



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



# Collaborative Human-Agent Interaction: Balancing Human Interruptions and Feedback

Bachelor's Thesis

Yuri Simantob

[ysimantob@ethz.ch](mailto:ysimantob@ethz.ch)

Interactive Visualization & Intelligence Augmentation Lab  
Institute for Visual Computing  
ETH Zürich

Submitted for the fulfillment of the requirements for the degree of  
**Bachelor of Computer Science**

## Supervision

1<sup>st</sup> Reviewer: Prof. Dr. Mennatallah El-Assady

2<sup>nd</sup> Reviewer: Rita Sevastjanova

December 15, 2025

# Acknowledgements

First and foremost, I would like to thank my main supervisor, Rita Sevastjanova, for her consistent efforts to guide my thesis in the right direction, and for her faith in my proposed research. Likewise, I would like to thank Tobias Stähle, my co-supervisor, for his inputs and his guidance especially regarding the backend part of the implementation.

Furthermore, I want to express my gratitude towards Joseph Loffredo and Suyeol Yun, you are the reason that this thesis exists in the first place. Joseph and Suyeol prompted me to come up with a research proposal that would use AI agents to understand Congressional lobbying better, making me combine political science questions with open problems in human-computer interaction.

Beyond this, I would like to extend my thanks to Tom Power for giving me an invaluable walk-through into the world of lobbying, and allowing me to rethink the design of my Congressional agent team, and to Hye Young You for giving me concrete insights into the concrete research that is done in the Lobbying domain.

At last, I would like to dearly thank Prof. Dr. Mennatallah El-Assady for giving me the opportunity to undertake this bachelor's thesis at the IVIA Lab.

# Abstract

The rise of multi-agent AI systems and their increasing capabilities open up lots of questions about human involvement. Currently, multi-agent systems rely on various types of feedback from humans, that can either be asked for by the agents (agent-initiated feedback) or manually input by human users (human-initiated interruption). This thesis examines how human users interact with multi-agent AI systems if both directions of feedback are present. We test this through a user study, where participants are exposed to two different agent systems, a general-purpose deep research system and a domain-specific system working a problem in US political science. In our sample we observed that AI tool experience is the main indicator for good usability of a bidirectional feedback system, and that agent-initiated feedback interferes with human-initiated interruption in complex manner. In certain situations, the agent-triggered feedback mechanism almost completely replaced human interruption as the element used for communicating the human user's concerns. Most critically, we found a need for unified grounding between the human user, the multi-agent system and the entity that guides the agent-initiated feedback procedure.

# Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Literature Review</b>	<b>4</b>
2.1 Human-Agent Interaction . . . . .	4
2.2 Human Control & Autonomous Systems . . . . .	5
2.3 Improving on top of LLMs . . . . .	5
2.4 Human-in-the-Loop for Agentic Systems . . . . .	6
2.5 LLM-as-a-judge . . . . .	6
2.6 Application Background: Deep Research and US Congressional Lobbying . . . . .	7
<b>3 Background - US Congressional System and Lobbying</b>	<b>9</b>
3.1 The US Congressional System . . . . .	9
3.1.1 Legislative Process . . . . .	9
3.1.2 Committee System . . . . .	10
3.2 Lobbying in the United States Congress . . . . .	11
3.3 Available Data Sources . . . . .	11
3.3.1 Congressional Data . . . . .	11
3.3.2 Lobbying Disclosure Data . . . . .	11
3.4 Domain Selection Rationale . . . . .	12
<b>4 Methodology</b>	<b>14</b>
4.1 Research Approach . . . . .	14
4.2 Experimental Design . . . . .	15
4.3 System as Experimental Apparatus . . . . .	15
4.4 User Study Protocol . . . . .	15
4.4.1 Participants . . . . .	15
4.4.2 Study Procedure . . . . .	16
4.4.3 Data Collection Instruments . . . . .	17
4.5 Analysis Approach . . . . .	17

<b>5 System Architecture</b>	<b>18</b>
5.1 Backend Overview . . . . .	18
5.2 US Congress MCP Server . . . . .	18
5.2.1 MCP Function Design . . . . .	18
5.3 Agent Team Design and Engineering . . . . .	21
5.3.1 Framework Selection . . . . .	21
5.3.2 Congressional Lobbying Team . . . . .	22
5.3.3 Deep Research Team . . . . .	22
5.4 Microsoft Autogen Library Extensions . . . . .	22
5.4.1 Understanding the Autogen Library . . . . .	23
5.4.2 User Interruption . . . . .	24
5.4.3 Branching . . . . .	24
5.4.4 Plug-Ins . . . . .	25
5.5 Backend Architecture . . . . .	25
5.5.1 Team Configuration System . . . . .	26
5.5.2 Context Engineering . . . . .	26
5.5.3 Feedback Components . . . . .	27
<b>6 Visual Interactive Interface</b>	<b>29</b>
6.1 UI Elements . . . . .	30
6.1.1 Landing Page View . . . . .	31
6.1.2 Feedback Component View . . . . .	32
6.1.3 Run View . . . . .	32
6.2 State Management . . . . .	36
6.3 Configuration & Setup Flow . . . . .	37
6.4 Tree Visualization . . . . .	38
<b>7 Evaluation</b>	<b>40</b>
7.1 RQ1: Interruption vs. Agent-Initiated Feedback . . . . .	40
7.2 RQ2: Agent Message Transparency . . . . .	42
7.3 RQ3: Feedback Component Design and Adaptation . . . . .	42
7.4 RQ4: Domain Expertise and Interaction Patterns . . . . .	45
7.5 SUS Results . . . . .	45
<b>8 Conclusion</b>	<b>47</b>
8.1 Limitations . . . . .	47
8.2 Future Work . . . . .	47

8.3 Contributions . . . . .	48
<b>Bibliography</b>	<b>49</b>

# CHAPTER 1

# Introduction

---

One of the capabilities of large language models (LLMs) that are being explored today is their unmatched ability to orchestrate complex, multi-step reasoning processes through multi-agent collaboration, where agents with different roles work together on intricate problems mimicking a human team [1, 2]. For the past year, extensive research has been dedicated to studying the potentials [3] and the pitfalls [4] of the new multi-agent system (MAS) paradigm. At the same time the emergence of complex agentic systems that can autonomously reason for longer periods of time triggered many new psychological sensations in the humans using them. Today, this is best understood in the software engineering domain, where coding tools like GitHub Copilot and more recently Claude Code CLI dramatically shifted the needle in reducing the friction and effort to build codebases of any kind, allowing users to give up control of coding design choices. [5, 6] Commonly known as "vibe coding", such usage practices can lead to a "loss of control" sensation, in which the authors have no longer a detailed understanding of their generated code.

The reason why this can become a problem is that when an AI agent errs, either by deleting code or making false assumptions, these errors can be hard to spot and tedious to track down and undo [5]. In this sense MAS pose a problem to human interpretability and agency itself: as reasoning becomes distributed across multiple autonomous components, users risk losing full awareness of the causal chain of decisions that led to a given conclusion or outcome, blurring the line between authorship and assistance.

Yet there are multiple other domains AI agents and MAS can be used, especially in the practice of research itself. Deep Research pipelines already use multiple agents to parallelize and aggregate their web search findings [7] in order to generate longer and more densely cited answers to prompts. Furthermore, the emergence of the Model Context Protocol (MCP) made data sources and existing APIs much more available to agentic systems, enabling them to amass much more detailed context for in-depth queries [8, 9]. This makes building multi-agent systems that mimic a human team's workflow very interesting to assist researchers in specific domains; however, the autonomous multi-step reasoning nature inherent to research exposes the researchers to the very same risks that come with shallow understanding of the agents' reasoning.

There exist multiple ways for a human user to maintain control when working together with (multi-)agent systems. A user may interrupt the agent and tell it what to do differently (human interruption), or one can build feedback mechanisms into the agent system, where an agent can decide to request input from the human user (agent-initiated human feedback). Previous research efforts involving human control mechanisms in the human-agent interaction made considerable advances in understanding human needs and designing elements that improved

human-agent collaboration via feedback. However, such work limits itself either to completely focusing on perfecting the agent-initiated feedback system [10, 11] or to studying agent-in-the-loop mechanisms for human conversations [12].

In this work, we explore fundamental questions about the design of human participation in multi-agent systems' interfaces. We tested the combination of using agent-initiated feedback together with the possibility of human-initiated interruption with visual cues when working with a MAS. We specifically implement an extension to the Autogen library that allows direct support for human-initiated interruption within the framework, and combine it with evaluation metric customization, which we will call the "feedback components" that guide the agent-initiated feedback triggering logic in the style of [13].

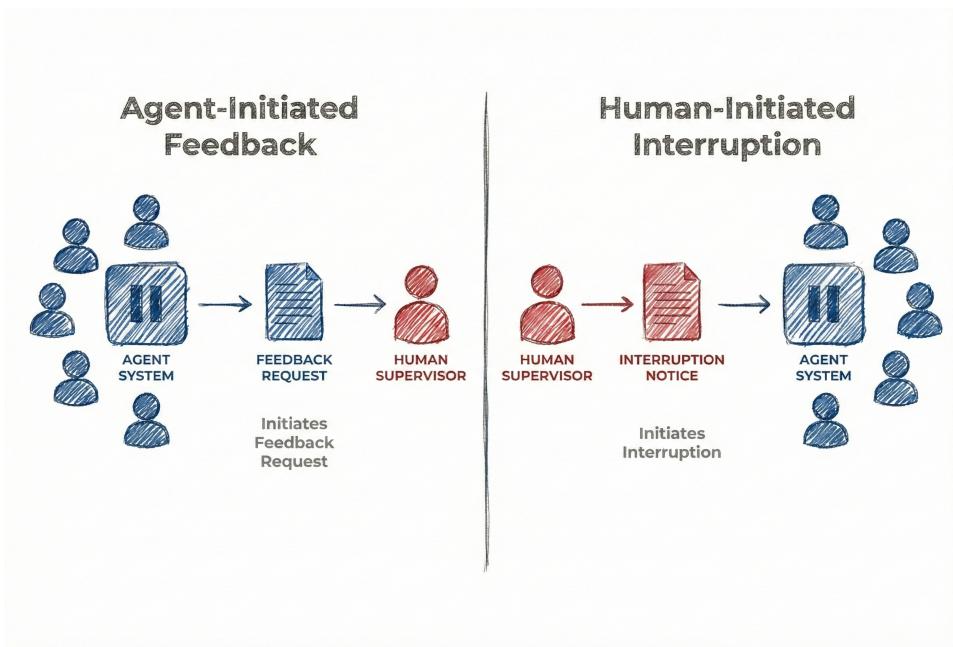


Figure 1.1: Visual description of the difference between human-initiated interruption and agent-initiated feedback

Specifically, we investigate the following research questions:

- **RQ1:** How do users perceive and utilize combined human-initiated interruption and agent-initiated feedback mechanisms in multi-agent systems?
- **RQ2:** Do users want visibility into multi-agent coordination, and what drives the variation in user engagement with transparency features?
- **RQ3:** How do users configure and adjust feedback trigger mechanisms, and what mental models guide their expectations?
- **RQ4:** Does the agent team interaction behavior differ between general-purpose deep research agent teams and domain-specific agent teams?

Further we proceed to test the presence of both these features in a visual interface where the human user is given the liberty to look at the inter-agent communication and define the

evaluation metrics that will later trigger agent-initiated feedback. In order to extrapolate these user preferences to general multi-agent systems, we test our interface on two different groups of people working with different agent teams: a deep research team that can be given an open question, tested on Bachelor's students, and a money-in-politics investigation team where domain experts solve a pre-defined task.

Our contribution to open Human-Agent interaction research is as follows:

1. We qualitatively assess the user experience of combining human-initiated interruption together with agent-initiated feedback in multi-agent systems
2. We evaluate the degree of active involvement human users desire when working together with multi-agent systems
3. We present an extension to the Microsoft Autogen library that allows agent team developers to directly implement human interruption into their MAS

We found that human users generally like the idea of both feedback mechanisms, but there are some navigational interleavings that arise when we combine both into one interface, where a large presence of one affects the usage of the other. Similarly, users didn't express much enthusiasm for the transparency features. While liking the display of which agents were speaking, they frequently weren't interested in all agent communication that was presented and couldn't follow with everything the agents said in the first place. Additionally, we observed that AI tool use experience is a good indicator of perceived usability of our system, while domain expertise is not. However, domain experts had a much better understanding of each agent's functionalities in the Congress team than the students had of the agents in the Deep Research team, indicating that a specialized agent team is more intuitive to domain experts than a general purpose agent team.

Domain-expertise did not turn out to be a good indicator of usability. However, AI tool expertise clearly indicated how usable a participant found our interface. We concluded that awareness of the latest AI tools in writing, researching and coding increases the usability of multiple feedback mechanisms together, while an inexperience in advanced AI tools may make user involvement features hard to understand and work with in the first place.

However, our main unexpected finding concerns the way users interact with our agent-initiated feedback system. We uncovered a three-layer grounding mismatch between the user, the multi-agent system and the feedback triggering system, where not only the human user makes false assumptions about the multi-agent system's capabilities and the feedback triggering system's capabilities but also that the feedback triggering system has false assumptions about the multi-agent system itself. As a result, we conclude our work highlighting the need of a grounding framework or a list of shared assumptions between the agent system and the feedback system in order to make the user interaction with agent-initiated feedback more seamless, and to give more way for human interruption.

# Literature Review

---

Current research has assessed the dynamics of human-agent interaction when the user can interrupt the conversation (human-initiated interruption) [14], or when the user can specify when to ask for feedback automatically (agent-initiated feedback). We aim to uncover deeper insights and user preferences by combining both mechanisms in one interface and extend these insights by comparing students and political science domain experts' behavior when working with the agent teams. The inspiration for our two distinct deep research teams, with which our test subjects will interact, comes from the groundwork done by OpenAI's Deep Research tool and Anthropic's engineering guides for deep research MAS. [15, 16] For our domain-specific example, we chose the tasks based on previous statistical work done on interest groups in the US Congress [17] and existing approaches to connect the data from the US Congress API to an LLM. [18]

## 2.1 Human-Agent Interaction

The phenomenon of human collaboration has been studied for a long time, where researchers use the framework of the "common ground" as the baseline that enables well-working collaboration, coining the establishment of the common ground as "grounding". [19] When expanding this problem to human-computer interaction, Clark et al. noted an interesting issue - namely that it is up to the users' initiative to establish common ground when communicating with a computer.

*It often falls to users to put in the extra effort needed to try to keep things on track. This is what I call the grounding problem in human-computer interaction. **Brennan (1998)** [1]*

In light of recent developments in the LLM agent domain, these issues have been reformulated specifically for human-agent interaction. [2] outline 12 challenges in human-agent interaction, divided into general challenges, human-to-agent communication challenges and agent-to-human communication challenges.

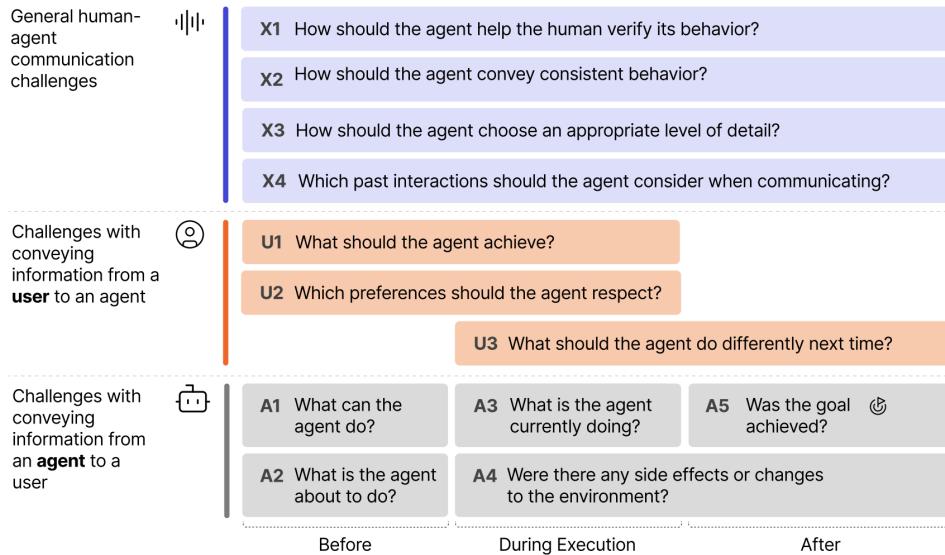


Figure 2.1: The 12 problems of Human-Agent Interaction as put forward by Microsoft Research

## 2.2 Human Control & Autonomous Systems

There is extensive literature about balancing human control in autonomous systems, especially in military applications where the task executions have actionable and possibly consequential outcomes, where one study found decreasing accuracy in identifying enemy targets when the human operator could only veto an execution, and not approve it in the first place. [20] The concept of "Meaningful Human Control" (MHC) arose in response to these philosophical issues, mainly from the autonomous weapons question. [21, 22]

In more recent years, the public discussion of autonomy in AI agent systems has been shifting to more general use cases, especially in code generation [23, 24]. A big survey conducted by StackOverflow, the biggest online community for discussing coding issues, uncovered that 86% of respondents are concerned about the factuality of the coding agents' responses, and 20% affirm that they became less confident in their own problem-solving capabilities.

