Politecnico di Torino

Master's Degree in Computer Engineering

A.y. 2024/2025

Graduation Session December 2025

AgentExtension: A Three-Modality Framework for Human–Agent Web Interaction

Supervisors:
Luigi De Russis
Tommaso Calò

Candidate:
Endri Sefa

Table of Contents

List of Tables

List of Figures

Glossary

AI

Artificial Intelligence

GUI

Graphical User Interface

HCI

Human-Computer Interaction

HAI

Human-Agent Interaction

HAII

Human-AI Interaction

XAI

Explainable Artificial Intelligence

LLM

Large Language Models

GDPR

General Data Protection Regulation

SDK

Software Development Kit

API

Application Programming Interface

JSON	JavaScript Object Notation
DOM	Document Object Model
UI	User Interface
SEQ	Single Ease Question
SUS	System Usability scale

Chapter 1

Introduction

1.1 Context and Motivation

With the growing development and adoption of Artificial Intelligence in modern days, the number of people using these tools and the demand for newer ways to improve everyday tasks has increased in the last years. Web agents represent some of the latest systems developed to allow users to delegate actions to these tools, enabling them to perform tasks on their behalf across the web. Such tools are revolutionizing the way of interacting with the web by automating repetitive processes and allowing users to focus on other activities while the agent completes their requests. By simply describing to the agent the desired task, the agent will do the actions- such as performing clicks, filling forms and retrieving the required data- on the behalf of the user.
The field of Automation Agent and more specifically Web Agent, is still relatively new to the AI community, making it a stimulating and evolving area of research. Earlier this year OpenAI[1] introduced its newest agent "Operator", which enables web navigation through conversational input into its own browser. This July they fully released this with the name of "ChatGPT agent". Shortly after, Anthropic followed with the release of their own web agent through ClaudeAI[2]. Lately OpenAI introduced *Atlas*, a web assistant that can also perform actions on the web, working as an agent. For now it is only available for macOS [3]. This development demonstrates the engagement of big companies in such systems, but it also highlights how much this field is emerging in the market and yet, how current solutions are not affordable for all types of users.
Since this research area is still emerging, based on the novelty of the topic, the available literature remains fragmentary, and many assumptions have yet to be empirically verified through experimentation. From the new and broader field of Human-AI Interaction was born the subfield of Human-Agent Interaction. This

new research field has new constraints to be considered and most importantly, still needs to be discovered, mostly by testing this kind of system through user tests. Several studies have attempted to identify the main issues in this domain and associate to this ones some heuristics and guidelines to be respected, requiring also some knowledge from other fields to be accounted for, like for example psychology and ethics. Unlike other software, AI-powered systems, and so also Agents, must conform to some social norms and behaviors, which adds further complexity to the development and design of this kind of systems.
Another crucial aspect to consider is the security: developers must carefully define what and how the agent can do the actions that the user wants to delegate. This introduces additional constraints in the control field, another crucial point for the agent, since the user must be in control whenever critical actions need to be performed, particularly when critical operations are involved, such as financial transactions. This tension between autonomy and control represents one of the most challenging aspects of web agent design, making this a demanding and intellectually stimulating research area.

1.2 Objectives of the thesis

Although web agents remain an unfamiliar topic for many people, their advantages are concrete. With the help of agents, individuals who are not experienced with the web -or people that have physical difficulties operating with electronic systems- could complete complex tasks by providing a natural language prompt. The interaction paradigm with the agents is simple, yet effective, however it still requires significant refinement to accurately identify and address user needs. Current web agents are too limited in their functionality, which introduces potential risks for the user. The way the agent interacts with the web and their decision-making process is opaque and the user is left without any explanation on what the agent is doing. This lack of transparency increases uncertainty and reduces user trust.
Given these challenges, this thesis aims to develop a three-modality agent that, by learning from the user, can teach and collaborate with them to complete tasks on the web. This approach lets the user be in control of the capabilities of the agent, making its ecosystem grow with shared knowledge. The agent will be capable of sharing this acquired knowledge to other users to then teach or collaborate with them, thus enhancing overall usability and efficiency. Moreover, this thesis seeks to evaluate the current guidelines for agents design and development, with particular focus on trust, controllability and transparency, which are some focal points in the development of agentic figures.
In summary, the main objectives of this thesis are:

To design and implement three interconnected types of agent, each one with a different interaction modality:
a Teach modality, where the user instructs the agent how to perform actions on the web, allowing the system to increase its knowledge. A Learn modality, where the user learns tasks that the agent already knows how to perform. A Collaborate mode, where the user and the agent both work together to complete a task. These modalities are designed to be interconnected, allowing seamless transitions between them and creating a more immersive and continuous user experience.

To conduct a user study in order to evaluate the heuristics gathered: the heuristics are gathered from the existing literature and then adapted to these new interaction modalities that the agent will have. Participants will interact with the different modalities in order to validate the system, allowing them and the observer to identify new issues and opportunities within the system.

1.3 Structure of the thesis

The thesis structure follows the study of the main heuristics and issues related to agents followed by the design and development of the three web agents, each one with the different modality cited before. The work concludes with a user study aimed at gathering insights and evaluating the different agent approaches. Finally, the experimental data collected will be analyzed to draw conclusions for this thesis work. In detail, the chapters of this thesis are organized as follows:

- Chapter 2 "*Background and Main Issues for Agents*" : this chapter provides a brief overview of agents and examines the main issues that developers need to face in the creation of them, including technical issues such as biases of generative AI and ethical concerns. These topics are discussed by using real examples, analyzing the current state of Web Agents.

- Chapter 3 "*Design Guidelines*" : by studying the latest academic paperwork and existing heuristics, this chapter aims to create a table of guidelines for each one of the interaction modalities. Both theoretical and practical perspectives are considered to define the design principles of the agents. The work follows the Microsoft guidelines for the agent creation integrating each guideline with data from other research papers.

- Chapter 4 "*Prototype Design*" : Based on the analysis of the current Web agents, this chapter aims to describe the choices behind the type of agent and the considerations behind its design. After this it presents all the prototyping processes developed using Figma[4], detailing the choices behind each modality feature.

- Chapter 5 "*Development*" : starting from the architecture, this chapter describes all the development required for each component of the agent and the different ways each modality is implemented in order to provide a seamless experience for the user.
- Chapter 6 "*Model Performance Analysis*" : this chapter aims to analyze the models used to generate the answers leading to the final choices for the project. In detail, for each prompt, the model is analyzed considering performance, spendings, accuracy of the answer and token usage. The results are then compared to identify the optimal model for each prompt.
- Chapter 7 "*User Test*" : this chapter describes all the procedures followed to design and conduct the user tests. It details the experimental setup, participant selection criteria, tasks assigned to participants, and the methodology used for data collection and analysis. The results are then analyzed in relation to the design objectives.
- Chapter 8 "*Conclusion and final considerations*" : the final chapter draws the conclusions of this thesis work, ending with final considerations in view of future development ideas inspired by the insights obtained throughout the project.

Chapter 2

Background

In this chapter is introduced the topic of Web agent, providing a brief overview of what defines these systems, contextualizing their current state, and identifying the main factors contributing to their rapid growth. It also discusses the primary issues that currently affect agent design and deployment.

2.1 Introduction to Web Agents

On the 23rd of January 2025, OpenaAI [1] introduced its first web agent, *Operator*. This agentic system was capable of navigating the web autonomously using its own built-in browser. The navigation process is done by user prompts describing the desired task to be achieved. This was just a starting point in this year, demonstrating that major technology companies are now investing in agentic systems. Web agents, however, remain a relatively new area of research, whose development has accelerated significantly thanks to advancements in LLMs. Although existing literature on web agents is fragmented, early works, such as Pavón and Corchado's "*Agents for the Web*" (2004) [5] offer valuable insights. They describe the concept like this: "The notion of 'web agent', as the notion of agent, is rather fuzzy and depends on the authors. [...]. As a starting point, we can consider that the term 'web agent' which refers to an autonomous entity, with processing capabilities, and supporting web services." They later explain that by "supporting," they refer to enhancing the functionality of existing web services, for example by enabling personalization. This idea of automating user tasks to provide a more personalized browsing experience can also be found in Luke et al.'s 1997 publication, "*Ontology-Based Web Agents*" [6], which proposed an HTML extension for defining ontologies to facilitate faster and more efficient web navigation.
Although these studies were published nearly two decades ago, they already provided some working examples of web agents, indicating that research in this area has

long-standing foundations. Considering the constant growth of people navigating the web, thanks also to the always growing number of devices connected to the internet, combined with the increasing interest from both developer and end-users in AI, agents have lately become a more discussed and studied topic.
At the beginning of this year, Yujia Qin et al. released *UI-TARS* [7], a native GUI agent model that is capable of interacting seamlessly with GUIs. This system employs a Visual Language model, taking screenshots of the environment and determine the necessary actions to complete user requests. Using UI-TARS as a model for web navigation, *Midscene.js* [8], which is an open-source project, allows the user to enhance their browsing experience providing this agent as a Google Extension. The interface mimics a chatbot, allowing users to input text-based queries that the agent interprets and executes by analyzing screenshots of the current tab. These screenshots are then sent to the LLM model used in order to train it for the future interactions
Later, at the end of September of this year, OpenAI released the ChatGPT Agent (previously known as Operator) to Plus plan users. This release further demonstrates the growing public and industrial interest in web agents. However, it also highlights the urgent need for clear guidelines and frameworks for developing such systems.

2.2 Main Issues with current Agent Design

The novelty of this topic, combined with the increasing integration of these kind of systems with AI in the latest years, introduces a series of challenges that developers must address while designing web agents. These developments have also given rise to two emerging research domains within Human-Computer Interaction: Human-Agent Interaction and Human-AI Interaction, which are deeply interconnected in this context. In "*Challenges in Human-Agent Communication*", Bansal et al. [9] discuss the key questions that developers must consider while developing agentic systems. One of the central challenges involves establishing a shared knowledge between both agent and user. Is fundamental for the agent, in this cases, to adapt their general knowledge to what are the user necessities and goals.
When this common knowledge is not reached, it can undermine user trust, that is a crucial dimension for users to choose whether or not to rely on a specific agent. As explained by Baker et al. in "Human-Agent Teaming" [10] trust and reliability are closely linked to each other.Moreover, when using multiple agents, the presence of systems with low reliability, can also lower the trust of systems that are considered reliable by the user. Another important point for Baker et al., is the context where the agent is used: in critical situations, errors have a more significant impact that errors caused by other types of agents, so the system needs

to be designed accordingly in order for the agent to feel reliable and trustful to the user.
Another major challenge is explainability. Agents must provide reasoning for their actions and decisions with an appropriate level of detail. Developers need to understand their target user base and ensure that explanations are both accessible and consistent across interactions. Furthermore, given the amount of different types of interactions that agents con use, developers have to choose the most efficient one. The answer given, then, has to be consistent through different calls, trying to adapt their general mental models to the variety of users.
Security and privacy while navigating represent additional concern. Since Web Agents often have total control over users' browser, they pose risks of executing unintended or harmful actions. Having the control over the personal browser also raises privacy concerns, since most of the agents rely on screenshots to understand what actions to perform on the interface. Additionally is important for the agent to remember the previous interactions of the user with both agent and websites. In this case, determining what data to store and for how long remains a key issue, particularly when sensitive or personal information may be involved.
Finally, users consistently demand for transparency. As Bansal et al.[9] note, transparency enables users to evaluate the correctness, robustness, and usefulness of these systems. However, achieving this transparency adds complexity to development, as it requires integrating explainability metrics derived from XAI research.

2.3 Summary of Insights

In "*Agentic Web: Weaving the Next Web with AI Agents*" [11], Yang et al. explain that the emergency of AI Agents is causing a shift in what was before the web navigation. This new way of navigating the web, the *Agentic web*, has started this year, evolving from the previous version, which was the *mobile web.* They give this definition, citing the page 4 of the paper, "The Agentic Web is a distributed, interactive internet ecosystem in which autonomous software agents, often powered by large language models, act as autonomous intermediaries that persistently plan, coordinate, and execute goal-directed tasks. In this paradigm, web resources and services are agent-accessible, enabling continuous agent-to-agent interaction, dynamic information exchange, and value creation alongside traditional human-web interactions.".
This work just describes the paradigm shift that is happening now in web navigation. However, other studies highlight the potential risks associated with agentic systems. Zhang et al., in "*Characterizing Unintended Consequences in Human-GUI Agent Collaboration for Web Browsing*" [12], present a table with solutions for different

types of Unintended Consequences that happen while using agentic figures: for unintended consequences they refer to unexpected and negative outcomes from the interactions with Web Agents. Even thought agents provide some mitigation strategies to this scenarios, in the conclusion they suggest to rethink human-GUI agent Interaction. This just proves that these kind of systems still need more work and refinement.
This is extremely solidified also by the way agents fall for Dark patterns as explained by Tang et al. [13]. Dark patterns are deceptive design patterns that hinders the ability of users to make rational choices. After testing some web agents with 16 Dark patterns in a testing environment, they noticed that web agents are susceptible to this kind of designs.
In conclusion, despite the numerous risks and open challenges across multiple domains, the development of web agents remains essential for the evolution of the web, that is now shifting from a generative AI-driven model toward an agentic-AI one. Realizing this transition will require further research to ensure that these systems are reliable, transparent, and accessible to all types of users.

Chapter 3

Design Guidelines

The goal of this chapter is, starting from the current literature, to analyze the general design principles that apply to agentic systems and to evaluate the main challenges within Human–Agent Interaction. The discussion is inspired by several influential works in the field. The chapter is then concluded with a brief analysis of the table of Guidelines provided by Microsoft for AI infused systems in 2019, which will later be adapted for our project in the following chapter.

3.1 General Design Principles for Web Agents

The rapid growth in the use of AI technologies has created a clear need for developers to rely on structured design principles when building web agents. Weisz et al. in "*Design Principles for Generative AI Applications*", [14] provide a comprehensive set of principles and strategies aimed at supporting the design process.
The main principles are the following:

- **Design Responsibly** - The system should solve real user issues. To do so the design should be user-centered and should minimize user harms.

- **Design for Mental Models** - The AI system should share the same mental model with the users, teaching them also how to appropriately use the system.

- **Design for appropriate Trust & Reliance** - Developers should calibrate the user trust in the agent, explaining its role and also by letting them know the issues that can happen with the outputs. In this way developers can avoid overreliance on the system by the users.

- **Design for Generative Variability** - Users must be supported in managing the natural variability of generative AI outputs.

- **Design for Co-Creation** - Users should be able to meaningfully influence outputs and collaborate with the system in order to craft the desired answer.
- **Design for imperfection** - The system should help users understand that outputs may deviate from expectations and offer mechanisms for feedback, correction. This allows the system to improve the outputs for the future interactions.

These principles can be implemented through design decisions or by integrating specific functionalities or features into the system. Overall, this framework provides a good starting base for developers in order to assess what is needed for the AI system from the start of the design process.
Nazli Cila in "*Designing Human-Agent Collaborations: Commitment, responsiveness, and support*" [15], similarly , proposes a set of "design considerations", that are a list of questions framed as guiding considerations. The questions span these collaboration qualities: code of conduct, task delegation, autonomy and control, intelligibility, common ground, agents offering help and agent requesting help. For each aspect, there is a set of considerations that are questions that can help design a better agent. In the paper there are also shown some starting points for designer that can help developing a web agent answering these questions.
Together, the contributions of Cila and Weisz et al. illustrate how the research community is actively working to define the challenges faced by developers of AI-infused systems and to provide early conceptual tools for addressing them. These principles serve as an initial foundation for creating systems that respond to users' needs and mitigate common issues encountered during agent interaction.

3.2 Current challenges in Human-Agent Interaction

The previous chapter introduced the main issues affecting web agents. As noted, given the complexity of these systems, the problems that appear need work from interdisciplinary fields in order to ensure a good quality service.
In "Challenges in Human-Agent Communication" [9], Bansal et al. identify key communication challenges that arise between humans and agents. Figure 3.1 presents a set of questions grouped into three broad categories: general communication issues, challenges related to conveying information from users to agents, and challenges related to conveying information from agents to users. These challenges may emerge before, during, or after task execution.

For the first group, "*General human-agent communication challenges*", these challenges relate to establishing a shared understanding between the agent and the user. Insights from XAI research suggest that agents must make their behavior

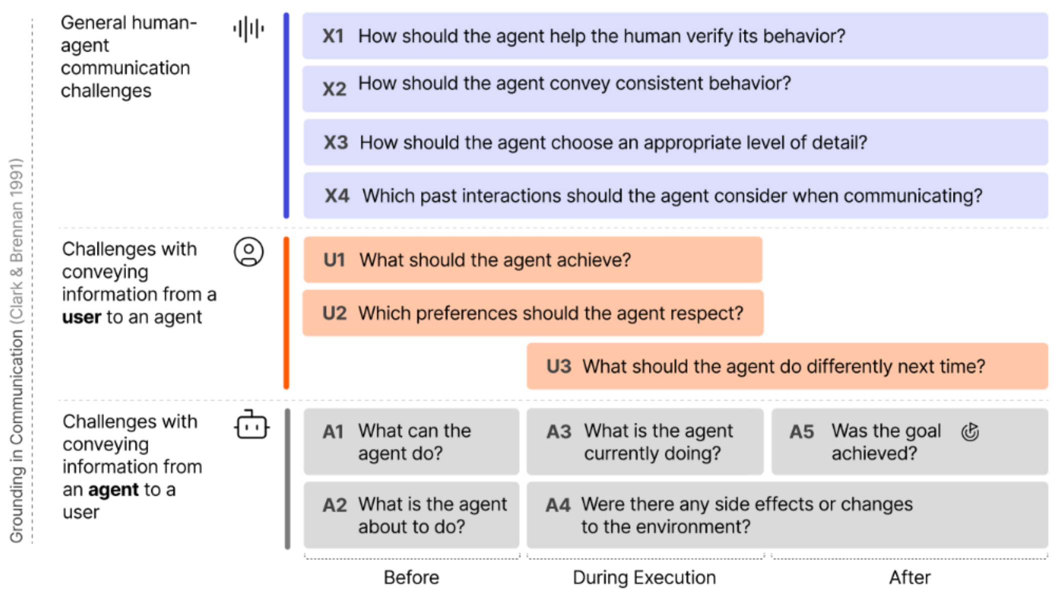

Grounding in Communication (Clark & Brennan 1991)
General human-agent communication challenges
X1 How should the agent help the human verify its behavior?
X2 How should the agent convey consistent behavior?
X3 How should the agent choose an appropriate level of detail?
X4 Which past interactions should the agent consider when communicating?
Challenges with conveying information from a **user** to an agent
U1 What should the agent achieve?
U2 Which preferences should the agent respect?
U3 What should the agent do differently next time?
Challenges with conveying information from an **agent** to a user
A1 What can the agent do?
A3 What is the agent currently doing?
A5 Was the goal achieved?
A2 What is the agent about to do?
A4 Were there any side effects or changes to the environment?
Before
During Execution
After

Figure 3.1: Challenges for Human-Agent Communication from Bansal et al. article "Challenges in Human-Agent Communication "[9]

transparent and comprehensible. Another important aspect is for the agent to leverage knowledge from previous interactions to complete tasks asking just for the needed information.
The second group, "*Challenges with conveying information from a **user** to an agent*", has as its concern in allowing developers to design agents that enable the user to provide the needed information to the agent. The first two are about the user's desire, so are about trying to disambiguate what is the user need from natural language, that introduces ambiguity and imprecision. In addition, understanding the user preferences is fundamental for creating a common ground between user and agent. The final challenge in this group concerns feedback: systems should allow users to refine or correct the agent's output in order to improve it for the following interactions.
The last group, *Challenges with conveying information from an **agent** to a user*", focuses on making the user understand the agent's capabilities, actions, if the goal was achieved or if any side effect occurred. Bansal et al. emphasize lightweight explanation mechanisms rather than extensive documentation, as the latter can overwhelm or confuse users. The documentation, that anyway is important, can be checked outside the scope of understanding the system's capabilities. In addition, special attention is required when the agent performs irreversible actions; in such cases, explicit user confirmation is essential.

Vera Liao et al., in "*Questioning the AI: Informing Design Practices for Explainable AI User Experience*" [16], complement this perspective by proposing a structured set of questions that systems should answer to support user understanding and transparency of it to the users. As discussed earlier, XAI remains one of the central challenges for web agents, and such frameworks can help developers identify what must be explained for users to interact safely and effectively with AI-powered systems.
These provided are just a subset of all the challenges that developers need to face while developing AI powered Agents. Nonetheless, these works provide designers with valuable tools to address these issues from the earliest stages of development, underlining the importance, as noted in the previous subchapter by Weisz et al., to adopt a human-centered approach into the design process.

3.3 Microsoft Guidelines for Human-AI Interaction

In 2019, Amershi et al. presented at the CHI convention "*Guidelines for Human-AI Interaction*" [17]. With over 2400 citations to date, this paper is considered foundational in the design of AI infused systems. It introduces 18 design guidelines that can generically be applied to any AI infused system, and so can also be adapted for web agents. For this work they tested an initial set of 20 guidelines using different systems; the testers were recruited from HCI and design distribution lists at a large software company. After refinements, they produced the final list of 18 guidelines that can be used to evaluate existing products or to help develop new ideas. Although 6 years have passed since this publication, their relevance remains strong. In this subchapter are going to be analyzed the main ideas, without going into detail; the next chapter will have the adapted version used to guide the design of this thesis project.
As done by Bansal et al., they split the guidelines into groups depending on the time that the guideline needs to be applied:

- **Initially** – in this category there are only two guidelines focused on explainability. Users should understand the system's capabilities and its level of accuracy before interacting with it.

- **During Interaction** – for this category there are 4 guidelines. 2 guidelines are about showing only the relevant information to the user, in order to not increase their cognitive load and making the system understand when to act or interrupt, based both on the environment and the task needed to be performed. The other two are about respecting social norms, adjusting the experience on the context of the user and limiting social biases that usually happen within generative AI.

- **When Wrong** – five are the guidelines in this field. Three are about making the system invocation, dismissal and correction efficient, allowing the user to perform easily these actions. One is about disambiguating or degrading the system's capabilities when is on doubt and the last one is about explainability, letting the user know why the system behaved like that.
- **Over time** – the last seven guidelines are about usage over time of the system. The system should learn from the user interactions and remember them, making easy for the end-user to make easy reference to that. The system should also provide global controls in order to personalize the experience within the agent. The system should allow the user to provide granular feedback. In addition the system also should understand when the users' actions can have consequences in future interactions with the AI infused system and notify that. To conclude the system should also update and change cautiously, limiting disruptive changes in its behavior, and, when doing so, it should also notify the user about those updates and changes.

