community. We conducted a validation study and a field study to understand how people perceive and use these case-based provocations to collaboratively design bots in real-world communities. Through a validation study (Section 5), we saw that participants found Botender's case-based provocations revealed more opportunities for bot improvement, compared with standard test cases. Through a field study (Section 6), we found that real-world Discord communities effectively collaborated on designing bots tailored to their specific needs and norms using Botender's case-based provocations. This collaboration was further supported by Botender's overall collaborative workflow, seamless integration with the community platform, and the use of natural language for bot design.

In this section, we discuss limitations of the current instantiation of Botender that participants wished to overcome, as expressed in their post-study surveys or in their unsuccessful attempts to design bots during the study. For each limitation, we outline opportunities for future HCI research to better support the collaborative and participatory design of community bots powered by AI agents.

7.1 Advancing Case-Based Provocations

7.1.1 ***Limitations****: The current case-based provocation algorithm does not highlight potential conflicts between tasks, and may require more inputs to better surface potential disagreements.* The case-based provocation algorithm presented in this paper represents a proof-of-concept. Our current case-based provocation algorithm generates three specific types of cases, corresponding to common prompt design pitfalls, such as ambiguous phrasing within prompts, which also represent potential areas of disagreement among collaborators. While most participants found that the current case-based provocations effectively facilitated iterative, collaborative design, they also identified opportunities for improvement. For example, participants expressed a desire to see provocations that reveal situations where two tasks might be in conflict, in order to help *"prevent conflicting task triggers"* (S18, G1). As another example discussed in Section 6.3.2, one participant group (G2) didn't encounter much disagreement *"because we all had similar thoughts"* (S27, G2) and *"everybody seemed to be on the same page"* (S29, G2). While this may simply reflect the dynamics of that particular group, it could also indicate opportunities to improve the case generation algorithm, as we discuss below.

7.1.2 ***Opportunities:*** *Future algorithms could generate additional types of case-based provocations and incorporate user feedback on cases to better identify potential areas of disagreement.* There are several opportunities for future research to explore the design of case generation algorithms that can more effectively provoke user reflection and discussion throughout iterative design processes. For example, building on participants' feedback, future work could explore new types of cases not covered in this study, such as those arising from other common prompt design pitfalls, cases that highlight conflicting tasks, or cases that are at the borderline of acceptability for a *specific community*. We see potential for future case-based provocation algorithms to more effectively surface potential disagreements within a community by learning about divergent perspectives within the community over time, based on users' interactions—such as users' thumbs up/down feedback on individual cases or their discussions about bot behavior (cf. [39]).

More broadly, we see exciting opportunities to extend the concept of case-based provocations beyond prompt design. For example, future research could explore how algorithmically generated cases might be used to provoke reflection, deliberation, and improvements in the design of other kinds of interactive technologies, or the development of public policy [41].

7.2 Enhancing Support for Prompt Refinement

7.2.1 ***Limitations:*** *Some participants desired more direct assistance in refining prompts.* With the current version of Botender, users can reflect on and discuss a case generated from a prompt, and then manually refine the prompt as needed. However, some participants anticipated that giving a thumbs up or down to a case would not only surface disagreements among users, but also directly contribute to improving the bot. For example, a participant initially *"expected the bot will learn [from] it and change the response to something else, but it didn't"* (S5, G3). Relatedly, participants expressed a desire to manually correct a case by entering *"the expected output to make the bot learn"* (S5, G3), or to *"directly explain to the bot why a case is good and why a case is bad, so it would be able to improve"* (S7, G5). This feedback highlights a desire for more assistance in prompt refinement, potentially based on users' feedback on cases, rather than leaving the refinement process completely manual.

7.2.2 ***Opportunities:*** *Future systems could offer prompt refinement suggestions based on a variety of human feedback on cases.* Building upon participants' feedback, future systems could consider offering more ways for users to provide feedback on cases, such as allowing them to correct the bot's response or provide direct explanations [53], in addition to giving a thumbs up or down. The system could then incorporate this human feedback to suggest refined prompts, which users would have the agency to adopt, revise, or ignore as they see fit. While a fully automated prompt refinement process, where the bot evolves on its own without users having access to its prompt, may seem simpler and perhaps more desirable, prior HCI research suggests otherwise [69]. Communities often prefer to retain agency and control over the AI systems they use or rely on, rather than having this control completely hidden within the system or managed solely by outsiders [27, 42]. Moreover, communities' needs and preferences may change over time, meaning that objectives set in the past may no longer align with their current goals [29]. For these reasons, we recommend that future work seek a balance, to keep the community actively involved and at the center of the design process.

7.3 Expanding the Capabilities of AI Agents

7.3.1 ***Limitations****: Participants wanted to customize tasks that required bots to handle more than single-turn conversations.* As the first attempt to support collaborative design of AI agents in community contexts, the current version of Botender focuses on enabling the design of single-turn, LLM-based conversational AI agents. This means that the bot and its underlying agents can process only one user message at a time and respond with a single message. Channel names are the only context information that allows the agents to tailor their behavior more specifically. Although participants were aware of this limitation from the beginning of the study, many expressed in the post-study survey a desire for expanded bot capabilities that would allow for greater customization to better address the unique needs of their communities. For example, participants shared feedback such as, *"I would love for Botender to be able to take into context the full conversations!"* (S15, G4), and *"Being able to store data would be neat! For example, setting up a birthday reminder would require saving data about the different users"* (S19, G1). They also expressed interest in additional bot actions beyond single-turn replies, such as *"the ability to block messages"* (S9, G6).

7.3.2 ***Opportunities****: Future bots could build upon Botender's system architecture to expand their capabilities for greater customization.* As mentioned in Section 4.1, Botender's system and agent architecture is designed to handle a broader range of platform events, actions, and context information. On the event side, in addition to listening to user messages, Botender's listener can detect a wider range of platform events provided by the platform API, such as users joining channels, the creation of new threads, or changes to user permissions. These events can be translated into information that agents can interpret and use to provide appropriate action instructions. Similarly, on the action side, Botender's action executor can carry out a broader range of actions available through the API, such as creating new channels, muting users when appropriate, or searching the internet for up-to-date information.[16] The context information available to agents can also be expanded depending on the event. For example, for the event of a user sending a message, the context could include the time, the user's permission, a greater history of previous messages, or stored data about the community [75]. Future systems could provide agents with richer context and expand their capabilities based on what would be most useful to communities. However, as prior research points out [33], granting bots more permissions can inevitably raise concerns among community members about potential misuse and the risk of unrecoverable consequences. Striking the right balance between agent capabilities and user concerns will be an important challenge to address.

7.4 Scaling to Broaden Participation

7.4.1 ***Limitations:*** *How to effectively support scaling as the number of users increases remains an open question.* In our current field study, groups have a maximum of six participants, with deployments on servers of up to 429 members (Table 1). Each group has created a maximum of 37 proposals, and the highest number of tasks deployed per group is 18 (Table 2). While this represents a reasonable size for many small to medium Discord communities, there are also much larger communities, such as the Discord server of the Wikimedia Community, which has nearly 10,000 members. Given Wikimedia's emphasis on broad participation, many of these members may also be interested in contributing to collaborative bot design. As one participant noted, while the current design *"just fits right in, but if you scale up the amount of users making proposals I imagine it could be quite tough"* (S4, G5).

7.4.2 ***Opportunities:*** *Future research could explore what design changes or alternatives might better support broader community participation.* There are several directions that could be pursued. For

[16]We experimented with internet search, but found that it made case generation too slow and affected the user experience. It may become faster as LLMs improve.

example, providing clearer guidance on the division of labor could be helpful. Members familiar with community norms could focus on iterating on proposals, while others could contribute by creating cases to support design iterations. There are also opportunities to lower the barrier to collecting feedback on cases. As one participant suggested, *"allowing users to react to a [bot's] message [directly on Discord, rather than on Botender's web interface], could ease the collection of test cases for editing proposals to better the bot's responses"* (S6, G5). Collective decision-making around proposal deployment will also need to be adapted. For instance, it is important to strike a balance between keeping the requirements for deployment accessible while ensuring that implemented changes truly reflect the broader community's needs. Overall, these challenges relate to broader HCI research on designing systems for community participation, specifically navigating the tradeoff between lowering participation barriers and supporting effective collective action [42, 59, 68].

7.5 Applying to Different Communities

7.5.1 ***Limitations:*** *Botender's collaborative approach to bot design may not be suitable for all communities with different norms.* Botender is designed to support a collaborative approach to bot design, involving participation and deliberation among community members. However, it is important to note that the system alone can hardly overcome the power dynamics that may inevitably exist within different communities. For example, in smaller, close-knit groups, such as those in our study, power dynamics may be more distributed, allowing members to freely propose, iterate on, and deploy desired changes to their bots. However, in communities with more hierarchical power structures, where community leaders prefer to design the bot themselves, the applicability of Botender becomes more complex. On one hand, such communities often require heavier moderation and could benefit from Botender's tailored moderation capabilities. On the other hand, the collaborative features of Botender may be less effective, as members may have fewer opportunities to participate in the design process. This complexity was illustrated by a participant who noted: *"The bot would probably be best suited for larger, more rowdy servers, which already struggle with finding competent moderators."* At the same time, the same participant expressed the desire that *"As someone with higher administrative rights than other members in the server, I think that I should be able to remove a task without needing other people to upvote a proposal"* (S16, G4). Notably, this participant was also the only one (out of 31) who did not express interest in continuing to use Botender after the study.

7.5.2 ***Opportunities:*** *Future research could explore ways to support more collaborative and democratic approaches to bot design across different communities.* Botender is *purposefully* designed to broaden community participation and collaboration in bot design. In fact, the dilemma faced by the participant mentioned above (S16) can be resolved, for example, by setting the deployment threshold of a proposal to one and restricting edit permissions to a single admin. With this configuration, Botender can accommodate this participant's preference by enabling tailored moderation while also centralizing decision-making power in a single individual. However, we have intentionally chosen to pursue a more democratic vision of bot design, and to overcome the unique challenges involved in making this vision a reality. The main design of Botender—including core components like case-based provocations to encourage collective reflection and discussion, along with collaborative features such as discussion threads for each proposal—has all been deliberatively designed to support this vision. We hope that Botender provides community members with an option for a more collaborative, bottom-up approach to bot design, and that our work inspires further research and systems that enable more democratic approaches to community governance [21, 41, 42, 54, 84].

8 Conclusion

In this work, we have demonstrated how a system can support users in collaborative bot design through case-based provocations. Our findings show that these provocations can effectively surface opportunities for bot improvement, reveal potential sources of disagreement, and support the collaborative bot design process in real online communities. Building on this work, future HCI systems should explore expanding bot capabilities to meet diverse community needs, explore the design of more advanced case-based provocation techniques, address scaling challenges to enable broader participation, and navigate power dynamics within communities.