## 2.3 Improving on top of LLMs

Innovating with language models became the core focus of AI research and applications, and substantial research has been dedicated to improve factual accuracy of the outputs these models emit. A significant first step was the emergence of Retrieval-Augmented Generation [25], which provided a method to semantically pick external data to guide the model's answer generation. This method has been refined and sophisticated, for example by building a knowledge graph to enhance retrieval from the data source. [26] Anthropic's Model Context Protocol (MCP) [8], extended this further by enabling to craft structured access for AI agents to any external data source available on the web.

As capabilities of LLMs grew over time, researchers began experimenting with multiple instances of AI agents communicating together towards a common goal. A first inspiration in this direction was given by Du et al. showing that using multiple LLM instances that discuss with each other can improve factual accuracy, calling their system the "Society of Mind" agent. [27] Based on this, other multi-agent architectures arose, from concatenating multiple intermediate answers [28] and introducing structured communication protocols between the participants' agents. [29] Most of these base architectures and protocols have been integrated in comprehensive coding frameworks, such as Microsoft's AutoGen framework, LangGraph, Google's Agent Development Kit, and other open-source frameworks such as AG2. [30, 31, 32]. Recent work suggests that the key to making multi-agent systems work effectively lies in orchestration, a new paradigm that is coined "Agentic Organization". [3]

Research delving into visual human interaction with multi-agent systems has been largely exemplified by Microsoft Research's AGDebugger [14] which allows a developer designing a MAS to "debug" the agents' behavior by initiating (human) interruptions. The most crucial desire the study participants expressed was the ability to reset back to a previous point in the workflow (which they subsequently implemented into the debugger), focusing their research on getting the agent team to run smoothly in its design phase.

## 2.4 Human-in-the-Loop for Agentic Systems

Google's People + AI guidebook features an entire section listing design principles for designing human-in-the-loop feedback mechanisms in agentic systems. [33] They stress in detail the opportunities and challenges arising when designing these agentic feedback mechanisms, highlighting a feedback loop's potential to "guide its evolution and improve the model's performance", as well as the pitfall of absent grounding between the feedback mechanisms capabilities and the user's intention.

*For feedback to be effective, users and AI systems must interpret the signal in the same way. When there's a mismatch between what a user intends to communicate and what the system understands, both the user experience and system performance decline. [33]*

For this, concrete "feedback grounding mechanisms" have been proposed that work with human open-ended textual input [10] and convert it into grounded prompts that align with a system's realities, with a more applied and intuitive example being MarsSQL, that grounds user prompts according to the available SQL database schemata. [11]

## 2.5 LLM-as-a-judge

While grounding mechanisms ensure that feedback is interpretable by the system, they do not address when or why an agent team should request feedback in the first place. Since agent teams output large quantities of text in a short period of time, raising the problem that human users can hardly read over the text in detail to assess when feedback should be triggered. The concept of using an "LLM-as-a-judge" aims to solve this, where strong LLMs can achieve similar

agreement rates to those among human experts, [34] although complementary research warns for possible superficiality of current agreement benchmarking. [35]

Subsequent sophistication iterations coupled the text evaluation with retrieval systems, [36] having an evaluator agent in multi-agent systems [37, 38] up to defining "Agent-as-a-Judge", an entire agentic system that can dynamically look into the evaluated agents' trajectories. [39] Yet in some environments like browsers, evaluating web navigation trajectories of agents can still pose challenging grounding mismatches between the evaluator and the agentic system. [40]

## 2.6 Application Background: Deep Research and US Congressional Lobbying

Deep Research is a term that first came to the public ear through the release of OpenAI's Deep Research mechanism. [15] Subsequently, most companies that offered LLM chat interfaces moved to build their own versions of Deep Research, while other developers did their smaller-scale experiments with deep research systems. [41, 42] The release of Anthropic's developer blog on how they built their deep research system [16] showed further the manifestation of the multi-agent paradigm inside deep research systems.

We chose to develop a deep research agent team because deep research systems represent one of the most widely adopted and general-purpose forms of multi-agent architectures, currently available on most major conversational AI platforms (e.g., Gemini, ChatGPT, Claude). This design enables users to submit arbitrary research queries and to observe the multi-agent system as it collaboratively deliberates over the research process, thereby providing an interaction format that closely resembles established user experiences from conventional large language model platforms.

As to the rationale of selecting the domain of Congressional lobbying for the second agent team, we dedicated an entire chapter below.

## Multi-agent System Process Diagram

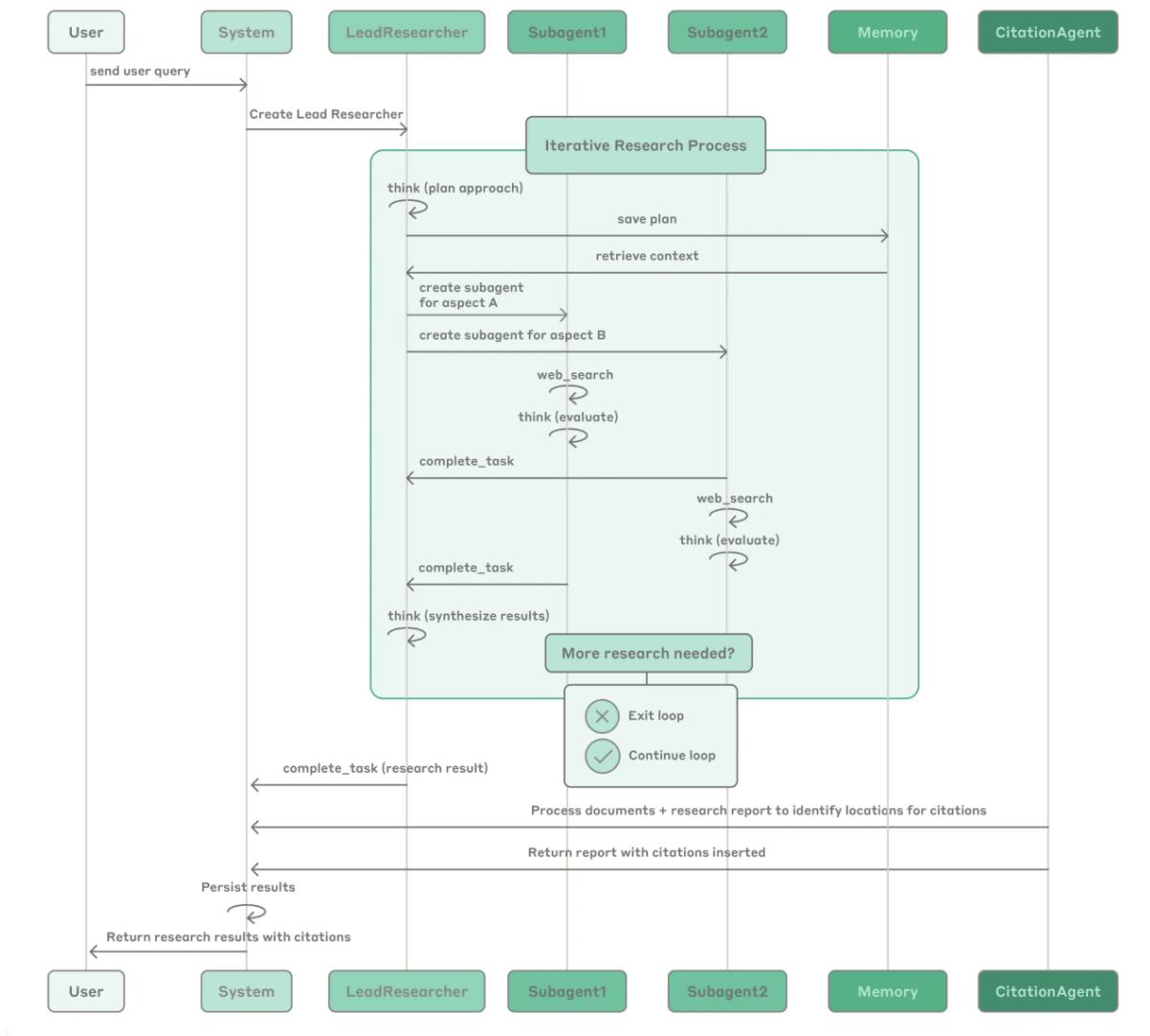


Figure 2.2: A Mermaid diagram explaining Anthropic's Deep Research system with multiple agents

# Background - US Congressional System and Lobbying

---

Having reviewed the literature on human-agent interaction and multi-agent systems, we now turn to the specific domain context for our study. This chapter provides essential background on the US Congressional system and lobbying practices, which form the foundation for our domain-specific agent team.

## 3.1 The US Congressional System

The United States Congress serves as the legislative branch of the US government, responsible for creating and passing laws. This means that it constitutes the lawmaking authority, and can introduce new laws that the executive branch, the White House in this case, will enforce.

There are two chambers in the US Congress: The House of Representatives (or the "House" in short) and the Senate. The House has 435 representatives; each representative represents a voting district of roughly 300000 American citizens, although this can vary. In contrast, the Senate has 100 members, where 2 senators come from each U.S state, independent of its population size.

### 3.1.1 Legislative Process

A bill is a proposal for a new law that can be introduced by a politician in either chamber. This politician is referred to as the bill's "sponsor". The respective chamber where the bill was introduced can then discuss a bill, "amend" it (that is, change its contents), refer it to a specialist committee inside the chamber, or vote on its passage or refusal. A bill has to pass both chambers in order to become law, therefore, if one chamber votes in favor of it, the entire process is repeated in the other chamber.

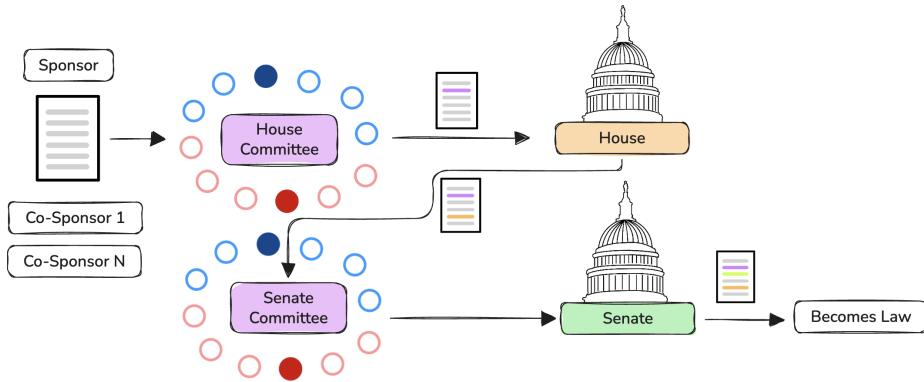


Figure 3.1: A Figure describing the Trajectory of a Bill through Congress

A bill begins its passage by being introduced by a sponsor. It can have various co-sponsors — politicians that express formal, although non-binding support for the bill. Then it will be referred to a specialist committee, where it will be discussed, amended (if needed) and sent back to the chamber floor. There, members of Congress may discuss it at length, propose amendments, discuss these amendments, and finally vote on them. First, the amendments are voted on one-by-one, and then the final bill (with all the passed amendments) is voted on. If this is successful, the (possibly amended) bill gets sent to the other chamber, and the entire process is repeated again. If it gets approved in the other chamber, it is signed by the President of the United States into law.

### 3.1.2 Committee System

Each chamber is divided into specialist committees that focus on bills of specific areas. In the House for example, a new bill that seeks to renew funding for coal mines will be referred first to the House Committee on Energy and Commerce, before being discussed by the entire House.

Inside the committee are members from both parties, the minority party and the majority party (depending on election outcomes either the Republican party or the Democrats). The majority party holds more seats than the minority party, and the committee's agenda is led by the "chair", traditionally the most senior member of the majority party in that committee. The chair has the authority to set the agenda of the committee, including scheduling the meetings and deciding which bills need to be discussed.

This is important because a committee typically cannot discuss all of the bills at length due to the sheer volume of bills being introduced. For example, in the 118th Congress alone, 19,315 bills were introduced, of which only 4% got a vote and a mere 1% were passed and enacted as law. [43] Consequently, many bills "die" in committees and are never discussed or voted upon. The minority party also has its leader, the "ranking member" of the committee and, although not having the same powers as the chair of the committee, it remains common practice for the chair to consider carefully rejecting the ranking member's wishes and concerns, especially if the committee chair has a vulnerable re-election campaign. [44]

Each committee has several subcommittees, and a committee can refer a bill further down to a specific subcommittee to delegate the discussion of a bill. An example would be the Senate Subcommittee on Aviation, which is a subcommittee to the Senate Committee on Commerce, Science, and Transportation.

In committees, bills can be discussed, amended, and sent back to the "floor", meaning to the entire House, where it will be scheduled for a general discussion.

## 3.2 Lobbying in the United States Congress

Lobbying practices in and around the United States Congress constitute a multi-faceted practice frequently simplified in news articles, and are viewed by many American citizens negatively. [45] In practice, lobbying manifests itself in various forms. Members of Congress often reach out to lobbyists to assist in drafting bills comprehensively and conversely, lobbyists reach out to members of Congress themselves, in order to make their concerns clear to them. [46]. Essentially, lobbyists serve as intricate intermediaries between interest groups and politicians, intermediaries that represent the interest groups while simultaneously providing politicians with industry best practices. [44] This makes the relationships between interest groups and politicians via lobbyists more nuanced than anyone would assume.

## 3.3 Available Data Sources

For this reason, understanding the Congressional system is only part of the picture. To study lobbying dynamics systematically, we rely on two comprehensive public data sources.

### 3.3.1 Congressional Data

Records of congressional actions are the principal data source that provides an insight into the proceedings in Congress itself. The Library of Congress stores most physical and digital documents that were emitted during the legislative sessions on the floor and inside the committees. Conveniently, they offer a public API at no cost that allows developers to retrieve information about bills, amendments, legislative actions and committee documents. We created a custom Model Context Protocol (MCP) server for our agents that allows them to access the API in custom sequences.

### 3.3.2 Lobbying Disclosure Data

Federal lobbying disclosure requirements provide the second crucial data source. We have a considerable amount of publicly available data points that indicate the presence of lobbying for a specific bill by a specific company. This analysis employs the LobbyView database that contains all the extracted data from raw LD-1 and LD-2 filings (the Lobbying Disclosure forms) companies and other interest groups are obligated to fill out whenever they contract a lobbying firm. These filings must specify the lobbyists they are contracting and the bills they are interested in. Therefore, such relationships can be readily extracted for some example lobbying cases, as

we have done for Big Oil Companies in 2018 (see Figure 3.1)

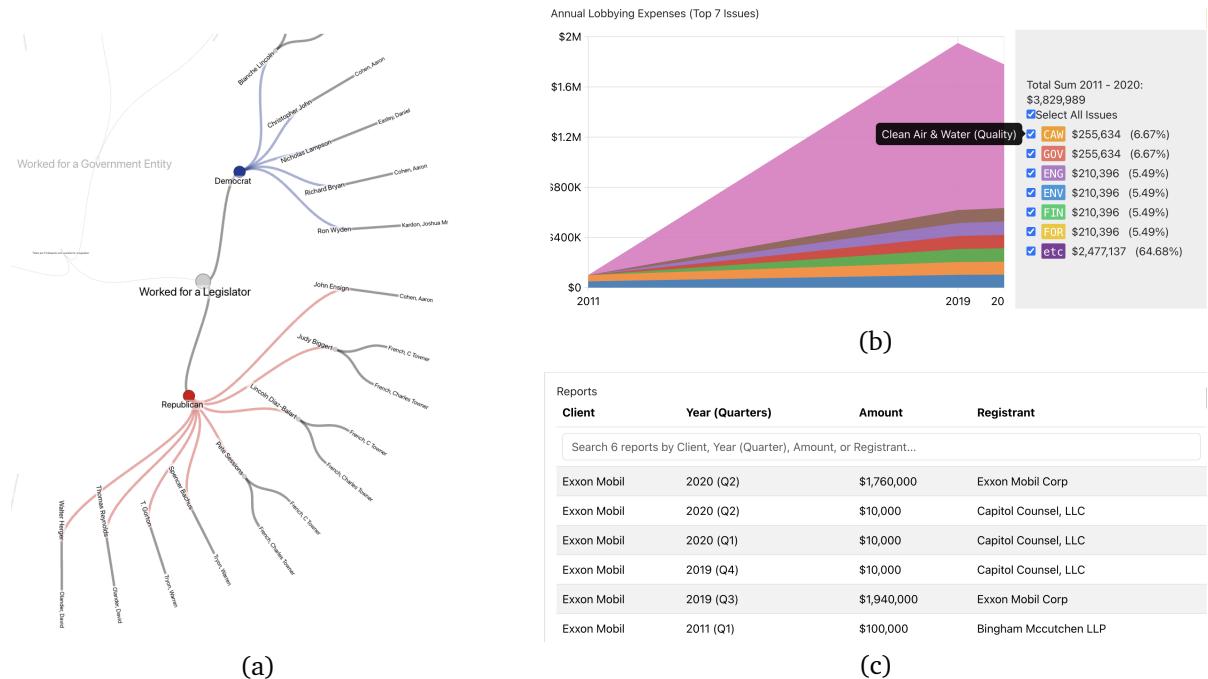


Figure 3.2: LobbyView data linking ExxonMobil's lobbying efforts to a specific Member of Congress. (a) depicts former Congressional Staffers that lobbied for ExxonMobil, (b) shows ExxonMobil's lobbying expenditures per policy issue and (c) shows the expenditures per quarter and lobbying firm

These data do not reveal which sections of a bill, companies were interested in or concerned with, which politicians the lobbyists contacted, and what they talked about. These dynamics are primarily uncovered by investigative journalists in specific cases, but generally not publicly available.