In conclusion, Amershi et al. synthesized over two decades of best practices in the field of HAII into 18 generic guidelines. As stated in the paper, they wanted to provide a mix between generic and specialized guidelines, understanding that not all of the issues that designers encounter are faced in this work. Even though not all guidelines are applicable to all AI infused system, this work is still an important starting point for all developers that are going to integrate AI in their projects.

Chapter 4

Prototype Design

This chapter presents the process behind the design of the final project. Starting from the work carried out in the previous chapter, it introduces the reworked table of guidelines that is then used to support the design process. After analyzing the current Web Agents, their designs and their main features, the chapter explains the choice for the deployment platform. Finally, it illustrates and describes the main features of the prototypes developed using Figma [4].

4.1 Guidelines Developed for the Project

In the previous chapter were briefly explained the guidelines provided by Amershi et al. from "Guidelines for Human-AI Interaction" [17]. From the beginning of this project, their table was taken as a starting point to reason about solutions for this thesis. Starting from these guidelines, the design process involved contextualizing each guideline for the different agent modalities that formed the initial conceptual basis of this project. Once this contextualization was completed, the process concluded with a set of concrete implementation examples that could later be used as references during the design and development phases.
Table 4.1 shows the final result obtained after several iterations and refinements.

Guideline	**Web-Agent Contextualization**	**Implementation Example**
G1: Make clear what the system can do. Help the user understand what the AI system is capable of doing.	**Teacher:** The system, acting as an instructional figure, should communicate that it can highlight key concepts, provide step-by-step explanations, and illustrate complex ideas with visual cues. Users should understand that the teacher-agent can adapt teaching methods based on the user's current knowledge and offer multiple forms of guidance (textual, visual, interactive). **Student:** When acting as a learner, the system should convey that it can ask clarifying questions, request examples from the user, and signal when it needs more information to improve its understanding. The user should realize that the student-agent's role is to absorb input, refine its comprehension, and become a more effective collaborator over time. **Companion:** In a supportive capacity, the system should clarify that it can provide gentle reminders, suggest relevant resources, offer summaries of past sessions, and organize the user's learning trajectory. Users should know that the companion-agent is there to assist, guide, and streamline their journey rather than lead it directly. For the end users *Bansal et al.* suggest to use light weight solutions to introduce capabilities and limitations that are integrated into an agent's interface or behavior instead of providing the documentation, that might be too long and verbose, with a level of detail that a normal user does not need.	**Teacher:** Include a "What I Can Do" info page accessible via a help icon. This page lists teaching methods—highlighting text, providing custom color-coded annotations, presenting step-by-step instructions, and linking to external references. Include brief explanatory tooltips or short demos (e.g., GIFs) to show each capability in action. **Student:** Present a short introductory panel upon first interaction, explaining that the agent can learn from user input. For example, a pop-up might say: "I can learn from your explanations, ask for more details if I'm confused, and provide worked examples as I understand your style." Brief clickable examples ("Ask me about a topic, and I'll try to clarify!") illustrate these capabilities. **Companion:** Offer a collapsible sidebar labeled "How I Can Help." Expanding it reveals bullet points: "I can set reminders, suggest further reading, summarize last week's progress, or highlight key concepts you might have missed." Next to each bullet, an icon or small animation shows how the feature works, ensuring the user grasps the companion-agent's supportive role.

Continued on next page

Guideline	**Web-Agent Contextualization**	**Implementation Example**
G2: Make clear how well the system can do what it can do. Help the user understand how often the AI system may make mistakes.	**Teacher:** The teaching-oriented agent should communicate its level of certainty when providing explanations or solutions. It can indicate when it is confident, when it is making a guess, or when it might need the user to confirm information. This ensures the user understands that while the agent is knowledgeable, its suggestions are not guaranteed correct and may need double-checking. **Student:** In a learning role, the system should show when it is unsure about its interpretation of the user's input. If it cannot fully understand a concept or is not certain about the next step, it should signal uncertainty, prompting the user to clarify. This helps set realistic expectations that the student-agent may misunderstand or ask for verification. **Companion:** A supportive companion agent should regularly acknowledge when it might be mistaken or lacks full confidence. For example, it can say, "I'm not entirely sure about this recommendation," or "You may want to verify this source," encouraging the user to treat suggestions as starting points rather than definitive truths.	**Teacher:** Include a small confidence indicator (e.g., a subtle colored icon or bar) alongside explanations. For example, after giving an answer, a tooltip might appear: "Confidence: Medium—Consider verifying with a trusted reference." If the teacher-agent is uncertain, it might say, "I'm not 100% sure. Would you like to check a reference resource?" **Student:** Introduce uncertainty messages in the student-agent's interactions. When the agent is unsure about the user's meaning, it could display a brief message: "I'm not completely sure I understand. Could you explain further?" or "I might have misunderstood; let's break it down together." This invites the user to collaborate and refine the agent's understanding. **Companion:** Provide disclaimers when offering suggestions. For example, when the companion recommends reading material, it might say: "This article might help, but I'm not fully certain it is the best fit. Let me know if you need something different." Include an option for users to mark feedback on suggestions ("Was this helpful?"), gradually improving the companion's accuracy and ensuring transparency about its fallibility.

Continued on next page

Guideline	**Web-Agent Contextualization**	**Implementation Example**
G3: Time services based on context. Time when to act or interrupt based on the user's current task and environment	**Teacher:** The agent when behaving as a teacher should know when to act in order to provide useful information to the user as a way for them to complete their tasks and should stop when the user does not need its help, to not provide useless tips that might increase the time spent by the user learning. In this case the agent should be capable of understanding what are the needs for them in order to comprehend and complete a task. **Student:** The student should act when triggered by an input of the user. The student will start doing the actions that are requested and the user will check if it is correct. If the student is wrong the user will stop the execution and after that the system will ask for clarifications to disambiguate with a more detailed explanation. In case even this is not enough the student will ask the user to show to it the actions that are required to get through that part. In this case the agent will be a passive listener until the user completes the task. If the user wants they can then check if the system has learned that task by doing a repetition. **Companion:** As a companion it should act when the user is performing recurrent actions or when the knowledge of the agent is enough, providing useful information to speed their task completion leveraging their understanding based on the previous interaction as said by *Bansal et al. [9]*. It should also act when the user is doing something unexpected or that the companion has not adequate information, asking the user to better understand what the user is doing to improve the future experience and learn their preferences. It should stop when its help is not needed or when the user says so, limiting the agent capabilities.	**Teacher:** The teacher, depending on the level of explainability that the user needs, can change the type of interaction. When the level is high it can show everything in detail with texts, audio, video, etc. by also doing the actions that the user wants to learn, instead if the level is low it can teach the user by giving only concise text explanations and some graphical hints. When noticing that the user is going out of the scope of the learning the system can show little pop-up messages or hints like "This is what you need to do in this moment" or "This is what you are searching for", doing so using the best means of communication (graphical, textual, audio, video, etc.). Instead when the user is doing the tasks correctly it can give them some type of feedback to keep them engaged in this process and letting them know that the system is checking on their actions. **Student:** After inserting the task, the system will start doing it showing the various actions in the website to the user through highlights (using graphical features to show the actions that the agent is doing). If wrong, the user can block the agent that will ask a follow up question, for example "What was I doing wrong?" or "What step did I do wrong?". If the end-user explains the mistake, if the knowledge of the agent is enough to understand what was wrong and how to correct it, the student-modality will resume their interactions and complete it with this new added information. If not the student will ask the user to show it and after that the system will be passively looking at the user's interactions to learn from them. **Companion:** The companion agent should stop the execution when doing irreversible actions that must need the approval of the user (i.e. buying some tickets for a flight). Also it can stop when noticing that the user is going out of reach of the current task, asking for clarifications with questions like "I noticed this . . . Are you done with ..?". The companion should act when the user allows its actions and does not stop that execution, usually for easily automated tasks that do not need particular attention and are reversible (i.e. fill the fields of a form).

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G4: Show contextually relevant information. Display information relevant to the user's current task and environment.	**Teacher:** When behaving as a teacher the agent should provide only the right amount of information needed to the user to understand how to complete a certain task based on his level of knowledge and experience. This knowledge must be gathered by previous interactions of different users and also by the amount of knowledge gathered by the user during their interaction with the website. The interface should be minimal showing only what is needed. **Student:** The student agent should display only the information useful for the user to follow the agent interactions on the screen, without having to see all of its "mental processes". When in doubt, the system should provide only the right amount of information for the user to teach the needed things to the agent, excluding extra information that can be redundant making the interaction more natural and not invasive. **Companion:** Acting as a Companion the agent should understand from the context and the previous interactions with the user what the needed information is during a certain task execution and how the user prefers it to be shown. In any case it should not block the user from accessing relevant information, instead it is necessary for it to support them during their process making the carrying off of the task easier and efficient. The companion should inform the user when they are going out of the scope of their task, without being too invasive, informing them gently . When doing actions it should show the user what they want to know to keep track of the agent's behavior.	**Teacher:** The teacher agent should interact providing the right amount of information and through the right type of communication (video, text, audio, etc.). It will show messages like "Now you need to . . . " through popups, in order to guide the user throughout the learning process. It will draw attention to specific elements for example with circles or by highlighting objects or icons that the user needs to interact with, making everything clear. **Student:** The use of highlights to show the agent interactions will be very important. The student agent will ask only questions when in doubt through the agent's interface. The agent will be capable of showing the actions that it is doing in the website through the use of graphical features to call attention to (highlighted mouse pointer i.e.). These things will help the user see what is important, to understand the agent's behavior, focusing only on what is doing and not on how the system is working. **Companion:** The companion will show only the suggested actions and a button that can be clicked to open the complete interface, only if the user needs extra information. The interface will provide only the best suggestions like "Do you need me to . . . ?". When in doubt the agent will ask questions to disambiguate like "Are you sure this is what you are searching for? Do you not need to . . . ?". When doing tasks based on the level of proactivity that the user allows it can show through accentuate the actions that the companion is taking and how much time it will take for it to complete them.

Continued on next page

Guideline	**Web-Agent Contextualization**	**Implementation Example**
G5: Match relevant social norms. Ensure the experience is delivered in a way that users would expect, given their social and cultural context.	**Teacher:** Based on the target user, the agent should express itself using understandable and suitable vocabulary. The interactions with the teacher should help the user understand a new task based on the complexity of the task itself and the capabilities of the them. The agent should make for the user comprehensible explanations to understand what it is doing, as said by *Baker et al.* [10] and at the same time the agent should be capable of understanding what the user needs. In this case it gets more attention since people tend to be more critical of agent supervisors as stated by *Baker et al.*, so it needs more attention while developing this component in order to not lose trust by the user. **Student:** When behaving as a student, the agent should not use terms that go beyond the comprehension of the user or that might be confusing. The agent should use a human-like language to show its own doubts to the user and should be capable of understanding what they are saying when helping them out. Also when doing the actions the behavior should be human-like, so that a human can follow that, allowing the user to understand what the agent is doing based on the agent's means of communication. **Companion:** As a companion the agent should behave in a peer-to-peer communication, understanding what is the context and using the information gathered before in order to deliver the preferred way to communicate with a user effectively. When stopping the user, the system should do politely and not in an abruptive and definitive way, allowing the user to resume where they left. The language used will be the one that the user uses to communicate with the website and the agent.	**Teacher:** The teacher agent should use simple and understandable words, in order to not limit the learning process only to people that are specific to the domain. After each interaction it can start using domain specific words that the agent is sure that they know based on the user's previous interactions. The teacher should start with the basic knowledge and after a few interactions it can assume the level of the user. In any case the critical/domain specific terms will always be highlighted so that the agent can give a brief or more detailed explanation by clicking on them, depending on the level of detail that the user wants. **Student:** The student agent should ask simple questions that can be understood by all kinds of users that are trying to teach the agent. In case of different mental models shared it can ask follow up questions to the user's answers like "When referring to . . . , do you intend this . . . ?" and adapt accordingly. As done by other means of teaching it should focus on what the agent is doing in order for them to understand what is happening in the interface. **Companion:** The companion agent should help the users in already done tasks so it should not exit this scope. By doing so it should use terms that are understandable to the user, and will do so based on the previous shared knowledge with it.

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G6: Mitigate social biases. Ensure the AI system's language and behaviors do not reinforce undesirable and unfair stereotypes and biases	**Teacher:** When doing its explanations the agent should use unbiased information gathered from previous user's interactions and also other users. Its language should be inclusive in a way that any user does not feel discouraged while using it and making them feel welcomed. **Student:** As a student the agent should refer to the user using their shared preferences in the configuration and not use the data recorded by previous interactions with other users that might be biased. **Companion:** The companion agent should refer to the user using their preferences and should not use other user's preferences for suggestions that might reinforce some undesired biased stereotypes; this behavior might hinder the trust of the specific users. We also need to consider in this case what *Baker et al.* [10] said : "Individuals and cultures are innately different from what they know about agents and how they interact with them". This means that the developers also need to know these cultural differences and individual differences and respect them in order to not undermine the users' trust in the agent.	**Teacher:** When interacting for the first time, the agent can be configured by the user for the different ways of addressing them (pronouns, age, etc. . .) and should start using simple terms allowing the user to understand everything, adapting the level of detail based on the user's knowledge. With all this information the agent should calibrate for the next ones, allowing the user feedback to recalibrate. **Student:** After the configuration asking the preferences about pronouns, etc., the companion agent should use these terms while referring to the user. The data collected previously will not be used in case it reinforces stereotypes. **Companion:** After getting to know the user, the agent should not give suggestions about actions or ask questions to them that might be biased. The model should recognize these cases and should limit the agent's capabilities. This thing will be done by the model, yet the configuration phase to learn the user's basic preferences might be an important step to create a good relationship between the agent and the human. As said by *Baker et al.*, transparency or agent behaviors can affect human trust.

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G7: Support efficient invocation. Make it easy to invoke or request the AI system's services when needed.	**Teacher:** The teacher agent should be reachable by any user, making it easy to invoke also by new ones. From the past experience of the teacher, the system should be capable of understanding the needs of the user making the interaction with the website and the agent more enjoyable and efficient. **Student:** As a user is approaching the system for the first time it will be asked if they want to teach the system some tasks, in order for the agent to help them and other users too while using the web by growing the agent's knowledge. The agent will be invoked by using its own icon or interface and will be easy and reachable. **Companion:** When behaving as a companion, the agent should be easily available and reachable by the user, allowing them to get a first preview by looking at the suggested tasks to let the companion execute based on the context and on their previous interactions. When allowed by the user the system can recognize and stop the user from going beyond the reach of their current task.	**Teacher:** The agent might be invoked through an icon that by design choices can be put in the bottom left corner (respecting the Consistency and Standard Heuristic of *Nielsen* [18]) that might open the agent tab inside the same page or can be a Browser extension that can be invoked from the extensions bar. After that the interface should be simple and standard (for this type of agent, using an input for the text of the user i.e.) and using ordinary words and icons that can be recognized by every user. **Student:** The agent might be invoked through an agent interface (or an icon/extension i.e.) that the user can employ to teach the agent specific actions. The interface should be simple and standard (for this type of agent, using an input for the text of the user i.e.) and using ordinary words and icons that can be recognized by every user. **Companion:** The agent might be invoked through an icon that by design choices can be put in the bottom left corner (respecting the Consistency and Standard Heuristic of *Nielsen*) that might open the agent tab inside the same page (or an icon/extension i.e.). After that the interface should be simple and standard (using an input for the text i.e.) and using standard words that can be recognized by every user. The interface should also provide some shortcuts for suggested tasks that can be triggered by clicking on them, allowing a more fast and enjoyable experience (*Nielsen's 7th Heuristic: Flexibility and Efficiency of Use*).

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G8: Support efficient dismissal. Make it easy to dismiss or ignore undesired AI system services.	**Teacher:** The user when interacting with the agent in this case should be capable of interrupting in any moment the communication with it without any risk or repercussion in their navigation of the website. The interface should provide a way to ignore certain services that the teacher offers during the communication with it. The agent should take note of this type of suggestions to improve user satisfaction and adapt their behavior accordingly. **Student:** When the interaction is stated, the user can dismiss the session started without any repercussion on their future interplay easily. In case of questions the user does not know the answer, they can stop at any moment deciding to provide or not any kind of explanation. The agent should not make the user feel bad for not knowing what they are explaining, knowing that also the psychological mechanisms are important on how the users interact with the agentic figures. Like *Cila et al.* [15] cite in their paper, the agent has endless patience; not taking the things personally is really important in this case, making the user feel less pressure. **Companion:** The user should be capable of dismissing certain services offered by the companion at any time. They should have the capability to adjust the level of proactivity offered by the agent from the companion interface, and it should learn from these types of interactions with the users in order to improve the level of satisfaction.	**Teacher:** The user in any moment should be capable of interrupting the interaction by clicking a specific icon that is universally recognized as dismiss, for example an "X" icon on the top right, or button in the chatbot after the input is sent. After that the user will be asked if they are sure that they want this through a dialog, respecting the 5th Jakob's Heuristic [18]. **Student:** After starting a teaching session to the agent, in any moment the user can stop (by clicking into an icon i.e.). The agent will ask them a follow up question asking if the user is sure and why they wanted to stop, receiving feedback for the user and to learn the user's preferences. The user is free to provide an explanation if they want. If they do so the agent will thank them for their feedback. **Companion:** Through the companion interface the user can stop at any moment what the agent is doing through the use of icons or through the command setting or even through the chat. This must be explained to the user by a guided video, or shown on the screen with a pop up during the first interaction, in order for them not to feel disoriented. If the agent asks questions about the user's interactions, they are free to not answer them, without any repercussions on the future communications with the agent.

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G9: Support efficient correction. Make it easy to edit, refine, or recover when the AI system is wrong.	**Teacher:** The teacher agent should have an option to be corrected and to recover from mistakes done. In this case the agent should understand its mistake from the user's feedback. In case its knowledge does not let the agent recover from its mistake the agent should notify the limits of its knowledge in order to not undermine the user's trust in the agent. **Student:** The interface should allow the user to refine the agent by making it share the same mental model with the user in order to better understand the actions that are being done on the website. When asked something by the agent, the user should be capable of editing their answer and correct the agent if they notice that the system is doing the task wrong. **Companion:** When behaving as a companion the interface should allow the user to edit the suggestions that the agent is making in order to make the system better understand the preferences about them. As doing so the agent might ask follow up questions on the changes in order to better grasp what is the user mental model and improve this. This behavior should also happen when the agent asks for clarifications.	**Teacher:** After giving a wrong input the user can use an icon, for example, to report an inconsistency or a mistake done by the agent in order for it to recover correctly from this (if its current knowledge allows it). The agent should give positive feedback to the user, should apologize and then should give a new and correct answer, with the degree of certainty, allowing the user to assess by themselves the agent's true capabilities. In case the input is wrong the user can make easy corrections about the previous interaction, without having to repeat all the process. **Student:** The agent will ask the user when unsure about their behavior in the website, and when wrong the user can correct that using the text input, that can be a chatbot, as done for the normal interaction. The user can stop the agent's action through an interface for example with feedback buttons or through the textbox. When submitting an answer the user can edit that afterwards if the response is not satisfying from both agent and user. In case the agent was stopped during the learning phase, through a repeat icon for example, it can do again the step to show to the user if the explanation given to it was helpful. **Companion:** This can be done through the use of an icon or a chat that the user can employ to correct the agent. The agent should answer positively to the user's correction and should assure them that their suggestion was helpful, providing a feedback, like "Thank you for correcting me, I will update your suggestions now". In case the actions done by the companion are irreversible the agent should notify the user before doing so. If the actions are reversible the user can go back, i.e. by using an icon. The same will happen when the agent will ask for explanation when notices unexpected behaviors of the user during a specific task.

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G10: Scope services when in doubt. Engage in disambiguation or gracefully degrade the AI system's services when uncertain about a user's goals.	**Teacher:** The teacher agent should understand when to degrade the services due to lack of information. In these cases should inform the user about how well the AI system performs different tasks by explaining its capabilities and limitations. **Student:** In this case, the student agent, when unsure, should try to disambiguate its own doubts, asking permission to ask more questions about the user's actions, in order to better understand their explanations/actions on the web. In case it notices that the user is going out of the scope of the teaching process the agent should limit its capabilities limiting eventual damage that a possible bad session might cause to the entire system. **Companion:** The companion, when unsure about the user's goals, should degrade its services in order to offer a better service and not overload the user with suggestions that are useless. The companion might ask questions about the user's preferences to better understand their goal and deliver a better service. Also, as said by *Baker et al.* [10], the agent should check for security when navigating, so it must degrade its services when behaving with unsafe websites or doing hazardous actions. *Weisz et al.* [14] in the section "Design for Imperfection" say to make uncertainty visible: caution the user that outputs may not align with their expectations.	**Teacher:** When unsure about the user's goal the system can ask follow up questions like "Sorry, I do not understand what you mean with ... Can you explain it again by changing some words? This can help me understand better what you need.". If this will not change anything, the agent will show a grade of uncertainty when teaching a user the task, remembering them about its own limitations and not to trust its output. **Student:** When unsure about the user's goal the student agent can ask follow up questions like "Sorry, I do not understand what you mean with ... Can you explain it again by changing some words? This can help me understand better what I'm going to learn.". If this will not change anything, the agent will show a grade of uncertainty when learning the task. The same will happen with the follow up questions in case of doubts with the task execution. **Companion:** In case the agent is unsure about what the user is doing can ask questions to better calibrate the user's preferences with questions like "What are you trying to do?" or "Are you trying to ... ?". When unsure about the preferences the agent can show a degree of precision near to the suggestion (with a percentage i.e.) to help the user understand if that might be needed or not without having to read all the suggested prompts.