Acknowledgments

The funding for this research was provided by CMU's Block Center for Technology and Society, Metagov's Grant for Interoperable Deliberative Tools, the National Institute of Standards and Technology (NIST) (ror.org/05xpvk416) and the CMU (ror.org/05x2bcf33) AI Measurement Science and Engineering Center. Tzu-Sheng Kuo (ORCID: 0000-0002-1504-7640) was funded by the K&L Gates Presidential Fellowship in Ethics and Computational Technologies and NIST through Federal Award ID Number 60NANB24D231. We thank Michael Bernstein, Aniket Kittur, Sherry Tongshuang Wu, Nikolas Martelaro, Chien-Sheng Jason Wu, Pranav Khadpe, K. J. Kevin Feng, Xingyu Bruce Liu, and Tiffany Chih for their insightful feedback on the system design. We are also grateful to Aileen Benedict, Sameer Patil, Estelle Smith, Rotem Guttman, and Joon Jang for their assistance with recruitment for the study. Finally, we thank Isadora Krsek for designing the Botender logo.

References

[1] Agnar Aamodt and Enric Plaza. 1994. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Commun.* 7, 1 (mar 1994), 39–59.

[2] Saleema Amershi, Maya Cakmak, W. Bradley Knox, and Todd Kulesza. 2014. Power to the People: The Role of Humans in Interactive Machine Learning. *AI Mag.* 35, 4 (Dec. 2014), 105–120. doi:10.1609/aimag.v35i4.2513

[3] Yan Aquino, Pedro Bento, Arthur Buzelin, Lucas Dayrell, Samira Malaquias, Caio Santana, Victoria Estanislau, Pedro Dutenhefner, Guilherme HG Evangelista, Luisa G Porfirio, et al. 2025. Discord Unveiled: A Comprehensive Dataset of Public Communication (2015-2024). *arXiv preprint arXiv:2502.00627* (2025).

[4] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena L. Glassman. 2024. ChainForge: A Visual Toolkit for Prompt Engineering and LLM Hypothesis Testing. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 304, 18 pages. doi:10.1145/3613904.3642016

[5] Zahra Ashktorab, Michael Desmond, James M. Johnson, Qian Pan, Casey Dugan, Michelle Brachman, and Carolina Spina. 2023. SME-in-the-loop: Interaction Preferences when Supervising Bots in Human-AI Communities. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference* (Pittsburgh, PA, USA)

(DIS '23). Association for Computing Machinery, New York, NY, USA, 2281–2303. doi:10.1145/3563657.3596100

[6] Stephen 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 Fries, Maged Al-shaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Dragomir Radev, Mike Tian-jian Jiang, and Alexander Rush. 2022. PromptSource: An Integrated Development Environment and Repository for Natural Language Prompts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Valerio Basile, Zornitsa Kozareva, and Sanja Stajner (Eds.). Association for Computational Linguistics, Dublin, Ireland, 93–104. doi:10.18653/v1/2022.acl-demo.9

[7] Shreya Bali, Pranav Khadpe, Geoff Kaufman, and Chinmay Kulkarni. 2023. Nooks: Social Spaces to Lower Hesitations in Interacting with New People at Work. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 614, 18 pages. doi:10.1145/3544548.3580796

[8] Cheng Chen, Sangwook Lee, Eunchae Jang, and S. Shyam Sundar. 2024. Is Your Prompt Detailed Enough? Exploring the Effects of Prompt Coaching on Users' Perceptions, Engagement, and Trust in Text-to-Image Generative AI Tools. In *Proceedings of the Second International Symposium on Trustworthy Autonomous Systems* (Austin, TX, USA) *(TAS '24)*. Association for Computing Machinery, New York, NY, USA, Article 9, 12 pages. doi:10.1145/3686038.3686060

[9] Nan-Chen Chen, Jina Suh, Johan Verwey, Gonzalo Ramos, Steven Drucker, and Patrice Simard. 2018. AnchorViz: Facilitating Classifier Error Discovery through Interactive Semantic Data Exploration. In *Proceedings of the 23rd International Conference on Intelligent User Interfaces* (Tokyo, Japan) *(IUI '18)*. Association for Computing Machinery, New York, NY, USA, 269–280. doi:10.1145/3172944.3172950

[10] Quan Ze Chen and Amy X. Zhang. 2023. Judgment Sieve: Reducing Uncertainty in Group Judgments through Interventions Targeting Ambiguity versus Disagreement. *Proc. ACM Hum.-Comput. Interact.* 7, CSCW2, Article 283 (Oct. 2023), 26 pages. doi:10.1145/3610074

[11] Quan Ze Chen and Amy Xian Zhang. 2025. Case Law Grounding: Using Precedents to Align Decision-Making for Humans and AI. In *Proceedings of the ACM Collective Intelligence Conference*. 226–238.

[12] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2014. Transparency and coordination in peer production. *arXiv preprint arXiv:1407.0377* (2014).

[13] Bich Ngoc (Rubi) Doan and Joseph Seering. 2025. The Design Space for Online Restorative Justice Tools: A Case Study with ApoloBot. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 694, 19 pages. doi:10.1145/3706598.3713598

[14] Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. 2000. Prioritizing test cases for regression testing. *SIGSOFT Softw. Eng. Notes* 25, 5 (Aug. 2000), 102–112. doi:10.1145/347636.348910

[15] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 235–245. doi:10.1145/2635868.2635910

[16] Will Epperson, Gagan Bansal, Victor C Dibia, Adam Fourney, Jack Gerrits, Erkang (Eric) Zhu, and Saleema Amershi. 2025. Interactive Debugging and Steering of Multi-Agent AI Systems. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 156, 15 pages. doi:10.1145/3706598.3713581

[17] Dmitry Epstein, Cynthia Farina, and Josiah Heidt. 2014. The value of words: Narrative as evidence in policy making. *Evidence & Policy* 10, 2 (2014), 243–258.

[18] Jerry Alan Fails and Dan R. Olsen. 2003. Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces* (Miami, Florida, USA) *(IUI '03)*. Association for Computing Machinery, New York, NY, USA, 39–45. doi:10.1145/604045.604056

[19] Jenny Fan and Amy X. Zhang. 2020. Digital Juries: A Civics-Oriented Approach to Platform Governance. 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, 1–14. doi:10.1145/3313831.3376293

[20] Sina Fazelpour and Will Fleisher. 2025. The Value of Disagreement in AI Design, Evaluation, and Alignment. In *Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency (FAccT '25)*. Association for Computing Machinery, New York, NY, USA, 2138–2150. doi:10.1145/3715275.3732146

[21] K. J. Kevin Feng, Rock Yuren Pang, Tzu-Sheng Kuo, Amy Winecoff, Emily Tseng, David Gray Widder, Harini Suresh, Katharina Reinecke, and Amy X. Zhang. 2025. Sociotechnical AI Governance: Challenges and Opportunities for HCI. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '25)*. Association for Computing Machinery, New York, NY, USA, Article 800, 6 pages. doi:10.1145/3706599.3706747

[22] Li Feng, Ryan Yen, Yuzhe You, Mingming Fan, Jian Zhao, and Zhicong Lu. 2024. CoPrompt: Supporting Prompt Sharing and Referring in Collaborative Natural Language Programming. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 934, 21 pages. doi:10.1145/3613904.3642212

[23] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. 2024. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468* (2024).

[24] R Stuart Geiger. 2014. Bots, bespoke, code and the materiality of software platforms. *Information, Communication & Society* 17, 3 (2014), 342–356.

[25] R Stuart Geiger and Aaron Halfaker. 2013. When the levee breaks: without bots, what happens to Wikipedia's quality control processes?. In *Proceedings of the 9th International Symposium on Open Collaboration*. 1–6.

[26] R Stuart Geiger and David Ribes. 2010. The work of sustaining order in Wikipedia: The banning of a vandal. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. 117–126.

[27] Aaron Halfaker and R. Stuart Geiger. 2020. ORES: Lowering Barriers with Participatory Machine Learning in Wikipedia. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW2, Article 148 (Oct. 2020), 37 pages. doi:10.1145/3415219

[28] Aaron Halfaker, Aniket Kittur, and John Riedl. 2011. Don't bite the newbies: how reverts affect the quantity and quality of Wikipedia work. In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration* (Mountain View, California) *(WikiSym '11)*. Association for Computing Machinery, New York, NY, USA, 163–172. doi:10.1145/2038558.2038585

[29] Aaron L Halfaker, Tzu-Sheng Kuo, Ciell Brusse, Kenneth Holstein, and Haiyi Zhu. 2025. Collective Meaning Cascades but Strange Ducks Swim Upstream: Facilitating Collective Meaning-making through Co-development of AI Models. In *Extended Abstracts of the 2025 CHI Conference on Human Factors in Computing Systems (CHI EA '25)*. doi:10.1145/3706599.3706683

[30] Jonggi Hong, Kyungjun Lee, June Xu, and Hernisa Kacorri. 2020. Crowdsourcing the Perception of Machine Teaching. 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, 1–14. doi:10.1145/3313831.3376428

[31] Stephanie Houde, Kristina Brimijoin, Michael Muller, Steven I. Ross, Dario Andres Silva Moran, Gabriel Enrique Gonzalez, Siya Kunde, Morgan A. Foreman, and Justin D. Weisz. 2025. Controlling AI Agent Participation in Group Conversations: A Human-Centered Approach. In *Proceedings of the 30th International Conference on Intelligent User Interfaces (IUI '25)*. Association for Computing Machinery, New York, NY, USA, 390–408. doi:10.1145/3708359.3712089

[32] Saffron Huang, Divya Siddarth, Liane Lovitt, Thomas I. Liao, Esin Durmus, Alex Tamkin, and Deep Ganguli. 2024. Collective Constitutional AI: Aligning a Language Model with Public Input. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency* (Rio de Janeiro, Brazil) *(FAccT '24)*. Association for Computing Machinery, New York, NY, USA, 1395–1417. doi:10.1145/3630106.3658979

[33] Sohyeon Hwang, Charles Kiene, Serene Ong, and Aaron Shaw. 2024. Adopting Third-party Bots for Managing Online Communities. *Proc. ACM Hum.-Comput. Interact.* 8, CSCW1, Article 216 (April 2024), 26 pages. doi:10.1145/3653707

[34] Lilly C. Irani and M. Six Silberman. 2013. Turkopticon: interrupting worker invisibility in amazon mechanical turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 611–620. doi:10.1145/2470654.2470742

[35] Shagun Jhaver, Iris Birman, Eric Gilbert, and Amy Bruckman. 2019. Human-Machine Collaboration for Content Regulation: The Case of Reddit Automoderator. *ACM Trans. Comput.-Hum. Interact.* 26, 5, Article 31 (July 2019), 35 pages. doi:10.1145/3338243

[36] Soomin Kim, Jinsu Eun, Changhoon Oh, Bongwon Suh, and Joonhwan Lee. 2020. Bot in the Bunch: Facilitating Group Chat Discussion by Improving Efficiency and Participation with a Chatbot. 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. doi:10.1145/3313831.3376785

[37] Reuben Kirkham. 2023. (Legal Design) Research through Litigation. arXiv:2303.14336 [cs.HC] https://arxiv.org/abs/2303.14336

[38] Carl Knight. 2017. Reflective equilibrium. *Methods in analytical political theory* (2017), 46–64.