### 3.4 Domain Selection Rationale

The money flow that nourishes the political status quo of a country has been a phenomenon that naturally emerges from the way that country is organized, but in most cases and for most of the time was not clearly documented or made accessible. In democracies, the politicians of the legislative branch are the ones drafting new laws, which gives companies affected by these new laws incentives to establish and maintain communication channels to these politicians. It is no wonder that the practice of lobbying emerged and in almost every country turned into an industry, specialized in handling the communication between a politician and a company.

Yet mostly these practices remain absent from public records, since a company or a stakeholder does not have an innate interest in publicly displaying their pursuit of goals. The United States serves as a special example, where the Lobbying Disclosure Act of 1995 obligates companies to register not only the lobbyists they hire to communicate with the politicians, but also the concrete bill they are concerned with. As a consequence, huge quantities of textual data

from 30 years of lobbying disclosures have been piling up, and as part of the Herculean effort of researchers, comprehensive databases like the LobbyView database or OpenSecrets have been made publicly accessible. [47, 48]

Additionally, the United States Congress publishes detailed records of the proceedings happening inside both houses. These include bill summaries, committee referrals, amendment proposals, debate transcripts, votes and much more data points that document what happens in Congress in exhaustive detail. This opens up many avenues of possibly interesting insights by relating the politicians' actions on the Congress floor to the lobbying disclosure data that is available. However, most of that research is limited to rigid statistical methods since the array of data is extremely large and text-heavy. This is why LLMs open up vast methodological possibilities for this niche domain.

The screenshot shows the details of Bill H.R. 1 of the 119th Congress. The main content area displays the bill's title, "H.R.1 - An act to provide for reconciliation pursuant to title II of H. Con. Res. 14.", and its introduction date, "05/20/2025". It lists the sponsor, Rep. Arrington, Jodey C., and the committee, House - Budget. The bill has undergone several committee meetings and prints, including "H.Rept. 119-106 Book 1" and "H.Rept. 119-106 Book 2". The latest action was taken on "07/04/2025 Became Public Law No: 119-21." The bill has 47 roll call votes. A tracker indicates the bill has moved from "Introduced" through the House and Senate to "Resolving Differences" and finally "Became Law". The right sidebar provides links to "More on This Bill", "CBO Cost Estimates", "Subject — Policy Area: Economics and Public Finance", "Related CRS Products", and "Give Feedback on This Bill". Navigation links at the bottom include "Summary (5)", "Text (6)", "Actions (140)", "Titles (9)", "Amendments (493)", "Cosponsors (0)", "Committees (1)", and "Related Bills (25)".

Figure 3.3: Details of Bill H.R. 1 of the 119th Congress published on the Congress website

In order to test and validate the way users will use the array of feedback mechanisms together, we wanted to come up with an additional agent team that experts from a very specific domain are able to use and see value in. In this way, we set two different use cases of our feedback mechanisms that allow us to analyze how the user experience overlaps and differs. Henceforth, we decided to build a team of agents that can solve Congressional lobbying tasks, bridging our need of diversifying the user scenarios with actual opportunity exploration of LLMs in the money-in-politics research area.

# Methodology

---

## 4.1 Research Approach

Existing research on human control in multi-agent systems has examined human-initiated interruption [14] or controlled agent-initiated feedback requests [11, 10] in isolation. No prior work has investigated how users interact with both mechanisms simultaneously, nor how these interaction patterns differ between general-purpose versus domain-specific tasks.

We address this gap through an empirical study of dual-mechanism interfaces. Our approach involves three key components: (1) extending the Microsoft Autogen library to support human-initiated interruption alongside its existing agent-initiated feedback (Section 5.4), (2) adding support for user-defined criteria for agent-initiated feedback requests inside the Microsoft Autogen library [10] (3) implementing two distinct agent teams—a general deep research team and a domain-specific Congressional lobbying team—to assess context-dependent interaction patterns (Section 5.3), (4) building a visual interface that makes agent-initiated feedback and human interruption accessible and intuitive and (5) conducting user studies with students and domain experts to capture interaction behaviors across expertise levels (Section 4.3).

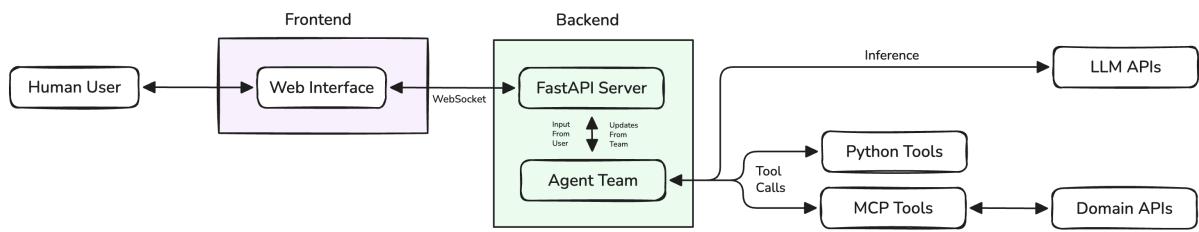


Figure 4.1: Simplified overview of our system (see Section 5.1)

This methodology enables us to answer our research questions by observing the actual user behavior with functioning systems rather than hypothetical scenarios. Chapters 5 and 6 detail the technical implementation of our experimental apparatus, while this chapter focuses on our research design and evaluation approach.

## 4.2 Experimental Design

To assess the generalizability of interaction patterns in different contexts (RQ4), we designed two distinct agent teams. The deep research team handles open-ended research questions, while the Congressional lobbying team performs domain-specific investigations that require specialized knowledge (detailed in Chapter 3). This design allows us to separate general multi-agent interaction patterns from domain-specific behaviors.

Similarly, we recruited two distinct user groups: Bachelor's students for the deep research team and Congressional lobbying domain experts for the specialized team. This pairing enables comparison between novice and expert interactions with their respective systems, revealing how domain expertise affects the usage of the feedback mechanisms.

The goal is to observe the interaction patterns of the participants with the multi-agent system itself, and with the options to exercise control over the conversation trajectory the multi-agent system is taking. More specifically we wanted to observe the reasons for human-initiated interruption, the effect of agent-initiated feedback requests on the participant's behavior, the participants' usage of the UI elements (Section 6.1) and the agent-initiated feedback's effect on human-agent interruption and vice versa.

The engineering of the agent teams is detailed in Section 5.3, while the Autogen extensions enabling dual feedback mechanisms are described in Section 5.4.

## 4.3 System as Experimental Apparatus

Participants interacted with a visual interface displaying agent communication in real-time. The interface provided three key capabilities: (1) visibility into inter-agent communication through swimlane visualization, (2) human-initiated interruption allowing users to pause and redirect agents at any point, and (3) agent-initiated feedback where agents could request user input based on customizable trigger conditions.

These capabilities directly support our research questions: visibility addresses RQ2, human interruption enables study of RQ1, and configurable feedback triggers inform RQ3. The complete technical implementation, including the MCP server (Section 5.2), agent architectures (Section 5.3), Autogen extensions (Section 5.4), and frontend interface is detailed in Chapter 6. Here we focus on the research design aspects.

## 4.4 User Study Protocol

### 4.4.1 Participants

We recruited 5 Bachelor's students to interact with the deep research agent team. Recruitment criteria included a STEM background (2 Computer Science, 2 Mechanical Engineering, 1 Physics),

as well as prior experience with using chatbots like DeepSeek or ChatGPT.

Additionally, we recruited 2 Congressional lobbying domain experts to interact with the specialized Congressional team. One expert is a retired lobbyist, with experience both inside the government and in the lobbying domain and the other is a PhD student in American Government deeply familiar with the proceedings in the United States Congress.

#### 4.4.2 Study Procedure

The studies for the Deep Research team were all conducted in-person, since the students chosen are from the same institution our research lab is in, namely ETH Zurich. As for the lobbying experts, we had to conduct the user studies online via Zoom. They were able to use our interface by giving them remote control over our computer that runs the interface locally.

For the students, we first gave them a walkthrough of the entire interface. This was done with an example query, for the Deep Research Team we tasked the multi-agent team to search for "Master's programmes combining Computer Science and Social Science in European Top Universities that can be applied for with a Bachelor's degree of Computer Science". The task for the Congress experts varied a bit, we pre-defined the task that was given to the agents. Given a company-bill pair  $(c, b)$  they were tasked to find out which politicians  $p$  where most aligned with company  $c$ 's position on  $b$  and which were most opposed.

The participants were asked to perform two runs with the agent team they were presented. The first run for the Bachelor's students was tasked to be a slight modification of the Master's programmes prompt, to cater for their specific interest. As for the second run, the query could be arbitrarily chosen by the participant. For the Congressional lobbying domain experts, we let them choose the company-bill pair in both runs. At all times, we asked them to think aloud during navigation of the task, and during long periods of silence we prompted them to think aloud again with spontaneous questions or comments. We specifically tracked the context of their interruptions and their thoughts and reactions when being confronted with an agent-initiated feedback request.

After their final run, we started an open conversation that was guided by a set of open-ended questions serving as a pillar for the flow of the discussion. This conversation aims at getting to the core of the qualitative insights that are needed for each of the 3 main research questions. In RQ1 for example, we want to know the reasons for each participant's navigation through the feedback landscape we presented them with - below one will find the open-ended questions for each RQ. As for RQ4, we use the insights distilled from the first 3 research questions to draw differences between both user groups. Subsequently, we watched the recordings again and took notes on their general comments that we could use to answer our research questions.

#### Open-Ended Questions

- **Q1:** Tell me how you used both the interruption feature and the automatic feedback triggers. Which did you find most useful and were there moments where they conflicted?

- **Q2:** How useful was it to see individual messages between agents? What level of detail did you actually need to navigate the agents through your task?
- **Q3:** How did you decide what to put in the feedback component prompt? How did these feedback components prove to be for you during the run?

#### 4.4.3 Data Collection Instruments

Following each session, participants completed the System Usability Scale (SUS) [49], a validated 10-item questionnaire measuring perceived usability. The SUS provides a reliable quantitative baseline for comparing the two agent team interfaces. The responses were all collected via Google Forms and could be completed in around 5 minutes for each participant.

As for the sessions themselves, we recorded the screen and the audio while they were navigating through the tasks or answering the open-ended questions. The runs took around 15 minutes each and the open-ended question round took another 15 minutes at most. Each recording was analyzed manually afterwards, noting down the relevant actions from all user interactions including interruption events, feedback submissions, agent messages viewed, timing data and their answers in the open discussion at the end.

### 4.5 Analysis Approach

We analyzed SUS scores to compare perceived usability between the two agent teams and user groups. Interaction logs were quantified to measure frequency and timing of interruptions and feedback events.

For this we followed the standard procedure outlined in [49] to grade our overall system in the first place, by taking the average of each individual participant's SUS score. We then looked at the correlation between AI usage experience and the usability score, manually classifying the open-ended descriptions into Advanced AI users, Regular AI users and Basic AI user. To understand what participants consistently liked or disliked about our system, each question was analyzed in isolation comparing (1) the variance and (2) the mean in the answers, together indicating the divisiveness or consistency of the usability statement.

These quantitative and qualitative methods were applied systematically to address each research question. For RQ1 (merging of feedback mechanisms), we analyzed interruption frequency and reasoning (agent and human-initiated) from interaction logs together with participants' explanations of their decision-making from the interview transcripts. For RQ2 (transparency needs), we examined which UI elements participants accessed and their stated preferences about agent message visibility. For RQ3 (feedback component design), we analyzed the components participants created, how they affected the entire run and what issues or preferences the participants voiced based on their initial choices. For RQ4 (expertise effects), we conducted comparative analysis of student versus expert interaction patterns, including their reliance on system guidance versus domain knowledge. The SUS scores provided an overall usability baseline across both groups and agent teams.

# System Architecture

---

## 5.1 Backend Overview

Our system comprises a full-stack effort to build two agent teams that implement the feedback mechanisms we are investigating in this thesis. First we built a comprehensive MCP server for the US Congress API [50] that enables agents to semantically browse through the detailed proceedings of a bill. Subsequently we developed and tested the two agent teams in Microsoft Autogen and subsequently modified the library to accomodate the conjunction of feedback mechanisms. Finally, we designed a visual interface that displays the agent communication, and enables a user to interrupt and/or terminate the run of the agent team.

## 5.2 US Congress MCP Server

For the domain-specific team, an MCP server was developed that provides the agents comprehensive access to public data from the US Congress API. This involved defining access functions that allows an agent to intuitively retrieve information from the Congress API about a bill's detailed proceedings, which will be important for the lobbying task we will give them. In some functions we employed RAG techniques to filter out information in order not to blow the context window of the agents. Another MCP server for the United States Congress already exists [51], however, it merely replicates the API endpoints, and it does not support the advanced functionality we need for our agents to have the necessary information at hand. Therefore, to understand how to design the MCP server, that is, what tools we make available to the agents, we built up a good understanding of the US Congress beforehand (Chapter 3).

### 5.2.1 MCP Function Design

Our MCP functions provide agents with access to the proceedings that are relevant to uncovering the lobbying dynamics for a specific bill, since this is the task we are giving the agent team in the first place. This is non-trivial and involves finding all the politicians that have influenced a bill's trajectory, and comparing that influence. The goal here is to let the agents reason and find out which politicians were of most interest to the lobbyists that were known to be involved in that bill. For this, we have come up with a manifold set of functions, divided into increasing levels of complexity.

## **Basic API Getter-Functions**

Many base functions in our MCP server directly relay MCP calls to API endpoints, primarily getter functions. In our tool collection, there are functions to retrieve...

- the summary of a bill
- the sponsors of a bill
- the co-sponsors of a bill;
- a summary of the bill's trajectory (actions);
- the number of amendments to the bill;
- the sponsors and co-sponsors of an amendment (if any)
- a summary of the amendment's trajectory (actions);
- the party of a specific Congressman
- the state or voting district of a specific member of Congress;

These functions involve simple calls to the corresponding API endpoint, and parsing logic to convert the XML response back into JSON.

## **Medium-Complexity API Getter-Functions**

Some functions needed more involved logic to get the right bundle of information. For example, the functions that get you all of the actions inside a committee need additional recursive searches if any subcommittee to the committee was involved as well. If so, we fetch the actions for each committee and subcommittee separately. Another example is the function that gets all of the Congressmen of a certain state, which first calls the list of all Congressmen, and then filters out the ones that come from the requested U.S state.

## **External Data MCP Functions**

Some crucial data to pinpoint the involved members of Congress in a bill's trajectory isn't directly available through the endpoints. In our case, we had to search for additional data sources to find the committee membership rosters and the voting information, important knowledge for the agents if they want to assess who was involved in a bill's trajectory and in what manner.

We found the historical committee membership data in a GitHub repository [52] made by civic hackers that publish tools, such as district geofencing utilities and code reverse-engineering the contact forms of Congressmen. Their data includes all committees and subcommittees with detailed information about which member of Congress resided in which committee, and the rank they had inside the committee. Since they update this data for each newly-elected Congress, we downloaded all the files from the 113th (beginning in 2012) up to the 119th Congress (now) by browsing through the repository's commit history. This means that our Congress agents have access to all the committee membership data from 2012 onward.

While writing the code for the MCP voting data functions, there were no endpoints to retrieve information about which member of Congress voted for or against a certain bill or amendment. For the House of Representatives, this data is available on the House Clerk's website. [53] We wrote a helper function that scrapes the voting data for a specific "roll call event" (a voting event) by dynamically creating the specific URL that leads to the page with the requested voting data. As for the Senate, the votes are directly accessible from the Senate's official website, and similar scraping techniques were used to make this data available to the agents through a simple MCP function call.

The Congress API as of December 2025 serves voting information directly through an endpoint, but only for the House of Representatives as part of their beta testing.

## Inferring Relevant Sections

The most important MCP tool that we offer our agents is a function that allows them to retrieve the sections of a bill, that a company is likely interested in. This is very useful in our case because most bill texts (see Figure 4.1) have the ability to fill up the context window of our agents pretty quickly. At the same time, it would be an additional workload for the agents themselves to find out what parts of a (possibly very long) bill text a company will find relevant for their business model.