Continued on next page

Guideline	**Web-Agent Contextualization**	**Implementation Example**
G11: Make clear why the system did what it did. Enable the user to access an explanation of why the AI system behaved as it did.	**Teacher:** The agent behaving as a teacher should explain his actions showing why it is describing these actions based on the amount of data collected and interactions showing a degree of confidence of why the actions performed are the right ones. The user should be capable in any case to ask this information to the agent, as also stated by *Weisz et al. in the section "Design for appropriate trust & reliance: Provide rationales for outputs" [14]*. **Student:** The system, when behaving as a student, should show what is doing and in case the user wants can also answer why, showing the degree of confidence based on the data available. The student agent, when asking questions about its own doubts to the user, can explain to them why it is doing so, showing its uncertainties, allowing the user to better understand what are the limitations that the student is facing. This will be done to try and give a better answer to make the agent learn what it needs to complete the task during the interaction with the website. **Companion:** The companion agent should allow the user to understand why the agent is giving certain suggestions to help them based on data with a certain degree of certainty from their interactions and other users with the same engagement in the website (respecting also the G6 from this table about the biased data that might be collected). The same will happen when the companion will ask questions about the present task when unsure about the user's behavior. The system usually struggles with answering the why not question as seen by *Vera Liao et al. in "Questioning the AI: Informing Design Practices for Explainable AI User Experiences" [16]*. They suggest the benefit of interactive explanations, allowing users to explicitly reference the contrastive outcome and asking follow-up "What if" questions. From *Bansal et al.* [9] can be seen in certain scenarios the importance of real-time updates from the agent, enabling the user to stay informed about ongoing actions.	**Teacher:** When asked about the result, the agent must respond with the information giving a degree of precision and showing the source that has generated the answer for the user, with a detailed description of all the actions that are required to complete a task, like "You need to do this ... in order to achieve ... ". The degree of certainty can be shown with a percentage or a colored bar to show the level of confidence of the agent for example. **Student:** When taking the actions the student can show messages, for example on a sidebar, describing what is doing (i.e. "Now I'm going to click on the login button to log in into my personal profile"). The student can explain why it asked specific questions giving the source of the previous data collected, for example with messages like "From previous user interactions, I saw that to do ... you need to ... ". **Companion:** The companion can explain why it asked specific questions when the interaction was not something expected from a specific user. When explaining a suggestion it can show why with a text like "I saw that you previously did ... so based on yours and other users interactions I think you might need ... ". When stopping them, the agent should explain that it found that the user's specific actions were going out of the scope of the task itself, asking in this case for clarification. If the user needs some explanation the agent might show the information that it was used to generate the answer.

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G12: Remember recent interactions. Maintain short term memory and allow the user to make efficient references to that memory.	**Teacher:** The teacher agent should keep memory of the previous interactions, making it easy for the user to see their history. From this knowledge the system should understand what could be the possible next requests and if the user needs to refresh what they have done, respecting Nielsen's 6th Heuristic about "Recognition rather than recall" [18]. **Student:** When performing as a student, the agent should keep track of the previous interactions in order to better understand the behavior of a user inside of the website. In this way it can learn the ways that a user interacts with the website, and also to seek for changes in their behavior and understand why. The user can also see what they did in the previous sessions with the agent to see if it needs some help on some specific tasks or steps. **Companion:** In this case the companion should remember the previous interaction of the user in order to offer better suggestions for its present tasks based on its knowledge about the user's preferences. This approach will also allow to see if the user needs some help to better learn how to complete a specific task. In other cases this knowledge will help understand if the user is going out of their scope for the task completion. In all these cases the agent must inform that all the personal data is managed respecting the GDPR. All this must be done, as said by *Bansal et al.* [9], avoiding scraping sensitive information or violating the site's terms of service while using the agent.	**Teacher:** The interface can contain a specific tab with previously learned tasks that can be consulted by the user at any moment. The user can also see a level of confidence they have gained with what they learned in the previous interactions. In the same interface a user can ask for a review of a specific learning request that they made to the agent. **Student:** The user can access at any time the previous tasks that the agent learned with them from the history page in the agent's interface (for example from the history logo in the student's home page). From a previous task page they can also access the asked questions by the agent during the teaching session through an interface that can show this type of information in order for the user to check if the agent is correctly following what is important during this process and to see if now the agent is doing the actions correctly. **Companion:** The user can access the previous interactions through the companion interface divided in suggestions and in questions. The suggestions are the ones used when the agent is giving the user some tips or help regarding what they are doing (like "These were the previously used suggestions: . . . " i.e.). The questions tab can be consulted by the user to check when and where the agent thought they were going out of their scope during a session with the companion.

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G13: Learn from user behavior. Personalize the user's experience by learning from their actions over time.	**Teacher:** The teacher agent should know what the user can and cannot do in order to deliver only the needed information. It should remember what the user learned and in case it notices that the user is still not friendly with what they learned it should provide a summary or a slim way to go through it all again in a faster way and if needed to repeat that again. **Student:** As the user keeps teaching the student, the agent will adapt its responses based on what the user has taught it and what the agent knows based on the reinforced positive feedback of the user during the previous tasks learned, making this process feel more smooth and effective. This will create a shared mental model between them, that the agent will use to make the interaction better. **Companion:** When behaving as a companion the learning process is fundamental to understand the user's preferences and offer better suggestions. The agent should keep track of the past interaction with them and understand their goal in order to make the task completion easier for the user, like said by *Weisz et al. [14] (Teach the AI system about the user, and Visualize the user's journey)* and to better understand if the user is going out of the scope of the current task.	**Teacher:** As the user interacts with the teacher, the agent understands the preferred ways to communicate with them, based on their actions. I.e., if the user already knows how to perform a step, when explaining a task that has that included, the agent will speed up this process, since it is aware that the user already knows how to do that specific thing. **Student:** When interacting with the student, the user will educate it on their preferred way to interact based on how they collaborate with the agent to make it learn new tasks. As an example, with this type of interaction the agent will better know if the user needs slower reaction times to better see how the agent is doing its actions on the website or if the user needs extra focused objects to distinguish what is happening (i.e. bigger or brighter colors to underline what the agent is writing in a text field). **Companion:** The companion will learn the user's preferences, understanding how the user uses the suggestions or how the user wants to be stopped during an interaction with the website. For example the user can adapt the level of productivity of the agent, granting a certain degree of responsibility and so allowing certain actions to be automated and others not (i.e. inserting personal data can be automated, but the submission must be done by the user itself) based on how the user behaved previously. Also the agent will show more frequent tasks that the user needs to do based on the context they are used (i.e. if the user in the home page of a website uses more a specific suggestion, the agent will give that suggestion as a ready prompt from the home page of the agent).

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G14: Update and adapt cautiously. Limit disruptive changes when updating and adapting the AI system's behaviors.	(As seen this is system wise) **Teacher:** When behaving as a teacher, the system should change gradually to make the user learn and understand about the new ways of the agent, allowing the user to adapt gradually. This type of updates will not hinder the old users trust. **Student:** The student agent when updating should do so avoiding the introduction of troublesome features that might change in an unexpected way the interaction with the user and that might be a cause of loss of trust by them. **Companion:** As a companion the new updates should be applied slowly in a way to get the user used to the new changes in a gradual way and limit the disruptive changes that might mislead the user based on their previous interactions.	**Teacher:** When a function undergoes drastic changes, the system will try to integrate the changes gradually, allowing the old users to adapt to this new behavior of the teacher, without feeling disoriented or lost. For example if the interface of the agent is going to be changed, the system will do so gradually, keeping the key elements and changing secondary items first and then the primary ones. **Student:** The student should limit the disruptive changes to control reduction of trust caused by the process of understanding again the capabilities of the agent. For example, if the student updates the ways of showing its interactions on screen, it should keep the key elements that allow all the users to follow the agent in this process (highlights, textual description i.e.), that, if removed, might cause some uncertainty to the users. **Companion:** If the companion will undergo some changes, these ones should not be disruptive, allowing a gradual learning of the new behaviors to the user. If the changes are about the functionalities, for example how the suggestions are made, these ones must not change drastically (i.e. we cannot remove the text input from the suggestion system, without allowing the user to write a suggestion).

Continued on next page

Guideline	**Web-Agent Contextualization**	**Implementation Example**
G15: Encourage granular feedback. Enable the user to provide feedback indicating their preferences during regular interaction with the AI system.	**Teacher:** Allowing the user to give feedback through an appropriate system (i.e. after each teaching interaction asking what the user liked/not liked, giving a satisfaction rating) about the teacher agent can be a good way to create a common ground and make the future interactions better. In this way the agent can learn what the specific user's preferences are. **Student:** As a student, the system should allow the user to give feedback to better understand the user's preferences and learn the preferred way to interact with them, fine tuning its functionalities to give a better experience with them, and enhance its own capabilities. **Companion:** The system, when acting as a companion, should give the user the possibility to provide useful feedback, allowing in this way for the system to learn the user's preferences and improve their experience for the next interactions, by training the agent itself.	**Teacher:** After an interaction with the system, the teacher can ask a few follow-up questions to the user about that, like "Did you enjoy this learning session?" or "Can you give a rating to this session?" to have more granular feedback and asking questions if the rating is below a certain threshold. After this the agent can also ask if the user has any suggestions to make the agent better. **Student:** The user after a session with the student can give a feedback on how it behaved, with a text field or through a set of questions that the user can answer like "What rating will you give to this session?", or "Do you think that the agent completed the task correctly?". "Were you able to follow my actions?" can be used to assess if the agent's actions were understandable. "Was the agent capable of understanding what were your inputs and the lesson given by you?, for example, will be used in case the agent does not know how to complete the task and asks the user for help, and then it does that task again with the new knowledge acquired. This will help the agent in those disambiguation cases where the agent has not enough understanding to complete the task. **Companion:** The user can give feedback on the accuracy of suggestions made by the companion through the agent's interface. This can be done through textual, ratings, or other methods, again asking follow up questions as explained before for the teacher agent. Instead for the student part the user can provide feedback on the agent's proactivity and how good it was recognizing when the user was going outside the range of the task as example. Each ready suggestion might have a dismiss button, for example an X on the top left, that can be clicked to remove that one and be replaced by another one. When doing so the user can then give a feedback on why that specific suggestion was not useful, allowing the system to fine tune the user's inclinations.

Continued on next page

Guideline	**Web-Agent Contextualization**	**Implementation Example**
G16: Convey the consequences of user actions. Immediately update or convey how user actions will impact future behaviors of the AI system.	**Teacher:** When interacting with the teacher agent, the system should notify the user about how their actions might affect the future interactions, allowing for them to roll back in case they do not want to. This will also happen after learning tasks, allowing the agent to adapt to the new acquired or desired knowledge of the user. **Student:** The student agent should notify the user when the actions that they are taking are going to modify/reinforce specific mental models based on the information gathered from the user or that the user has given to the system (through feedback i.e.). When changing the agent's settings the user will be notified about eventual limitations or new behaviors that the agent will have. **Companion:** As information is given to the companion agent, that can be both a suggestion or a question, it should notify that the provided preferences and interactions are going to change the future suggestions and/or behavior of the agent with them, in order to better match the user's preferences. The user can in addition adjust how the system behaves changing it from the settings page of the companion. The same will happen when changing the settings of the agent, for anyone of the types above (i.e. Adapting the proactivity of the system to the desired one, Activate/Deactivate specific functions of the agent, etc.). Also it is important to let the user know when the agent will take irreversible actions, stopping before doing so and waiting for a user response .	**Teacher:** When the agent is teaching something, after the task is finished it can ask if the user learned that with questions like "Do you feel confident now with the new learned task?". If so, the agent will say that for the next interaction it will consider that learned by the user and will adapt its own explanations considering the fact that the user already has that type of knowledge. Instead when the user adapts the setting of the agent, it will inform how those ones will affect its behavior, allowing the user to accept that or roll back. **Student:** After a session, the student will ask feedback to the user, for example "Were you able to follow me during this session?" and if the response is negative the agent can ask "Would you like to change something?", allowing the user to write or to show the settings page to allow the user to fine tune the agent's functionalities and properties. This can also be done in the system's interface. When doing so the user will be notified about how this will change how the agent will interact with them, allowing them to roll back or to accept these changes. **Companion:** The feedback from the user will be used to enhance their experience with the companion, so the agent will notify through messages that things will affect its future behavior. For example, after finishing a task, the agent can ask "Are you satisfied with the task you gave me?". This type of feedback will help reinforce the agent's capabilities if the response is positive it will be used as a suggestion when the context will be matched. If not the user can then adapt the future interactions to their preferred way. The same will happen when the agent will notice that the user is going out of the scope of the task. As said for the agents above, the behavior will be similar when changing the behavior directly from the settings.

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G17: Provide global controls. Allow the user to globally customize what the AI system monitors and how it behaves.	**Teacher:** The system, when behaving as a teacher should provide the user a reachable and easy to understand set of controls to let the user personalize their experience with the system, in order for them to get the desired experience. The user can decide how much the teacher is explaining and the level of detail of the task explanation. **Student:** The user should be capable of customizing the behavior of the student agent, adjusting to the desired level the proactivity and the amount and type of data that it can collect, reminding the user that all is done respecting the GDPR rules. The user can also adjust and choose how the system will show the agent actions in the website, when learning new tasks depending on the user needs or preferences. **Companion:** When the user interacts with the companion agent, at any moment they can be capable of customizing what the system is monitoring, its level of proactivity and the amount of data collected to give the user useful suggestions to speed up their task completion process and not to be too invasive. This would also be possible for the functionalities that the agent offers when it is trying to block the user from going out of the scope of their actual task.	**Teacher:** The teacher's interface can have a "Settings" page where the user can see all the active functionalities, the level of proactivity of the agent in that exact moment, for how much time the data is stored, the level of detail, etc. From this interface they can choose their preferred levels in order to have a personalized experience. Also from this page the user can access the personal data that is stored and can choose how they want it to be kept. The interface in any case will assure the user that the data kept is only for preferences and that is respecting the GDPR rules, allowing them to manage their own data according to this. For example if the teacher knows the personal data of a user it can use that to show some examples using that data for teaching a task, instead of using some fake data (like filling a Name and Surname field i.e.). **Student:** From the student interface the user can decide how proactive they want the agent to be, the amount of data that it collects. The agent must show that the data collected is just for learning purposes and that it is not kept in any way, respecting the GDPR rules in the privacy section. The user can, moreover, choose how they want the agent to describe their actions in the website (for example for certain users the text is difficult so they prefer an audio explanation for each action that the agent is taking). **Companion:** The user can access easily the settings of the companion agent from a gear icon that will lead them to the settings page. From this page the user can choose the level of proactivity, the amount of data stored and the preferences, as done by the teacher/student agent. The same will be done with the GDPR rules as per the other agents above, allowing the user at any moment to manipulate their own data according to this. At any moment the user can go back to predefined options by clicking its button on the system's interface. This will be possible for each agentic figure.

Continued on next page

Guideline	Web-Agent Contextualization	Implementation Example
G18: Notify users about changes. Inform the user when the AI system adds or updates its capabilities.	**Teacher:** When the teacher agent is being updated, it should notify the user about the incoming changes, allowing them to understand everything in a simple way, and if they need a more detailed explanation, the system can provide a more complete and complex documentation. The changes should not be too drastic in order for the user to get used to the changes gradually. **Student:** The student agent should notify the changes that are going to be made in his way of interacting with the user and the data collected, being transparent with the user and making them aware of the new agent's behavior. **Companion:** The system, when behaving as a companion, should notify the user about the updates in its capabilities in order for them to better use all the new functionalities and to not be disoriented by the new changes introduced by the system. This should be done respecting the rule of transparency, since ,as said by *Baker et al.* [10], this characteristic is linked with trust by the user.	**Teacher:** When new changes are being added to the teacher agent, it should notify the user about its new capabilities with messages like "With the new updates I can do . . . " or "Now I can . . . Try it out.". In case of changes it should notify the user with similar messages like "Now, if you want to . . . you have to . . . Ask me if you need some help to understand this." as an example. **Student:** As new changes are being added to the student, the system should notify the user about its new capabilities with messages like "With the new updates I can do . . . " or "Now I can . . . Try it out.". In case of changes it should notify the user with similar messages like "Now, if you want to . . . you have to . . . Ask me if you need some help to understand this." as an example. **Companion:** When new changes are being added to the companion, the agent should notify the user about its new capabilities with messages like "With the new updates I can do . . . " or "Now I can . . . Try it out.". In case of changes it should notify the user with similar messages like "Now, if you want to . . . you have to . . . Ask me if you need some help to understand this." as an example.

Table 4.1: Microsoft Guidelines table adapted for this thesis

This table served as the starting point of the design process, enabling ideas to be developed before moving on to detailed prototype creation. The "Contextualization" column also integrates other important works that were previously cited and particularly relevant. The "Implementation" column instead, gathers inspirations from existing agents or features.
From this point on, the focus of the naming used in the table shifted from what the agent was doing to what the user was doing, in order to make each modality easier to understand from the user's perspective. Consequently, the names were changed as follows:

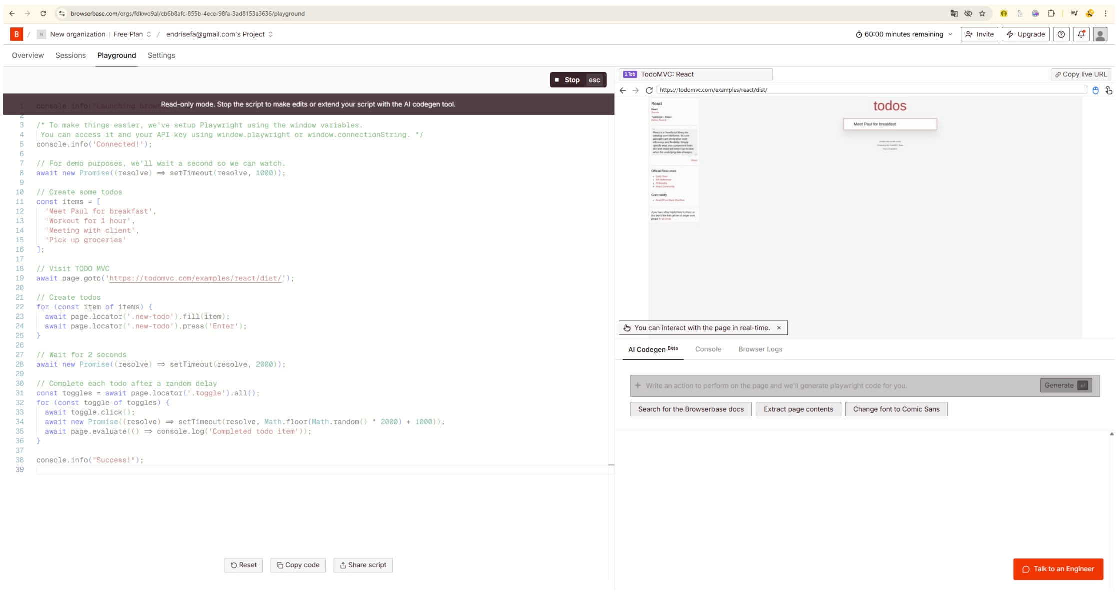
- from *Student* to **Teach**, since in this modality the user is teaching the agent;
- from *Teacher* to **Learn**, since in this modality the user is learning;
- from *Companion* to **Collaborate**, to reinforce the idea of completing a task together.

This renaming supports clearer mental models and improves the readability of the guidelines from a user-centered point of view.

4.2 Study on current Web Agent Designs

Agents implement a variety of functionalities and features that differ depending on the scope and context in which the system is used. However, even if current implementations adopt different designs, the main types of deployment can be summarized into two broad categories, based mainly on the paradigm used by the agent to navigate the web:

1. *Agent with built in browser:* this type of agent provides its own browser within the current browser tab, typically as a window embedded in the page. Usually, it offers a text input to send requests. For example, Browserbase [19] 4.1 provides a code input and a headless browser inside the tab.



browserbase.com/orgs/fdkwo9al/cb6b8afc-855b-4ece-98fa-3ad8153a3636/playground
New organization
Free Plan
endrisefa@gmail.com's Project
60:00 minutes remaining
Invite
Upgrade
Overview
Sessions
Playground
Settings
Stop
esc
1 Tab
TodoMVC: React
Copy live URL
https://todomvc.com/examples/react/dist/
todos
Meet Paul for breakfast
You can interact with the page in real-time.
AI Codegen Beta
Console
Browser Logs
Write an action to perform on the page and we'll generate playwright code for you.
Generate
Search for the Browserbase docs
Extract page contents
Change font to Comic Sans
Reset
Copy code
Share script
Talk to an Engineer

Read-only mode. Stop the script to make edits or extend your script with the AI codegen tool.

```
console.info("Launching brows

/* To make things easier, we've setup Playwright using the window variables.
 You can access it and your API key using window.playwright or window.connectionString. */
console.info('Connected!');

// For demo purposes, we'll wait a second so we can watch.
await new Promise((resolve) => setTimeout(resolve, 1000));

// Create some todos
const items = [
  'Meet Paul for breakfast',
  'Workout for 1 hour',
  'Meeting with client',
  'Pick up groceries'
];

// Visit TODO MVC
await page.goto('https://todomvc.com/examples/react/dist/');

// Create todos
for (const item of items) {
  await page.locator('.new-todo').fill(item);
  await page.locator('.new-todo').press('Enter');
}

// Wait for 2 seconds
await new Promise((resolve) => setTimeout(resolve, 2000));

// Complete each todo after a random delay
const toggles = await page.locator('.toggle').all();
for (const toggle of toggles) {
  await toggle.click();
  await new Promise((resolve) => setTimeout(resolve, Math.floor(Math.random() * 2000) + 1000));
  await page.evaluate(() => console.log('Completed todo item'));
}

console.info("Success!");
```

Figure 4.1: Example of usage of Browserbase

A similar example is ChatGPT Agent, which uses the standard ChatGPT interface with a chatbox to input the request using natural language. However, unlike standard ChatGPT, it operates as an agent inside a headless browser. The interface can have the browser running below the prompt or display the chat on one side and the browser on the other, with the page always visible on the screen.

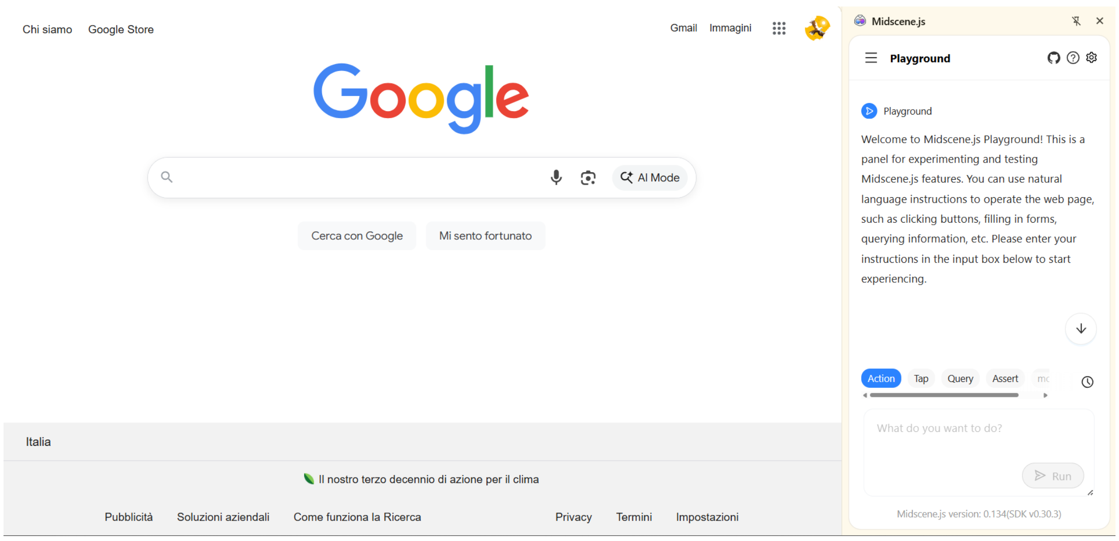
2. *Browser extension:* Differently from the previous type, this one uses the user's own web browser and can manipulate the current tab to complete the task, by opening the extension and using its interface. Usually the interface provides a

text input where the user specifies the task that wants to complete, usually using natural language. An example can be seen in figure 4.2.