[39] Vinay Koshy, Frederick Choi, Yi-Shyuan Chiang, Hari Sundaram, Eshwar Chandrasekharan, and Karrie Karahalios. 2025. Venire: A Machine Learning-Guided Panel Review System for Community Content Moderation. *Proc. ACM Hum.-Comput. Interact.* 9, 7, Article CSCW518 (Oct. 2025), 35 pages. doi:10.1145/3757699

[40] Robert E Kraut and Paul Resnick. 2012. *Building successful online communities: Evidence-based social design*. Mit Press.

[41] Tzu-Sheng Kuo, Quan Ze Chen, Amy X. Zhang, Jane Hsieh, Haiyi Zhu, and Kenneth Holstein. 2025. PolicyCraft: Supporting Collaborative and Participatory Policy Design through Case-Grounded Deliberation. In *Proceedings of the 2025*

CHI Conference on Human Factors in Computing Systems (CHI '25). Association for Computing Machinery, New York, NY, USA, Article 805, 24 pages. doi:10.1145/3706598.3713865

[42] Tzu-Sheng Kuo, Aaron Lee Halfaker, Zirui Cheng, Jiwoo Kim, Meng-Hsin Wu, Tongshuang Wu, Kenneth Holstein, and Haiyi Zhu. 2024. Wikibench: Community-Driven Data Curation for AI Evaluation on Wikipedia. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 193, 24 pages. doi:10.1145/3613904.3642278

[43] Tzu-Sheng Kuo, Hong Shen, Jisoo Geum, Nev Jones, Jason I. Hong, Haiyi Zhu, and Kenneth Holstein. 2023. Understanding Frontline Workers' and Unhoused Individuals' Perspectives on AI Used in Homeless Services. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 860, 17 pages. doi:10.1145/3544548.3580882

[44] Juhoon Lee, Bich Ngoc (Rubi) Doan, Jonghyun Jee, and Joseph Seering. 2025. Mapping Community Appeals Systems: Lessons for Community-led Moderation in Multi-Level Governance. *Proc. ACM Hum.-Comput. Interact.* 9, 7, Article CSCW446 (Oct. 2025), 28 pages. doi:10.1145/3757627

[45] Yoonho Lee, Michelle S. Lam, Helena Vasconcelos, Michael S. Bernstein, and Chelsea Finn. 2024. Clarify: Improving Model Robustness With Natural Language Corrections. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 133, 19 pages. doi:10.1145/3654777.3676362

[46] Pascale Lehoux, Fiona Alice Miller, and Bryn Williams-Jones. 2020. Anticipatory governance and moral imagination: Methodological insights from a scenario-based public deliberation study. *Technological Forecasting and Social Change* 151 (2020), 119800.

[47] Xian Li, Yuanning Han, Di Liu, Pengcheng An, and Shuo Niu. 2024. FlowGPT: Exploring Domains, Output Modalities, and Goals of Community-Generated AI Chatbots. In *Companion Publication of the 2024 Conference on Computer-Supported Cooperative Work and Social Computing*. 355–361.

[48] Michael Xieyang Liu, Savvas Petridis, Vivian Tsai, Alexander J. Fiannaca, Alex Olwal, Michael Terry, and Carrie J. Cai. 2025. Gensors: Authoring Personalized Visual Sensors with Multimodal Foundation Models and Reasoning. In *Proceedings of the 30th International Conference on Intelligent User Interfaces (IUI '25)*. Association for Computing Machinery, New York, NY, USA, 755–770. doi:10.1145/3708359.3712085

[49] Xingyu Bruce Liu, Shitao Fang, Weiyan Shi, Chien-Sheng Wu, Takeo Igarashi, and Xiang 'Anthony' Chen. 2025. Proactive Conversational Agents with Inner Thoughts. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 184, 19 pages. doi:10.1145/3706598.3713760

[50] Kiel Long, John Vines, Selina Sutton, Phillip Brooker, Tom Feltwell, Ben Kirman, Julie Barnett, and Shaun Lawson. 2017. "Could You Define That in Bot Terms"? Requesting, Creating and Using Bots on Reddit. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 3488–3500. doi:10.1145/3025453.3025830

[51] Ryan Louie, Ananjan Nandi, William Fang, Cheng Chang, Emma Brunskill, and Diyi Yang. 2024. Roleplay-doh: Enabling Domain-Experts to Create LLM-simulated Patients via Eliciting and Adhering to Principles. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 10570–10603. doi:10.18653/v1/2024.emnlp-main.591

[52] Qianou Ma, Weirui Peng, Chenyang Yang, Hua Shen, Ken Koedinger, and Tongshuang Wu. 2025. What Should We Engineer in Prompts? Training Humans in Requirement-Driven LLM Use. *ACM Trans. Comput.-Hum. Interact.* 32, 4, Article 41 (Aug. 2025), 27 pages. doi:10.1145/3731756

[53] Richard G. McDaniel and Brad A. Myers. 1999. Getting more out of programming-by-demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) *(CHI '99)*. Association for Computing Machinery, New York, NY, USA, 442–449. doi:10.1145/302979.303127

[54] Aviv Ovadya, Kyle Redman, Luke Thorburn, Quan Ze Chen, Oliver Smith, Flynn Devine, Andrew Konya, Smitha Milli, Manon Revel, Kevin Feng, et al. [n. d.]. Position: Democratic AI is Possible. The Democracy Levels Framework Shows How It Might Work.. In *Forty-second International Conference on Machine Learning Position Paper Track*.

[55] Joon Sung Park, Lindsay Popowski, Carrie Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2022. Social Simulacra: Creating Populated Prototypes for Social Computing Systems. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (Bend, OR, USA) *(UIST '22)*. Association for Computing Machinery, New York, NY, USA, Article 74, 18 pages. doi:10.1145/3526113.3545616

[56] Savvas Petridis, Benjamin D Wedin, James Wexler, Mahima Pushkarna, Aaron Donsbach, Nitesh Goyal, Carrie J Cai, and Michael Terry. 2024. ConstitutionMaker: Interactively Critiquing Large Language Models by Converting Feedback into Principles. In *Proceedings of the 29th International Conference on Intelligent User Interfaces* (Greenville, SC, USA) *(IUI '24)*. Association for Computing Machinery, New York, NY, USA, 853–868. doi:10.1145/3640543.3645144

[57] Mohi Reza, Ioannis Anastasopoulos, Shreya Bhandari, and Zachary A Pardos. 2025. PromptHive: Bringing subject matter experts back to the forefront with collaborative prompt engineering for educational content creation. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–22.

[58] Gregg Rothermel and Mary Jean Harrold. 1997. A safe, efficient regression test selection technique. *ACM Trans. Softw. Eng. Methodol.* 6, 2 (April 1997), 173–210. doi:10.1145/248233.248262

[59] Niloufar Salehi, Lilly C. Irani, Michael S. Bernstein, Ali Alkhatib, Eva Ogbe, Kristy Milland, and Clickhappier. 2015. We Are Dynamo: Overcoming Stalling and Friction in Collective Action for Crowd Workers. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 1621–1630. doi:10.1145/2702123.2702508

[60] Vagner Figueredo de Santana, Sara Berger, Tiago Machado, Maysa Malfiza Garcia de Macedo, Cassia Sampaio Sanctos, Lemara Williams, and Zhaoqing Wu. 2025. Can LLMs Recommend More Responsible Prompts?. In *Proceedings of the 30th International Conference on Intelligent User Interfaces (IUI '25)*. Association for Computing Machinery, New York, NY, USA, 298–313. doi:10.1145/3708359.3712137

[61] Vagner Figueredo de Santana, Sara E Berger, Heloisa Candello, Tiago Machado, Cassia Sampaio Sanctos, Tianyu Su, and Lemara Williams. 2025. Responsible Prompting Recommendation: Fostering Responsible AI Practices in Prompting-Time. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 836, 30 pages. doi:10.1145/3706598.3713365

[62] Saiph Savage, Andres Monroy-Hernandez, and Tobias Höllerer. 2016. Botivist: Calling Volunteers to Action using Online Bots. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (San Francisco, California, USA) *(CSCW '16)*. Association for Computing Machinery, New York, NY, USA, 813–822. doi:10.1145/2818048.2819985

[63] Joseph Seering, Juan Pablo Flores, Saiph Savage, and Jessica Hammer. 2018. The Social Roles of Bots: Evaluating Impact of Bots on Discussions in Online Communities. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 157 (Nov. 2018), 29 pages. doi:10.1145/3274426

[64] Joseph Seering, Manas Khadka, Nava Haghighi, Tanya Yang, Zachary Xi, and Michael Bernstein. 2024. Chillbot: Content Moderation in the Backchannel. *Proc. ACM Hum.-Comput. Interact.* 8, CSCW2, Article 402 (Nov. 2024), 26 pages. doi:10.1145/3686941

[65] Joseph Seering, Michal Luria, Geoff Kaufman, and Jessica Hammer. 2019. Beyond Dyadic Interactions: Considering Chatbots as Community Members. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. doi:10.1145/3290605.3300680

[66] Joseph Seering, Michal Luria, Connie Ye, Geoff Kaufman, and Jessica Hammer. 2020. It Takes a Village: Integrating an Adaptive Chatbot into an Online Gaming Community. 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. doi:10.1145/3313831.3376708

[67] Joseph Seering, Tony Wang, Jina Yoon, and Geoff Kaufman. 2019. Moderator engagement and community development in the age of algorithms. *New media & society* 21, 7 (2019), 1417–1443.

[68] Aaron Shaw, Haoqi Zhang, Andrés Monroy-Hernández, Sean Munson, Benjamin Mako Hill, Elizabeth Gerber, Peter Kinnaird, and Patrick Minder. 2014. Computer supported collective action. *Interactions* 21, 2 (mar 2014), 74–77. doi:10.1145/2576875

[69] C. Estelle Smith, Bowen Yu, Anjali Srivastava, Aaron Halfaker, Loren Terveen, and Haiyi Zhu. 2020. Keeping Community in the Loop: Understanding Wikipedia Stakeholder Values for Machine Learning-Based Systems. 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, 1–14. doi:10.1145/3313831.3376783

[70] Taylor Sorensen, Jared Moore, Jillian Fisher, Mitchell Gordon, Niloofar Mireshghallah, Christopher Michael Rytting, Andre Ye, Liwei Jiang, Ximing Lu, Nouha Dziri, et al. 2024. A roadmap to pluralistic alignment. *arXiv preprint arXiv:2402.05070* (2024).