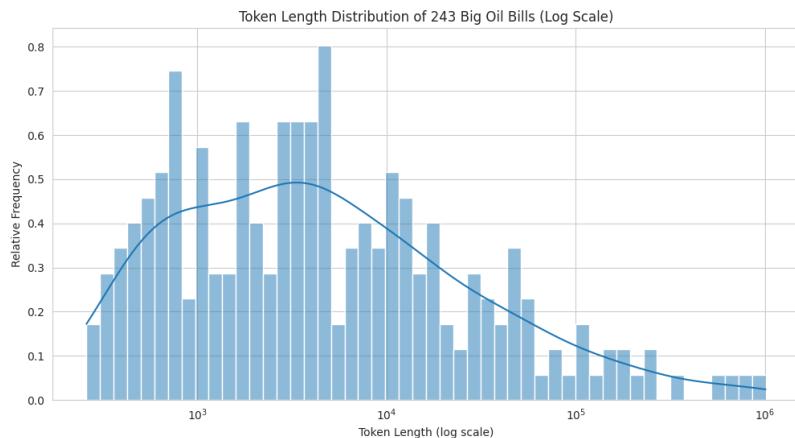


Figure 5.1: Distribution of text length of 243 bills that have been lobbied for by Big Oil Companies in 2018

This is why we designed a RAG pipeline that retrieves the full text directly and can extract the relevant sections directly for the agent. For that, we fetch the full text directly from the Congress API, and then chunk it into its sections. Each chunk will be indexed by calling the OpenAI `text-embedding-3-large` and stored into a Chroma database on the server. When the agent calls the relevant sections function with a bill-company pair, we fetch the summary of the bill and generate 5 questions that a company might have for that bill, based on the information we have from the summary. Each question will then be used to perform a retrieval query to the indexed sections. After doing this 3 times for 5 questions each, we return the sections that appeared at least in 2 of the 3 query runs, in order to make the outputs of that function more deterministic.

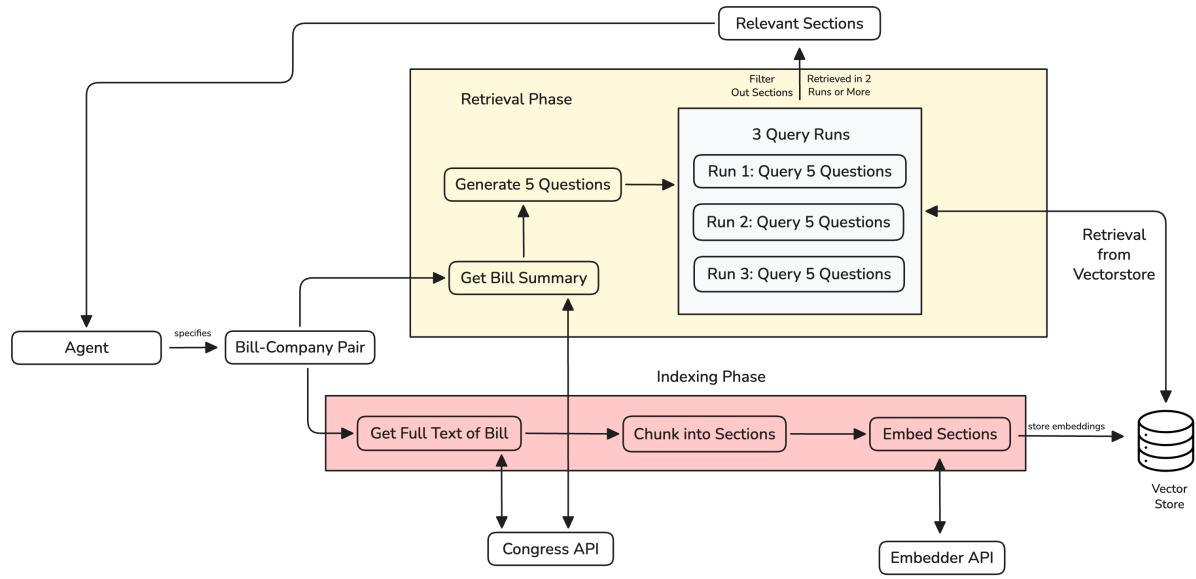


Figure 5.2: The retrieval pipeline we use to gather the relevant sections of a bill for an arbitrary Company-Bill pair

## 5.3 Agent Team Design and Engineering

The agent teams had to be designed and tested before we could use them in any user study. We began by looking at different multi-agent frameworks that would allow us to build the teams with the dynamics we wanted.

### 5.3.1 Framework Selection

We evaluated several multi-agent frameworks for implementing our dual feedback mechanisms. The first framework we considered was CrewAI, which allows you to define individual agents and subtasks all unified into a "crew". We found that the library was particularly good at defining a sequence of tasks and carrying them out one-by-one. However, we discovered that the sequence of tasks would need to be hardcoded when working with the library, and that it was mainly meant for building agent teams with pre-determined pipelines. This wouldn't have helped our use case, since we were aiming for a dynamic investigation, where the findings from the tool calls would determine the next steps.

Accordingly, we selected Microsoft Autogen, particularly the `autogen_agentchat` submodule because it has built-in modules for dynamic next-speaker selection. Initially, we did some experiments to determine which LLMs were best for correct tool calling.

Model & Message	GPT-4o	GPT-4o-mini	GPT-4.1	GPT-4.1-mini	GPT-4.1-nano
Raw	Yes	Partly	Yes	No	No
Filtered	Partly	No	Yes	No	No
Trivial Instruction	Yes	Yes	Yes	Yes	Yes

Figure 5.3: Small experiment with 3 instruction messages of decreasing complexity to see which models call the tools correctly

### 5.3.2 Congressional Lobbying Team

After this we built and tested the agent team for the Congressional lobbying task over 6 iterations.

Iteration	Description/Additions
1	3 agents & LLM selector
2	2 more agents
3	Complex manual selection logic instead of LLM
4	LLM selector with last message extraction
5	System Message & Task Message Refinements
6	Agent-Initiated Feedback included

Table 5.1: Initial Congress team iterations and what they added

At the sixth iteration, we had the bedrock for our agent team's member count, the tool calls each agent has access to, and the system messages each agent receives in order to manifest their identities. Only small modifications have been made after. We generalized this architecture into a YAML-based configuration system enabling rapid agent team specification (detailed in Section 5.5.1).

### 5.3.3 Deep Research Team

For the deep research team, we took a template from an existing GitHub repository, that used multiple agents to do journalism research. [54] We converted the code originally written for the AG2 framework into Autogen instructions, removed the code executor agent and implemented an extension that enables one to set "allowed transitions" from one agent to the other in Autogen. Eventually we observed that it works better by removing the "allowed transitions" feature, and kept the next-speaker logic similar to the Congress team.

## 5.4 Microsoft Autogen Library Extensions

Although the Microsoft Autogen library is intuitive to use for our open-end multi-agent team use case, it only supports agent-initiated feedback interruption. In order to make our agent teams ready for the research we are interested in, we had to extend the library to accommodate both directions of feedback. Two further modifications were required: including reasoning for each

agent-initiated feedback request and context engineering for the agent teams such that the next speaker selection made semantically more sense. We refactored these modifications away from the core Autogen library code into "plug-ins" whose usage is optional.

### 5.4.1 Understanding the Autogen Library

We worked with two submodules of the Autogen library: `autogen_core`, which is the main pillar that contains the low-level definitions and mechanisms governing the agents, and "`autogen_agentchat`", where the logic for team coordination resides. There are multiple group chat modules that Agentchat offers, differing from each other through their orchestration mechanisms.

A group chat (used interchangeably with "team" in this work) is a collection of agents that can "speak" one after the other. When we say "speak", we mean that an agent can either just generate text, call tools (provided that any are made available), or summarize the results obtained from the tool calls. Each group chat has its **group chat manager**, which selects the next agent to speak after another agent spoke. This group chat manager can be either a trivial mechanism that selects the agents in a round-robin fashion, or it can implement an involved selection process with LLM calls and subsequent logic determining the next suitable agent to speak.

Moreover, the group chat manager is the instance in control of the entire agent team run. Every time an agent finishes speaking, it is the manager who receives the response and transmits it to all the other agents. Generally speaking, the manager is in charge of the "event loop", it (1) selects an agent to speak (by the logic specified), (2) waits for the agent's response, (3) examines the agent's response for a pre-determined termination condition, (4) publishes the agent's response to the other agents, and (5) selects the next agent. This loop continues until some agent's message meets the termination condition.

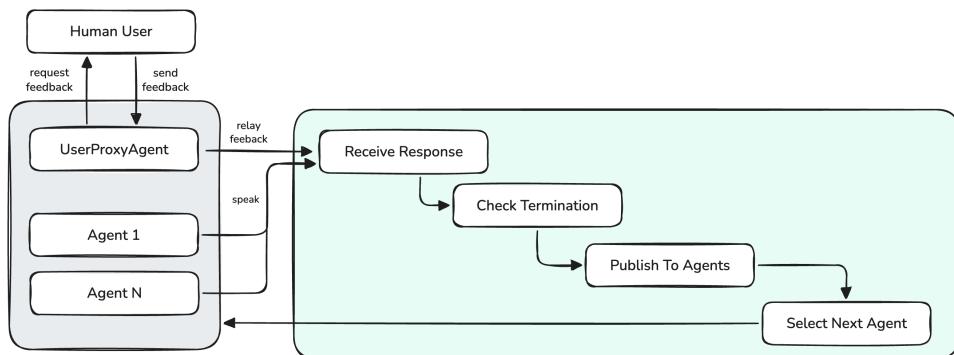


Figure 5.4: The role of the Group Chat Manager inside the event loop of the agent team run

The agent library contains many special agents with distinct functionalities. One of the most important ones for our work is the `UserProxyAgent`, which is the interface agent for the human user, that can be called in the case that the agents want to have feedback from the human user. The manager can select this agent just like any other agent, and, if selected, pause the run until the human user writes something into the specified input function.

### 5.4.2 User Interruption

Yet, as mentioned before, one functionality that the Autogen Agentchat library does not implement for its group chats, is the option to explicitly interrupt the agent team from outside, and send it feedback or additional instructions before resuming the conversation again. Existing functionality only allowed for interrupting and resuming again, without adding a message in-between.

We extended the library accordingly by adding two new events: "UserInterrupt" and "UserDirectedMessage". Based on these new events, we implemented additional signaling logic to harbor the human-initiated interruption and human-initiated feedback into Autogen's event loop:

- **Group Chat:** Instead of directly interrupting the run through the runtime, we explicitly send a UserInterrupt message to the manager
- **Group Chat Manager:** Sets an additional interruption flag upon UserInterrupt and sends a directed message to the desired agent upon UserDirectedMessage (provided that the interruption flag was set before)

All of this functionality is wrapped under a new agent module called the `UserControlAgent`, which has two attribute functions, one to call interrupt the agent team run (`UserInterrupt`) and one to subsequently write a direct message to an agent (`UserDirectedMessage`). This agent does not even need to be added to the team, it just needs the reference to the team, and executes its functionality from outside.

### 5.4.3 Branching

An additional thing that we wanted to implement is the ability to interrupt the stream at any point of the existing conversation. The only user study testing human interruption with a multi-agent system found that one of the most desired functionalities to probe and steer an agent team was precisely the aforementioned feature. [14] This turned out to be much more challenging than previously thought, because we would need to remove messages from the thread that the manager stores to keep track of the conversation. Every time the manager receives a response, it adds it to the message thread and continues from there. The issue we found is that the individual agents do not share a synchronized message thread with each other. This complicates the interruption at any conversation point, and the reversion back to that message in all of the agent's message threads.

This out-of-sync issue has multiple layers. First, the group chat manager stores text messages and tool calls in its message thread, whereas all the agents simply store the text messages which the other agents emitted. But more importantly, agents flush their individual message-thread copies as soon as they finish their turn. This design choice by the library's creators keeps context local to each agent and prevents the global thread from becoming cluttered. However, for our purposes, it introduces a mismatch: each agent's message-thread store becomes inconsistent with those of the other team members.

So we had to code a translation middleware that accounts for this mismatch. If the human interrupter wishes to re-set the conversation, say 10 messages up the manager's message thread,

we first count how many of these messages are pure text messages. Then we individually check for each agent when it last spoke, recalculate and send the respective re-setting value for each agent. This ensures that all agents have the same messages removed from their message thread stores when the human user "branches" the conversation.

#### 5.4.4 Plug-Ins

During the course of implementing the advanced feedback mechanism, we modified the Autogen Agentchat library code a lot to achieve our desired functionality. However, most of this code was to extend existing modules in Autogen, not to introduce new ones, and thus cluttered the library code. This is why we refactored a lot of code into plug-ins that can be dynamically activated and deactivated. For that, the core library code was modified only to incorporate the plug-ins if there are any, and continue with its core functionality otherwise.

We introduce two plug-ins: The context engineering plugin and a plugin for the agent-initiated feedback components that the user defines in the beginning of the run. Both plug-ins will be described in more detail in sections 5.5.2 (Context Engineering) and 5.5.3 (Feedback Components).

## 5.5 Backend Architecture

With the Autogen library extended for our purposes, we were able to build a pipeline that builds an agent team dynamically from the provided specifications. Based on these specifications—defined within a single `team.yaml` file—together with either a Python file specifying the agents' tool functions or the URL of an MCP server, an AutoGen agent team is instantiated with access to the corresponding Python or MCP tools.

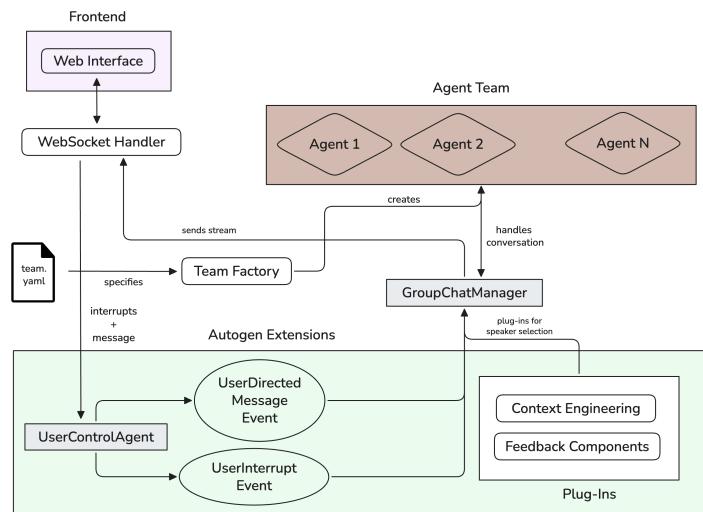


Figure 5.5: Detailed figure to show the extension we made to the Autogen library

### 5.5.1 Team Configuration System

Located inside `team.yaml` is almost all the important information to build the agent team. All of the agents' names, system messages, and available tools are defined and assigned there, the prompts that guide the group chat manager's next speaker selection can be specified, and the choice of LLMs is set there as well. Optionally, a `UserProxyAgent` can be included to enable agent-initiated feedback requests, and the main task assigned to the agent team may also be specified at this stage—as done for the Congress agent team—whereas for the Deep Research team the main task is defined at a later point in the pipeline.

In order to convert these specifications into an actual running agent team, we provide factory code that makes this happen. There are four different factories:

1. **Input Function Factory:** Links the `UserProxyAgent` to a function that defines the logic how human feedback should be requested from the agents
2. **Model Client Factory:** Creates the model clients from the specified model names. Supports Anthropic and OpenAI APIs
3. **Tool Function Factory:** Links the Python tool functions to the agents' workbench, making them callable by the agents
4. **Team Factory:** Builds the agents' from all of their specified attributes, calls the other factories, and integrates everything together into a working agent team

This modularity allows us to specify any agent team without code and build it directly. The input function factory and the tool function factory are relatively simple procedures that aggregate all the tool functions inside a function registry, which allows the Team Factory to load the functions into the agents' workbenches. In contrast, the team factory takes care of the non-trivial parts of the setup, especially those of the plug-ins.

### 5.5.2 Context Engineering

LLM orchestration is hard to get right, and Autogen's inherent functionality enables us to modify the prompt, where we can reference the agents' names, their descriptions, and the entire message thread. Yet, even after many prompting iterations, we encountered limitations as to how seamless next-speaker selection worked, often witnessing selections that made no sense in context. We suspected that this was due to the confusion that the LLM decides to decide the next speaker when all the messages ever sent are pasted into the prompt.

This is why we wanted to keep additional refined summaries that the manager can use for clearer next-speaker prompts, and the agents could use to have a better understanding of what happened in the run and where the entire team is headed. Those include three main fields:

- **State of the Run:** A high-level summary of what has already been found, the next steps, and the concrete next step
- **Tool Call Facts:** A bullet-point list of all the information the tool calls returned

- **Handoff Context:** Instructions that guide which agent should be invoked in which situation

We modified the logic inside Autogen’s group chat modules so that we can replace the entire message history with this shorter and more structured overview, reducing confusion for the next-speaker selection. The next-speaker prompt receives the State of the Run and the Handoff Context, supplying it with the concrete next step and the handoff instructions. The Tool Call Facts are used for the agent-initiated feedback components elucidated in the next section.