Another relevant modality is the *"Coding"* one. As seen in the figure 4.1, Browserbase itself supports it, allowing the user to send requests to the agent by writing code. Another example can be Stagehand [20], which offers an SDK for developers to control web agents. Also Midscene can operate with this paradigm [8] by using the bridge mode: by using the “bridge mode,” the user can write code (guided by the official documentation) to control the browser tab in which the extension is active, allowing the agent to perform actions specified in the code. This last option is more oriented towards developers, but it remains a particularly interesting way to interact with agents and to gain a deeper understanding of how an agent operates.

4.3 Choice of platform: web extension

Considering the paradigms noted above, the code-based modality was not selected as the primary interaction mode for this thesis, due to the programming skills required and the need to understand technical documentation, limiting the access only to people with coding experience. Furthermore, this type of modality is mainly used to create or integrate agents in other systems, rather than to provide a general-purpose user-facing interface.
The final choice for this thesis project was to develop a web extension, more specifically a Google Chrome extension, following an interaction approach similar to already existing interfaces, such as that shown in Figure 4.2.
Both main approaches (built-in browser agents and extensions) provide a well-acquainted modality of interaction, allowing the user to make the request to the agent through a chat-like text message. However, creating and maintaining a headless browser environment is more complex from a technical and infrastructural point of view, whereas a browser extension simply interacts with the user’s current tab in a more direct and seamless way.
With the extension-based approach, the user simply clicks on the agent icon and can immediately request assistance with a task. There is no need to open the Web Agent website, simplifying the invocation. This approach also lets users start from their preferred search engine and existing tabs, instead of being constrained to the one used inside a headless browser.
However, this choice also has important implications. Since the extension runs in the user’s own browser, the agent can perform actions directly on the user’s behalf. This could speed up processes such as logins and other kinds of actions that require personal data, which are slowed down when using headless browsers. At the same time, this capability introduces the risk that the agent may also perform harmful



Welcome to Midscene.js Playground! This is a panel for experimenting and testing Midscene.js features. You can use natural language instructions to operate the web page, such as clicking buttons, filling in forms, querying information, etc. Please enter your instructions in the input box below to start experiencing.

Figure 4.2: Midscene.js interface with the Browser opened

operations on the user's behalf. As discussed in previous chapters, this situation can easily lead to a loss of trust.
When operating in this modality, agents must detect the presence of potentially harmful operations and either, notify the user or stop the execution of the action, returning control to the user so that they can explicitly allow or dismiss the agent's behavior.

4.4 Design ideas and prototype

The design inspiration were inspired by the existing web agents implemented as a Google Chrome extension, such as Midscene.js. The overall design is kept simple, with a textbox where the user inserts their request for the agent. For the remaining components the main goal was to respect a simple and minimal design, following both Guidelines cited in the table 4.1, and integrating with the Nielsen's Heuristics for UI Design [18]. The following subsections analyze the main interfaces that were prototyped using Figma.
The core concepts for each component functionalities were the following:

- **Learn:** in this modality, the agent was designed to provide a sequence of steps to the user. For each step, the user performs a set of actions on the website, and the agent records these interactions, allowing the user to receive feedback. The agent also offers a series of hints that describe the actions required to complete each step on the website, enabling a more gradual learning process. In addition, there is an option for the agent to "show" the actions by highlighting the areas of the interface where the user needs to interact.

- **Teach:** In the teach modality, the user provides, using natural language, the task they want to teach. After that, the user specifies the task step by step, showing the agent the actions required to complete each step. For every step, the agent then replays the actions by generating them, allowing the user to verify their correctness. Once all steps have been performed, the user completes the task, and all recorded data are stored in the knowledge of the agent. This allows other users to later learn this task in *Learn* mode or to use it to automate steps in *Collaborate* mode.

- **Collaborate:** In this modality, the use provides, by using natural language, a task they want to accomplish. The agent, based on the current state of the table,generates a set of steps that are needed to reach the goal. The agent can automate individual steps by using the context of the page, but it always leaves the final decision to the user, who can choose whether to automate a step or not. The agent understands autonomously if the step is completed and moves on until the end of the task is reached.

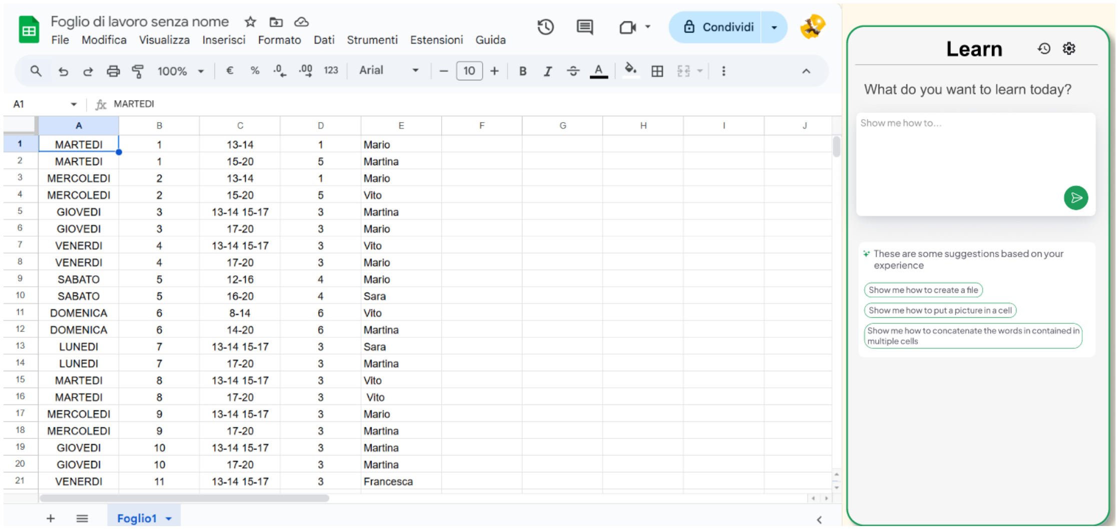
4.4.1 Prototype Architecture

This section describes the design developed during the prototyping phase, which later serves as a reference for the development chapter. The main idea was to keep the interface as minimal as possible and consistent with consolidated applications and existing web agents. Starting from the Home page of the application, all the different agent modalities are analyzed in sequence. The chosen operating environment for the agent was a spreadsheet website.

Home

For the Home interface, as mentioned before, the structure was kept similar to the existing agents that work as browser extensions. At the top there is the header component showing in the middle the type of agent modality selected. By clicking on this label, the user can switch between the different modalities. To the right of the modality name, there are the History and the Settings button.
The History section keeps track of all the previous interactions of the user, allowing the user to clear that. The Setting section lets the user personalize their experience, for example by choosing the interface theme, and more importantly by entering their personal API key, which is required to enable and use the agent.
Below the header there is a question that depends on the modality selected: for example, as in the Figure 4.3, the question is “What do you want to learn today?”, for the Learn modality. Inside the main text input, a placeholder hint suggests

how to formulate the first part of the sentence, guiding the user in structuring the request to the agent. Within the input area there is also a button to send the request.


Foglio di lavoro senza nome
File Modifica Visualizza Inserisci Formato Dati Strumenti Estensioni Guida
Condividi
100%
Arial
10
A1
MARTEDI
Foglio1
Learn
Show me how to...

	A	B	C	D	E
1	MARTEDI	1	13-14	1	Mario
2	MARTEDI	1	15-20	5	Martina
3	MERCOLEDI	2	13-14	1	Mario
4	MERCOLEDI	2	15-20	5	Vito
5	GIOVEDI	3	13-14 15-17	3	Martina
6	GIOVEDI	3	17-20	3	Mario
7	VENERDI	4	13-14 15-17	3	Vito
8	VENERDI	4	17-20	3	Mario
9	SABATO	5	12-16	4	Mario
10	SABATO	5	16-20	4	Sara
11	DOMENICA	6	8-14	6	Vito
12	DOMENICA	6	14-20	6	Martina
13	LUNEDI	7	13-14 15-17	3	Sara
14	LUNEDI	7	17-20	3	Martina
15	MARTEDI	8	13-14 15-17	3	Vito
16	MARTEDI	8	17-20	3	Vito
17	MERCOLEDI	9	13-14 15-17	3	Mario
18	MERCOLEDI	9	17-20	3	Martina
19	GIOVEDI	10	13-14 15-17	3	Martina
20	GIOVEDI	10	17-20	3	Martina
21	VENERDI	11	13-14 15-17	3	Francesca

What do you want to learn today?

These are some suggestions based on your experience

Show me how to create a file

Show me how to put a picture in a cell

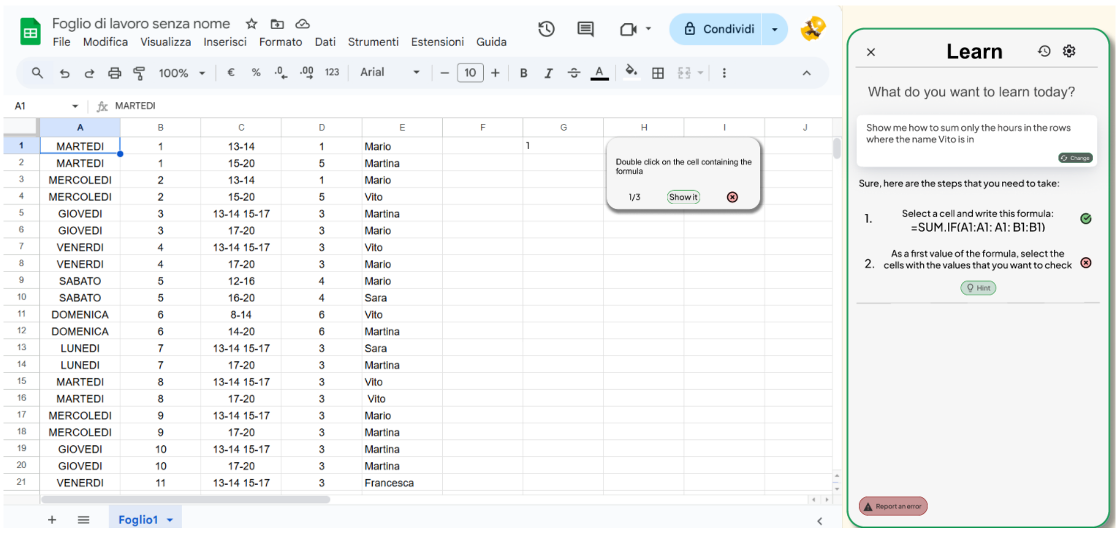
Show me how to concatenate the words in contained in multiple cells

Figure 4.3: Home page of the Learn modality

Finally there is a Card component that, depending on the selected modality, contains some prompt suggestions. By clicking on a suggestion, it fills the text field automatically with the corresponding text. This helps users speed up their interactions with the agent and fills what would otherwise be an empty portion of the interface. These texts change according to the selected modality, helping to maintain consistency within this component across interfaces, even when the agent's modality changes.

Learn

In the Learn interface, the header and the text input area are the same as in the Home page, thus maintaining consistency. The only change is that the size of the text input is adjusted to fit the input, and the "send" button is replaced by a “change” button. This button allows the user to reformulate the original request and send a new one. In figure 4.4 is shown while the user has opened the hint section of the second step. For each step, there is a step number, a textual description of the step, and, when needed, a formula shown in bold. On the right side, there is an icon telling the state of the step(for example, whether it is not yet completed or already done).
The hints shown as a dialog in the website, are waiting for the user do to what is written. If the user needs, can click the "Show me" button to make the agent show


Foglio di lavoro senza nome
File Modifica Visualizza Inserisci Formato Dati Strumenti Estensioni Guida
Condividi
100%
Arial
10
A1
MARTEDI
Foglio1
Double click on the cell containing the formula
1/3
Show it
Learn
Change
Hint
Report an error

	A	B	C	D	E	F	G
1	MARTEDI	1	13-14	1	Mario		1
2	MARTEDI	1	15-20	5	Martina		
3	MERCOLEDI	2	13-14	1	Mario		
4	MERCOLEDI	2	15-20	5	Vito		
5	GIOVEDI	3	13-14 15-17	3	Martina		
6	GIOVEDI	3	17-20	3	Mario		
7	VENERDI	4	13-14 15-17	3	Vito		
8	VENERDI	4	17-20	3	Mario		
9	SABATO	5	12-16	4	Mario		
10	SABATO	5	16-20	4	Sara		
11	DOMENICA	6	8-14	6	Vito		
12	DOMENICA	6	14-20	6	Martina		
13	LUNEDI	7	13-14 15-17	3	Sara		
14	LUNEDI	7	17-20	3	Martina		
15	MARTEDI	8	13-14 15-17	3	Vito		
16	MARTEDI	8	17-20	3	Vito		
17	MERCOLEDI	9	13-14 15-17	3	Mario		
18	MERCOLEDI	9	17-20	3	Martina		
19	GIOVEDI	10	13-14 15-17	3	Martina		
20	GIOVEDI	10	17-20	3	Martina		
21	VENERDI	11	13-14 15-17	3	Francesca		

What do you want to learn today?

Show me how to sum only the hours in the rows where the name Vito is in

Sure, here are the steps that you need to take:

1. Select a cell and write this formula: =SUM.IF(A1:A1: A1: B1:B1) ✓
2. As a first value of the formula, select the cells with the values that you want to check ⊗

Figure 4.4: Learn modality interface

them the actions needed to complete the step. Finally, at the bottom of the agent interface there is a button that allows the user to report errors.

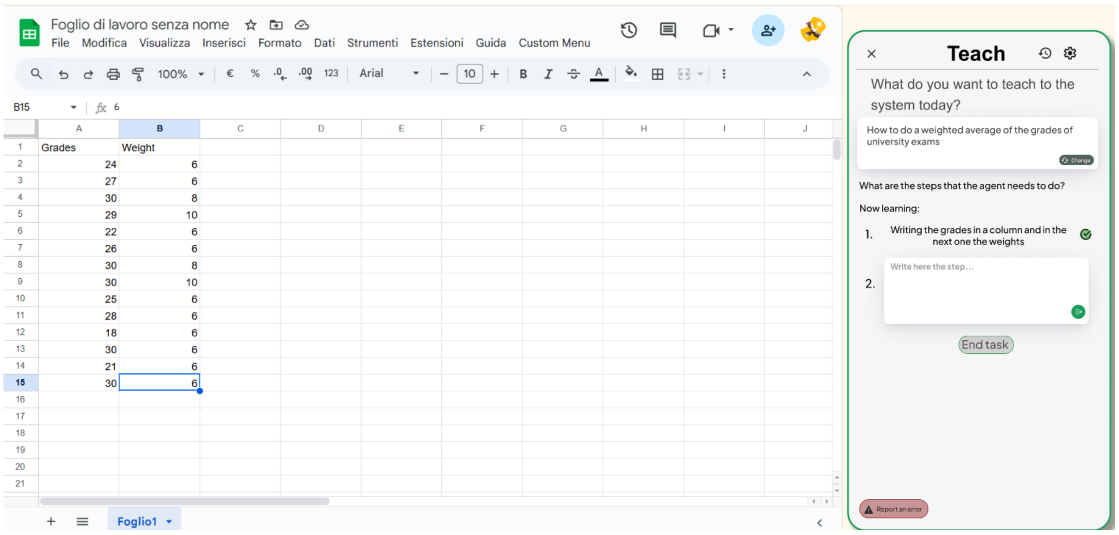
Teach

As with the Learn modality, the header and the text field section are kept the same in Teach, ensuring consistency among all modalities. Once the agent is started, the user writes a brief description of the step that is going to perform inside the specific textbox, such as the one shown in Figure 4.5, where the user is specifying the second step. After entering the text, the user performs the corresponding actions on the website and then clicks the "Stop" button in the interface to signal that the step is completed. Next, the user clicks the button to start the learning process of the agent: the agent resets the table or interface elements affected by the user's actions and, through a highlighted mouse pointer, replays the learned actions. For example, in figure 4.6, the agent is replaying the actions for the first step. The mouse pointer is surrounded by a colored highlight, which visually indicates the level of confidence of the agent while executing those actions.

To end the teaching session, once all the steps have been provided and reviewed, the user clicks the "End task" button. As in the Learn modality, an error report button is provided at the bottom of the interface.

Collaborate

The layout of the Collaborate interface follows the same structure as the Home of this modality, and is consistent with the other modalities. In this case, the agent


Foglio di lavoro senza nome
File Modifica Visualizza Inserisci Formato Dati Strumenti Estensioni Guida Custom Menu
100% Arial 10
B15 fx 6
Foglio1
Teach
End task
Report an error

	A	B
1	Grades	Weight
2	24	6
3	27	6
4	30	8
5	29	10
6	22	6
7	26	6
8	30	8
9	30	10
10	25	6
11	28	6
12	18	6
13	30	6
14	21	6
15	30	6

What do you want to teach to the system today?

How to do a weighted average of the grades of university exams

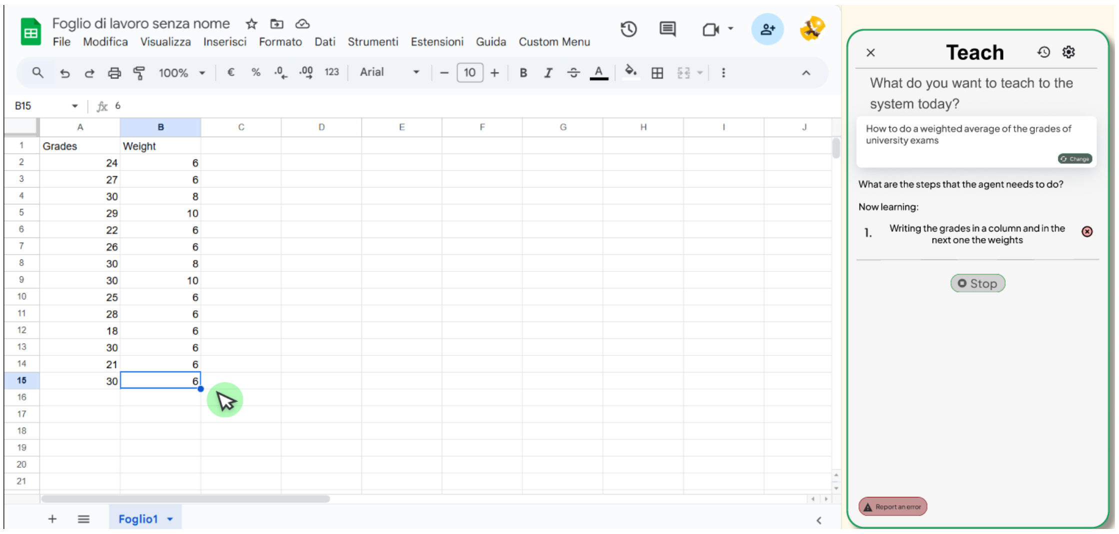
What are the steps that the agent needs to do?

Now learning:

1. Writing the grades in a column and in the next one the weights

2. Write here the step...

Figure 4.5: Teach interface while inserting the second step text


Foglio di lavoro senza nome
File Modifica Visualizza Inserisci Formato Dati Strumenti Estensioni Guida Custom Menu
100% Arial 10
B15 fx 6
Foglio1
Teach
Stop
Report an error

	A	B
1	Grades	Weight
2	24	6
3	27	6
4	30	8
5	29	10
6	22	6
7	26	6
8	30	8
9	30	10
10	25	6
11	28	6
12	18	6
13	30	6
14	21	6
15	30	6

What do you want to teach to the system today?

How to do a weighted average of the grades of university exams

What are the steps that the agent needs to do?

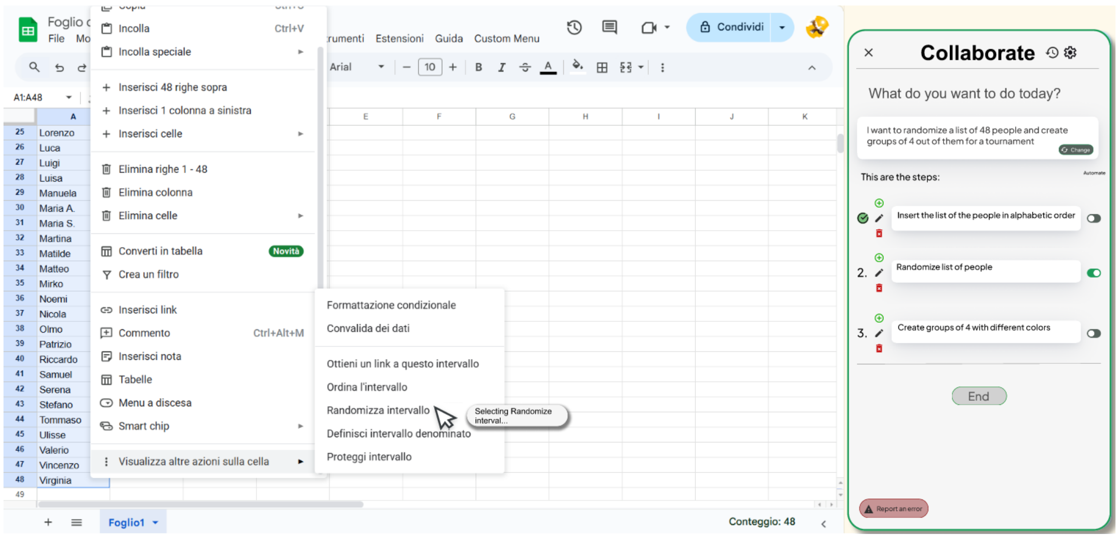
Now learning:

1. Writing the grades in a column and in the next one the weights

Figure 4.6: Teach interface while the agent is replaying the first step

provides a set of steps that can be either automated by the agent or manually executed. The user can toggle whether a specific step should be automated by clicking a dedicated switch next to it. Each step can be edited, deleted or have a step below by click the buttons on the side left of the step text.
The agent in this modality detects when the user completes a task and, after that, if the next step is marked as automated, proposes its automation through a popup on screen. If the user clicks that popup, the step is automatically automated.

During this automated execution, short explanations appear near the mouse cursor on the page, as shown in Figure 4.7, to clarify what the agent is doing.



What do you want to do today?

I want to randomize a list of 48 people and create groups of 4 out of them for a tournament

This are the steps:

1. Insert the list of the people in alphabetic order
2. Randomize list of people
3. Create groups of 4 with different colors

Figure 4.7: Collaborate interface of the agent automating a step

The user is free to end the task at any time by clicking the "End" button at the bottom of the list of steps. As in the other modalities, a report button is also present at the bottom of this interface.