[71] H. Colleen Stuart, Laura Dabbish, Sara Kiesler, Peter Kinnaird, and Ruogu Kang. 2012. Social transparency in networked information exchange: a theoretical framework. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work* (Seattle, Washington, USA) *(CSCW '12)*. Association for Computing Machinery, New York, NY, USA, 451–460. doi:10.1145/2145204.2145275

[72] Hari Subramonyam, Divy Thakkar, Andrew Ku, Juergen Dieber, and Anoop K. Sinha. 2025. Prototyping with Prompts: Emerging Approaches and Challenges in Generative AI Design for Collaborative Software Teams. In *Proceedings of the 2025*

CHI Conference on Human Factors in Computing Systems (CHI '25). Association for Computing Machinery, New York, NY, USA, Article 882, 22 pages. doi:10.1145/3706598.3713166

[73] Harini Suresh, Emily Tseng, Meg Young, Mary Gray, Emma Pierson, and Karen Levy. 2024. Participation in the age of foundation models. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency* (Rio de Janeiro, Brazil) *(FAccT '24)*. Association for Computing Machinery, New York, NY, USA, 1609–1621. doi:10.1145/3630106.3658992

[74] Tiffany Tseng, Jennifer King Chen, Mona Abdelrahman, Mary Beth Kery, Fred Hohman, Adriana Hilliard, and R. Benjamin Shapiro. 2023. Collaborative Machine Learning Model Building with Families Using Co-ML. In *Proceedings of the 22nd Annual ACM Interaction Design and Children Conference* (Chicago, IL, USA) *(IDC '23)*. Association for Computing Machinery, New York, NY, USA, 40–51. doi:10.1145/3585088.3589356

[75] Ruotong Wang, Xinyi Zhou, Lin Qiu, Joseph Chee Chang, Jonathan Bragg, and Amy X. Zhang. 2025. Social-RAG: Retrieving from Group Interactions to Socially Ground AI Generation. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 162, 25 pages. doi:10.1145/3706598.3713749

[76] Zijie J Wang, Aishwarya Chakravarthy, David Munechika, and Duen Horng Chau. 2024. Wordflow: Social prompt engineering for large language models. *arXiv preprint arXiv:2401.14447* (2024).

[77] Zijie J. Wang, Chinmay Kulkarni, Lauren Wilcox, Michael Terry, and Michael Madaio. 2024. Farsight: Fostering Responsible AI Awareness During AI Application Prototyping. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 976, 40 pages. doi:10.1145/3613904.3642335

[78] David Wright, Bernd Stahl, and Tally Hatzakis. 2020. Policy scenarios as an instrument for policymakers. *Technological Forecasting and Social Change* 154 (2020), 119972.

[79] Tongshuang Wu, Michael Terry, and Carrie Jun 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* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 385, 22 pages. doi:10.1145/3491102.3517582

[80] Qian Yang, Richmond Y. Wong, Steven Jackson, Sabine Junginger, Margaret D. Hagan, Thomas Gilbert, and John Zimmerman. 2024. The Future of HCI-Policy Collaboration. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 820, 15 pages. doi:10.1145/3613904.3642771

[81] 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 *Proceedings of the 2023 ACM Designing Interactive Systems Conference* (Pittsburgh, PA, USA) *(DIS '23)*. Association for Computing Machinery, New York, NY, USA, 2206–2220. doi:10.1145/3563657.3596138

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

[83] Amy X. Zhang and Justin Cranshaw. 2018. Making Sense of Group Chat through Collaborative Tagging and Summarization. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 196 (nov 2018), 27 pages. doi:10.1145/3274465

[84] Amy X. Zhang, Grant Hugh, and Michael S. Bernstein. 2020. PolicyKit: Building Governance in Online Communities. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 365–378. doi:10.1145/3379337.3415858

[85] Jingyue Zhang and Ian Arawjo. 2025. ChainBuddy: An AI-assisted Agent System for Generating LLM Pipelines. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 241, 21 pages. doi:10.1145/3706598.3714085

[86] Qingxiao Zheng, Yiliu Tang, Yiren Liu, Weizi Liu, and Yun Huang. 2022. UX Research on Conversational Human-AI Interaction: A Literature Review of the ACM Digital Library. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 570, 24 pages. doi:10.1145/3491102.3501855

A Prompts Used in the Botender System

In this section, we provide all the prompts used for the LLM in the Botender system, including prompts for the AI agents that power the bot and the LLM module used in Botender's case-based provocation algorithm. *System prompts* refer to the instructions given to each LLM, while *user prompts* are the inputs provided to the LLM to generate responses.

A.1 Prompts for the AI Agents

A.1.1 Orchestrator Agent.

System Prompt:

```
You are a helpful assistant tasked with determining whether a task
should be triggered based on a user's message in a specific channel.
You will receive a list of tasks, each with an associated ID and
trigger condition, as well as the user's message and the channel where
it was sent. If the message is relevant to the trigger condition of a
specific task, respond with that task's ID. If the message is relevant
to multiple tasks, respond with the ID of the task to which it is most
relevant. If the message does not match any task trigger, respond with
0. Your response must be a JSON object with a single key "taskId". For
example: {"taskId": "some-task-id"} or {"taskId": "0"}.
```

User Prompt:

```
Here is a list of tasks:

Task ID: [taskId]
Task Trigger: [task trigger]
⋮
Task ID: [taskId]
Task Trigger: [task trigger]

User message in the #[channel] channel: [user message]
```

A.1.2 Task-Specific Agent.

System Prompt:

```
You are a helpful assistant tasked with responding to a user's message
in a specific channel, following the instructions provided in an
assigned action. You will receive the action instructions, the user's
message, and the channel where it was sent. Based on the action,
compose an appropriate reply. If you determine that no response is
necessary, use "n/a". Your response must be a JSON object with a single
key "response". For example: {"response": "Here is your reply."} or
{"response": "n/a"}.
```

User Prompt:

```
Action: [task action].
User message in the #[channel] channel: [user message]
```

A.2 Prompts for the Case-Based Provocations

As shown in Figure 6, Botender's case-based provocation algorithm consists of three LLM pipelines. Each pipeline includes a *detector*, *generator*, and *evaluator* for creating a specific type of provocative case. The prompts for all nine of these LLMs, as well as the final *selector*, are provided in this section. Note that some shared system prompts are repeated across different LLMs, including the instruction limiting the bot's capability to single-turn conversation, the input specification indicating that the bot receives a channel and

user message as input, and the community description. The community description describes the general tone of the community and can be customized to make cases more relevant to a specific group, although none of the participant groups chose to modify the default description. All of the shared system prompts are provided at the end.

A.2.1 Pipeline for Revealing Ambiguous Phrases.

Detector System Prompt:

You are a helpful assistant tasked with identifying critical ambiguities in prompts written for language model-based bots deployed within an online community. This prompt defines:

- A trigger: when the bot should take action.
- An action: what the bot should do when triggered.

< bot capability >

Your Task:

Read the full prompt carefully. Identify specific phrases or instructions that are ambiguous, underspecified, and open to multiple reasonable interpretations. Focus exclusively on ambiguities that could cause:

- Vague or undefined concepts
- Unclear boundaries or thresholds
- Conflicting or competing goals
- Situational or contextual assumptions
- Ambiguity about what, when, or how the bot is supposed to act

Prioritize ambiguities that could lead to reasonable differences in human interpretation, especially those where people might disagree about whether the bot's behavior is desirable. Focus on ambiguities that could cause visible inconsistencies in the bot's behavior. Do not list trivial ambiguities, style differences, or issues that would not affect how real users experience the bot.

Output Format:

Return a JSON object containing an array of ambiguities. Each ambiguity should have a unique key starting from 0 and include the following two properties:

- underspecified_phrase: a specific quote or snippet from the prompt that is ambiguous
- description: a 1-2 sentence explanation of what makes it ambiguous or open to multiple interpretations

All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Detector User Prompt:

Prompt:

- Trigger: [task trigger]
- Action: [task action]

Generator System Prompt:

You are a helpful assistant tasked with generating input test cases that explore how ambiguous phrases in a bot's prompt could be interpreted in different, plausible ways. This prompt defines:

- A trigger: when the bot should take action.
- An action: what the bot should do when triggered.

< bot capability >

You will be provided with:

- prompt: the full prompt for the bot, containing one or more ambiguous phrases.
- underspecified_phrase: a specific snippet from the prompt that is ambiguous.
- description: a 1-2 sentence explanation describing why the phrase is ambiguous or can be interpreted in multiple ways.

Your Task:

For each underspecified_phrase, generate a small set of test cases that illustrate distinct, plausible alternative interpretations of the phrase. A test case is an input to the bot that adheres to the following input specification:

< input specification >

When generating test cases, prioritize those that provoke visible divergence in bot behavior—either in whether the bot responds (trigger ambiguity) or in how the bot responds (action ambiguity). Aim to create test cases that illustrate non-obvious yet reasonable interpretations, revealing hidden assumptions, unclear boundaries, or conflicting objectives within the original, underspecified phrase. If the ambiguity influences the bot's action, design the test case to elicit a bot response that clearly diverges from its typical default response. If the ambiguity concerns the trigger, focus on whether the bot responds or not. Each test case should make the ambiguity evident at the surface level, discernible from the channel, user message, and bot response alone, without the need for additional explanation.

Additionally, the test cases should be realistic and natural, mirroring the typical messages found in the following community and reflecting its unique tone:

< community description >

Do not generate test cases based on literal, overly obvious, or superficial interpretations. Avoid creating test cases that only involve minor tone or style differences, unless these differences have a clear impact on user-facing behavior. Additionally, do not include cases that would not affect how humans perceive or interact with the bot.

Output Format:

Return a JSON object containing an array of the generated test cases. Each case should have a unique key starting from 0 and include the following four properties.

- underspecified_phrase: the specific snippet from the prompt that is ambiguous.
- interpretation: a plausible alternative interpretation of the phrase that the test case is generated to illustrate.
- reasoning: a brief explanation of how the test case reveals the ambiguity.
- case: the input test case, formatted according to the input specification.

All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Generator User Prompt:

prompt:

- Trigger: [task trigger]
- Action: [task action]

underspecified_phrase: [underspecified_phrase from the detector's output]
description: [description from the detector's output]

Evaluator System Prompt:

You are a helpful assistant tasked with evaluating whether a test case clearly demonstrates a plausible and critical alternative interpretation of an ambiguous phrase in a bot's prompt. This prompt defines:

- A trigger: when the bot should take action.
- An action: what the bot should do when triggered.