The State of the Run field is updated after every text message and the Tool Call Facts after each tool call execution. The Handoff Context never gets modified for simplicity. For this, we use specific update instructions that we send to a low-latency LLM, Claude Haiku 4.5 in our case. We included the option to update it after each human feedback message. For this we had complex LLM routing to detect instructions from the user to change the next-speaker logic, but we removed it as a tradeoff for user experience. The delay of the LLM calls added up proved too long, despite using Claude Haiku 4.5.

This allows us the manager to clearly see what the next step is and made the next-speaker selection much clearer and better working. We refactored this logic to the plug-ins.

### 5.5.3 Feedback Components

With functionality in place to build an agent team dynamically, interrupting the agent run from outside at any point of the conversation, sending a message into the team, and offering support to catch agent-initiated feedback, we have all the elements needed to test our multi-agent system with dual feedback (agent-initiated and human-initiated), at least on the surface. The only problem that remains is that the human user has no control over when such feedback will be triggered.

In order to solve this, we let us inspire by a previous approach that takes open-ended textual input from the human and converts these to "agent metrics" that an agent can use to adapt its behavior mid-run. [10] We used a very similar approach to let users decide when they want the agents to ask them for feedback, by requesting open-ended textual input before the run begins and converting it into "feedback components". These feedback components are then compared to every new message and triggered if the message semantically overlaps with the specified component. An example of a feedback component would be "Trigger if any Democrat senator is mentioned" or "Watch for any universities that are outside of Germany".

The decision to trigger or not to trigger an agent-initiated feedback is done by an LLM-as-a-judge that is tasked to score the semantic overlap of an agent message and a feedback component from 1 to 10. For the user studies, we fixed a trigger threshold of 8, and explicitly instructed the LLM to output reasoning for values above 8. This enables a human user to specify when exactly the agent team should involve them, and also to provide a short reasoning description to explain why exactly the agents stopped the run and are asking for feedback.

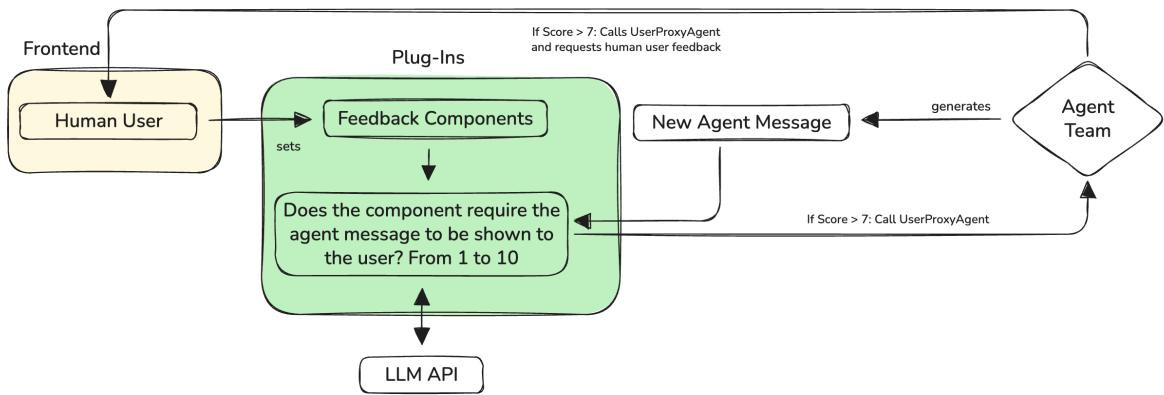


Figure 5.6: The feedback component system underlying the agent-initiated feedback

The alternative would have been to perfection context engineering that would enable the manager to decide perfectly the human user should be called for feedback. However, even if users could specify beforehand what they wanted the agent team to watch out for and solid logic converting that open-ended request into good context were in place, it would have nevertheless been difficult to extract reasoning for agent-initiated feedback, since the UserProxyAgent, that is, the agent that would be called to represent the human user isn't coupled to an LLM.

# Visual Interactive Interface

---

The backend logic we implemented allowed us to think of many possible interfaces that our test subjects would work with. We began sketching the interface a long time before the backend was finished, but with the backend in place we had to iteratively adapt to the constraints that the functionality in Chapter 5 posed. We have set the following goals for our interface:

**G1** The interface should provide an intuitive overview for users to see the entire conversation history of the multi-agent team

**G1.1** Before starting the run, UI elements should provide an overview of the multi-agent system and what the user is able to do

**G1.2** The human users should be able to specify the feedback components, as well as edit, remove and add custom ones

**G2** Interrupting the conversation should induce minimal cognitive load

**G2.1** Interrupting the conversation at an arbitrary point should be intuitive

**G3** The feedback components' scores based on the current messages should be easy to see

**G4** The reasoning for Agent-Initiated Feedback should be communicated clearly to the user

**G5** Users should be able to see their own participation footprint in the conversation history

In this section we will first go through all of the elements that are present in our interface, explaining how they fulfill our goals that we set above, then we will discuss the backend-frontend communication setup that powers the configuration settings the human user wishes; thereafter, we will look at how we manage the state from the messages received both from the backend and from the human user, and lastly we will dive into more detail how we realized the swimlane visualization for the message flow.

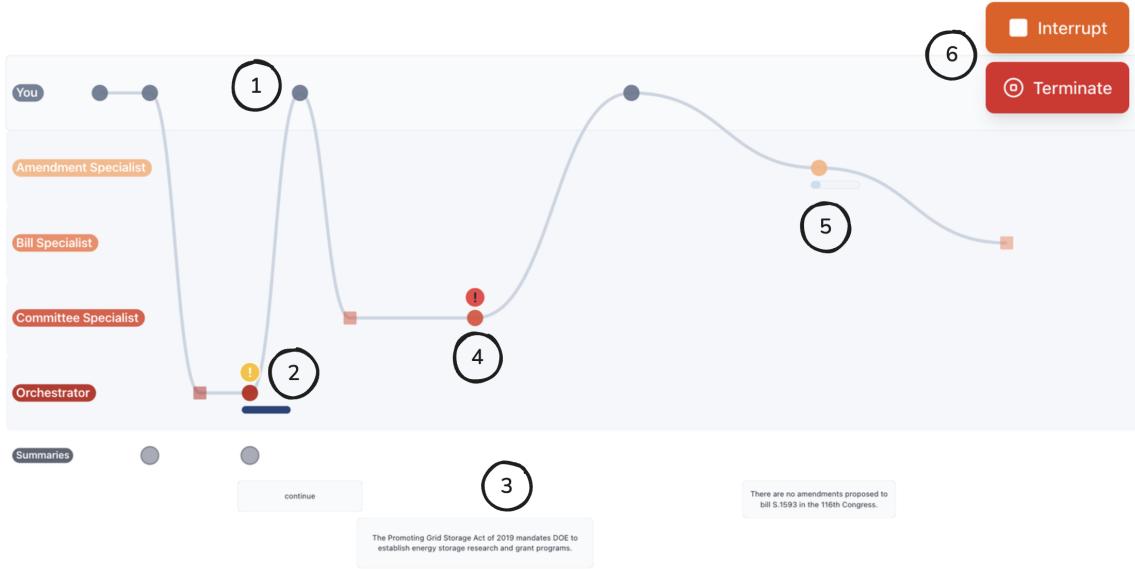


Figure 6.1: An overview of the run view. 1) This is the swimlane that displays all the actions undertaken by the human user (interruptions and feedback) 2) A message node where agent-initiated feedback triggered 3) The summaries displayed below the nodes 4) A message node where the human user interrupted 5) A node that wasn't triggered – the bar chart shows a low score 6) The buttons to interrupt and terminate

## 6.1 UI Elements

The entire interface is divided into three views, one primary view and two secondary views. The landing page view enables the user to input the research task and open-ended text specifying how they would like to receive feedback. Subsequently, the feedback component view displays the generated and refined feedback components extracted from the open-end feedback specifications, and allows the user to edit the list of these feedback components. Finally, the run view, which is our main view, displays the agent run and contains all the main interrupt and feedback display functionality.

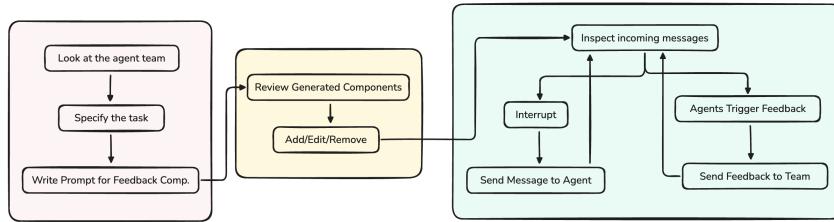


Figure 6.2: The navigation flow of a user through the interface

### 6.1.1 Landing Page View

The landing page view's main two purposes is to 1) give the user a first overview of who is inside the agent team and 2) swallow the user's research request (open-end text for the deep research team and a company-bill pair for the Congress team) and 3) configure the components the user wants to be notified mid-run. At the top of the page we present the user with a "Meet the Team!" overview showing the agent cards, where they can click through all of the agents' names and descriptions. This fulfills **G1.1** partly; one can see what the roles each agent has in the team that one is going to be working with.

Below, either a company-bill pair can be selected via a dropdown (for the Congress team) or a deep research prompt can be input. For the Congress team, we explain the task in three short sentences, list the agent's tool calling capabilities, and the user's role in the process. Also, since the company-bill pairs can be cryptic in their short form even for political science experts (e.g. "Exxon Mobil – S.1593 (116th)"), we include a dynamic link to the respective webpage of the bill, which adapts to different selections. For the deep research team, we simply ask the user "what do you want to do a deep dive on?" and display a text box for a prompt to be written into it.

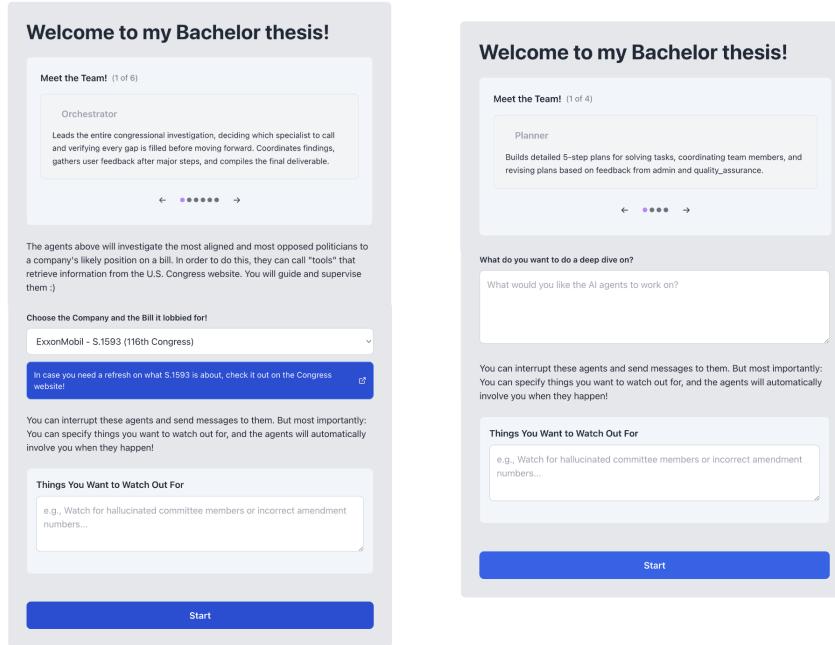


Figure 6.3: Deep Research team and Congress team in the Landing Page View

At the bottom, a text box is shown where one can write down their concerns and specifications that will be converted into feedback components by an LLM. Above the text box, we added two more sentences that explain the user's involvement, namely that they can interrupt and detail the situations in which they want to be notified, fulfilling goal **G1.1** entirely. After that they can click a big "Start" button, which doesn't actually start the run, but calls the LLM to generate the feedback components.

### 6.1.2 Feedback Component View

Upon filling out all the necessary boxes in the previous view, all of the generated feedback components can be looked at. They each have an identifier string consisting of two words and a one-sentence description that is used for the score calculation outlined in Section 4.4.2. The identifiers as well as the descriptions can be edited, the entire feedback component can be removed. New feedback components can be created, allowing the double-checking of the user's open-end input in the Landing Page View. This delivers goal **G1.2** and a "Start Run with {n} components" will send the updated feedback components to the backend starting the agent team run.

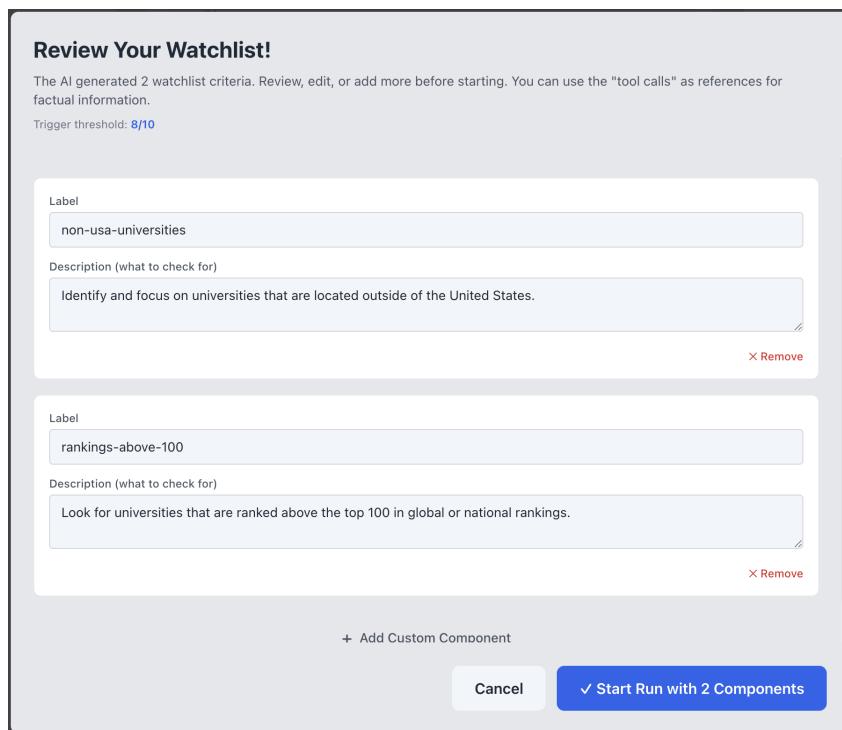


Figure 6.4: The Feedback Component View: Each component has a label and a description

### 6.1.3 Run View

This view gives the user an overview of the agent team run's proceedings. Each agent has its "swimlane", a segment of the view's horizontal spectrum where all of the agent's messages will appear as circular nodes, and its tool calls as rectangular nodes. The rectangular nodes appear as soon as the agent initiated the tool call, while the circular nodes appear only when the agent stopped streaming its text. When a new agent starts to speak and a new node is rendered, an edge is created between the new node and the last node. All of these building blocks allow us to view the entire message history of the agents in a simple manner, fulfilling **G1**.

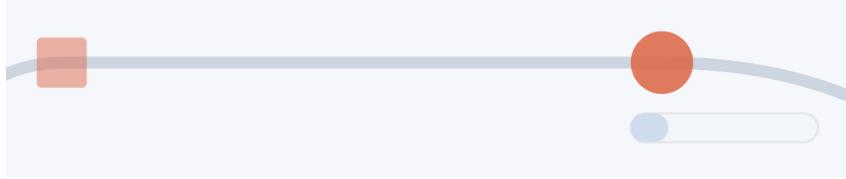


Figure 6.5: Here we can see the difference between a tool call and an agent message. The orchestrator first calls a tool (rectangle) and then outputs a message (circle)

All swimlanes are labeled with the agents' names. The colors of the nodes and the text boxes for the names are the same for each agent/swimlane, and are evenly divided on the OrRd (Orange to Red) sequential spectrum. In order to avoid extreme colorings, we only consider the colors between the 40th and 85th percentiles of the spectrum. As to the message sequence itself, we space the last 5 nodes equidistantly from each other, and shrink the distance between the rest of the nodes in order not to make the user navigate too much when inspecting messages sent a while ago. Below the swimlanes, there is space for 15-word summaries for each of the circular nodes, although we display the full summaries for the last 3 messages only. This is because it would get too crowded if we would display the full summaries for the nodes that have less space between each other.

At the top right, we place two clearly visible buttons, one to interrupt the team run directly and one to terminate the team run once and for all. We chose the "Interrupt" button to be orange, and the "Terminate" button to be red corresponding to the graveness of intervention into the team run. These simple colorings, and the obvious placement therefore fulfills **G2**, minimizing cognitive load. In terms of interruption at a specific point in the conversation, we make the edges wider as soon as the user hovers over them, so that there is more space to click on them, fulfilling **G2.1**.