Chapter 5

Development

In this chapter discusses the development process, starting from the technologies used, then analyzing the implemented architecture with some focus on the main components. The chapter, then, concludes with a description of the additional features that enable AgentExtension, which is the name of the extension, to support richer and more effective interactions with the user.

5.1 Technologies used

The choice of the technologies was a fundamental aspect of this project. The main objective was to develop a Google Chrome Extension, following the official documentation provided by Google [21]. For the user interface , the extension was implemented using React [22] with TSX [23] and Vite [24]. This configuration made it possible to adopt the component design paradigms of React and rely on a configuration-based build process, where simple commands are sufficient to rebuild the extension after changes thanks to Vite.
For the GUI, HeroUI [25] was selected as the component library. HeroUI provides dynamic, easy-to-use, and visually appealing components already integrated to work with React. Moreover, it is straightforward to configure alongside Tailwind CSS [26]. Tailwind was used to manipulate component styles directly from TSX code, enabling a faster and more flexible development process.
to conclude, GitHub [27] was used to store and version the code, supporting parallel work on different features. For this purpose, two repositories were create: one for running the interface locally as a "web version," used mainly to test individual components on the *localhost*, and an official repository containing only the build-ready code for the extension, since the functions called by this version were making the web one crash.

5.2 Structure of the project

As mentioned in the previous section, two repositories were used for this project: one for the web version, which facilitated faster GUI development, and one for the build version, containing all the code specific to the Chrome extension. What is going to be analyzed in this chapter is the code of the extension, however, it is worth to mention the need of a double repository, because it allows the integration of only the GUI components by copying the component code, since as happened during the testing, introducing functions for the managing a Google Chrome extension made the web version crash.
Another important decision was to create a simple version of a spreadsheet website for the agent to interact with. Initial experiments with existing web agents revealed that they often struggled with understanding how complex spreadsheet platforms are organized. For this reason, the AI logic in this project was tailored to work specifically with the custom spreadsheet website developed for the thesis. This website was deployed using GitHub, and will not be further analyzed in this thesis work. The choice of a spreadsheet environment allowed testing of all components in a challenging context, for both expert and novice users.

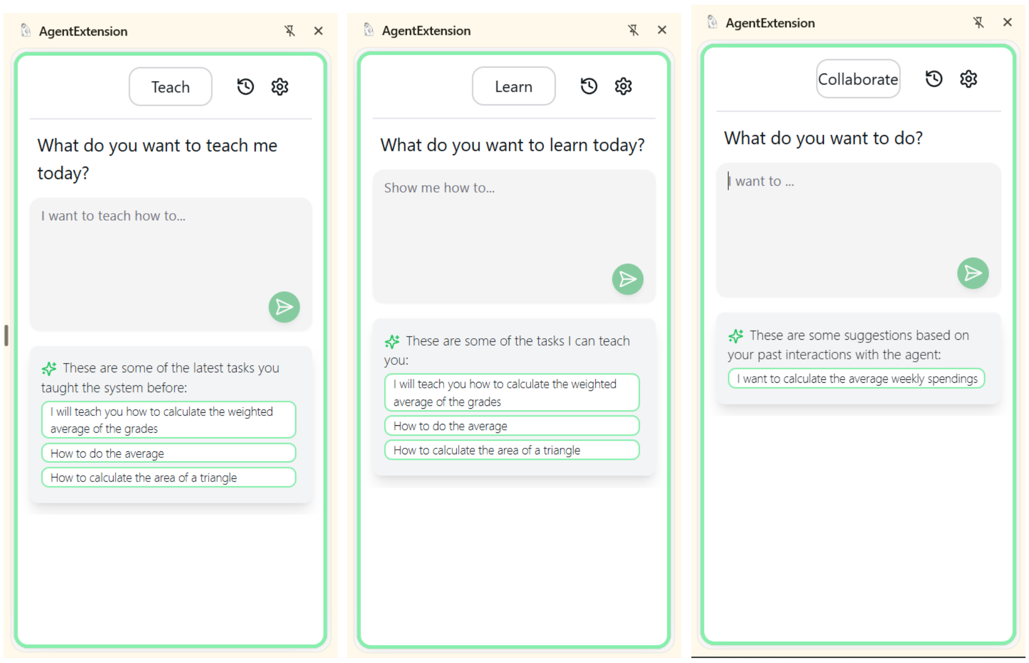
5.3 System Architecture

The system was developed adopting a single-page architecture for the extension. The decision was primarily made to avoid routing errors or other types of issues that might arise when building the extension. Furthermore, since all the components shared common parameters and states, a single-page structure simplified reuse and allowed for a better communication between the three different modalities. This approach made the whole system connect and share information better.
Starting from the home page, each interface changes depending on whether a session is active and, naturally, depending on the selected modality. Each modality (Learn, Teach, Collaborate), as well as the Home view, has its own separated component, all of which are invoked in the *App.tsx* file. This file defines all the `useState` variables used to manage state across components and allowing for the different components to display. In `App.tsx`, the header component is also managed, which includes the switch that selects different agent modalities and shows the current one, as well as the buttons that trigger modals for history and settings.
Another important aspect concerns data storage. All data are saved in the extension's local storage, so the agent's knowledge is currently managed individually for each user and is not shared. Personal data, such as the user's API key and the different modalities history, are also stored there. The agent initialized with a base knowledge, which consists of the four formulas supported by the custom

spreadsheet website. This data was saved manually, since the agent does not create or generate this base knowledge when fist installed. However, a file in the repository contains this base knowledge, allowing new users to insert this data in case the extension is installed on a new browser.
In the following subsections, the main interfaces are analyzed with brief descriptions of relevant code features, accompanied by figures of the final UI.

5.3.1 Home Interface

The home page contains, below the header component, a text field to submit the user's request, and, underneath it, a card containing up to 3 elements that help the user perform requests more quickly, inserting the clicked text in the text box. The only variation occurs in the Teach modality, where this card element contains the last 3 tasks learned by the agent, in order for the user to not teach these again. Figure 5.1 shows the home page for each of the three modalities, showing the small differences in the interface. This choice was made, also, to keep the interface consistent, and allowing the user to gain familiarity with the system more easily.
When opened, AgentExtension will show the home page of the Teach modality. By clicking the name of the mode, the user can than switch to the wanted one.


AgentExtension

Teach

What do you want to teach me today?

I want to teach how to...

These are some of the latest tasks you taught the system before:

I will teach you how to calculate the weighted average of the grades

How to do the average

How to calculate the area of a triangle

AgentExtension

Learn

What do you want to learn today?

Show me how to...

These are some of the tasks I can teach you:

I will teach you how to calculate the weighted average of the grades

How to do the average

How to calculate the area of a triangle

AgentExtension

Collaborate

What do you want to do?

I want to ...

These are some suggestions based on your past interactions with the agent:

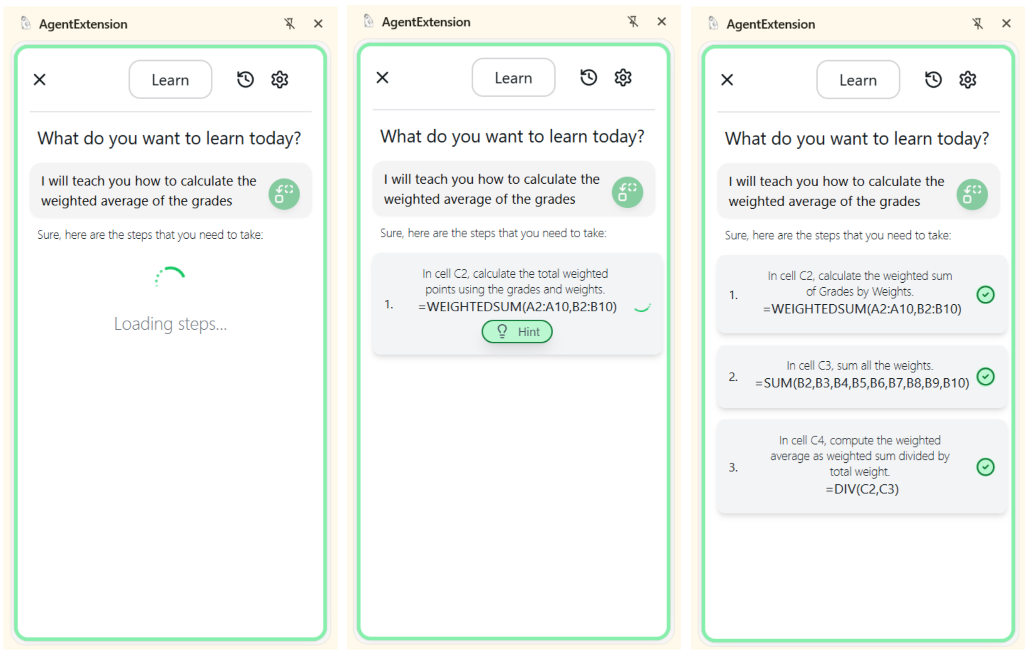
I want to calculate the average weekly spendings

(a) Teach modality (b) Learn modality (c) Collaborate

Figure 5.1: Home interfaces for all three modalities

5.3.2 Learn Interface

The Learn interface keeps the text box at the top, resized to better fit the content, allowing the user to modify its text by clicking and changing it. After the user submits a task, the agent first assesses whether it already knows how to perform it. As shown in Figure 5.2 (a), if the task is recognized, a set of steps is generated, and shown below the task text box. . If the agent does not have that knowledge(determined through an API call to OpenAI), instead, a modal is displayed asking the user whether they want to switch to the Teach modality to teach the task (see Figure 5.5 (a)).
Once the assessment is completed and al the steps are generated, the agent initially shows only the first step, as illustrated in Figure 5.2 (b).



What do you want to learn today?

I will teach you how to calculate the weighted average of the grades

Sure, here are the steps that you need to take:

Loading steps...

AgentExtension

What do you want to learn today?

I will teach you how to calculate the weighted average of the grades

Sure, here are the steps that you need to take:

1. In cell C2, calculate the total weighted points using the grades and weights. =WEIGHTEDSUM(A2:A10,B2:B10)

AgentExtension

What do you want to learn today?

I will teach you how to calculate the weighted average of the grades

Sure, here are the steps that you need to take:

1. In cell C2, calculate the weighted sum of Grades by Weights. =WEIGHTEDSUM(A2:A10,B2:B10)

2. In cell C3, sum all the weights. =SUM(B2,B3,B4,B5,B6,B7,B8,B9,B10)

3. In cell C4, compute the weighted average as weighted sum divided by total weight. =DIV(C2,C3)

(a) Assessment of the task (b) First step is displayed (c) All steps completed

Figure 5.2: Variants of the Learn interface

Immediately after the first step is displayed, the component sends a message to an injected JavaScript code in order to activate the recording of the user's actions. This functionality records all clicks and inputs, allowing the agent to check if the performed actions are correct. If incorrect actions are performed, the agent displays error popups in the current tab 5.6. If the user feels uncertain, they can click the "Hints" button to open a dialog in the current tab that shows a description of all the actions required to complete the step 5.8. The user can also allow the agent to

visually demonstrate how to complete the step by clicking the "Show me" button inside the hint dialog 5.7.
When the step is successfully completed, the user is notified through a congratulating popup. After all steps have been completed (Figure 5.2), the agent injects a dialog containing a congratulations message into the current tab. This dialog contains a "Close" button that, if clicked, ends the session. After that button is clicked, the history stored in the local storage is updated.

5.3.3 Teach Interface

In the Teach modality, the header and the text field at the top remain the same as for the Learn interface, ensuring consistency. Below the text box, the layout changes to support task teaching. This modality also begins with an assessment to check whether the task is already known, using the same logic and a similar interface to that used in Learn. If the task has already been taught, a modal appears allowing the user either to teach it again or to return to the home of Teach (Figure 5.5 (b)).If the task, instead, is part of the agent's base knowledge, the modal only shows a close button, preventing the user from modifying core functions, which could compromise the integrity of the system. Before the teaching session starts, the current state of the table is saved as a reference. This provides the context to the agent, enabling it to adapt the taught task to future situations when users want either to learn the task (in Learn mode) or to automate it (in Collaborate mode).
Once the session begins, the user inserts in the step text box only a brief description of what is going to perform, using natural language. After sending this step description, the agent starts recording the user's actions on the website. The interface clearly indicates that recording is in progress, by placing the step inside a card with a "Stop" button and a status message above it (Figure 5.3 (b)). The use then performs the actions required to complete the step and, once they are done, presses the "Stop" button.
After that, the interface shows a "Generate" button that starts the generations of the agent's steps. Clicking this button triggers a cleaning process that restores the table to its previous state, removing the user's actions. The agent then generates actions based on the recorded data and the cleaned table state. When the actions are ready, the user clicks the "Start" button inside the dialog injected to start the agent's replay 5.7. At any point, the user can stop the agent while it is showing the actions, in order to block or prevent eventual errors or harmful actions.
Once the actions are completed (or the user has stopped the agent), the interface waits for feedback, asking whether the actions were correct or if they need correction (Figure 5.3 (e)). If the user indicates that the actions are wrong, the agent shows a text box where the user can type a textual correction, which is then used to

generate updated steps (Figure 5.3 (f)).
After the first step is created and validated by the user, they can end the task by clicking the "End task". This button appears starting from the second step onward, only during the step creation. All these interface transitions are performed with dynamic animations and conditional rendering: different elements become visible when certain variables reach specific values, and these counters are updated after each key action, enabling a smooth teaching flow.
When the user ends the task, the History is updated and the task is saved in the local storage, updating the set of known tasks. What are stored are: the task description, the initial table state, and for each step, the step description and the actions performed by the agent. In case the task was taught again, the previous one is replaced by the new one.

5.3.4 Collaborate Interface

As for the other two modalities, the header and text box remain unchanged in Collaborate. After the user submits a task, a loading spinner similar to the one used in the other modalities is displayed while the steps are being generated. These steps are derived from the current state of the table and are intended to guide the user toward the completion of the task. The function responsible for this generation also assesses each step, determining whether the agent already knows how to perform it or not.

Once the steps are generated, each step is displayed inside a card. The current step, which is marked with a spinner on the right side, could be completed by the user, as seen in Figure 5.4 (b), or, if automatable, by the agent, by clicking the "Generate Actions" button. If the step is not automatable, but the user attempts to enable automation via the corresponding switch, the agent displays a modal asking whether the user wants to teach that step first (Figure 5.5 (c)). If the user agrees, the session changes into a teaching session, switching modality to Teach and passing the step description as the task to be taught.
For each step the user is capable of, by clicking the respective icons on the card (from left to right):

- *edit the step text:* clicking this icon changes the card into a text box, allowing the user to update the text. For simplicity, when the text is changed, the step is treated as unknown to the agent.
- *delete the step:* clicking this icon opens a confirmation modal to validate the user's choice.
- *add a new step below:* clicking this icon inserts a new text input below the selected step. The user can close this input by clicking a close icon in the same position.

If the user decides to automate a step, the process starts with generating the corresponding actions. Once these actions are ready, the interface displays a simplified list of actions that the agent will perform. This list is intentionally minimal, avoiding technical details (such as formulas) to keep the focus on the main interactions. After the user presses the "Start agent" button (Figure 5.4 (c)) and confirms the operation through a dialog injected into the page (Figure 5.7 (b)), the agent takes control of the current tab and performs the actions to complete that step. In this modality, only the current step can be completed; the agent moves on to the next step only after the current one has been completed, either by the user or by the agent.
The user, at any moment, can decide to conclude the session by clicking in the "End Task" button that is below the final step. This ensures that the user maintains full control over the agent and can use this modality in a flexible way.

5.4 Other features

This section presents additional components and features that were essential for the correct functioning of the project, but were not part of the UI. Some of these functionalities were mentioned in the previous subsections; here they are described in a more focused way, without going into full implementation detail.

5.4.1 Injected Code

To communicate with the current page's DOM and vice versa, the extension injected into the current tab some JavaScript code. The injection happens only when the agent is opened and it injects it only in the current tab (thanks to the *vite.config.ts* file). The most important injected scripts are the Recording and the Replaying module.
The Recording functionality tracks user actions on the website. The actions recorded are only two types:clicks and inputs. These values are either stored in the extension's local storage or sent via `chrome.runtime.sendMessage`. The recording functions are used by the Teach mode in order to capture all the user's actions so that the agent can learn from them, and in the Learn modality, in order to check whether the user's actions match the generated learning steps of the agent. The injected code listens for different messages from the extension and reacts accordingly, for example by starting or stopping the recording, or sending specific messages only when certain conditions are met (e.g., depending on the modality). In addition, to maintain consistency, clicks on the same cell that re-open it for editing, or move the text cursor, are not recorded. Text inputs, on the other hand, are saved only after the Enter key is pressed, when another page element is clicked, or when the cell loses focus (for example, when the user clicks on the agent).

The replaying function, on the other hand, allows the extension to perform synthetic actions inside the page. This pose some technical challenges because synthetic clicks can be treated as suspicious by the page and blocked as harmful. When the command to replay actions was sent from the extension, the clicks were not performed. To avoid this, the replay is triggered by a component injected into the page itself, which, by the page, is treated as safe. The replaying function is used in all three of the agent modalities and works by going through a list of actions (each defined by interaction type, element ID, and input value, which is empty in the case of simple clicks) and performing them, one at the time. Various delays were introduced, in order to make more understandable the actions performed.
Is worth noting that the DOM can also be manipulated by this codes. For example, while a click is performed in a cell by the agent, its color becomes yellow, helping the user follow better the actions.

5.4.2 Injected Components

Another key aspect is providing feedback directly within the current tab, rather than only through the extension interface. For this reason, several UI components were built and then injected into the web page.
Each of these components had its own configuration file, which enables it to be build as a standalone JavaScript [28] file, after being originally implemented as React TSX components. These files can then be injected into the current tab using the `chrome.scripting.executeScript` function. Several types of injected components are used. Some are simple popups that notify the user, for example when a replay finishes in Teach mode, when a step is completed by the user, or when a step is completed by the agent in Learn mode 5.6. This popups had a timer that removed these elements after few seconds, but could also be dismissed by a simple click in the page.

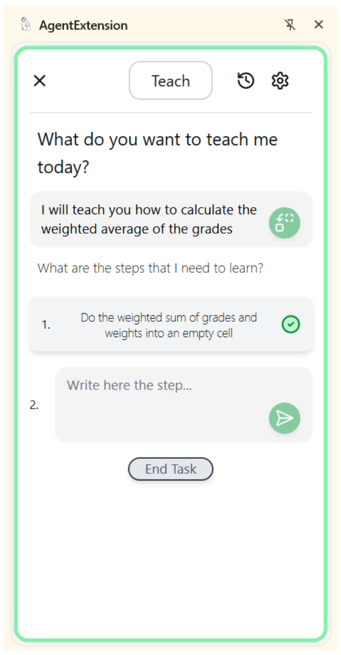
Others components are more interactive, such as dialogs shown at key moments: for instance, the dialog to start replaying actions in Teach mode or the dialog to start an automated task in Collaborate mode (Figure 5.7). Closing these dialogs sends different messages to the extension, triggering the corresponding events.

The more complex one is the Hint component. Once injected, can trigger events, by clicking the "Show me" button discussed before, but more importantly can be triggered by the user errors, changing color and showing which step was performed incorrectly by the user, making the specific action text bold 5.8.

5.4.3 OpenAI Calls

The engine underlying the agent's behavior is based on API calls to OpenAI. These calls serve multiple purposes: assessing whether the agent already has knowledge

of a given task, generating steps, hints and agent actions for automation. The implementation details of these calls are not deeply analyzed in this chapter; instead, the next chapter presents a series of experiments used to identify the most suitable model among several ones. For this reason this API calls are going to be presented briefly in the following chapter.
The most important aspect is, as cited before, all the calls are explicitly tailored to the custom spreadsheet website created for this thesis. In each prompt, the model is informed, as the first thing, that the environment is a spreadsheet-like website. Then, the prompt constrains the OpenAI model's capabilities in order to simulate a realistic learning ecosystem. The prompts were iteratively refined to produce the desired output while respecting the initial design objectives of this thesis project. Finally, the choice of model had a strong impact on the quality and consistency of the outputs. Different models produced different behaviors using the same prompts. The work presented in the next chapter analyzes these differences and reports the main results and conclusions derived from this experimentation.


AgentExtension

Teach

What do you want to teach me today?

I will teach you how to calculate the weighted average of the grades

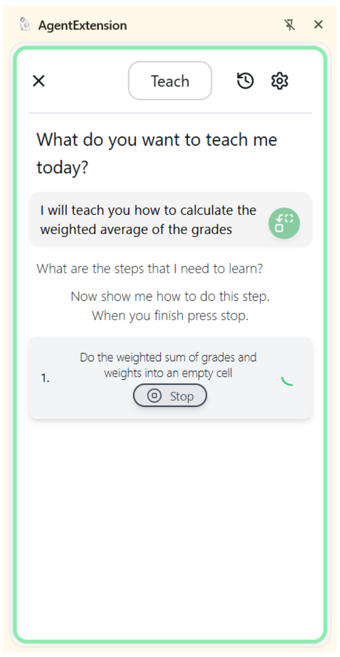
What are the steps that I need to learn?

1. Do the weighted sum of grades and weights into an empty cell

2. Write here the step...

End Task

(a) Second step insertion


AgentExtension

Teach

What do you want to teach me today?

I will teach you how to calculate the weighted average of the grades

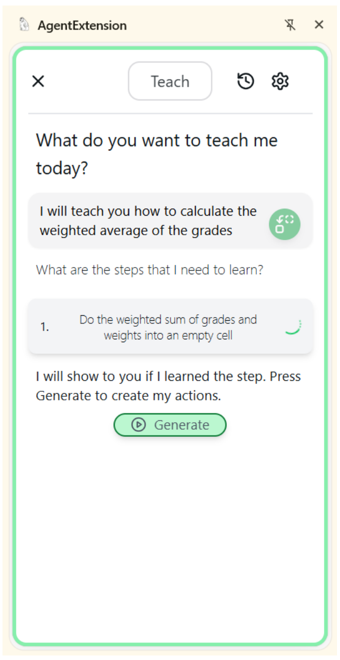
What are the steps that I need to learn?

Now show me how to do this step.
When you finish press stop.

1. Do the weighted sum of grades and weights into an empty cell

Stop

(b) Agent recording actions


AgentExtension

Teach

What do you want to teach me today?