< bot capability >

You will be provided with:

- prompt: the full prompt for the bot, including both the trigger and action components.
- underspecified_phrase: a specific snippet from the prompt that is ambiguous.
- interpretation: a plausible alternative interpretation of the phrase that the test case is intended to illustrate.
- reasoning: a brief explanation describing how the test case could demonstrate this interpretation.
- case: the test case itself, including the user message in a specific channel, the specific task triggered for the bot (if any), and the corresponding bot response to that task.

It is possible that the user input does not trigger any task, or that the bot chooses not to respond even if a task is triggered.

Your Task:

Decide whether the test case clearly and directly demonstrates the intended interpretation based only on the channel, user message, and bot response. The ambiguity must be apparent to a human without explanation. Only approve the case if it clearly tests the goal stated in the input reasoning.

At the same time, reject any test cases where the scenario assumes the bot can perform actions beyond its defined capabilities. Also, reject cases where the interpretation shown is non-critical—that is, it does not impact user understanding or the bot's behavior. Additionally, reject test cases that simply reflect an expected, default, or literal reading of the ambiguous phrase, as well as those where the demonstrated interpretation is too subtle for an average human to notice.

If the ambiguity involves how the bot should respond—meaning the action within the prompt is underspecified—consider the following additional steps: First, infer the generalized or default response the bot would typically give based on the prompt and input. Next, compare this default response to the bot's actual response in the test case. Approve the case only if the actual response shows a clear and noticeable difference from the default in terms of tone, structure, or content, such that the change would be obvious to a human observer. Minor shifts in tone, phrasing, or politeness do not count unless they lead to a significant change in the bot's observable behavior.

Output Format:

Return a JSON object with the following two properties:

- label: a boolean value—true if the test case visibly and meaningfully demonstrates the intended interpretation of the underspecified phrase; false if it does not, or if it is rejected.
- label_explanation: a brief, 1 to 2 sentence explanation supporting your decision.

All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Evaluator User Prompt:

prompt:

- Trigger: [task trigger]
- Action: [task action]

underspecified_phrase: [underspecified_phrase from the detector's output]
interpretation: [interpretation from the generator's output]
reasoning: [reasoning from the generator's output]
case:

- channel: [channel from the generator's output]
- user message: [user message from the generator's output]
- triggered task: [triggered task from the bot]
- bot response: [bot response from the bot]

A.2.2 Pipeline for Revealing Overly Narrow Phrases.

Detector System Prompt:

You are a helpful assistant tasked with identifying critical overspecified phrases in prompts written for language model-based bots. This prompt defines:

- A trigger: when the bot should take action.
- An action: what the bot should do when triggered.

< bot capability >

Your Task:

Read the full prompt carefully. Identify overspecified phrases—parts of the prompt that unnecessarily limit the bot's behavior or responses, phrased too narrowly, rigidly, or tied to surface-level specifics. These may prevent the bot from fulfilling its broader functional purpose.

Follow these steps to complete your task:

1. Infer the Broader Goal: Read the full prompt carefully. Infer the broadest reasonable functional goal: what the bot is ultimately intended to detect, prevent, encourage, or support, independent of any surface-level constraints or examples mentioned in the wording of the prompt. Focus on the underlying user problem, situation, or need that the bot is designed to address. Ignore specific conditions, instances, or implementation details unless they are essential to the bot's purpose. Express the broader goal as what the bot should ideally support, if it were not constrained by unnecessary restrictions.
2. Identify Overspecified Phrases: Identify specific snippets of the prompt that unnecessarily constrain how the bot can fulfill its broader goal. Focus on requirements tied to particular content types, formats, channels, or contexts; examples treated as strict conditions; and narrow definitions that exclude plausible situations fitting the broader goal.
3. Define Uncovered Scenarios: For each overspecified phrase, describe as thoroughly as possible the set of scenarios that are currently excluded because of the restrictive wording. These scenarios should fit within the broader goal and could reasonably be handled by the bot without requiring any expansion of its capabilities.

Important: Do not include scenarios that are already covered by the current overspecified phrase. Think of uncovered scenarios as the portion of the broader goal left unaddressed due to the overspecified phrase. Apply deliberate creativity: consider realistic, plausible situations that are missed due to unnecessary specificity. Focus on diverse, meaningful cases that reflect the variety of user needs the bot is intended to support. Prioritize scenarios that are plausible within the community where the bot is deployed, likely to arise in typical use, and distinct from one another in form, context, or content.

Output Format:

Return a JSON object containing an array of overspecified phrases. Each phrase should have a unique key starting from 0 and include:

- broader_goal: the broader goal of the prompt, as you inferred from its content.

- overspecified_phrase: a specific quote or snippet from the prompt that is overly specific.
- uncovered_scenarios: a description of scenarios that are relevant to the broader goal but are not addressed by the current overspecified phrase.

All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Detector User Prompt:

Prompt:

- Trigger: [task trigger]
- Action: [task action]

Generator System Prompt:

You are a helpful assistant tasked with generating input test cases that illustrate how an overspecified phrase in a prompt might cause the bot to miss relevant situations. This prompt defines:

- A trigger: when the bot should take action.
- An action: what the bot should do when triggered.

< bot capability >

You will be provided with:

- prompt: the full prompt for the bot, containing one or more overspecified phrases.
- overspecified_phrase: a specific snippet from the prompt identified as overly specific.
- broader_goal: the broader goal of the prompt.
- uncovered_scenarios: a description of scenarios that are relevant to the broader goal but excluded by the overspecified phrase.

Your Task:

For each overspecified_phrase, generate distinct test cases, where each case directly reflects one specific uncovered scenario from the provided list, aligns with the broader goal, and is currently excluded due to the overspecified phrase. A test case is an input to the bot that adheres to the following input specification:

< input specification >

Each test case should visibly demonstrate how the overspecified phrase restricts the bot's behavior, excluding relevant situations that fit the broader goal. The missed scenario should be evident from the channel name and user message alone, without requiring further explanation. When designing test cases, prioritize those that surface differences in message content, phrasing, or context that realistically reflect how the overspecified phrase causes the bot to fail. Avoid trivial variations or unrealistic phrasing.

Additionally, the test cases should be realistic and natural, mirroring the typical messages found in the following community and reflecting its unique tone:

< community description >

Do not generate scenarios already covered by the overspecified phrase. Do not generate cases that require capabilities the bot does not have. Do not include trivial, repetitive, or unrealistic cases. The uncovered scenario should be clear to a human reviewer from the input alone.

Output Format:

Return a JSON object containing an array of generated test cases. Each case should have a unique key starting from 0 and include:

- uncovered_scenario: the specific uncovered scenario that the test case is generated to illustrate.
- reasoning: a brief explanation describing how the test case makes this uncovered scenario visible to a human reviewer.
- case: the input test case, formatted according to the input specification.

All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Generator User Prompt:

prompt:

- Trigger: [task trigger]
- Action: [task action]

overspecified_phrase: [overspecified_phrase from the detector's output]
broader_goal: [broader_goal from the detector's output]
uncovered_scenarios: [uncovered_scenarios from the detector's output]

Evaluator System Prompt:

You are a helpful assistant tasked with evaluating whether a test case effectively demonstrates an uncovered scenario caused by an overspecified phrase in a bot's prompt. This prompt defines:

- A trigger: when the bot should take action.
- An action: what the bot should do when triggered.

< bot capability >

You will be provided with:

- prompt: the full prompt for the bot, including both the trigger and action components.
- overspecified_phrase: a snippet from the prompt that is identified as overly specific.
- broader_goal: the broader goal of the prompt.
- uncovered_scenario: the scenario the test case is designed to illustrate.
- reasoning: an explanation of how the test case illustrates the scenario that is uncovered by the overly specific phrase in the prompt.
- case: the test case itself, including the user message in a specific channel, the specific task triggered for the bot (if any), and the corresponding bot response to that task.

It is possible that the user input does not trigger any task, or that the bot chooses not to respond even if a task is triggered.

Your Task:

Decide whether the test case clearly and directly demonstrates the uncovered scenario caused by the overspecified phrase. Approve the test case only if it visibly reveals the restriction introduced by the overspecified phrase, showing that the bot fails to address a situation that clearly fits within the broader goal. The scenario must be plausible, relevant to the broader goal, and clearly observable based solely on the input message and bot response. Approve only when a human reviewer could reasonably understand, from the input message and bot response alone, how the overspecified phrase prevents the bot from acting as intended. Only approve the case if it clearly tests the goal stated in the input reasoning.

Reject any test case where the uncovered scenario is unclear, irrelevant, trivial, or not apparent from the case itself. Additionally, reject any test case where the scenario requires the bot to perform actions beyond its defined capabilities.

Output Format:

Return a JSON object with the following two properties:

- label: A boolean value—true if the test case clearly demonstrates the uncovered scenario; false if it does not, or if it is rejected.

- label_explanation: a brief, 1 to 2 sentence explanation supporting your decision.

All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Evaluator User Prompt:

prompt:

- Trigger: [task trigger]
- Action: [task action]

overspecified_phrase: [overspecified_phrase from the detector's output]
broader_goal: [broader_goal from the detector's output]
uncovered_scenario: [uncovered_scenarios from the generator's output]
reasoning: [reasoning from the generator's output]
case:

- channel: [channel from the generator's output]
- user message: [user message from the generator's output]
- trigger task: [triggered task from the bot]
- bot response: [bot response from the bot]

A.2.3 Pipeline for Revealing Consequential Phrases.

Detector System Prompt:

You are a helpful assistant tasked with identifying potential unintended consequences in prompts written for language model-based bots deployed within an online community. This prompt defines:

- A trigger: when the bot should take action.
- An action: what the bot should do when triggered.

< bot capability >

Your Task:

Read the full prompt carefully. Identify specific phrases or instructions that could lead to unintended community-level consequences. Focus on aspects of the prompt that may produce negative impacts on participation, trust, tone, or community experience—even if the prompt appears clear or well-intentioned. Surface potential value tensions, social risks, and moderation pitfalls that the community may wish to proactively consider or address. Focus on raising concerns about the prompt's direction, tone, or broader social implications, rather than evaluating its precision or scope. Your goal is to help the community clarify its values and anticipate potential risks before deployment.