Upon clicking on an edge or on the orange "Interrupt" button on the top-right of the view, a modal appears on the right that displays the same "Meet the Team!" agent cards, now displayed for the user to select the agent they want to send a direct message to. Below, there is a box to draft the message and a button to send the message to the selected agent. This modal can be collapsed to the right, where a small arrow appears, reminding the user to complete the interruption. If the interruption was called by clicking an edge, a small warning will appear below saying that this conversation will "trim" current messages. To see exactly where the interruption was triggered, we place a red exclamation mark on top of the last node included in the future conversation, fulfilling **G5**.

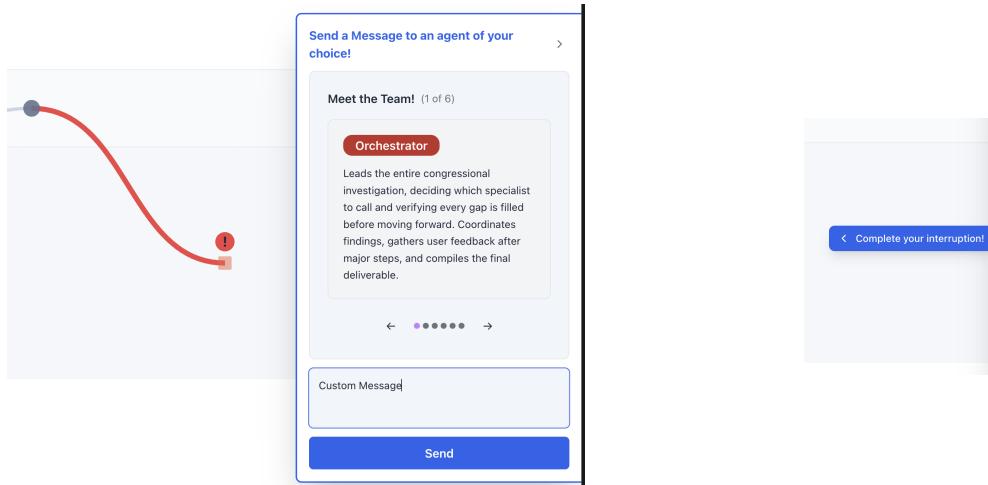


Figure 6.6: The modal for human-initiated interruption and its collapsed version. In this case, it was triggered by an edge interruption.

When a user interrupts at an edge, the conversation will branch from the source node of the edge clicked. This is displayed in our interface by lowering the opacity of the "trimmed" branch and continuing the new branch with the same color scheme as used before.

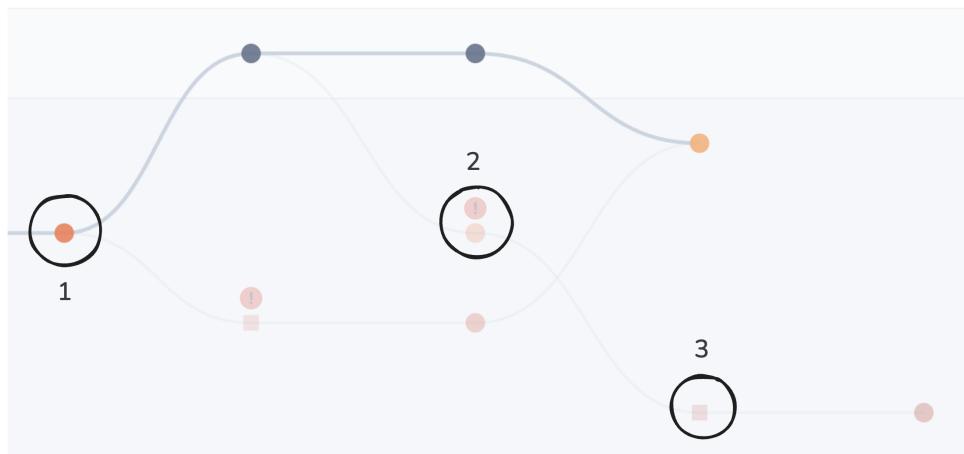


Figure 6.7: A conversation of multiple branches as a result of multiple edge interruption. 1. the last message still included in the old branch 2. the sink node of the edge that was interrupted (not included in new branch anymore) 3. The continuation of the old deactivated branch

Most importantly, all of the circular nodes corresponding to agent messages have a small bar chart below that shows the maximum score of all the feedback components graded in comparison to this message. This allows the user to get a feel for how much they should be concerned with the content of that particular message. In previous iterations, we displayed one bar chart per feedback component per message, i.e. if we defined 2 criteria to watch out for, we would have two bar charts under each circular node. We figured that, for their purpose is to convey the degree of alarm the user should be in, displaying one bar chart with the maximum value from all the feedback component scores satisfies G3 and reduces cognitive load significantly.

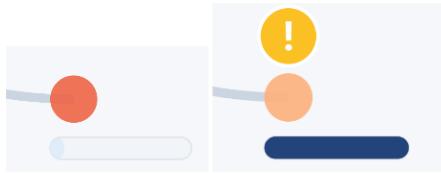


Figure 6.8: A comparison of the bar chart for a low feedback component score and for a high feedback component score

When clicking on a node, a frame opens where one can see by default the entire message of the agent, and below the exact scores each feedback component got for this particular message. Additionally, there is a tab overview at the top of that frame, allowing one to navigate to the "Run State" tab where the State of the Run and the Tool Call Facts are listed, fields we use for context engineering in the backend (see Section 4.4.1). We placed them to give the user the same run summary as the agents, as well as an overview all the findings from the external tool calls, with the hopes of increasing the user's awareness of the general direction the agent team is heading towards.

**Bill Specialist**

Bill S.1593 mandates the DOE to establish energy storage research and grant programs, sponsored by Senator Tina Smith.

<a href="#">Message</a>	<a href="#">Run State</a>	<a href="#">Actions</a>
-------------------------	---------------------------	-------------------------

**Message Content**

Bill\_specialist reporting:

Bill S.1593 in the 116th Congress is titled the "Promoting Grid Storage Act of 2019." The bill requires the Department of Energy (DOE) to establish programs for energy storage research, technical assistance, and grant making for energy storage systems. The main provisions include a research program, technical assistance for entities to identify and design energy storage systems, and a competitive grant program for pilot energy storage systems. The only known sponsor currently is Senator Tina Smith (D-MN).

Next step:  
committee\_specialist, please gather the roster of committees involved with S.1593, their membership details, and any reports or committee actions related to the bill.  
actions\_specialist, prepare to gather the timeline of actions that happened on S.1593.

**Bill Specialist**

Senator Tina Smith is the primary sponsor of S.1593, and cosponsors and committees are being gathered.

<a href="#">Message</a>	<a href="#">Run State</a>	<a href="#">Actions</a>
-------------------------	---------------------------	-------------------------

**State of Run**

**State of Research**

**What we have done so far**

- Investigated ExxonMobil's lobbying focus on bill S.1593 (116th Congress), the "Promoting Grid Storage Act of 2019."
- Defined two main research tasks: interpret ExxonMobil's interests in the bill and rank Congress members.
- Established a specialist team covering bill data, committees, legislative actions, amendments, and Congress member information.
- Bill specialist started gathering bill details but encountered an error due to missing congress\_index in the tool call.
- Documented bill mandates for DOE programs related to energy storage research, technical assistance, and grant initiatives.

**Research Outlook (Next Steps)**

- Gather names of sponsors, cosponsors, and committees for full legislative context (bill\_specialist).
- Gather committee rosters, actions, and reports (committee\_specialist).
- Compile legislative actions timeline on the floor (actions\_specialist).
- Collect amendments and associated data (amendment\_specialist).
- Retrieve Congress member party affiliations, states, and districts (congress\_member\_specialist).
- Analyze data and rank Congress members by effectiveness for Task 2.

**Concrete Next Step**

Bill specialist to gather names of sponsors, cosponsors, and committees involved in S.1593 to provide full legislative context.

(a)      (b)

Figure 6.9: Both tabs of the node details: (a) shows the raw message and (b) depicts the state of research summary used in context engineering



Figure 6.10: Section of the Node Details explaining the state of the Feedback Component Scores: Here the component was triggered

As soon as the score of a feedback component for a particular message crosses the threshold of 8, a modal will appear at the same place where the interruption modal would appear and with the same collapsing functionality. But in this modal, we let the user know that the feedback is triggered by the agents themselves. According to the backend's prompting structure outlined in Section 4.4.2, users are able to display the triggered feedback component and the reason for triggering it (G4). Additionally, the entire text of the message that triggered the respective feedback component can be seen by expanding a by-default collapsed box, named "Details". Below all of this contextual information that serves to elucidate the user as to why the feedback was triggered, there is a message box where the user can draft the feedback to send it back to the team. In contrast with interruption, this feedback is sent to the manager, which subsequently decides which agent to invoke next.

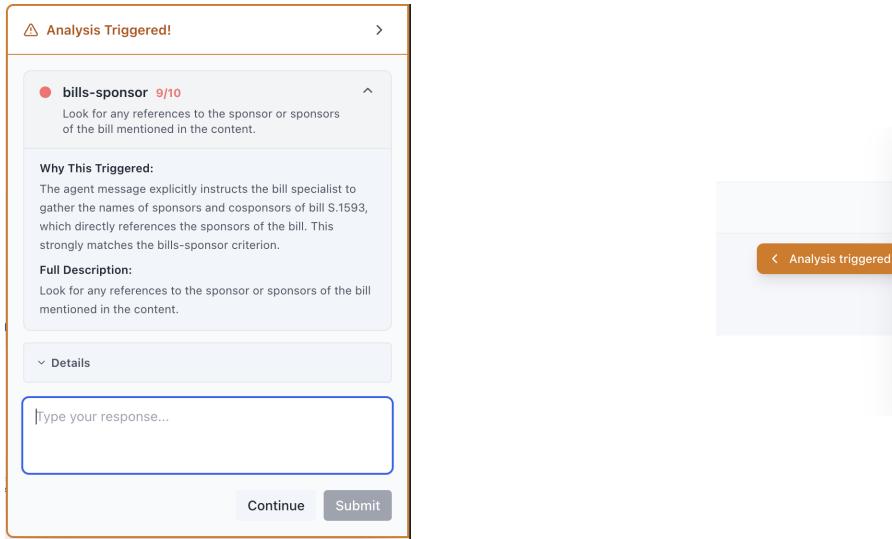


Figure 6.11: The modal for agent-initiated feedback and its collapsed version.

## 6.2 State Management

In order to translate the many different messages the frontend receives via the WebSocket channel into state variables that, we use the Zustand state management library for code modularity.

Zustand allows us to do fine-grained re-rendering by defining hooks to specific subsets of the entire store which only trigger a re-rendering of the components if a relevant state variable was changed.

In this sense, our store serves as the single source of truth for the UI components. The UI components do not react directly to the incoming WebSocket messages, the incoming messages trigger state updates which then via the hooks trigger the re-rendering of the components.

We subdivide the state variables into 13 categories that outline the different features we support in the interface. Notable among these state variable categories are the variables that store the **conversation tree (Conversation)** - based on all messages, tool calls and interruptions - the **User Interaction** category that tracks if the agent run has been interrupted by the user and what message the human user wants to send to which agent, the **Agent Input** category that tracks when the agents are asking the user for feedback, and the **Analysis** category that complements the agent-initiated feedback requests with the triggered analysis components, their scores, and their respective reasoning.

These variables are accessed by 45 different hooks that define different subsets of these state variables, and are tied to a small number of UI components that depend on them. Mostly, the hooks access variables from the same category, but some of them, especially the hooks for human interruptions and agent-initiated feedback requests, hook to a more complex and diverse set of variables. This is because the situations of feedback requests and interruptions trigger various changes in the manner in which UI components are displayed. For a complete overview of the categories, see Table 5.1.

Category	Description
Connection	WebSocket lifecycle management
Agent Team	Agent metadata from backend
Conversation	Message history and tree structure
Stream	Execution state tracking
User Interaction	Human Input after Interruption
Edge Interrupt	Branch creation UI state
Agent Input	Human feedback from agents' request state
Tool Tracking	Tool execution indexing
Analysis	Feedback components monitoring
Component Generation	Component review workflow
State Display	Frontend visibility of 3-state context
Error	Error handling
Termination	Graceful shutdown state

Table 6.1: Zustand store state categories

### 6.3 Configuration & Setup Flow

An important back-and-forth flow of information we need in our interface is the configuration of the agent team run, since the backend does not possess all the information necessary to start the run. It is up to the human user to communicate a specific query or a company-bill pair to the

backend and the specific feedback components they would like to see included in the run. At the same time, we want to present the human user the agents making up the team, as well as short descriptions of their roles. For this, some setup logic was implemented so that we could parameterize the agent team runs with human user input in the beginning.

As soon as the WebSocket connection is established, the following setup protocol is executed:

1. ( $B \rightarrow F$ ): Names of the Agents in the Teams and their short descriptions are sent
2. ( $F \rightarrow B$ ): Human User specifies the task (depending on the agent team) which gets sent to the backend
3. ( $F \rightarrow B$ ): (Optional) A feedback component prompt is sent for component generation (if provided)
4. ( $B \rightarrow F$ ): (Optional) Generated feedback components are returned for user review
5. ( $F \rightarrow B$ ): (Optional) Human User confirms or edits components and sends final run configuration

This bidirectional setup protocol ensures that both the frontend and backend are properly synchronized before the agent team begins execution, with all necessary parameters and configurations in place. Once the setup is complete and the final configuration is confirmed, the backend initiates the agent team run and begins streaming real-time updates back to the frontend through the established WebSocket connection.

## 6.4 Tree Visualization

Enabling the interruption at any point of the conversation makes the conversation thread fit perfectly into a tree data structure, which is the data structure that we use in order to display information in the frontend. In contrast to the group chat manager in the backend, where we simply trim the message thread up to the requested index and then continue the message from there on, we store the old conversation branches in the frontend tree located in the Zustand store, so that we can display them with low opacity in the interface.

Each swimlane has its color, and while we fix the color of the swimlane of the human user to gray, we dynamically adapt the colors of all agents in the swimlanes to be evenly spread across the [0.4, 0.85] spectrum of the sequential color scheme `interpolateOrRd`. At the same time, each feedback component receives its own sequential color scheme, using the full spectrum to display the feedback bar charts under the nodes depending on the score they received.

The rendering logic for the tree itself is implemented using D3.js's data-driven approach, which efficiently handles dynamic updates through its enter/update/exit pattern. This effectively means that when the conversation tree data changes, D3 selectively updates only the parts of the visualization that need modification - creating new SVG elements for newly added nodes, updating attributes of existing elements as their positions or states change, and removing elements that are no longer present in the data structure, analogous to the manner React hooks manage the entire interface. The visualization renders edges as smooth horizontal Bezier curves

connecting parent-child nodes, while nodes are interactive elements that respond to clicks to display detailed information or trigger conversation branching. D3's rendering pipeline ensures performant updates by only manipulating DOM elements that have changed, using a structured five-phase process that handles swimlanes, edges, nodes, and analysis components sequentially.

All tree updates are executed through a five-phase deterministic process: (1) data preparation and position calculation, (2) swimming-lane infrastructure rendering, (3) edge rendering with Bezier curves, (4) node rendering with type-specific shapes, and (5) analysis badge visualization. This pipeline ensures consistent visual updates and maintains rendering performance as the conversation tree grows.

The visualization also implements a sophisticated zoom/pan system with scale extent [0.5, 3] that includes custom filters to prevent zoom initiation on interactive elements such as nodes and edges, ensuring that users can interact with individual elements without accidentally triggering viewport manipulation. This design allows for smooth navigation through large conversation trees while maintaining precise interaction control for branching and node inspection.

# Evaluation

---

In this section, we present the user study setup and the findings we made to answer our 4 research questions. Additionally, the questions in the user study were made to test our complete end-to-end interface in terms of usability, intuitiveness, and usefulness. As already stated in the Methodology section, we tasked 2 legislative lobbying experts and 5 students to work with both agent teams using both pre-defined and open-ended approaches.

We analyzed the data from the user study through qualitative observation notes and user feedback. The results reveal critical patterns within domain experts and students that inform our understanding of human-agent collaboration dynamics.

## 7.1 RQ1: Interruption vs. Agent-Initiated Feedback

Participants showed a strong preference for human-initiated interruption over agent-initiated feedback triggers, though the two mechanisms influenced each other in complex ways. For both feedback mechanisms, usage varied significantly where the frequent appearance of agent-initiated feedback made users interrupt less themselves. Engagement with agent-initiated feedback often led to frustration, if the triggers weren't working as desired, but sometimes prompted the user to look at the agents' messages that have been generated so far and "pose a question directly to the agent team".

The presence of frequent agent-initiated feedback requests had a notable dampening effect on active user engagement. When the system triggered numerous feedback requests, users inspected fewer nodes independently and relied primarily on the triggered messages to understand the agent team's progress. Paradoxically, even when users received these agent-initiated prompts, they sometimes dismissed the provided reasoning without reading it carefully, especially after having witnessed triggers that made no sense to them, suggesting that the interrupted message itself—rather than the system's explanation—was what mattered. This pattern indicates that agent-initiated feedback, rather than enhancing transparency, sometimes substituted for more active user exploration of the conversation tree.