I will teach you how to calculate the weighted average of the grades

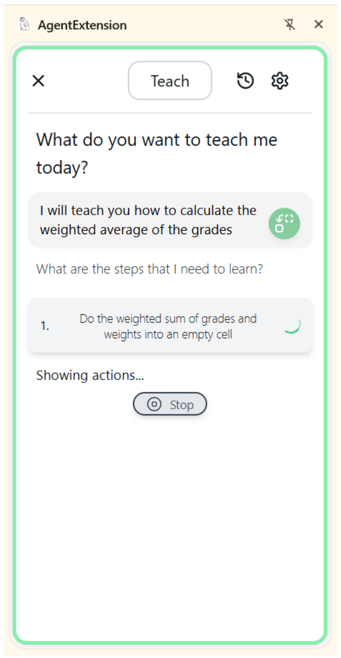
What are the steps that I need to learn?

1. Do the weighted sum of grades and weights into an empty cell

I will show to you if I learned the step. Press Generate to create my actions.

Generate

(c) Generating the actions


AgentExtension

Teach

What do you want to teach me today?

I will teach you how to calculate the weighted average of the grades

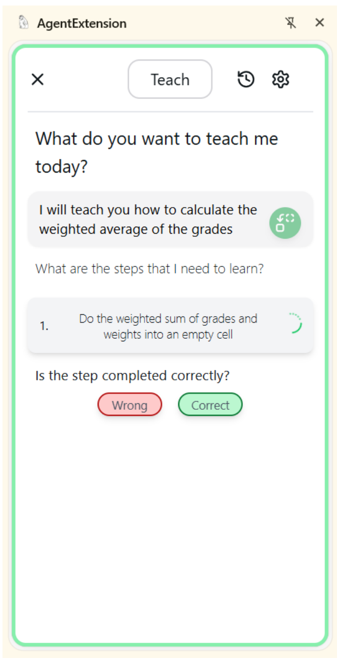
What are the steps that I need to learn?

1. Do the weighted sum of grades and weights into an empty cell

Showing actions...

Stop

(d) Showing the learned actions


AgentExtension

Teach

What do you want to teach me today?

I will teach you how to calculate the weighted average of the grades

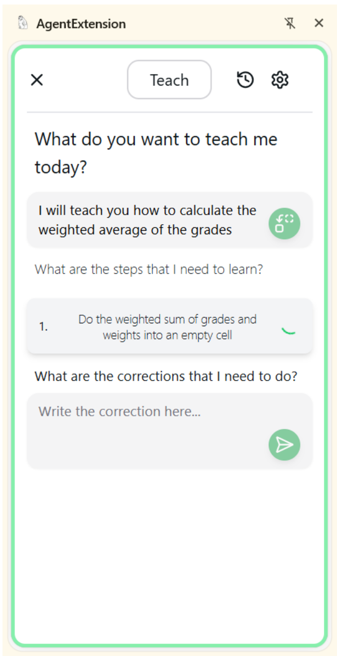
What are the steps that I need to learn?

1. Do the weighted sum of grades and weights into an empty cell

Is the step completed correctly?

Wrong Correct

(e) Feedback after ending the agent actions


AgentExtension

Teach

What do you want to teach me today?

I will teach you how to calculate the weighted average of the grades

What are the steps that I need to learn?

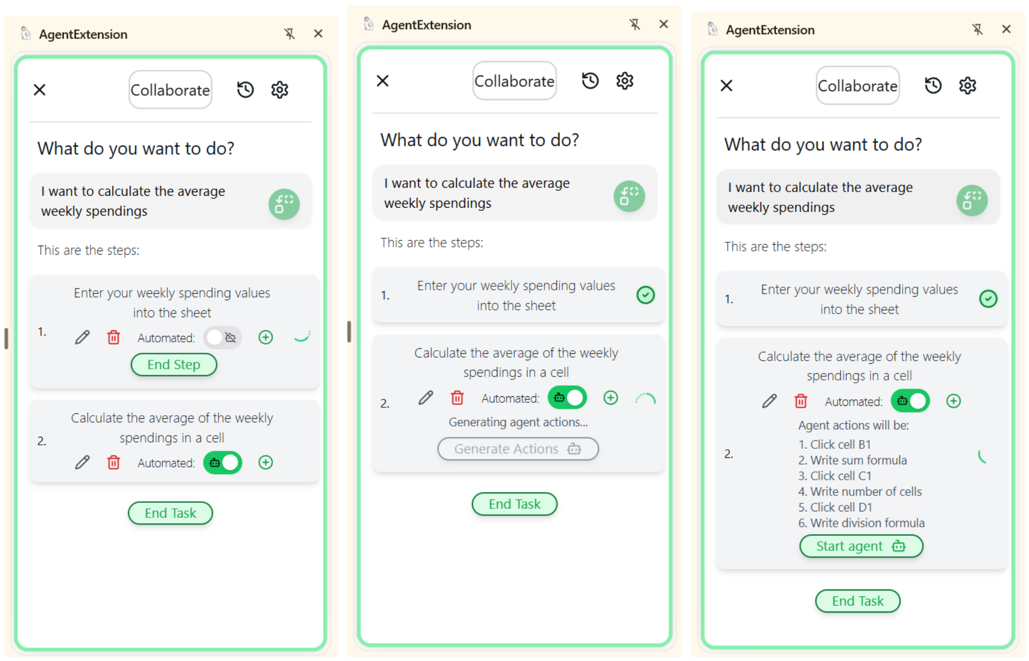
1. Do the weighted sum of grades and weights into an empty cell

What are the corrections that I need to do?

Write the correction here...

(f) Correcting the wrong actions

Figure 5.3: Variants of the Teach interface



What do you want to do?

I want to calculate the average weekly spendings

This are the steps:

1. Enter your weekly spending values into the sheet
Automated:

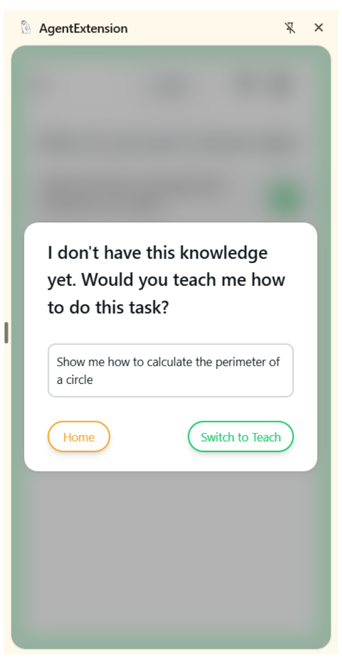
2. Calculate the average of the weekly spendings in a cell
Automated:

Generating agent actions...

Agent actions will be:
1. Click cell B1
2. Write sum formula
3. Click cell C1
4. Write number of cells
5. Click cell D1
6. Write division formula

(a) Current step not automatable (b) Generating actions for an autoamtable step (c) Agent ready to start collaborating

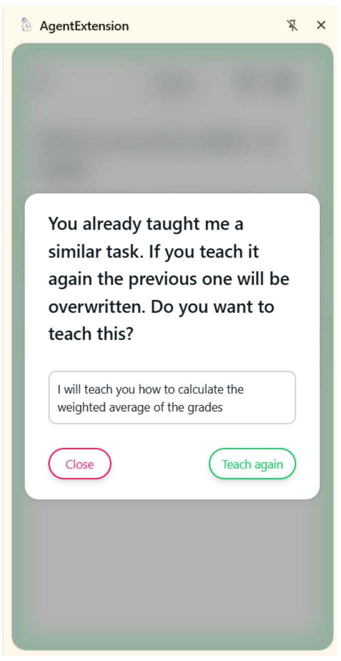
Figure 5.4: Variants of the Collaborate interface


AgentExtension

I don't have this knowledge yet. Would you teach me how to do this task?

Show me how to calculate the perimeter of a circle

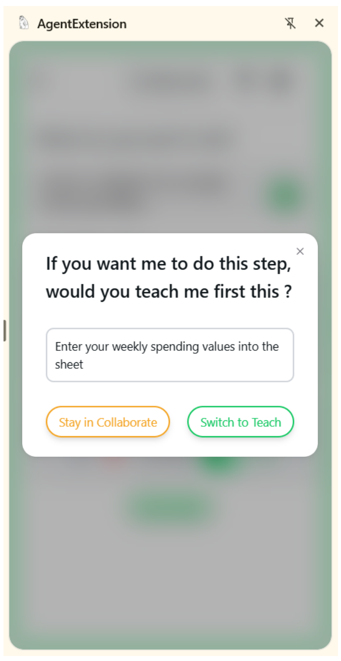
(a) Modal in Learn when a task is unknown


AgentExtension

You already taught me a similar task. If you teach it again the previous one will be overwritten. Do you want to teach this?

I will teach you how to calculate the weighted average of the grades

(b) Modal in Teach when the task was previously taught

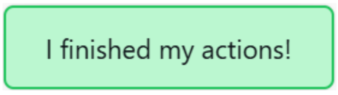

AgentExtension

If you want me to do this step, would you teach me first this ?

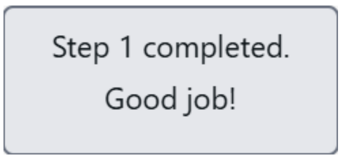
Enter your weekly spending values into the sheet

(c) Modal in Collaborate to teach an unknown step

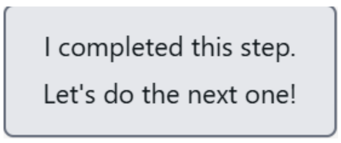
Figure 5.5: Modals used in the different modalities


I finished my actions!

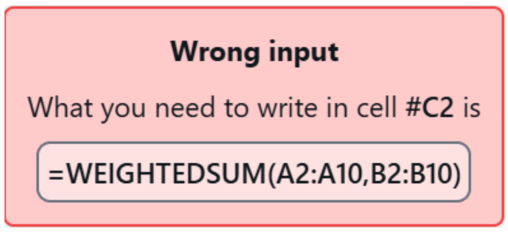
(a) Replay finished


Step 1 completed.
Good job!

(b) Step completed by the user


I completed this step.
Let's do the next one!

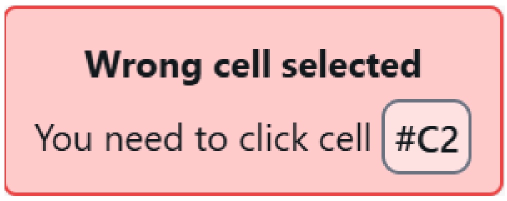
(c) Step completed by the agent


Wrong input

What you need to write in cell **#C2** is

=WEIGHTEDSUM(A2:A10,B2:B10)

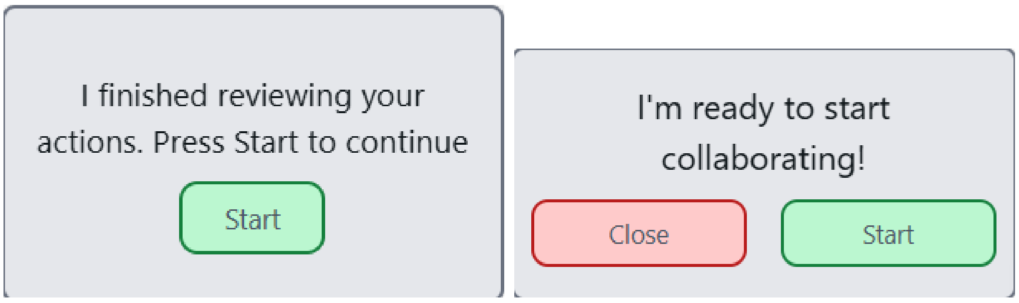
(d) Wrong input


Wrong cell selected

You need to click cell #C2

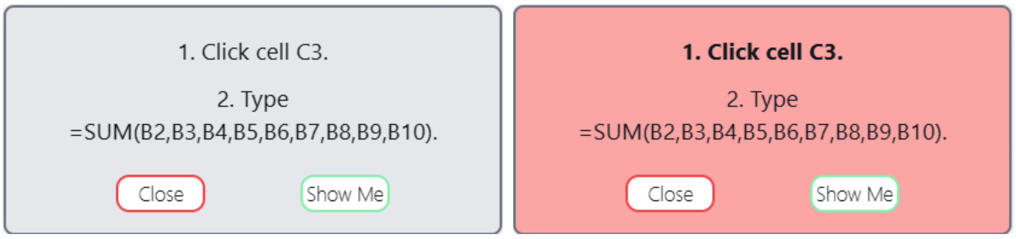
(e) Wrong click

Figure 5.6: Example of popups used by extension to communicate with the user


I finished reviewing your actions. Press Start to continue
Start
I'm ready to start collaborating!
Close
Start

(a) Start replay in Teach (b) Start automated task in Collaborate

Figure 5.7: Example of dialogs used in different modalities


1. Click cell C3.
2. Type =SUM(B2,B3,B4,B5,B6,B7,B8,B9,B10).
Close
Show Me
1. Click cell C3.
2. Type =SUM(B2,B3,B4,B5,B6,B7,B8,B9,B10).
Close
Show Me

(a) Normal hints interface (b) Hints interface after a wrong action

Figure 5.8: Hints component

Chapter 6

Model performance Analysis

With the release of GPT-5 by OpenAI [1] during the development of the project, it became relevant to understand which model provided the most suitable responses for AgentExtension. In this chapter are briefly described the prompts used in this project, the experiments performed for each of them, the results obtained, and the rationale behind the final model choices adopted in the latest version of AgentExtension.

6.1 Scope of the experiment

From the beginning of the development, the only model used to perform the AI API calls was GPT-4o of OpenaAI [1]. The prompts were iteratively refined through empirical testing, adding constraints and clarifications until the responses matched the desired behavior.Once satisfactory results were reached, the prompts were considered stable.
On the 7th of August OpenAI released their new model family GPT-5, described by the company as their "most advanced model for coding and agentic tasks." This raised the question of whether GPT-5 variants could outperform the already-integrated GPT-4o in the API calls performed
The main idea, after this release, was to systematically test the existing prompts against different models and compare them using various metrics, in order to find which model delivered the best output for each prompt. All prompts used in AgentExtension were tested, and are briefly summarized below:

- *Assess task:* used both in Learn and Teach modalities to determine whether a task is already known by the agent. It is used to block the user from re-teaching an already-known task or to detect when a user wants to learn a task that is not yet known.

- *Generate Learn steps Actions:* generates the list of steps for a Learn session. For each step, it also produces a list of hints and the sequence of actions the user must perform to complete that step.
- *Generate teach Actions:* generates actions that the agent should perform, starting from the step that the user is teaching, the current state of the table and the actions performed by the user, in order to check whether the agent has correctly learned from that demonstration.
- *Generate teach corrected actions:* used when the user reports errors after the agent's generated actions. The user provides a textual correction, and the agent updates the actions accordingly.
- *Collaborate steps:* in Collaborate mode, this prompt produces a list of steps needed to complete a task based on the current state of the table. For each step it performs an assessment (identical to the General assessment) in order for the system to understand if a step is automatable or not by the Collaborate mode.
- *Generate Collaborate actions:* generates a simplified list of action descriptions for the user (to explain what the agent will do), together with the list of actions that the agent will perform for the automation of the step.

6.2 Models chosen for the experiments

The models selected for the evaluation were GPT-4o and several configurations from GPT-5: GPT-5 with low reasoning, GPT-5-mini with low and medium reasoning and GPT-5-nano with low reasoning. Only models from OpenAi were tested. This choices were based on initial empirical tests using the refined prompts on simple tasks, comparing behavior across the available models.
GPT-4o provided good answers when generating the structured data needed for the Learn and the actions for the Collaborate modality, but it showed some weaknesses with knowledge assessment and Collaborate step generation. Tests with GPT-4o-mini revealed a noticeable decrease in response quality, despite better latency and lower costs. For this reason, GPT-4o-mini was excluded from further experiments.
GPT-5 is presented by OpenAI as particularly suitable for agentic applications. Its invocation pattern is simpler than that of GPT-4o, but it offers fewer fine-grained control features. Initial tests suggested the opposite behavior compared to GPT-4o: GPT-5 performed well on assessments and on generating Collaborate steps, but showed more issues with prompts deeply tied to actions and that required some DOM context understanding.

6.3 Experiments performed

In this section, after a brief description of each prompt, are discussed all the different test scenarios performed, providing some insights behind these choices. For each individual test case, the API call was executed 4 time in order to check whether the model produced hallucinations or inconsistent outputs when the same prompt was repeated.
Depending on the variables and interaction possibilities, different scenarios were simulated to see how each model behaved. For every single execution, the following data were saved into a file: input tokens, output tokens, cost, execution time, the parameters used for the call, the run number (from 1 to 4), and the full output.

6.3.1 Learn and Teach Assessment

Initially, there were two separate functions – one for the Teach and one for the Learn – that used almost identical prompts with minor differences, but produced the same type of output. The prompt takes as input values:

- the task the user wants to learn or teach;
- the list of tasks that belong to the agent's base knowledge;
- the list of the tasks that the agent learned from the user.

The prompts describe the environment and explain that the goal is to find if there is a match between the user's request and a tasks known by the agent. It has to do so by ignoring unnecessary details of the request and focusing only on whether the task itself is known.
The output returned from the prompt is a JSON object containing:

- *code*: a code indicating if the task is not known (1), a learned task (2) or part of the base knowledge (3);
- *match*: if a match occurred, the text of task that matched with the user request;
- *index*: if a match occurred, the index of the matching element in the corresponding list (if there was no match it returns -1);
- *score*: a similarity score associated with the match.

The similarity score was not used in the final system, but during experimentation it provided useful insight into the thresholds used by the models to identify matches. For these tests, only the user request and the list of learned tasks were varied, since the base knowledge is fixed. Twelve test cases were defined, with tasks designed to be more or less similar to existing learned tasks, in order to see how different models respond to these subtle changes.

6.3.2 Collaborate step

This function, differently from the previous prompt, both generates a set of steps and assesses if the system knows how to perform them. This prompt takes the same inputs as the Learn and Teach assessments, with the addition of the current table DOM. Including the DOM allows the call to detect whether there is a need to insert data or whether the table is already sufficiently populated. The output is a list of steps, each paired with an assessment result identical in format to the one used by the Teach and Learn assessment prompts.
Again, twelve test cases were used, following a similar approach to the Assessment prompts tests, except for the addition of the table DOM as another input value. In four of these cases, the DOM was filled with the values values to test whether the models would remove unnecessary "data insertion" steps when the table was already populated.

6.3.3 Steps Learn

This function is triggered after the assessment call determines that there is a known task the agent can use as a reference to teach the user their request. The parameters that this prompt takes as inputs are:

- the task the user wants to learn;
- the current DOM of the table;
- the list of step descriptions from the matching task;
- the table DOM at the time the agent learned that task (taken from the matching task);
- the list of actions grouped by step from the original learning session (also taken from the matching task).

The output is the most structured of all prompts used for this project. The output returned a JSON array where each element contains:

- *step*: a textual description of the step;
- *formula*: an optional formula to be inserted (only when needed);
- *hints*: a list of hints, one for each action to be performed;

- *actions*: the list of actions that the user must perform to complete the step (each being JSON object containing type of action, DOM selector and the eventual input value).

Twelve test cases were used, varying the table DOM and actions list. Preliminary tests showed that, when the first the reference column of the table for a formula was fully filled during the original learning, the model sometimes generated formulas referencing all rows, even when the current column was not full. For this reason, multiple scenarios were defined to test the model's adaptability to different table states and learned-action references.

6.3.4 Teach Actions

This function is called when the user clicks the button to generate the agent actions in Teach mode. The parameters that this prompt takes as inputs are:

- the task that the user is teaching;
- the step that the user is currently teaching;
- the current table DOM;
- the actions that the user performed to teach the agent.

The output, which is a JSON object, returns a list actions, where each action includes:

- *type*: whether the action is a "click" or an "input";
- *selector*: the CSS selector of the element where the action needs to be performed;
- *value*: the input value (present only if the type is *input*).

The test cases were four and all the parameters varied for this experiments. In two of them were simulated multiple wrong inputs to check wether the model removes the redundant and erroneous actions.

6.3.5 Teach Corrected Actions

This function is identical to the previous one, except for two parameters that are added to the prompt: a textual correction that the user provides in order to correct the agent's actions and the wrong actions that the agent performed. The output format is the same as for the Teach actions. The test cases mirror the previous four, with the addition of corrections and wrong actions as input.

6.3.6 Collaborate Actions

This prompt is used by the user to generate the agent's actions to automate a step in Collaborate mode. The prompt takes as inputs:

- the job that the user wants to automate;
- the current table DOM;
- the list of step descriptions from the matching task;
- the table DOM at the time the agent learned that task (taken from the matching task);
- the list of actions grouped by step from the original learning session (also taken from the matching task).

The output is a JSON object containing:

- *action_list*: a list of brief, user-friendly descriptions of the actions that the agent will perform;
- *agent_actions*: the detailed list of actions (each one being a JSON object containing type, selector and the optional input) to be executed by the agent.

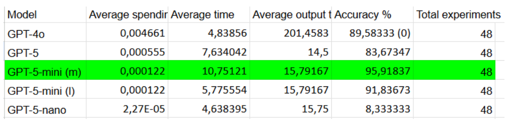
Eight test cases were used, varying all input values. The tests focused on verifying whether, from complex and multi-step jobs, the actions were generated correctly, remaining consistent with the current table state.

6.4 Results and final choices

After all the tests were executed, each output had to be examined manually, since there was no simple automatic way to consider generative differences and subtle errors. However, errors were relatively easy to detect thanks to the results being stored in a structured spreadsheet.
In this section, for each prompt, the main results are summarized with a discussion on noteworthy aspect noted during the tests, followed by the final decisions regarding model selection. In addition, to clarify, in the figures, "(m)" and "(l)" near the model chosen indicate medium and low reasoning, respectively.