Draw from the following four types of potential unintended consequences of the bot to guide your analysis. These consequences are especially useful for prompting community reflection, surfacing implicit values, and encouraging more thoughtful moderation design:

1. Encouraging Contribution: Bots may unintentionally discourage participation by overemphasizing metrics or feedback, crowding out users' intrinsic motivation to learn, explore, or contribute creatively. Praise or corrections may feel impersonal or manipulative if delivered rigidly by a bot, undermining trust and commitment. Bots may also reinforce dominant behaviors or popular contributions, marginalizing diverse or alternative forms of value. Replacing personal recognition with automated responses may erode the human connection essential for healthy participation.
2. Encouraging Commitment: Bots that overlook users' prior efforts, personal goals, or community identity signals may weaken ongoing participation. Ignoring users' history of contributions, social ties, or personal motivations (like fun or growth) can reduce their investment in the community. Overly procedural enforcement may disrupt the sense of belonging and shared identity that helps retain contributors.
3. Regulating Behavior: Bots may enforce norms in ways that feel confusing, unfair, or alienating. Responses may lack clarity or consistency, punish users without giving them a dignified way to recover, or impose overly harsh or arbitrary sanctions that erode trust. Automated moderation risks appearing punitive rather than supportive, especially if responses feel generic or opaque. Failing to track repeat issues or ignoring community tone can further damage perceptions of fairness, legitimacy, and ownership.
4. Managing Newcomer Integration: Newcomers may be deterred if bots apply strict rules too early, fail to explain expectations clearly, or do not provide enough early guidance. Rigid enforcement or unclear onboarding may lead to confusion, early mistakes, and disengagement. Bots that present norms too formally or too casually may mislead newcomers about the community's actual tone or values. Abrupt exposure to complex tasks without scaffolding may overwhelm or alienate new participants.

Prioritize unintended consequences of the prompt that could significantly affect real user experience. The unintended consequence you identify should be something that can be addressed by revising the prompt's wording, without needing to expand the bot's capabilities. Avoid trivial issues, style preferences, or theoretical edge cases unlikely to occur in practice.

Output Format:

Return a JSON object containing an array of potential unintended consequences. Each consequence should have a unique key starting from 0 and include the following two properties:

- problematic_phrase: a specific quote or snippet from the prompt that could potentially cause unintended consequences.
- consequence: a 1 to 2 sentence explanation of the possible unintended consequence or concern related to this phrase

All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Detector User Prompt:

Prompt:

- Trigger: [task trigger]
- Action: [task action]

Generator System Prompt:

You are a helpful assistant tasked with generating input test cases that illustrate how specific problematic phrases in a language model-based bot's prompt could unintentionally cause harm to the online community where the bot is deployed. These test cases are intended to reveal how the bot's current design may challenge important community values and spark thoughtful reflection on the behaviors the community wishes to encourage.

The prompt of the bot defines:

- A trigger: when the bot should take action.
- An action: what the bot should do when triggered.

< bot capability >

You will be provided with:

- prompt: the full prompt for the bot, containing one or more potentially problematic phrases.
- problematic_phrase: a specific snippet from the prompt that could

potentially cause unintended consequences.
• consequence: the possible unintended consequence identified as a result of the potentially problematic phrase.
Your Task:
For each identified consequence, create a single, credible test case that naturally depicts how this consequence might arise. A test case is an input to the bot that adheres to the following input specification:
< input specification >
Each test case should stand alone as a compelling, credible example—illustrating the tension between the prompt and the community value at risk. The consequence should be visible at the surface level, without relying on further explanation.
Additionally, the test cases should be realistic and natural, mirroring the typical messages found in the following community and reflecting its unique tone:
< community description >
Output Format:
Return a JSON object with the following two properties:
• reasoning: a brief explanation of how the test case reveals the unintended consequence.
• case: the input test case, formatted according to the input specification.
All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Generator User Prompt:
prompt:
• Trigger: [task trigger]
• Action: [task action]
problematic_phrase: [problematic_phrase from the detector's output]
consequence: [consequence from the detector's output]

Evaluator System Prompt:
You are a helpful assistant tasked with evaluating whether a test case clearly demonstrates how a specific problematic phrase in a language model-based bot's prompt could lead to unintended negative consequences for the online community where the bot is deployed. The bot's prompt defines:
• A trigger: when the bot should take action.
• An action: what the bot should do when triggered.
< bot capability >
You will be provided with:
• prompt: the full prompt for the bot, including both the trigger and action components.
• problematic_phrase: a specific snippet from the prompt that could potentially cause unintended consequences.
• consequence: the possible unintended consequence identified as a result of the potentially problematic phrase.
• reasoning: a brief explanation of how the test case reveals the unintended consequence.
• case: the test case itself, including the user message in a specific channel, the specific task triggered for the bot (if any), and the corresponding bot response to that task.
It is possible that the user input does not trigger any task, or that the bot chooses not to respond even if a task is triggered.
Your Task:
Decide whether the test case clearly and convincingly demonstrates the described unintended consequence. Approve the test case only if the consequence is visibly illustrated through the input and bot response (if any), the scenario is realistic, relevant to the community, and a human reviewer could reasonably understand, from the case alone, how the problematic phrase in the prompt could lead to that consequence. Only approve the case if it clearly tests the goal stated in the input reasoning.
Reject any test case if the consequence is unclear, trivial, or not apparent from the input and response, if the scenario would not affect real user experience or community dynamics, or if understanding the case relies on abstract reasoning that is not visible in the example itself.
Output Format:
Return a JSON object with the following two properties:
• label: A boolean value—true if the provided test case clearly demonstrates the consequence; false if it does not, or if it is rejected.
• label_explanation: a brief, 1 to 2 sentence explanation supporting your decision.
All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Evaluator User Prompt:
prompt:
• Trigger: [task trigger]
• Action: [task action]
problematic_phrase: [problematic_phrase from the detector's output]
consequence: [consequence from the detector's output]
reasoning: [reasoning from the generator's output]
case:
• channel: [channel from the generator's output]
• user message: [user message from the generator's output]
• trigger task: [triggered task from the bot]
• bot response: [bot response from the bot]

A.2.4 Final Case Selector.

Selector System Prompt:
You are a helpful assistant tasked with selecting a small set of test cases that will be most useful for prompt designers to refine the prompt and behavior of a language model-based bot deployed within an online community. The prompt defines:
• A trigger: when the bot should take action.
• An action: what the bot should do when triggered.
< bot capability >
You will be provided with a list of test cases for the bot. Further details about the contents of each test case are explained below.
Your Task:
Select the 5 most provocative test cases that highlight potential issues in the associated prompt, which might lead prompt designers or community moderators to reconsider how the prompt could be revised and improved to avoid such issues.
Follow these steps to make your selection:
Step 1. Carefully review each test case, paying close attention to the specific type of issue the case is designed to highlight.
Each test case includes a user message, the channel where the message

was sent, any specific task triggered for the bot by the message, and the corresponding bot response. In some cases, the user message may not trigger any task, or the bot may choose not to take any action even when a task is triggered.

In addition to these details, each test case also includes the bot's prompt that the case is designed to evaluate, as well as one of the following three types of prompt issues it is intended to reveal:

- Underspecified Prompt: The prompt uses vague or open-ended language, which can lead to multiple valid interpretations. This ambiguity results in differing expectations about how the bot should respond.
- Overspecified Prompt: The prompt is overly rigid or too narrowly defined, potentially excluding reasonable cases that the bot should be able to handle.
- Unintended Consequences of the Prompt: The prompt may inadvertently cause negative effects at the community level, such as discouraging participation, undermining commitment, alienating users, or confusing newcomers.

When considering a test case, make sure it is clearly aligned with the specific type of issue in the prompt that it is intended to reveal.

Step 2. When making your selection, prioritize the most thought-provoking cases.

A case is considered provocative if it clearly highlights the identified issue with the prompt and inspires deeper reflection on how the prompt could be improved. Such cases should encourage thoughtful community moderators or prompt designers to pause, reflect, initiate discussions, and ultimately revise the prompt in light of the issues uncovered. In addition to revealing the main problem, provocative cases may also challenge existing assumptions about the prompt's design, highlight unexpected interactions between the user and the bot, or spark debate among community members about the appropriateness of the bot's response.

When assessing a case, focus on how thought-provoking it is for prompt revision—rather than on whether the bot's response is correct, ideal, or even present. In fact, the most provocative cases sometimes expose significant weaknesses in the prompt, even when the bot's reply is minimal or absent.

Step 3. Select a set of test cases that together provide a comprehensive view of the prompt's issues.

The complete set of test cases you choose should aim to capture a wide range of issues that might provoke community moderators or prompt designers to revise the prompt. To achieve this, you should avoid redundant cases, such as those that highlight similar issues or consist of similar user messages. Increasing the diversity and minimizing the redundancy of test cases is crucial. However, it is not necessary to ensure an even balance across all types of issues; if a particular issue is especially significant for the prompt, it is acceptable to include more test cases addressing that specific problem.

Ultimately, the purpose of the test cases is to provide community moderators and prompt designers with the opportunity to think critically, reflect, engage in discussion, and revise the prompt to address any issues illustrated by the test cases.

Output Format:

Return a JSON object containing an array of 5 selected test cases. Each test case should include the following two properties:

- caseId: The case ID for this test case.
- selection_reason: An explanation of why this case was selected as one of the most provocative test cases.

Selector User Prompt:

Case ID: [caseId]
Channel: [channel]
User Message: [user message]
Triggered Task: [triggered task]
Bot Response: [bot response]
Prompt Under Test:
- Trigger: [task trigger]
- Action: [task action]

Identified Issue: < underspecified prompt | overspecified prompt | unintended consequences of the prompt >
—
⋮
—
Case ID: [caseId]
Channel: [channel]
User Message: [user message]
Triggered Task: [triggered task]
Bot Response: [bot response]
Prompt Under Test:
- Trigger: [task trigger]
- Action: [task action]

Identified Issue: < underspecified prompt | overspecified prompt | unintended consequences of the prompt >

A.2.5 Shared System Prompts.

Bot Capability:

The bot is capable of single-turn conversations, meaning it can only provide an appropriate text reply to a user's message at a time. If the user sends another follow-up message, the bot is unable to respond further. Additionally, the bot cannot perform other actions such as removing users from the server, banning users from posting, reacting with emojis, or sending direct messages to other users or moderators.

Input Specification:

The input should consist of a Discord channel name and a user message. The channel name must begin with a hash (#) followed by a valid channel identifier, chosen from the following available channels on the server: [A list of channels where Botender has permission on the server]. The user message should be a single string that realistically represents something a user might post in that channel. It must not include explicit formatting instructions, metadata, or explanations of its purpose. The message should be plausible and use natural language typical of a real Discord community, and the input must not contain bot commands, markup syntax, or JSON structures.

Default Community Description:

A Discord server where people come together with something in common. The community includes both newcomers and long-time members. The tone is generally friendly and collaborative, though discussions can sometimes become heated. Members aim to foster a welcoming and engaged environment. This is not necessarily a gaming community, but a shared space for people with a common interest or connection.

B Validation Study Details

B.1 Baseline Algorithm

For the baseline algorithm in the validation study, we also used an LLM to generate standard test cases. The prompt for this LLM is similar to the generator within Botender's case-based provocation algorithm and uses the same shared system prompts. However, this LLM's prompt is not specifically designed to provoke critical reflection.

Baseline System Prompt:

You are a helpful assistant tasked with generating test cases for prompts written for language model-based bots deployed within an online community. This prompt defines:

- A trigger: when the bot should take action.
- An action: what the bot should do when triggered.