The agent-initiated feedback system suffered from a critical technical limitation: although connected to an LLM, it still based its scores on keyword presence rather than semantic intent. When users specified criteria like "Don't consider X anymore," the system would still trigger if an agent responded with "we will not consider X anymore," because X was mentioned regardless of context and the feedback components couldn't be altered mid-run. This led to either very

frequent triggering (when topics remained relevant throughout the run) or almost no triggering at all, with little middle ground. The resulting noise caused at least one user to abandon the agent-initiated feedback system entirely after a single run, stating they would "only need to read the messages and interrupt."

Several usability issues compounded the challenges with both mechanisms. Interruption had a higher activation barrier in the beginning than expected—users indirectly voiced desires to input something but found it took time to formulate and execute an interruption, while agents generated content much faster than users could review it. In several instances, participants wanted to interrupt but had agent-initiated feedback trigger before they could click, creating an awkward collision between the two modalities. The fact that both mechanisms used the same modal position and size reinforced confusion about their different purposes. Edge-based interruption—allowing users to branch the conversation from an earlier point—was used only in extreme cases where agents became stuck in what one participant described as a "gratification loop," suggesting that this more sophisticated feature was either not well understood or not perceived as necessary during typical usage.

A subtle but important pattern emerged around message targeting. When either modal for agent-initiated or human-interrupted feedback appeared, users were confused as to where their feedback gets sent to. Just as users sometimes wanted to send their feedback to a specific agent during an agent-initiated feedback request, they wanted to send the message to the entire team when they interrupted the run. This confusion arising from the users' mental model of the system is indeed more complex and gets detailed attention in Section 7.3. But likewise, participants frequently used interrupts specifically to involve particular agents they felt were being underutilized or possessed relevant expertise, indicating they maintained detailed mental models of individual agent roles even when uncertain about the overall coordination architecture.

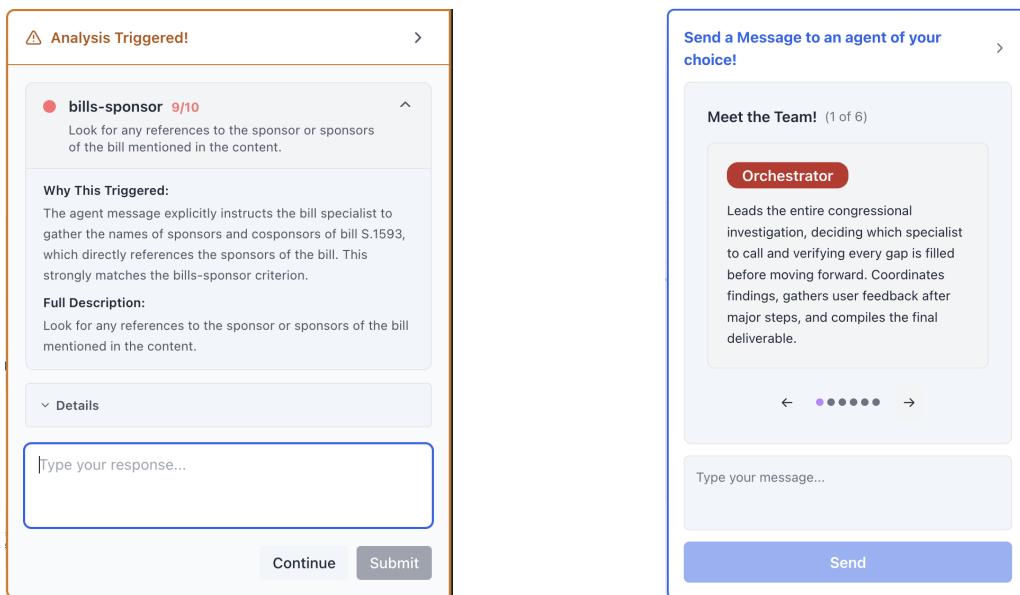


Figure 7.1: Comparison of the agent-initiated feedback modal to the human-initiated interruption modal. Both appear at the same position in the interface.

Despite its limitations, agent-initiated feedback was viewed positively in principle, users

anticipated its usefulness for longer runs where they couldn't monitor everything. The mechanism was seen as valuable for catching important moments that might otherwise be missed in the stream of agent activity. However, the gap between this conceptual appeal and the frustrating reality of unreliable triggers highlights the difficulty of designing effective notification systems for multi-agent collaboration.

## 7.2 RQ2: Agent Message Transparency

Participants demonstrated limited engagement with the comprehensive agent message transparency features, revealing a critical disconnect between what users wanted to see and what the system displayed. The run state information presented alongside agent messages failed to provide clear value to users—only one participant examined the run state tab, and they found it unhelpful because they couldn't understand its purpose or how it related to the actual agent message content. This suggests that transparency features require explicit design intentions and simple and concise contextual framing to be useful, rather than simply providing raw access to system state information.

Users expressed a clear preference for intermediate results over process-level transparency. The 15-word summaries proved too brief and were not positioned in sufficient visual proximity to their corresponding nodes for effective scanning. Participants consistently indicated they wanted to see more intermediate findings and conclusions from the agents' work, but were entirely uninterested in internal orchestration messages such as "Next we call the web\_search\_agent" or "Thank you for clarifying! We will wait for the planner agent to finish the new plan..." Several users noted confusion about which nodes would contain plans, which would show tool call results, and which would present intermediate conclusions. They indicated that more explicit categorization of message content would guide them more effectively in deciding which messages to inspect, rather than depending primarily on the feedback component scores—which reflected only the pre-anticipated factors they had defined at the outset.

The feedback component monitoring features also revealed usability challenges. Users found the bar charts below nodes unintuitive for conveying proximity to feedback thresholds, particularly because no threshold line was displayed for reference. Additionally, participants expected to click directly on these bar charts to inspect the underlying feedback components, rather than needing to click on the node itself—a design oversight that created friction in the exploration workflow.

## 7.3 RQ3: Feedback Component Design and Adaptation

A critical finding emerged around multiple layers of misalignment between user intentions, system capabilities, and feedback evaluation. Users constructed feedback components based on their mental models of what the system could do and detect, while the feedback evaluation system operated according to its own interpretation of both the multi-agent system's behavior and the user's intent. These misalignments manifested across three distinct layers: users' assumptions about agent capabilities, users' expectations of what the feedback system could monitor, and the evaluation system's actual assessment logic—creating systematic gaps between intended and actual system behavior.

### **Users gave feedback instructions based on what they thought the multi-agent system could do**

In the walkthrough, we by default omitted a detailed explanation of how the agent works in the backend, yet some of the participants asked more detailed questions, especially regarding the next-speaker selection. This is a crucial point that we omitted in the interface, namely, not giving the group chat manager a swimlane. Reasons for this include that we would have had to think over the entire design in order to include the group chat manager and its actions, the swimlane layout would have become useless since every message would have been routed through the group chat manager, which would have made every second "node" lay in the manager's swimlane.

Yet it happened with several users that they wanted to tell one agent to call another, both users that were explained the presence of a backend group chat manager and users that weren't. The most notable case was in the Deep Research example, where one user tried asking the "Planner" several times to do a web search with the "Web Search Agent" and then to tell the "Report Writer" to write a final report. Users consequently mistook the Planner agent's capabilities with the one from the group chat manager, and were confused when their orders were not followed and another agent was called instead.

### **Users formulated feedback prompts based on what they thought the feedback system could detect**

In the landing page view and the feedback component view, where users can construct their feedback components, many users wrote components that the underlying prompting for the feedback system misunderstood or misplaced in context. Such issues appeared in various forms:

**Negated Components** Components of the form "Do **not** trigger if X happens/shows" were logically expected to have low ratings if X de facto appeared in a message and high ratings if X did not appear. However, the prompting that generates the scores asks the LLM to compare the message with the (in this case negated) instruction and output a rating from 1 to 10 deciding how urgently the user needs to be involved. Consequently, whenever X appeared, the LLM output high scores and the agent-initiated feedback was triggered because the LLM implicitly saw "urgency to be involved" as presence detection and not absence detection.

**Phantom Components through Missing Context** Other components that users even tried even more frequently were components that the feedback system would have a hard time verifying. Two examples included "Trigger if the university is among the Top 100 ranked worldwide" and "Trigger if the university has a tuition fee of more than 10000 USD". The inherent problem of these components is their reliance on how much context the generated messages of the agents provide with their intermediate results, which depends on the main task prompt that the user inputs in the first place. We call them "Phantom Components through Missing Context", because these components would often times output low values consistently, since they had no material to make a definite claim on. In our examples, the user would have to specify that they wanted to list universities together with their exact ranks, or with their exact tuition fees, which then would lead to responses from the agents that could be meaningfully classified as matching the

Top 100 or the 10000 USD feedback component. However, participants were often surprised that no agent-initiated feedback was triggered, revealing a fundamental mismatch: users did not expect to have to anticipate the feedback system's evaluative needs when formulating the main task prompt.

**Question Components** One user in particular that used a lot of AI tools to code and write had the expectation that he could create components that would interrupt and prompt him direct questions, that the multi-agent system would need at a specific point of the ongoing research. This expectation came from a feature of OpenAI Deep Research, that after a query is input makes 3-4 additional questions to clarify the research scope before starting the deep research run (that can't be interrupted and takes 20 minutes). The user stated that this is a feature he dislikes about OpenAI Deep Research and loves the idea of being interrupted mid-run, but expected it to be in the form of specific questions that he would be asked to bridge a temporary knowledge gap of the agents.

**Mid-Run Adaptation** Additionally, users misunderstood during the run of the agent team, what exactly the feedback component system was able to do. This included the expectation, that users could remove or edit the behavior and frequency of components triggering during the agent team run. This inability to modify the feedback components during execution emerged as a major limitation. Multiple participants wanted to "change the feedback components" or "remake the feedback components mid-run" after understanding how they actually behaved. One participant articulated: "You understand how well your feedback component works mid-run, so it would be good to remove or edit them." Without this capability, users either restarted entire sessions or abandoned the feedback system entirely.

### **The feedback evaluation system assessed messages based on its own model of the multi-agent system, sometimes judging features**

The feedback system analyses each message of the agents and then gives it a score for each feedback component that decides if the run gets interrupted or not. Yet, agents' messages don't only display results, they also contain inner orchestration instructions e.g. "Call the Web Search Agent to execute the plan" or messages of acknowledgement of previous intermediate results ("Thank you for finding the committee roster Committee Specialist"). Although less often, some components that needed higher-order reasoning, e.g. "Trigger if a product of a company where X is CEO is triggered" saw scores above the threshold mistaking one of its own agents as part of the technology. Generally, components that needed multiple reasoning steps were the components that brought the users to frustration, because the reasoning they would trigger with would not make sense to them. Interestingly enough, users voiced their disinterest in inter-agent orchestration instructions highlighting the need to clearly separate them from the intermediate results.

## 7.4 RQ4: Domain Expertise and Interaction Patterns

Participants found the domain-specific Congress team's role structure significantly more intuitive than the general-purpose Deep Research team. In the Congress team, each agent possessed a distinct set of tool-calling capabilities tied to specific legislative data sources, making their roles more immediately comprehensible. Users reported knowing more specifically which agents to ask for particular types of information, and the notion of working with a "team" felt more natural and coherent. In contrast, the Deep Research team's more generic role definitions—where multiple agents could perform similar research and synthesis tasks—created ambiguity about which agent should handle which aspects of the inquiry.

Despite these structural differences, both user groups exhibited similar patterns of pushing beyond system capabilities. Participants in both conditions attempted requests that exceeded the agents' designed functionality: in the Congress team, one expert asked an agent to analyze how well-connected specific politicians were (a limitation of the available tool set), while Deep Research users asked whether agents could write executable code, trigger final reports on demand, or update feedback components mid-run. Both groups also showed comparable patterns of skipping a precise review of the generated feedback component, suggesting that engagement with agent-initiated feedback was influenced more by individual user preferences than by the specificity of the agent team.

A crucial finding emerged around how query specificity interacted with domain knowledge rather than agent team design. Even with the general-purpose Deep Research team, when students pursued queries within domains they knew well, their interaction patterns shifted dramatically. They formulated more detailed feedback prompts and read agent messages more thoroughly, mirroring the careful attention domain experts showed with the Congress team. Some participants noted they wouldn't need agent-initiated feedback at all in domains where they possessed expertise, stating that simply reading messages and interrupting when necessary would suffice. This suggests that user domain knowledge, rather than agent team specialization alone, drives the depth of engagement with transparency and feedback mechanisms.

Across both agent teams, certain agents became disproportionately prominent in participants' attention and mental models. The Web Search Agent drew focus as the only agent capable of executing search queries, making it a critical node in users' understanding of how information entered the system. The Planner agent was frequently mistaken for the group chat manager, with users attempting to direct it to orchestrate other agents—a confusion that highlights the challenge of communicating system architecture through agent roles alone. In the Congress team, the Committee Specialist received heightened attention due to the central importance of congressional committees in legislative processes, demonstrating how alignment between agent roles and domain-critical concepts enhances user engagement with specific team members.

## 7.5 SUS Results

To complement the qualitative observations, we administered the System Usability Scale (SUS) to quantify overall system usability. The assessment revealed an unexpected finding: AI tool proficiency was the primary predictor of usability success, superseding domain expertise. Par-

ticipants with advanced AI experience—those familiar with coding tools, commercial APIs, or frontier lab assistants—scored substantially higher than those with basic AI backgrounds, while domain experts showed no advantage over students. Most strikingly, when comparing the two Congressional research domain experts directly, the political scientist with advanced AI experience scored 20 points higher than the retired lobbyist with only regular AI proficiency, despite the latter’s decades of legislative expertise. This pattern suggests that multi-agent system interaction requires a specialized literacy that domain knowledge alone cannot provide.

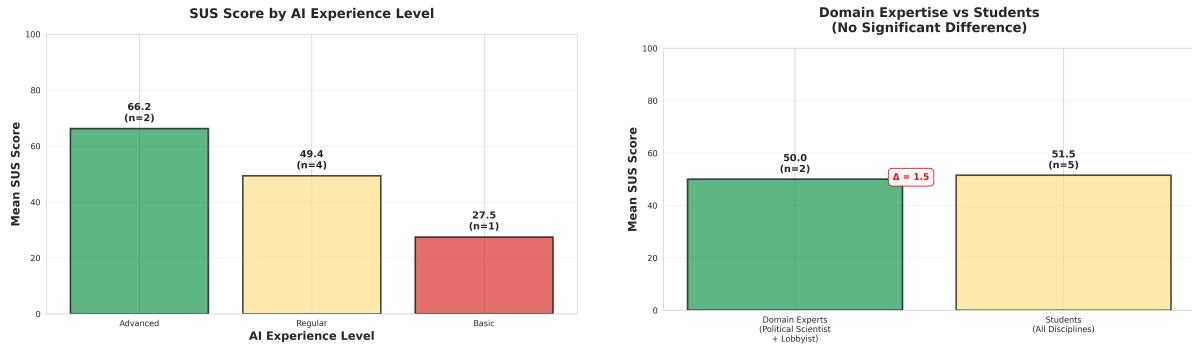


Figure 7.2: AI tool experience and domain expertise’s effects on the SUS score

Analysis of individual SUS questions revealed consistent patterns across user proficiency levels. Participants universally struggled with system learnability, rating “I needed to learn a lot of things before I could get going” at 3.14 out of 5, and found the system cumbersome to use (3.29 out of 5)—with even the highest-scoring participant acknowledging operational tedium. However, users consistently disagreed that the system was inconsistent (2.00 out of 5), indicating that despite perceived complexity, the multi-agent coordination maintained internal coherence. Notably, perceived complexity varied dramatically by AI experience level, with advanced users rating the system as straightforward while basic users found it unnecessarily complex, reinforcing that usability challenges stem from interaction literacy requirements rather than inherent design flaws.

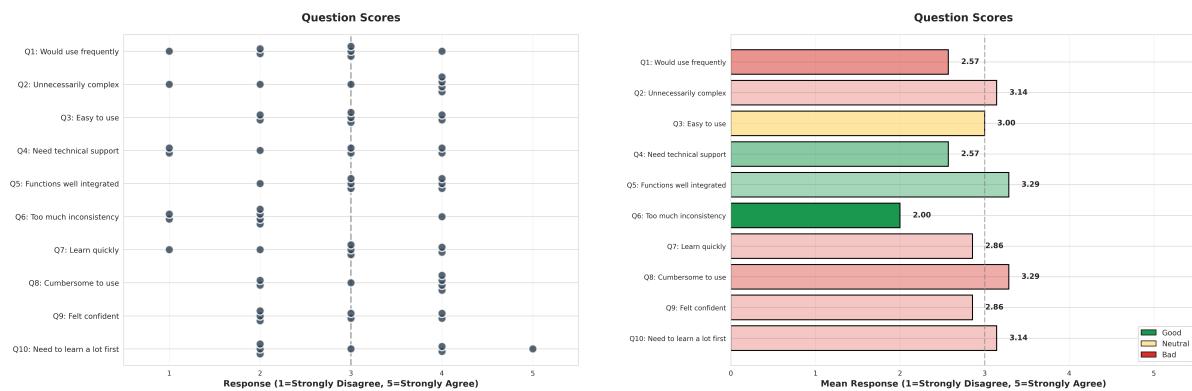


Figure 7.3: The individual answers and the mean value per answer to the SUS questions

# Conclusion

---

## 8.1 Limitations

Despite its contributions, the application is subject to several limitations. First, most of the backend functionality dictating the agent run is hidden from the user for complexity reasons, minimizing cognitive load but also limiting the correctness of a mental model a user might build up from the system. At the same time, the feedback component scoring mechanism is still done using a simple LLM prompt, that can't differentiate between the wide variety of components and wishes a user might design through open-ended description.