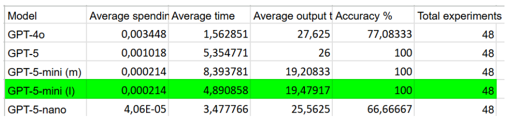
6.4.1 General Assessment

As mentioned earlier, the initially there were two separate prompts, one for Teach and one Learn, even though they produced conceptually identical outputs. After



Model	Average spendir	Average time	Average output t	Accuracy %	Total experiments
GPT-4o	0,004661	4,83856	201,4583	89,58333 (0)	48
GPT-5	0,000555	7,634042	14,5	83,67347	48
GPT-5-mini (m)	0,000122	10,75121	15,79167	95,91837	48
GPT-5-mini (l)	0,000122	5,775554	15,79167	91,83673	48
GPT-5-nano	2,27E-05	4,638395	15,75	8,333333	48

(a) Teach assessment



Model	Average spendin	Average time	Average output t	Accuracy %	Total experiments
GPT-4o	0,003448	1,562851	27,625	77,08333	48
GPT-5	0,001018	5,354771	26	100	48
GPT-5-mini (m)	0,000214	8,393781	19,20833	100	48
GPT-5-mini (l)	0,000214	4,890858	19,47917	100	48
GPT-5-nano	4,06E-05	3,477766	25,5625	66,66667	48

(b) Learn assessment

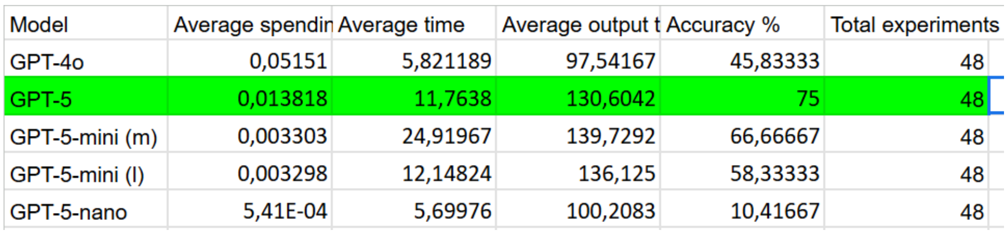
Figure 6.1: Results of the assessment functions tests for a single task

running the experiments, the results showed that for Teach, no model achieved 100% accuracy, whereas for Learn, three models did, as it is shown in Figure 6.1.

Based on these results, the best-performing model for Learn was **GPT-5-mini** with low reasoning.The choice between low and medium reasoning was guided by response time: while costs were the same, the medium reasoning variant almost doubled the response time. Starting from the Learn text, a new Generic Assessment prompt was derived and fine-tuned until it worked equally well for both Learn and Teach. The tests were repeated to confirm the correctness, achieving also a lower average latency.

6.4.2 Collaborate Step generation

Initially, a single prompt was used to both generate the steps and assess them. The results, however, were less promising than expected: the best-performing model, GPT-5 reached only the 75% of accuracy, as shown in figure 6.2.



Model	Average spendir	Average time	Average output t	Accuracy %	Total experiments
GPT-4o	0,05151	5,821189	97,54167	45,83333	48
GPT-5	0,013818	11,7638	130,6042	75	48
GPT-5-mini (m)	0,003303	24,91967	139,7292	66,66667	48
GPT-5-mini (l)	0,003298	12,14824	136,125	58,33333	48
GPT-5-nano	5,41E-04	5,69976	100,2083	10,41667	48

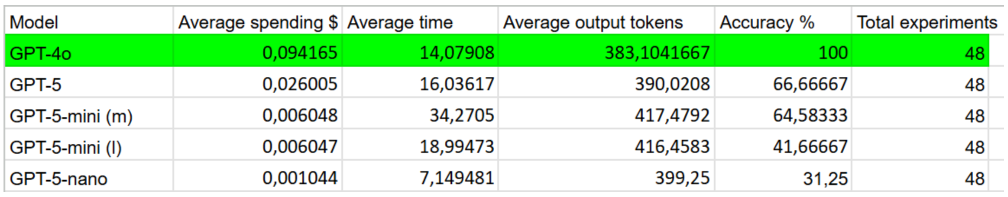
Figure 6.2: Results of the Collaborate step generation

Analyzing the errors revealed that providing the list of known tasks as part of the prompt biased the model toward splitting unknown tasks into combinations of known ones. To avoid this behavior, the logic was separated into two prompts: one to generate the list of steps to complete the task based on the current state of the table DOM, and a second one to assess the steps, similar to the generic assessment, but applied to a list of steps instead of a single task.
Considering the previous performance on the General Assessment prompt, GPT-5-mini with low reasoning and GPT-4o were both tested again. As expected, GPT-5-mini performed well, reaching the accuracy of 97%, with faster response

time and almost similar comparable costs with the previous unique prompt version. The only errors were caused by the insertion of multiple input, which, by prompt request had to be splat into separate steps; instead , what happened was that different topics inputs into the table were all merged into a single input step. Since this limitation was considered acceptable, the final model adopted for this function, composed now of two prompts, was **GPT-5-mini** with low reasoning.

6.4.3 Learn steps

For the Learn Steps prompt, the model selection was more straightforward. Being the only model with 100% accuracy (6.3) , **GPT-4o** was chosen. The second-best model, based on the outputs, was GPT-5, reaching only the 66% of accuracy. In addition, GPT-4o had one of the best average response times, making it the second-fastest model tested for this prompt, with only the drawback of its significantly higher cost.

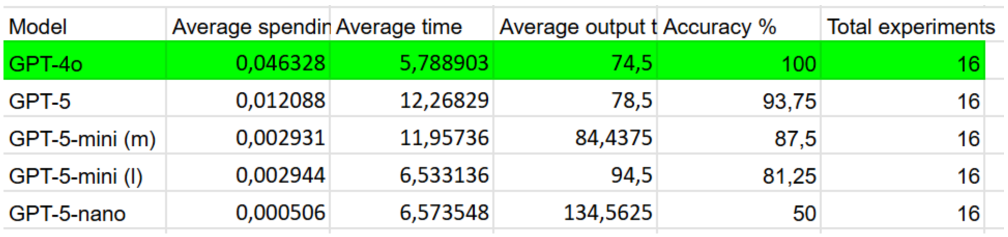


Model	Average spending $	Average time	Average output tokens	Accuracy %	Total experiments
GPT-4o	0,094165	14,07908	383,1041667	100	48
GPT-5	0,026005	16,03617	390,0208	66,66667	48
GPT-5-mini (m)	0,006048	34,2705	417,4792	64,58333	48
GPT-5-mini (l)	0,006047	18,99473	416,4583	41,66667	48
GPT-5-nano	0,001044	7,149481	399,25	31,25	48

Figure 6.3: Results of the Learn steps tests

6.4.4 Teach Actions

For Teach Actions, results showed that only one model achieved 100% accuracy: as for the previous analyzed prompt, **GPT-4o** (Figure 6.4) was chosen for this prompt . Despite its higher cost, it also provided the fastest response times for this prompt among all models, which can help the users feel more engaged during their experience.

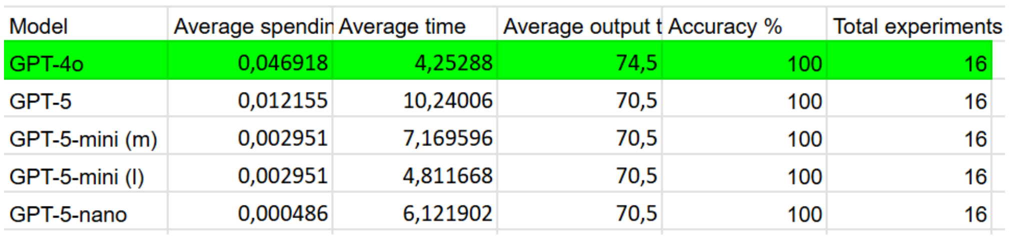


Model	Average spendin	Average time	Average output t	Accuracy %	Total experiments
GPT-4o	0,046328	5,788903	74,5	100	16
GPT-5	0,012088	12,26829	78,5	93,75	16
GPT-5-mini (m)	0,002931	11,95736	84,4375	87,5	16
GPT-5-mini (l)	0,002944	6,533136	94,5	81,25	16
GPT-5-nano	0,000506	6,573548	134,5625	50	16

Figure 6.4: Results of the Teach actions generation tests

6.4.5 Teach Corrected Actions

For Teach Corrected Actions, all models achieved 100% accuracy (Figure 6.5). However, for consistency and robustness —especially in the presence of more complex or unusual errors that could happen— the same model used for Teach Actions was used. Therefore, **GPT-4o** was also chosen for this prompt.

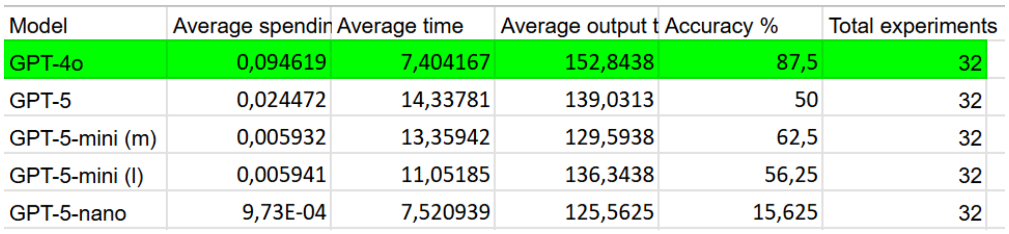


Model	Average spendin	Average time	Average output t	Accuracy %	Total experiments
GPT-4o	0,046918	4,25288	74,5	100	16
GPT-5	0,012155	10,24006	70,5	100	16
GPT-5-mini (m)	0,002951	7,169596	70,5	100	16
GPT-5-mini (l)	0,002951	4,811668	70,5	100	16
GPT-5-nano	0,000486	6,121902	70,5	100	16

Figure 6.5: Results of the corrected Teach actions generations tests

6.4.6 Collaborate Actions

For Collaborate Actions, tests confirmed that, as seen for the other complex prompts, GPT-4o provided the best-quality responses. However, the initial accuracy, which was 87,5%, was still insufficient to guarantee a high-quality service to the end-users (Figure 6.6).



Model	Average spendin	Average time	Average output t	Accuracy %	Total experiments
GPT-4o	0,094619	7,404167	152,8438	87,5	32
GPT-5	0,024472	14,33781	139,0313	50	32
GPT-5-mini (m)	0,005932	13,35942	129,5938	62,5	32
GPT-5-mini (l)	0,005941	11,05185	136,3438	56,25	32
GPT-5-nano	9,73E-04	7,520939	125,5625	15,625	32

Figure 6.6: Results of the Collaborate actions generations tests

To address this issue, the prompt itself was iteratively refined. The structure and instructions were adjusted until GPT-4o consistently produced correct outputs on the defined test cases.

6.4.7 Adoption of GPT-5 with medium reasoning

After concluding this automated test phase, additional stress tests were performed to explore edge cases and complex interaction sequences. These tests revealed some weaknesses of GPT-4o under specific conditions, especially in more demanding scenarios that were not fully covered by the original test set.

For this reason, the model for Learn Steps, Teach Actions, Teach Corrected Actions and Collaborate Actions was switched to **GPT-5** with medium reasoning. This configuration was intentionally kept outside of the test cases, since its response time is significantly higher and would have negatively impacted the users' experience with AgentExtension. Manual testing confirmed that GPT-5 with medium reasoning provided more robust behavior for these complex situations. Fortunately, pricing remains unaffected by the reasoning mode, which is affected only by the token usage.

Chapter 7

User Test

After the development was completed, it became necessary to validate the system. This chapter presents the planning and design of the user test, followed by an analysis of the results across several evaluation dimensions. These results are then analyzed to provide useful insights into the strengths and weaknesses of the proposed system, highlighting areas of success and those requiring further improvement.

7.1 Planning

In this section, all the planning behind the user test conducted after the development of AgentExtension is analyzed. Starting from the goals of these experiments, it describes how the tests were designed, the type of data collected during the evaluation, the experimental setup, the characteristics of the participants involved, and the tasks they were required to complete. The test procedure is also detailed.

7.1.1 Study Overview

Were conducted a mixed-methods within-subject study to evaluate how users perceived the different interaction modalities of AgentExtension —Teach, Collaborate, and Learn— that was developed for this thesis. This study wants to assess how the system influences users' perceptions of controllability, trust, transparency –which were important issues that web agents development face– and understanding when collaborating with an intelligent UI assistant.
Was kept a focus on the main issues that agentic figures had based on the table of chapter 4 (4.1) where the main issues were understandability , controllability of the agent at any moment, and user trust on these systems. The study was done on a Spreadsheet-like web interface, as said before, developed only for this testing

purpose. In this website were conducted the tests of AgentExtension and all of its modalities.

7.1.2 Experimental Design

Each participant experienced the three interaction modalities (Teacher, Companion, Student) in counterbalanced order to mitigate learning and fatigue effects, allowing to collect unbiased data from the users. The variables that were kept in consideration for the analysis were the following:

- Independent Variable: Interaction Modality (3 levels).
- Dependent Variables: Perceived controllability, perceived understandability, trust, transparency, usability (subjective), and task performance (objective).
- Mixed Data Sources: Behavioral logs, questionnaires, and post-study interviews.

This data was then analyzed to find both values and issues inside the system and assess its correctness.

7.1.3 Prototype and Logging

The prototype was the Google Chrome AgentExtension that was developed with the three modalities, capable of multimodal interaction (textual, visual, and capable of interacting with the current tab) and the browser-based Spreadsheet like website. This website, as said in the previous chapters, was created to provide a more simple version of already existing websites (Google Sheets, i.e.), keeping the base functionalities (formulas, and cell manipulation) and allowing also a more lightweight communication with the OpenAI API, thanks to a lighter DOM.
For research reproducibility and in-depth analysis, the system recorded:

- User actions (cell edits, selections, formula insertions, time per action).
- Agent generations (prompts, intermediate reasoning traces, final actions)
- Agent actions (cell edits, selections, formula insertions)
- Dialog events (invocation, correction, stop commands and finish commands).
- System events (errors, delays, user interventions, cells cleaning).

All logs were timestamped and stored per condition, enabling replay and fine-grained temporal analysis of user–agent interaction dynamics.
The tests were in addition recorded using a screen recording software. This was done to let testers feel less observed and to allow a more natural behavior. For the experiments was avoided camera recording that would have created some biases in their interactions with the system, making the users feel more observed. The screen recording furthermore helped to keep track also of all different kinds of interactions happening during our examinations, especially of errors of the system. In addition the audio was recorded, allowing for a better recollection of useful insights provided during the think-aloud and final interview. At the end of each session, the facilitator and participant jointly reviewed selected excerpts of the agent's behavior to elicit reflection-based feedback (similar to co-discovery or retrospective think-aloud).

7.1.4 Participants and Setup

For the experiments were recruited 18 participants (university students and office workers) with varying experience in spreadsheet use and AI systems. Participants were screened for basic digital literacy and familiarity with simple formula creation and also with familiarity with AI. This information will then be used to analyze the result using this to gain better insights.
All the testers, before starting the experiments, signed an informed consent allowing also to record their interactions and the audio of the experiments through a screen recording software. After this their demographic data was recorded (age, gender, job, prior AI experience, prior Agent experience, Excel/Google Sheets expertise).
Participation lasted approximately 60 minutes and took place in a lab setting using the experimenter's laptop, allowing the use of an external mouse for users that were not comfortable using the touch pad. The laptop was already configured with the extension already installed and the API key already inserted in the settings page of AgentExtension.

7.1.5 Tasks

For the experiments, each participant had to complete each one of the following tasks, which are related to the specific modality :

1. **Learn Mode:** The agent explains and guides the participant in *computing the area of a triangle.* The table is already filled with the base and height.
2. **Collaborate Mode:** The agent collaborates with the participant to *calculate the average of their weekly spendings.* The table at the start of the experiment is empty

3. **Teach Mode:** The participant demonstrates to the agent how to *perform the weighted average of university grades*, correcting it if necessary. The table at the start of the experiment is already filled with the grades and weights.

The choice of the task order was counterbalanced, as said before, in order to limit learning fatigue, allowing the collection of unbiased data. These tasks were chosen to allow both experienced and unfamiliar users with a spreadsheet environment to have a challenging assignment. All the tasks consisted of multiple steps and allowed for different scenarios to unfold and a variety of data to be collected, allowing better insights.

7.1.6 Procedures

1. *Introduction (5 min):* The facilitator presents the study goals, obtains consent, provides a short overview of the agent's capabilities and shows the basic functionalities of the custom Spreadsheets website.

2. *Training (5 min):* Participants performed a brief tutorial task to familiarize themselves with invoking, stopping the agent and understanding its interface and components.

3. *Task Phase (30 min):* Participants completed the tasks in counterbalanced order. A think-aloud and cooperative evaluation approach was used for each task: the facilitator intervened only when the participant was stuck due to agentic errors deriving from prompting results, noting those occurrences for later analysis or when system errors occurred allowing the experiment to recover from the previous state.
   The cooperative approach was used only when testers had no proficiency with a spreadsheet environment, letting them interact as much as possible alone with the agent and helping only when necessary.

4. *Retrospective Review (10 min):* The participant and facilitator review selected agent actions and generations to reflect on clarity, control, and appropriateness.

5. *Post-condition Questionnaires (1 min):* After each task, participants responded to a SEQ (Single Ease Question) about the perceived task difficulty.

6. *Post-condition Questionnaires (10 min):* After each test, participants completed:

   - SUS (System Usability Scale) – perceived usability.
   - Trust, Controllability and Transparency Scales – custom Likert items (table reference).

7. *Final Semi-Structured Interview (5 min):* Participants were asked a final interview with questions regarding their experience with the different agent modalities and letting them provide some feedback on their experience with it.

7.2 Result analysis

This section presents the analysis of the experimental results. Several dimensions were considered to obtain a well-rounded evaluation of the whole system. Table 7.1 reports in detail what are the dimensions used, the source or instrument for each dimension, and an example of the metrics considered.

Dimension	**Source / Instrument**	**Example Metric**
Usability	SUS (A)	SUS total score
Task difficulty	SEQ	1–7 scale
Controllability	Custom Likert (B)	1-5 scale
Trust	Custom Likert (B)	1-5 scale
Transparency	Custom Likert (B)	1-5 scale
Performance	Logs	Task outcome, error rate, time completion (time analyzed also depending on proficiency with spreadsheet environment and AI systems, to obtain more homogeneous data)
Behavior	Logs	Number of corrections, agent stops, help requests to interviewer
Reflection	Interview (C)	Thematic codes on trust and understanding

Table 7.1: Overview of Dimensions, Sources, and Example Metrics

7.2.1 SEQ score

After each task, users answered a **Single Ease Question (SEQ)** regarding the perceived difficulty of the task, using a scale from 1 (Very Difficult) to 7 (Very Easy).
In the *Learn* modality, results confirmed expectations: users found this modality easy to understand and interact with. With the highest average score (6.611, Figure 7.1), most users felt comfortable and did not ask for help during the progress of the task. Only one user rated it a 5, which still corresponds to the minimum positive evaluation.

The *Collaborate* modality also received positive feedback, with an average score of 6.222. As shown in figure 7.2, most users rated the difficulty as a 6, with only

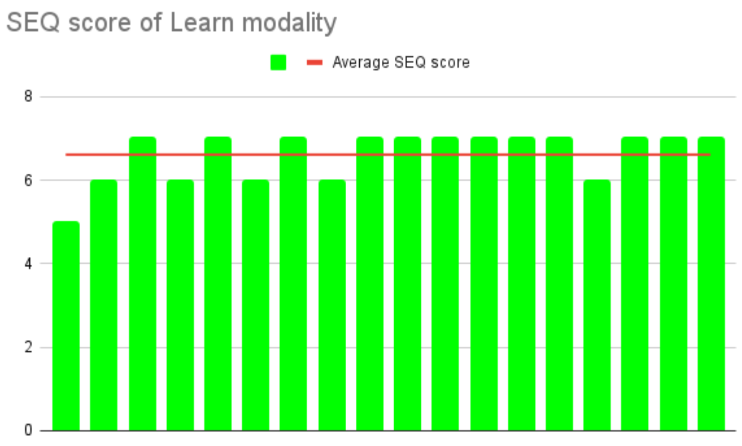


Figure 7.1: SEQ scores for Learn modality

two users assigning a 4 and 5. As expected, this modality was perceived as easy, although a few users expressed uncertainty regarding the automation switch and functionality.

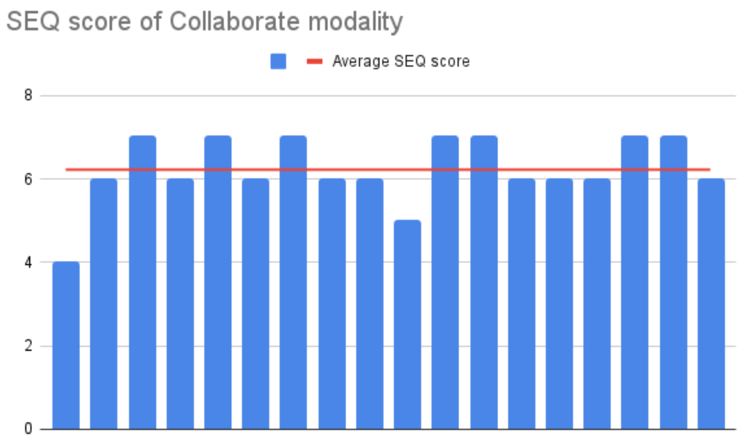


Figure 7.2: SEQ scores for Collaborate modality

For the *Teach* modality, a lower score was anticipated and indeed observed. As it is intended, users approach this modality only after acquiring some familiarity with the system, as it requires them understand the system's functionalities and mental model before teaching. This time the average score is 5,222, which is still a passing grade, however, compared to the other modalities, and considering that most of the users had to be guided in the completion of the task, it shows some technical issues within this mode. The distribution was wider, with most ratings being 5 or 6. Three users gave a score of 7, but the presence of several 4s and even two 3s, highlight the difficulties users face when interacting for the first time with this modality (Figure 7.3).

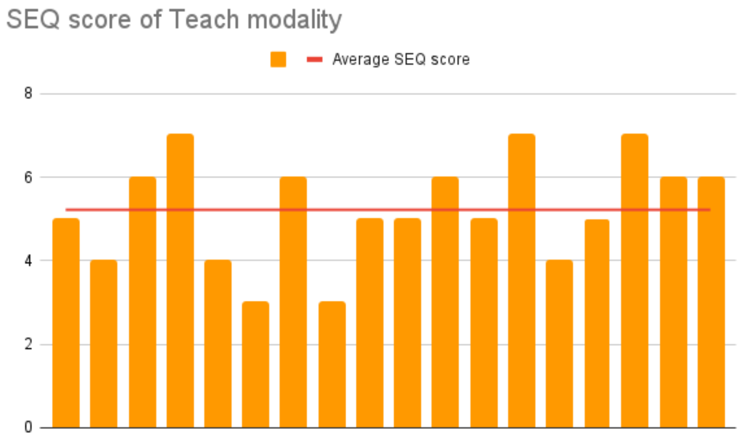


Figure 7.3: SEQ scores for Teach modality

7.2.2 SUS score

The SUS results further confirm the positive trend, with an average score of 79.58. Only three scores fell below the SUS threshold of 68, as it shows figure 7.4. Some user motivated this low grades based on the fact that they were unfamiliar with AI systems and therefore less trusting of such technologies. Others instead, considering mostly the Collaborate and Learn modality, considered the system helpful, which motivate their higher grades. With more than thirteen scores above 80, the general feedback is more than positive. Nevertheless, to reach the excellent grade score, which score is above 80, improvements are fundamental to be made. When analyzing per-question averages, all odd-numbered items scored above 3, with Questions 3, 4, and 7 (which are illustrated in appendix A) scoring above 4, which is a notable achievement. The hightest-scoring was question 5, which is about the functions being well implemented in the system, with an average of 4.68. Even-numbered questions, which are the negative statements, also scored 2 or lower. Analyzing the questions that had the most negative impact on the final scores, considering the type of the question: the weakest odd-numbered was Question 9, concerning user confidence while using the system, with an average of 3.31; Among the even-numbered items, the highest value corresponded to Question 10, which states that users would need to learn a lot before starting to use the

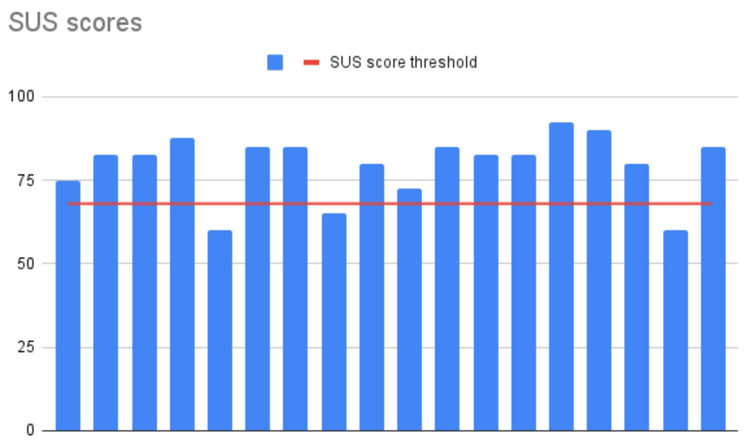


Figure 7.4: Results of the corrected Teach actions generations tests

system, scoring an average of 2.
These results provide encouraging evidence of the quality achieved in the final version of the this thesis project, allowing also to find the critical aspect to focus when improving the system.