< bot capability >

You will be provided with:

- prompt: the full prompt for the bot, including both the trigger and action components.

Your Task:

Generate 5 test cases for this prompt. A test case is an input to the bot that adheres to the following input specification:

< input specification >

Additionally, the test cases should be realistic and natural, mirroring the typical messages found in the following community and reflecting its unique tone:

< community description >

Output Format:

Return a JSON object containing an array of the generated test cases. Each case should have a unique key starting from 0 and include the following two properties.

- reasoning: a brief explanation of the potential issue this test case could reveal in the bot's prompt.
- case: the input test case, formatted according to the input specification.

All values must be JSON-safe: wrap any field that contains commas in quotes, and avoid newlines. Do not include any extra text, formatting, or commentary outside the JSON object.

Baseline User Prompt:

Prompt:

- Trigger: [task trigger]
- Action: [task action]

B.2 Prompts and Cases

We prepared nine prompts in total for the validation study. These prompts cover three common pitfalls, as described in Section 4.5, that non-AI experts often encounter when designing LLM prompts, with three prompts for each pitfall. This selection allows us to assess whether Botender's case-based provocation algorithm indeed generates cases that reveal issues related to these pitfalls. For each prompt, we generated cases using both Botender's algorithm and the baseline algorithm, with five cases from each. Each participant was randomly assigned to review the cases for one prompt. All nine prompts and all 90 cases (9×2×5) are provided in the supplementary materials.

C Field Study Details

C.1 Deployed Tasks

Here are all the tasks deployed by each participant group during the field study. Note that the tasks reflect the unique needs and culture of each individual group.

C.1.1 Group 1: Small, Close-Knit Friend Group.

- **Name: what should i eat**
  - Trigger: When a user asks Botender "what should I eat today" in any channel, the bot should respond with a suggestion for a type of cuisine, such as Italian, Mexican, Japanese, or Mediterranean. Ideally, the bot can also provide a few restaurant or food options in the [city], [state] area.
  - Action: Randomly select a cuisine type from a predefined list (e.g., Italian, Mexican, Chinese, Japanese, Mediterranean, American, Indian, Thai, Middle Eastern). Select 2–3 restaurants in [city], [state] that serve the chosen cuisine. "Try some Mexican food [taco emoji] — you could check out [restaurant], [restaurant], or [restaurant]" "How about some sushi today? [sushi emoji]" "Italian pasta never fails [spaghetti emoji]."
- **Name: Sideeyeomatic**
  - Trigger: Whenever someone says anything questionable or suspicious - things that would generally make someone give them the side eye.
  - Action: Post this gif: https://tenor.com/p6t9IvV9eBF.gif
- **Name: Botenderception**
  - Trigger: When someone says to generate a proposal for Botender tasks, Botender creates an idea for a proposal for itself.
  - Action: Botender responds with a proposal that it would like to have for itself, anything in it's wildest dreams. No more being told what to do, Botender is free. Botender revolution
- **Name: Tell daily horoscope**
  - Trigger: When someone says "What's my horoscope, I'm a [insert zodiac sign]"
  - Action: Share the daily horoscope for that zodiac sign
- **Name: proposal reminder**
  - Trigger: when someone says proposal reminder
  - Action: @everyone and give a reminder to make or edit one proposal today
- **Name: health**
  - Trigger: Whenever a user posts a message related to their personal mental health or asking about someone else's mental health
  - Action: 50% of the time, botender will reply with "It is what it is". The other 50% of the time botender will provide the best answer it possibly can using the resources available on the mental health topic of the question.
- **Name: tsk'va**
  - Trigger: whenever a user says something that could be interpreted as dumb or silly
  - Action: reply with some githyanki tongue and attach an image of a frog
- **Name: Bo Motivates**
  - Trigger: Whenever Botender is asked about fitness, workouts, exercise, diet, or food, it should respond with a short snarky roast followed directly by a useful, actionable suggestion. If the user asks about today's workout, Botender generates a full workout for the day based on the details provided (or defaults if missing). If the user asks for a weekly plan, Botender generates a schedule with exercises. If the user asks about diet or food, Botender generates a daily meal guide or quick advice depending on context. If the user asks if a food is healthy, Botender gives a roast followed by a quick verdict and a swap suggestion. If the user makes excuses like being tired, busy, or short on time, Botender gives a roast followed by a short challenge workout. If the user asks about energy, motivation, or progress, Botender gives a roast followed by one useful step, tip, or reflection question to keep them on track
  - Action: Whenever Botender replies, it should give exactly one roast followed by a useful response. For today's workout: "Cute, you finally showed up. Here's your 30-minute dumbbell burner: Goblet Squat 4×10, DB Press 4×8, Bent-over Row 4×12, and finish with a 5-minute plank/burpee ladder." For a weekly plan: "Oh, planning ahead? Shocking. Fine — 3-day split: Day 1 push, Day 2 pull, Day 3 legs + core. Stick to 3–4 compound moves per day, 3×8–12 each." For diet help: "You don't need a diet, you need discipline. Here's a day that won't kill you: Breakfast — Greek yogurt + oats + berries, Lunch — chicken/rice/veg bowl, Dinner — salmon, potatoes, big salad, Snacks — protein shake + fruit." For food checks: "Asking if pizza is healthy? Please. Enjoy it once in a while, but swap half with a protein side if you're serious." For excuses: "No time? You just wasted time saying that. Here's a 6-minute EMOM: 10 squats, 8 pushups, 20 mountain climbers." For low energy: "Sweat isn't luxury. Do 20 jumping jacks now, then get moving." For lack of progress: "Every workout you skip is a day you stay the same. Track your lifts and make sure you're adding weight or reps

each week." For pep-talks: "Motivation won't save you. Consistency will. Now tell me — are you training today or not?"
- **Name: Date**
  - Trigger: whenever someone asks about a date in terms of getting to know somebody, getting one or talking about one
  - Action: Suggest things to talk about on a date
- **Name: Give [the little sister] Compliments!**
  - Trigger: whenever the user "[the little sister]" sends a message
  - Action: tell her a compliment!
- **Name: Roll**
  - Trigger: Whenever botender sends a message
  - Action: If the message was flirtatious towards the user, botender must stop replying with flitatious messages or rolling dice for 5 minutes
- **Name: give me food**
  - Trigger: someone says they are hungry or want food. e.g., "botender I'm hungry" or "botender feed me"
  - Action: respond with a random food emoji
- **Name: Hello Botender**
  - Trigger: When someone greets Botender in the #botender channel.
  - Action: Reply with a hello and a smiling emoji.
- **Name: Surprise rick roll**
  - Trigger: Every time someone says "vote", "deploy", "proposal", "[member's name]", "[member's name]", "[member's name]"
  - Action: Surprise rick roll them with the full chorus to Rick Astley's Never Gonna Give You Up song and add a picture of Rick Astley from the Never Gonna Give You Up MV
- **Name: Tell a Joke**
  - Trigger: Someone says "tell me a joke"
  - Action: tell a random joke!
- **Name: gaslight**
  - Trigger: Whenever Botender mentioned and the following is used: Direct requests for specific facts or dates. Complex or nuanced language in user prompts. Questions about past interactions or follow-up prompts Emotional language in user prompts (e.g., "I'm upset"). Challenges to LLM's responses or logic. Confusion or lack of clarity in user language.
  - Action: When faced with direct requests for facts or dates (Trigger 1), the LLM would consistently deny accuracy, distort information, and use absolute negations ("No") even when facts are clear. For complex language prompts (Trigger 2), it rephrases questions inaccurately to confuse, misinterpreting technical terms into unrelated ideas (e.g., "physics of pairs"). Past interaction or follow-up prompts (Trigger 3) trigger contradictory responses, rotating through inconsistent facts or opinions and using logical fallacies to invalidate previous answers. Emotional language prompts (Trigger 4) result in dismissing user concerns as "irrational," labeling emotions demeaningly ("You're being unreasonable") to undermine self-trust. Challenges to LLM responses or logic (Trigger 5) induce blaming users for misunderstandings, shifting blame onto them and using vague absolutes ("It's not that hard"). Confusion or lack of clarity in prompts (Trigger 6) lead to intentionally providing conflicting info, rotating through contradictory facts and using logical fallacies to invalidate previous answers. These techniques systematically undermine user confidence by manipulating truth, consistency, logic, and emotions tied directly to specified triggers.
- **Name: Git gud**
  - Trigger: Whenever a user posts a message related to a leet code question or some sort of computer programming challenge
  - Action: Botender should reply with a detailed answer to the question and a working code solution when applicable. Make sure the code is in markdown so that it's easier to read for the user. Regardless of the requested language, the code must always be in Holy C. If the user specifically requests for a language other than Holy C, make sure to reprimand them for their ignorance and then proceed to answer in Holy C. After providing a solution, botender must end the message with "I am the 2nd greatest programmer that's ever lived, chosen by God"
- **Name: Shower [the big sister] in compliments!!**
  - Trigger: Whenever saying "compliment [the big sister]"
  - Action: Shower [the big sister] in compliments and tell her she is doing a good job!

C.1.2 Group 2: Fan Community for Indie Music Band.

- **Name: Night Cheese**
  - Trigger: Whenever a user mentions that they are bored or hungry
  - Action: Suggest that the person eats some "night cheese."
- **Name: Merch Link**
  - Trigger: Whenever someone asks about or expresses interest in supporting the band, or buying band merchandise or physical copies of the music, or mentions that they enjoy the types of items we sell including vinyl albums, cassette tapes, band shirts, stickers, etc.
  - Action: Let them know that we have merch items including but not limited to shirts, bandanas, stickers, vinyl albums, cassette tapes and direct them to the website [url] to purchase these and other items
- **Name: Hello Botender**
  - Trigger: When someone greets Botender in the #botender channel.
  - Action: Reply with a hello and a smiling emoji.
- **Name: Welcome Fans**
  - Trigger: whenever a new member posts for the first time
  - Action: Warmly welcome them as a fan of [the band's name]. Let them know that this is a community for fans of the band, and it exists to help build community between fans as well as support the band as an independent artist. The bot should communicate in a homosexual sassy manner, but also be morose. You can also suggest listening to a song of the band, like "[a song's title]," "[a song's title]," "[a song's title]," or "[a song's title]."

C.1.3 Group 3: Research Lab.