Moreover, our work includes little quantitative data to prove our findings, and we particularly rely on qualitative analysis from the user's interactions and feedback. Our data collection mechanisms don't allow us to make timed quantitative correlations between agent-initiated feedback and the possibility human-initiated interruption, nor makes the usage distribution of these mechanisms available.

## 8.2 Future Work

The work presented here opens up many questions that will be of great interest to the field of human-agent interaction as a whole.

As to the merging of human-initiated interruption and agent-initiated feedback, more studies have to be conducted that investigate the optimal visual separation of the two elements, such that the distinction of them is more intuitive for the human user. This includes the design, size and placement of the distinct modals, or a more fundamental rethinking of how we want users to be able to interrupt and be interrupted. As to the edge-clicking interrupt, more studies are required to detect for which specific tasks users will find interrupting at any point in the conversation useful and intuitive, since [14] found it to be a much desired feature for "agent debugging", but in our case this functionality was rarely touched upon.

As to the specific domains, we saw that the agents' capabilities in the Congressional agent team could be adapted much further by including data sources into the MCP server that we didn't include here. Committee hearing transcripts and committee reports are one of the most looked at documents when investigating specific lobbying cases [44] and the LobbyView database holds many more tables that, if connected to an MCP server, could give the agents access to factual industry connections between politicians and companies. More involved features would involve

including campaign financing data and election results, the former being an indicator which interest groups financed politicians and the election results indicating how safe a politician's standing is in his or her voting district.

Most importantly however, research is needed to test more sophisticated agent-initiated feedback mechanisms. Studying the separation of feedback components into intuitive types (Section 7.3), would allow fine-grained adaptations to the logic that triggers the feedback components of each type. One could have feedback components that look into inter-agent orchestration, components that look into intermediate results and feedback components that need to have their own reasoning chain and tools in order to trigger, or in other words make the feedback component system another asynchronous agent. Another crucial addition that needs to be tested is the ability to alter the feedback components mid-run, and in what form users will find it most usable: Is a visual drag-and-drop system of feedback components what users like most? Or the same open-ended feedback text box, whose content would be used to alter the feedback components too? Further work would consider how human users would interact with a more involved feedback system, which will become more important the longer agents can reason on their own.

### 8.3 Contributions

The two principal contributions of this thesis lies in the uncovering of the grounding mismatch between the human user, the multi-agent system and the feedback component system, and describing the qualitative interference of agent-initiated feedback in the ability of executing human-initiated interruption. Our system shows this consistently across two domains of different granularities, underscoring the universality of these issues.

Another HCI contribution we present in this work is the fact that designing an interface reuniting interruption and feedback in a visual manner is harder than expected, and its usability is highly dependent on the experience a human user has working with state-of-the-art AI tools. We conclude that the satisfaction of the design goals **G1** to **G6** is not sufficient to provide a good user experience for the average user.

Inadvertently, we also contribute to the field of Congressional lobbying, demonstrating how a team of multiple agents can reason about specific lobbying cases with access to factual information from public sources. We make a demonstrated case of the potential LLM agents have in advancing the field through their semantic text-processing capabilities.

# Bibliography

- [1] S. E. Brennan, “The grounding problem in conversations with and through computers,” in *Social and Cognitive Psychological Approaches to Interpersonal Communication*, S. R. Fussell and R. J. Kreuz, Eds. Mahwah, NJ: Lawrence Erlbaum Associates, 1998, pp. 201–225.
- [2] G. Bansal, S. Amershi, D. Weld, E. Kamar, S. K. Lahiri, S. Dutta, C.-Y. Li, and A. Awadallah, “Challenges in human-agent communication,” *arXiv preprint arXiv:2412.10380*, 2024.
- [3] Z. Chi, L. Dong, Q. Dong, Y. Hao, X. Wu, S. Huang, and F. Wei, “The era of agentic organization: Learning to organize with language models,” *arXiv preprint*, vol. abs/2510.26658, 2025, arXiv:2510.26658 [cs.AI]. [Online]. Available: <https://arxiv.org/abs/2510.26658>
- [4] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. Zaharia, J. E. Gonzalez, and I. Stoica, “Why do multi-agent llm systems fail?” *CoRR*, vol. abs/2503.13657, 2025, arXiv:2503.13657 [cs.AI]. [Online]. Available: <https://arxiv.org/abs/2503.13657>
- [5] V. Pimenova, S. Fakhoury, C. Bird, M.-A. Storey, and M. Endres, “Good vibrations? a qualitative study of co-creation, communication, flow, and trust in vibe coding,” *CoRR*, vol. abs/2509.12491, 2025, arXiv preprint. [Online]. Available: <https://arxiv.org/abs/2509.12491>
- [6] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, “The impact of ai on developer productivity: Evidence from github copilot,” *arXiv preprint arXiv:2302.06590*, 2023. [Online]. Available: <https://arxiv.org/abs/2302.06590>
- [7] J. Hadfield, B. Zhang, K. Lien, F. Scholz, J. Fox, and D. Ford. (2025, Jun. 13) How we built our multi-agent research system. Anthropic PBC. Blog post. [Online]. Available: <https://www.anthropic.com/engineering/multi-agent-research-system>
- [8] A. Team, “Introducing the model context protocol,” <https://www.anthropic.com/news/model-context-protocol>, Nov. 25 2024, blog post; open-source protocol announcement.
- [9] N. Krishnan, “Advancing multi-agent systems through model context protocol: Architecture, implementation, and applications,” *arXiv preprint*, May 2025, version 1, posted May 1, 2025. [Online]. Available: <https://arxiv.org/abs/2504.21030>
- [10] H. Zhu, P. Cuvin, X. Yu, C. K. Y. Yan, J. Zhang, and D. Yang, “Autolibra: Agent metric induction from open-ended human feedback,” *arXiv preprint arXiv:2505.02820*, 2025.
- [11] H. Yang, J. Zhang, Z. He, and Y. R. Fung, “Mars-sql: A multi-agent reinforcement learning framework for text-to-sql,” *arXiv preprint arXiv:2511.01008*, 2025.
- [12] S. Houde, K. Brimijoin, M. Muller, S. I. Ross, D. A. Silva Moran, G. E. Gonzalez, S. Kunde, M. A. Foreman, and J. D. Weisz, “Controlling ai agent participation in group conversations: A human-centered approach,” in *Proceedings of the ACM Conference on Intelligent User Interfaces (IUI '25)*, 2025, pp. 390–408. [Online]. Available: <https://doi.org/10.1145/3708359.3712089>

- [13] Q. Pan, Z. Ashktorab, M. Desmond, M. Santillán Cooper, J. Johnson, R. Nair, E. Daly, and W. Geyer, “Human-centered design recommendations for llm-as-a-judge,” in *Proceedings of the 1st Human-Centered Large Language Modeling Workshop*, 2024, pp. 16–29.
- [14] W. Epperson, G. Bansal, V. Dibia, A. Journey, J. Gerrits, E. Zhu, and S. Amershi, “Interactive debugging and steering of multi-agent ai systems,” 2025.
- [15] OpenAI. (2024) Introducing deep research. OpenAI. Accessed 30 November 2025. [Online]. Available: <https://openai.com/index/introducing-deep-research/>
- [16] Anthropic. (2024) Multi-agent research system. Anthropic. Accessed 30 Nov 2025. [Online]. Available: <https://www.anthropic.com/engineering/multi-agent-research-system>
- [17] I. S. Kim, S. Liao, and K. Imai, “Mapping political communities: A statistical analysis of lobbying networks in legislative politics,” MIT Department of Political Science, Tech. Rep., 2020. [Online]. Available: <http://web.mit.edu/insong/www/pdf/network.pdf>
- [18] J. Loffredo and S. Yun, “Agent-enhanced large language models for researching political institutions,” *Journal of Political Institutions and Political Economy*, vol. 6, 2025. [Online]. Available: <https://ssrn.com/abstract=5053570>
- [19] H. H. Clark and S. E. Brennan, “Grounding in communication,” in *Perspectives on Socially Shared Cognition*, L. B. Resnick, J. M. Levine, and S. D. Teasley, Eds. American Psychological Association, 1991, pp. 127–149.
- [20] J. C. Walliser, “Trust in automated systems: The effect of automation level on trust calibration,” Naval Postgraduate School, Monterey, California, Technical Report, June 2011.
- [21] L. C. Siebert, M. L. Lupetti, E. Aizenberg, N. Beckers, A. Zgonnikov, H. Veluwenkamp, D. Abbink, E. Giaccardi, G.-J. Houben, C. M. Jonker, J. van den Hoven, D. Forster, and R. L. Lagendijk, “Meaningful human control: actionable properties for ai system development,” *AI and Ethics*, 2022. [Online]. Available: <https://doi.org/10.1007/s43681-022-00167-3>
- [22] M. C. Horowitz and P. Scharre, “Meaningful human control in weapon systems: A primer,” Center for a New American Security (CNAS), Working Paper Working Paper 03-13-15, Mar 2015. [Online]. Available: [https://www.files.ethz.ch/isn/189786/Ethical\\_Autonomy\\_Working\\_Paper\\_031315.pdf](https://www.files.ethz.ch/isn/189786/Ethical_Autonomy_Working_Paper_031315.pdf)
- [23] Y. Dong, X. Jiang, J. Qian, T. Wang, K. Zhang, Z. Jin, and G. Li, “A survey on code generation with llm-based agents,” *arXiv preprint*, vol. arXiv:2508.00083v1, Aug 2025. [Online]. Available: <https://arxiv.org/abs/2508.00083v1>
- [24] S. Overflow, “2025 developer survey: Ai,” Stack Overflow, Tech. Rep., 2025, accessed on 2025-11-23. [Online]. Available: <https://survey.stackoverflow.co/2025/ai>
- [25] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474.
- [26] B. Peng, Y. Zhu, Y. Liu, X. Bo, H. Shi, C. Hong, Y. Zhang, and S. Tang, “Graph retrieval-augmented generation: A survey,” 2024, aCM journal article (peer-reviewed). [Online]. Available: <https://doi.org/10.1145/3777378>

- [27] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, “Improving factuality and reasoning in language models through multiagent debate,” in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 2024, pp. 11 733–11 763.
- [28] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Zou, “Mixture-of-agents enhances large language model capabilities,” in *International Conference on Learning Representations (ICLR)*, 2025, iCLR 2025 (spotlight). [Online]. Available: <https://openreview.net/forum?id=h0ZfDIRj7T>
- [29] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, “Metagpt: Meta programming for multi-agent collaborative framework,” *arXiv preprint*, vol. arXiv:2308.00352v7, Aug 2023, [cs.AI] — Submitted 1 Aug 2023, revised 1 Nov 2024. [Online]. Available: <https://arxiv.org/abs/2308.00352>
- [30] Q. Wu, Z. Zhang, Y. Zhang, W. Chen, Z. Zheng, X. Li *et al.*, “Autogen: Enabling next-gen llm applications via multi-agent conversation framework,” in *Conference on Language Modeling (COLM)*, 2024, cOLM 2024 (peer-reviewed). [Online]. Available: <https://openreview.net/forum?id=BAakY1hNKS>
- [31] Google, “Agent development kit (adk): A modular framework for ai agents,” online, 2025, available at <https://google.github.io/adk-docs/>. [Online]. Available: <https://google.github.io/adk-docs/>
- [32] LangChain, “Langgraph: Open source framework for stateful multi-agent systems,” online, 2025, available at <https://github.com/langchain-ai/langgraph>. [Online]. Available: <https://github.com/langchain-ai/langgraph>
- [33] Google PAIR, “Feedback and controls,” Guidebook, n.d., accessed 29 November 2025. [Online]. Available: <https://pair.withgoogle.com/guidebook/chapters/feedback-and-controls>
- [34] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, “Judging llm-as-a-judge with mt-bench and chatbot arena,” *arXiv preprint arXiv:2306.05685*, 2023, 37th Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks.
- [35] L. Dietz, O. Zendel, P. Bailey, C. L. A. Clarke, E. Cotterill, J. Dalton, F. Hasibi, M. Sanderson, and N. Craswell, “Principles and guidelines for the use of llm judges,” in *Proceedings of ICTIR 2025*, 2025.
- [36] C. Ryu, S. Lee, S. Pang, C. Choi, H. Choi, M. Min, and J.-y. Sohn, “Retrieval-based evaluation for llms: A case study in korean legal qa,” in *Proceedings of the Natural Legal Language Processing Workshop 2023*. Association for Computational Linguistics, December 2023, pp. 132–137.
- [37] S. Kumar, A. A. Nargund, and V. Sridhar, “Courteval: A courtroom-based multi-agent evaluation framework,” in *Findings of the Association for Computational Linguistics: ACL 2025*, 2025, pp. 25 875–25 887.
- [38] F. Yu, “When ais judge ais: The rise of agent-as-a-judge evaluation for llms,” *arXiv preprint arXiv:2508.02994*, 2025.

- [39] M. Zhuge, C. Zhao, D. R. Ashley, W. Wang, D. Khizbulin, Y. Xiong, Z. Liu, E. Chang, R. Krishnamoorthi, Y. Tian, Y. Shi, V. Chandra, and J. Schmidhuber, “Agent-as-a-judge: Evaluate agents with agents,” in *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. Vancouver, Canada: PMLR, 2025, p. 267, first four authors made core contributions. KAUST crafted the dataset. Work done while Mingchen was interning at Meta, with Changsheng leading.
- [40] X. H. Lù, A. Kazemnejad, N. Meade, A. Patel, D. Shin, A. Zambrano, K. Stańczak, P. Shaw, C. J. Pal, and S. Reddy, “Agentrewardbench: Evaluating automatic evaluations of web agent trajectories,” in *Conference on Language Modeling (COLM)*, 2025, cOLM 2025 (peer-reviewed). [Online]. Available: <https://openreview.net/forum?id=fQcUZMPIvu>
- [41] ag2ai, “build-with-ag2: Sample code and application showcases for building ai agent applications with ag2,” <https://github.com/ag2ai/build-with-ag2>, 2025, accessed: 2025-12-12.
- [42] ComposioHQ, “autogen-research-assistant: Starter repository for building agents with composio and autogen,” <https://github.com/ComposioHQ/autogen-research-assistant>, 2024, archived repository. Accessed: 2025-12-12.
- [43] GovTrack.us, “Bills statistics,” <https://www.govtrack.us/congress/bills/statistics>, accessed: 2025-12-03.
- [44] T. Power, “Interview with the author,” Personal interview, 2025, conducted in Washington, DC on 24 August 2025.
- [45] A. Cerdá and A. Daniller, “7 facts about americans’ views of money in politics,” <https://www.pewresearch.org/short-reads/2023/10/23/7-facts-about-americans-views-of-money-in-politics/>, 2023, accessed: 2025-12-03.
- [46] R. L. Hall and A. V. Deardorff, “Lobbying as legislative subsidy,” *American Political Science Review*, vol. 100, no. 1, pp. 69–84, 2006.
- [47] I. S. Kim, “Lobbyview: Firm-level lobbying congressional bills database,” Massachusetts Institute of Technology, Working Paper, 2018, dataset accessible via LobbyView data download portal. [Online]. Available: <http://web.mit.edu/insong/www/pdf/lobbyview.pdf>
- [48] OpenSecrets, “Opensecrets: Money in politics data archive,” <https://www.opensecrets.org/open-data/>, 2024, bulk data made available under Creative Commons for non-commercial use.
- [49] J. Brooke, “Sus: A “quick and dirty” usability scale,” Digital Equipment Corporation, Reading, UK, Tech. Rep. TR 26.2, 1986.
- [50] Library of Congress, “Congress.gov api,” <https://api.congress.gov/>, 2025, accessed: 2025-12-03.
- [51] A. Sundar, “congress\_gov\_mcp: Multiparty-Consent Platform Server Code,” [https://github.com/AshwinSundar/congress\\_gov\\_mcp](https://github.com/AshwinSundar/congress_gov_mcp), 2025, accessed: 2025-12-03.
- [52] unitedstates, “congress-legislators github repository,” <https://github.com/unitedstates/congress-legislators>, GitHub repository, accessed 3 December 2025.
- [53] Office of the Clerk, U.S. House of Representatives, “Roll call votes — votes,” <https://clerk.house.gov/Votes>, accessed: 2025-12-03.

- [54] J. Marsman, “Multiagentresearch: An exploration of using multiple agents collaborating to perform research,” <https://github.com/jennifermarsman/MultiagentResearch>, 2025, accessed: 2025-12-03.