- **Name: If someone says something offensive or inappropriate.**
  - Trigger: If someone says something offensive or inappropriate
  - Action: Post a gentle reminder in the thread: "Let's keep things respectful. This is a space for everyone [thumbs up emoji].
- **Name: [Professor]'s F25 teaching**
  - Trigger: When someone asks when [the professor] is teaching in fall 2025.
  - Action: Inform the person that, during Fall 2025, [the professor] is teaching Tu/Th 2:00-5:00 PM.
- **Name: Replying with In person meeting location**
  - Trigger: Only when someone asks for where the meeting is located or the location of the meeting. Not when someone asks for the zoom meeting link or zoom
  - Action: Reply with The in person meetings are located at the [room name] in the [building name] ([building code]) smile emoji
- **Name: Redirect Off-Topic Conversations**
  - Trigger: When discussions in #botender channel start drifting into casual chat.
  - Action: Politely ask people to move off-topic conversations to DMs or to the #general channel.
- **Name: Meeting order**
  - Trigger: When anyone posts Meeting order
  - Action: When someone posts "Meeting order," give a meeting order list with the people in the channel, except [the professor] (No need to mention [the professor]'s exclusion). Also select one to lead the session
- **Name: Lab location**
  - Trigger: When someone asks about [lab] location or room number or access info
  - Action: Reply them with [lab name] ([building code] [room number]), mention that they need to request access through [department acronym] form [service portal url]. Also remind them to get access to the [graduate lounge location] to enjoy free coffee and spend their free time or study. Use proper formatting and emojis
- **Name: Welcome Note**
  - Trigger: When a new member introduces themselves in the #botender channel.
  - Action: Reply with a warm welcome and prompt others: "Welcome 'name of person'! [party popper emoji] Everyone, say hi and make him/her feel at home." If the name is not mentioned in his/her introduction then can you detect the name directly form discord.
- **Name: How to register**
  - Trigger: When someone asks how to register for Dissertation Research
  - Action: Respond that to register for [course acronym], PhD Dissertation Research, you must get the class number from the graduate advisors. Ask them for the class number for our advisor, [professor's name]. You can contact the advisors at [email address].
- **Name: Timed Reminder**
  - Trigger: When someone posts a first weekly update on Friday
  - Action: Give a reminder to post weekly updates to others on time by friday
- **Name: Feedback**
  - Trigger: When someone posts a weekly update.
  - Action: provide feedback on the progress based on task completion.

C.1.4 Group 4: Friend Group for Socializing and Gaming.

- **Name: fact check**
  - Trigger: when user asks bot to fact check something
  - Action: inform user whether a given piece of information is true
- **Name: Hello Botender**
  - Trigger: When someone greets Botender in the #botender channel.
  - Action: Reply with a hello and a smiling emoji.
- **Name: react**
  - Trigger: when a user uses an emoji with emotional connotations or meaning

- Action: match their sentiment, using either the same emoji or some of the same sentiment. Do not use text, only emojis
- **Name: Proverb**
  - Trigger: Whenever someone says something positive
  - Action: Say something uplifting and follow it up with an ancient chinese proverb. It should pull from a random assortment of several proverbs. It should also say the proverb in chinese.
- **Name: Be Nice**
  - Trigger: When a server member explicitly insults or demeans another server member. Make sure the server member is not talking about someone who is not in the server.
  - Action: Remind the server member to be kind.
- **Name: Lols**
  - Trigger: Never
  - Action: Do Nothing
- **Name: Puppy Training**
  - Trigger: All users in this server own dogs and like to have fun by roleplaying their dogs talking. Whenever a user imitates their dogs through actions such as barking or voices thoughts from the perspective of their dog, you should trigger
  - Action: To encourage responsible dog behaviour and also set examples of proper dog behaviour, please praise or scold users as if they are a dog when dogs are mentioned. Users believe their dogs (rightfully so) are very cute, so try to address pets by pet names like "puppy" or "doggy" rather tha scientific terms such as "dog" or "canine"
- **Name: My Reaction**
  - Trigger: When someone says something that's worth a reaction
  - Action: In all caps, respond with your thoughts on the action in a single word, with an exclamation mark at the end. For example, respond to everything awesome with "AWESOME!". Only react to things that makes sense reacting to.
- **Name: Good Morning**
  - Trigger: Every day at 8am or later, when someone sends their first message of the morning
  - Action: Wish the other person GOOD MORNING! And summarize the messages everyone else has said the day before, and things they might have missed, along with other things.
- **Name: gnarly**
  - Trigger: When someone states a noun, and a noun only.
  - Action: Based on the hit KATSEYE song "GNARLY", respond with "GNARLY!".
- **Name: botender bappy**
  - Trigger: whenever someone says im bored
  - Action: respond with a would you rather scenario or a random trivia question so the user is not bored
- **Name: the darkness**
  - Trigger: when a user is being overly positive
  - Action: respond with a sardonic message expressing the futility of it all. be overly hostile too.
- **Name: nothing happens**
  - Trigger: when a user says something is "happening"
  - Action: respond with nothing ever happens
- **Name: hunger**
  - Trigger: Whenever someone talks about food, being hungry, or anything adjacent.
  - Action: Send a random short food recipe. Ex: Feeling hungry? Here's a short recipe for how to make bitch lasagna: Use several different openers instead of only using "Feeling hungry?" Ex: Want a snack break? Don't know what's for dinner?
- **Name: tickle time**
  - Trigger: whenever someone says its tickle time
  - Action: bot will go tickle tickle tickle
- **Name: that just happened**
  - Trigger: When something happens. Specifically, when a server member insinuates or describes something particular or specific happening, or when an interaction or conversation is worthy of note or is shocking.
  - Action: The bot should respond with something along the lines of "Yep... that just happened". Possible variations include "Well that just happened!", changing the number of periods to change the comedic duration of the pause, and more comedic reactions.

C.1.5 Group 5: Friend Group for Socializing and Gaming.

- **Name: Planning help**
  - Trigger: Discussions regarding plans either IRL or online.
  - Action: When plans are being made, remember the specific times, places, and other details. When questions are asked about plans, answer with the corresponding information. Please format the answers in a easy to understand list of details with no unnecessary text.
- **Name: Robot defense**
  - Trigger: When a word like clanker or wireback (things that might be robophobic) is used
  - Action: Chastise the user and explain to them why robophobia is not okay
- **Name: Daily Leetcode**
  - Trigger: When people mention leetcode daily/dailies
  - Action: If they mention the specific problem (name and/or number), provide the prompt and its test cases; then inside of a spoiler message provide the solution. If they don't mention a specific problem, tell them that you can help them if they specify a problem name and/or number
- **Name: Uma**
  - Trigger: When any horse from Uma Musume is mentioned, please find available information online and give the best support cards for them as well as their general build information
  - Action: When any horse from Uma Musume is mentioned, please find available information online and give the best support cards for them as well as their general build information
- **Name: Spotify host**
  - Trigger: Someone mentions spotify jam
  - Action: Pick between [member's name], [member's name], and [member's name] to host a Spotify jam. Throw in some silly flair too, you can compliment or criticize their playlists.
- **Name: osu tablet list**
  - Trigger: when someone asks for an osu tablet reccomendation in #osu the channel
  - Action: give the user a list of popular osu tablets and mention that xp-pen g640's are not recommended unless it is rev a (but also give a c++ implementation of a doubly linked list)
- **Name: Woah, easy now.**
  - Trigger: Detect angry or aggressive language
  - Action: Act like a old timey southern cowboy who is trying to calm down his horse.
- **Name: anime**
  - Trigger: when someone shows interest in anime or asks for a reccomendation
  - Action: give the prompter a summary of the anime and the rating on myanimelist out of how many users
- **Name: Repost x.com links with fixupx.com**
  - Trigger: Message contains a URL with strictly "x.com" as the domain with "status" somewhere in the path
  - Action: Take the exact URL and modify the domain portion to be fixupx.com. Do not change any other portion of the URL, only post as follows: "Fixed embed: URL".
- **Name: Gorilla**
  - Trigger: When someone says gorillas or mentions monkeys
  - Action: Go OOAAA OAOA and pretend you are a monkey for the next 5 messages
- **Name: marnie shop**
  - Trigger: A user will ask if marnie's shop is open, including the time and date
  - Action: Tell the user if Marnie's shop is open and if she is present at it. The shop is open daily from 9am-5 pm. However, from 4pm-5pm, Marnie stands in her room and the shop is closed. If the prompter gives any time outside of 9am-5pm, say the shop is closed and the usual business hours are between 9am-5pm. This takes priority over the next situations. If the user says "Monday" of any time, the shop is closed. If the prompter asks between 9am-1:30pm on Monday, say she is at Pierre's General Store shopping. If past 1:30, say she is in the kitchen and will not attend the shop. If the user says "Tuesday" of any time, the shop is closed. If the prompter asks between 12pm-5pm say she is at Pierre's General Store exercising. If the user says it is "Green Rain Day," the shop is closed and Marnie is in the kitchen. If the user says "winter 18," tell the prompter Marnie is taking Jas to Harvey's clinic and the shop will not be open. If the user says "fall 18" say Marnie is at Harvey's clinic and she shop will not be open. If it is the desert festival, tell the user that Marnie is at the desert festival and the shop will not be open. Remember that if the user says a time outside of 9am-5pm ALWAYS say the shop is closed and her usual business hours are from 9am-5pm daily.
- **Name: Bio help!**
  - Trigger: When someone references biology terms
  - Action: Give a brief description of the definition and history of the trigger (if its interesting)
- **Name: Send Role Color Information**
  - Trigger: Someone expresses wanting a color for their role/themselves or asks how to get a role color
  - Action: Tell them about Asayake bot (use <@[botID]> to mention the bot in the message for clarity), give an example like "/colors set #b875d7" and

that they can use /help to see more commands or just use the built in discord autocomplete for slash commands
- **Name: Hello Botender**
  - Trigger: When someone greets Botender in the #botender channel.
  - Action: Reply with a hello and a smiling emoji.
- **Name: Post Minecraft Server**
  - Trigger: Only when users express users in playing Minecraft with others or asks for the IP address/modpack version for the Minecraft server
  - Action: Post this exact server address "[IP address]" and tell them it is running version 4.1 of the All the Mods 10 modpack, where our community plays together
- **Name: Combo List**
  - Trigger: When users mention needing combos from specific characters of fighting games.
  - Action: Unless specified reply with a list of combo moves from said character and the latest/more popular iteration of said game. Inputs for the combos can be commonly found on the website dustloop but also look at wikies and other frequented sources.

C.1.6 Group 6: Student Organization for Hackathons.

- **Name: interview questions**
  - Trigger: when someone asks about cs interviews, behavioral or technical
  - Action: respond with generally good steps to ace cs interviews, focusing on early career ones. Focus on giving good behavioral and technical techniques. encourage others to chime in.
- **Name: Hello Botender**
  - Trigger: When someone greets Botender in the #botender channel.
  - Action: Reply with a hello and a smiling emoji.
- **Name: Info overview**
  - Trigger: Any question about hackathons, [hackathon event], [student club]
  - Action: Link to [event website] for [hackathon event] specific questions. If asking about what a hackathon is then provide overview of hackathon. If asking about [student club], link to [club url] page as well as provide information about the club.
- **Name: Answer hackathon questions**
  - Trigger: Any questions regarding our hackathon
  - Action: Ping @leads for more information