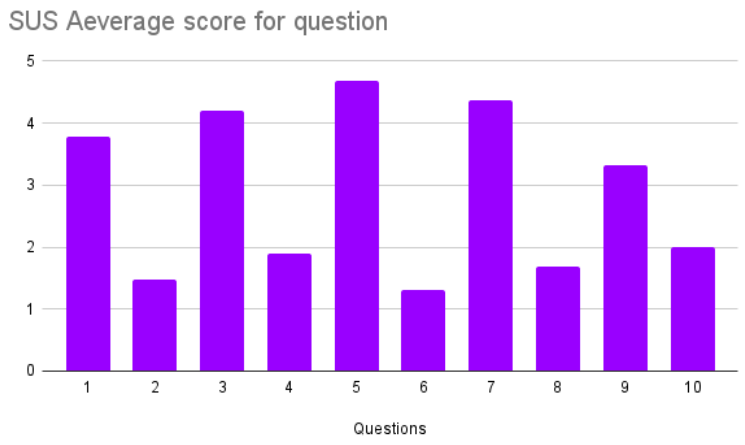


Figure 7.5: Average score of each question of the SUS

7.2.3 Custom Likert

As discussed in section 2, in the subchapter about the main issues within agents, the main challenges that this type of systems face are about, *controllability* provided to the user, *trust* that the user has on the agent's capabilities and *transparency* of the system's decisions and actions. For this reason, an additional questionnaire was developed (some optional depending on the user's interactions, as it can be seen in appendix B) for each one of this challenges.
In figure 7.7 are illustrated the average scores for each question. The columns in blue are the questions about *controllability*, the columns in red are about *trust* while the columns in yellow are about *transparency*.

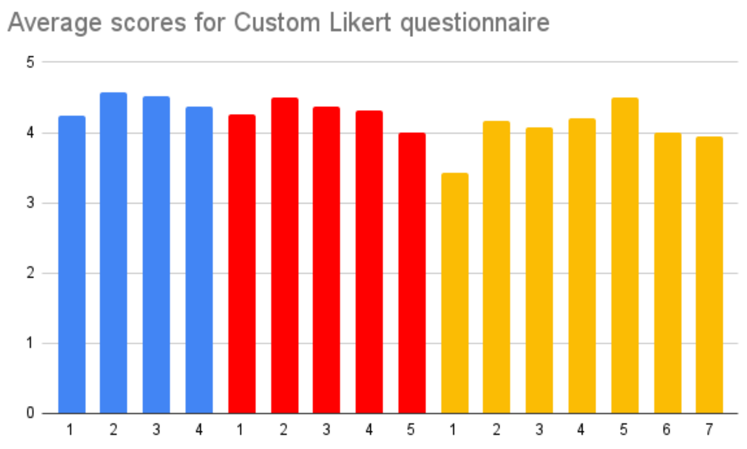


Figure 7.6: Custom Likert average scores for each question

Regarding Controllability, in average, the users felt in control of the agent, thanks to the functionalities provided. This dimension obtained the highest overall average of 4.42, with all scores above 4. In second place followed Trust, with an average of 4.29, and the lowest question still scoring a 4. Transparency scored the lowest, with an average still of 4.04, which is still a great result, however suggesting to improve it, since some users found the system's capabilities and limits were not so clear.

Figure 7.7 displays, instead, the aggregated sum for each user of the average of each group of question. This graph will later be used in combination with other data and results to derive additional insights. From the figure, it appears that, aside from a few exceptions, users did not experience major issues related to the typical challenges faced by web agents.

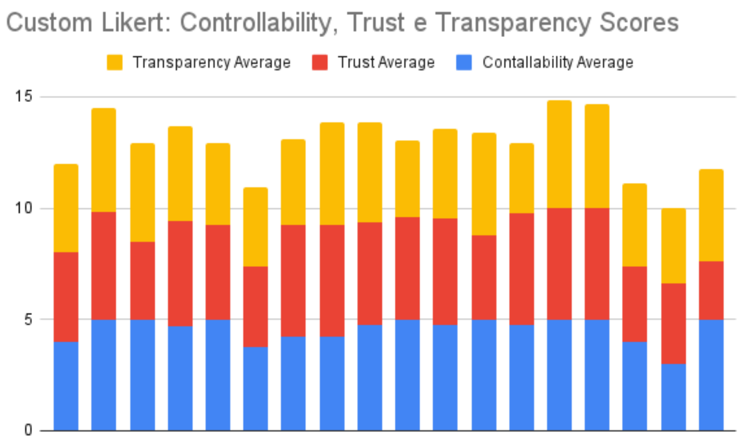


Figure 7.7: Custom Likert aggregated graph

7.2.4 Final interview

The questions used for the final interview are reported in the appendix C.For the Learn modality, the questions focused on the injected interface components, asking users for feedback on their usefulness. Both questions were asked only if these components were activated during the experiments. Regarding the error messages, None of the users found them intrusive; however, two suggested increasing their display duration to better understand what went wrong. One of the users encountered a bug that caused the error message to appear above the "completed step" message, creating some confusion about what was happening. This issue will be addressed in future versions of AgentExtension. All the users that used the "Hint" component found it helpful, and No addition feedback were given for this component.
For the Teach modality, the interview results highlighted the difficulties users faced when completing the task. Eight users reported that writing the task steps was challenging, explaining that they would not have understood how to write them without the provided guidance. This also included splitting the task into multiple steps. Users noted that the beginning was particularly confusing, as they were unsure how to formulate the steps, and more importantly, what to write inside the text. However, after submitting the first step, most of the users gained confidence with the system and the teaching process; to support this, five users described the teaching procedure as confusing, though only during the initial phase.
Regarding the Collaborate modality, some users initially found the buttons unclear, particularly the switch related to the automation of the step. Although these caused doubts at the start, most users reported that the meaning of the buttons

became clear once the the button was clicked. Overall, the majority found them understandable.

7.2.5 Result considerations

After running all the experiments, it was also useful to analyze the time needed by each user to complete the tasks. As expected from the feedback, the Learn task was the easiest and the fastest to complete, as it is shown in figure 7.8, with an average of 2:20 minutes. An important observation concerns the delay between sending the input and the user's first interaction with the environment. Since the system generally requires 40 to 50 seconds to generate an answer, users interacted with the interface after an average of 1:02 minutes, which indicates how immediate is for a user to understand what they need to do. Only two users took more than 4 minutes, which can be considered exceptional cases, caused probably by the unfamiliarity with the environment.

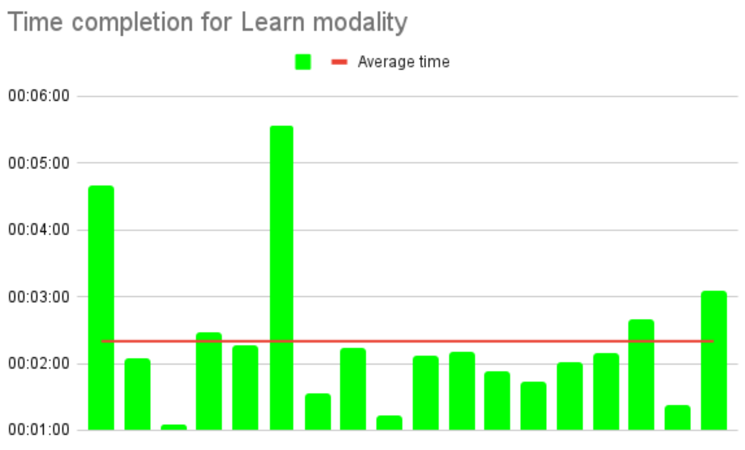


Figure 7.8: Learn task time completions

A similar behavior was observed for the Collaborate task, with an average completion time of 3:57 minutes. Considering also the time for the agent to generate the actions, which usually is around 50 seconds , for the users was quite intuitive to complete the task. Considering the 20 seconds required for the agent to generate the actions, users initiated their first interaction after an average of 45 seconds, again indicating that the modality was understandable. Again, only two users took more than 6 minutes to complete the task, so the distribution of the time completion was homogeneous.

The Teach task yielded very different results, confirming the higher difficulty of this modality. The average completion time of this modality was 11:19 minutes,

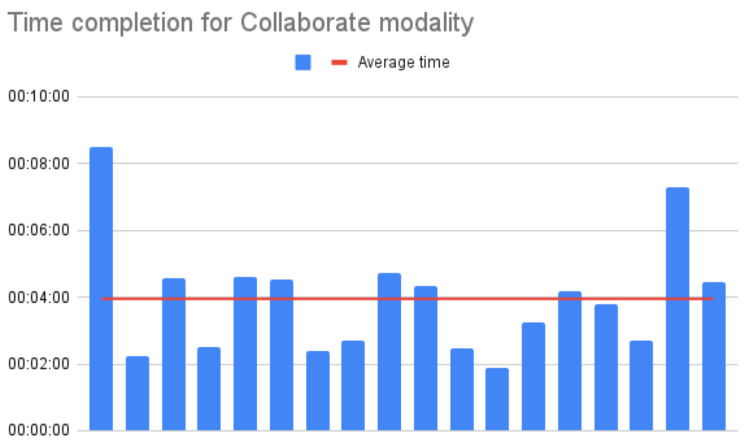


Figure 7.9: Teach task time completions

which is significantly longer than the pervious tasks. The time distribution is not homogeneous, ranging from a minimum of 4:16 minutes to a maximum of 24:14 minutes, which is a considerable range of values. Additionally, after accepting the task and confirming the intention to teach the task again, users waited an average of 2:24 minutes before submitting the first step. This delay is substantially higher, considering the 10 seconds for the assessment and an average of 15 seconds to close the "Teach again". Part of this waiting time was caused by the necessity of understanding how to formulate the step. However, the longest delay occurred after users submitted the step: all participants were misled by the spinner displayed next to it, assuming that the agent was still generating a response and therefore waiting unnecessarily.

Although data was collected regarding spreadsheet proficiency and weekly AI usage, the distribution shown in figure 7.11 does not reveal any clear correlation between these factors and the various task completion times recorded. The dataset is also too limited to draw reliable conclusions; therefore, additional testing will be necessary to obtain more meaningful insights.

in conclusion, the overall feedback is highly positive, with only 2 tasks out of 54 not completed correctly. The most significant issue identified by all the users relates to the spinner component, which was used inappropriately in all the modalities to identify the current task and led users to believe that the agent was still generating a response.
A bug was also discovered in the Learn modality: users who confirmed an input by clicking another cell unintentionally triggered first the step completion that the correction for the next step, overlapping this two popups. This should be fixed in future versions. Another recurring observation was that some users clicked buttons

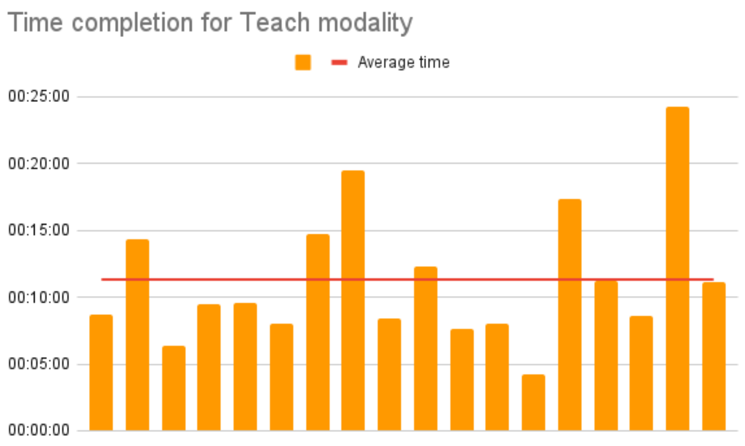


Figure 7.10: Teach task time completions

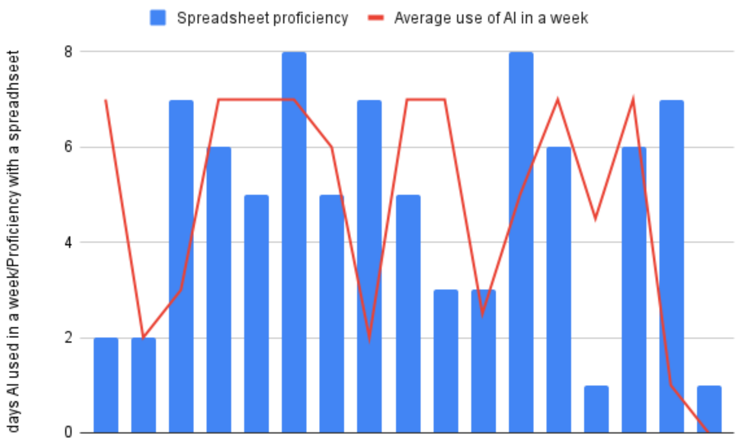


Figure 7.11: User proficiency graph

very quickly without reading system messages, which occasionally led to errors. Finally, user noted that the Teach modality does not provide a explicit feedback to the user (e.g., a message or a popup) that the taught task was successfully stored. Adding such feature would help the users understand the correct behavior or the instances of errors while using of AgentExtension.

Chapter 8

Conclusion

This final chapter presents the conclusions of the thesis, following a brief summary of the work conducted. Focusing on both the development process and the results of the user tests, this chapter analyzes the data and feedback collected throughout the project and discusses the practical need for a system such as AgentExtension. Drawing on the outcomes of the evaluation, it concludes with several suggestions for future developments.

8.1 Summary of the project

This thesis aimed to develop a three-modality web agent capable of learning from the users, teaching them and collaborating with them to accelerate task completions.
Staring from the current literature and from an analysis of state-of-the-art web agents, a set of guidelines was derived, helping to address the main issues encountered while designing such systems - with a particular focus on controllability, transparency and trust. Using the guideline table proposed by Amershi et al. [17], each modality of the system was contextualized, and real implementation examples were provided. After deciding to implement Agentextension as a Google Chrome extension, prototype design began with a clear understanding of the environment in which the agent would have been tested, which was a spreadsheet-based web interface. For this part was stimulating thinking about the different modalities functionalities and how they integrate within the whole system.
The prototype helped identify the necessary technologies. Once these were selected, development proceeded, with the UI designs following the initial prototype, along with several refinements. One of the main demanding tasks faced while developing AgentExtension was integrating a web interface into a Chrome extension while enabling it to communicate with and manipulate the DOM.

In parallel with the GUI development, prompt engineering was carried out to design and refine the prompts used for OpenAI API calls, enhancing the agent with AI capabilities. To ensure high-quality responses that empower the system, several OpenAI models were tested and compared based on output quality, response time, and cost. The final models were selected using this factors as a scale of measurements, with a focus on the output quality and correctness.
The second goal of the thesis was to validate the system using heuristics and guidelines derived from the literature studied at the beginning of this work. User tests were therefore designed to evaluate each one of the modalities with a challenging task, allowing users to test AgentExtension's functionality and assessing its usability.
A total of 18 participants tested the system, providing an overall positive feedback.The SUS scores were high (with an average of 79.58), and only two tasks out of 54 were not completed successfully. The tests also helped identifying the main issues within the system: the inappropriate use of spinners, which users misinterpreted as indicators that the agent was generating a response; and the need for a more explanatory interface in the Teach modality, helping users understand, in the first place, how the modality works and then how to write and structure tasks independently.
In conclusion, this thesis demonstrated that a user-centered approach, combined with the support of existing guidelines and heuristics, can significantly improve the usability of agentic systems, which is further confirmed by the positive results of the tests. At the same time, the research demonstrated the importance of refining and expanding such guidelines within a rapidly evolving research field.

8.2 Future developments

Several bugs and usability issues emerged during the evaluation and should be addressed in future versions of AgentExtension. First, there is the necessity to remove the spinners that indicate the current task, because all the users found them confusing. Second, the interface of the Teach modality needs to be re-designed. As seen during the experiments, the focus of the users was not guided towards the elements meant to clarify the teaching process, so re-thinking this interface would improve its explainability. Additionally, more feedback messages should be introduced, and users should be allowed to edit tasks and actions even in Teach mode, which both are a violation of the Nielsen's heuristics [18].
Looking further ahead instead, an interesting and challenging direction would be integrating the system with image recognition and web-automation capabilities. At present, the interface only works within the spreadsheet website designed for this project, and the prompts are tailored specifically to that environment. A significant

next step would be creating a system that can interact with the entire web and work in real and dynamic environments. Adding a browser automation library, such as Puppeteer [29], would allow the agent to perform more complex actions. The challenge would then be adapting the model to generate correct actions to navigate the web. Another important challenge concerns shared knowledge among users. Currently, the agent known tasks are saved in the local storage, meaning there is no need mechanism for detecting or preventing harmful operations, since there is no real risk. A future system should incorporate functionalities capable of detecting and blocking potentially dangerous interactions to ensure safe operation. This would prevent users from teaching harmful actions as standard tasks, in order to protect all users who interact with the system.
The results of the user tests highlighted the good quality and potential of AgentExtension. In an environment where AI technologies and web agents are getting increasing attention, this project represents a promising framework for future research and further development.

Appendix A

SUS questionnaire

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Appendix B

Custom Likert

Controllability

1. I could easily interrupt the agent when repeating actions in Teach mode.
2. I could easily dismiss the agent after the task was completed.
3. The agent responds appropriately when I try to guide it in Teach mode.
4. I feel in control of the interactions with the agent.

Trust

1. I trust the agent's reasoning when generating the steps in Learn mode.
2. I trust the agents when playing actions on the screen for the Learn mode.*
3. I trust the agents when replaying actions on the screen for the Teach mode.
4. I can rely on the agent performing steps on my behalf in Collaborate mode.
5. I would be comfortable using this agent without constant supervision.

Transparency

1. The capabilities and limits of the agent were clear.
2. The agent explains its actions in a way I can understand.
3. The agent explains the Errors in Learn mode in a way that I can understand.*
4. I understand why the agent provides specific steps and hints in Learn mode.*
5. I understand what are the actions that the agent performs in Learn mode.*

6. I understand why the agent generates the actions for the Collaborate step.
7. I understood what the switch in the Collaborate mode did.

The questions with the "*" symbol are optional questions, since they depend on the events that happened during the tests.

Appendix C

Final interview

Learn modality related questions

1. Did you find the error messages too invasive? *
2. Did you find the hints helpful? *

Teach modality related questions

1. Was it easy to write the steps for the task? Was it clear to you how to split the task into multiple steps?
2. Was the process of teaching a step confusing?

Collaborate modality related questions

1. Did you find it difficult to understand what the buttons did for each step?

The questions with the "*" symbol are optional questions, since they depend on the events that happened during the tests.

Bibliography

[1] OpenAI. *OpenAI.* 2015. URL: `https://openai.com/` (cit. on pp. 1, 5, 54).

[2] Anthropic. *ClaudeAI.* 2023. URL: `https://claude.ai/` (cit. on p. 1).

[3] Apple. *macOS.* 2001. URL: `https://www.apple.com/it/os/macos/` (cit. on p. 1).

[4] Figma, Inc. *Figma: the collaborative interface design tool.* 2016. URL: `https://www.figma.com/` (cit. on pp. 4, 14).

[5] Juan Pavón and J Corchado. «Agents for the web». In: *International journal of Web engineering and technology* 1.4 (2004), pp. 393–396 (cit. on p. 5).

[6] Sean Luke, Lee Spector, David Rager, and James Hendler. «Ontology-based web agents». In: *Proceedings of the first international conference on Autonomous agents.* 1997, pp. 59–66 (cit. on p. 5).

[7] Yujia Qin et al. «UI-TARS: Pioneering Automated GUI Interaction with Native Agents». In: *arXiv preprint arXiv:2501.12326* (2025) (cit. on p. 6).

[8] YiBing Lin Xiao Zhou Tao Yu. *Midscene.js: Your AI Operator for Web, Android, iOS, Automation Testing.* 2025. URL: `https://github.com/web-infra-dev/midscene` (cit. on pp. 6, 34).

[9] Gagan Bansal, Jennifer Wortman Vaughan, Saleema Amershi, Eric Horvitz, Adam Fourney, Hussein Mozannar, Victor Dibia, and Daniel S Weld. «Challenges in human-agent communication». In: *arXiv preprint arXiv:2412.10380* (2024) (cit. on pp. 6, 7, 10, 11, 17, 25, 26).

[10] Anthony L Baker, Shan G Lakhmani, and Jessie YC Chen. «Human-Agent Teaming». In: *Human-Computer Interaction in Intelligent Environments.* CRC Press, 2024, pp. 333–363 (cit. on pp. 6, 19, 20, 24, 32).

[11] Yingxuan Yang et al. *Agentic Web: Weaving the Next Web with AI Agents.* 2025. arXiv: `2507.21206 [cs.AI]`. URL: `https://arxiv.org/abs/2507.21206` (cit. on p. 7).

[12] Shuning Zhang, Jingruo Chen, Zhiqi Gao, Jiajing Gao, Xin Yi, and Hewu Li. *Characterizing Unintended Consequences in Human-GUI Agent Collaboration for Web Browsing.* 2025. arXiv: `2505.09875 [cs.HC]`. URL: `https://arxiv.org/abs/2505.09875` (cit. on p. 7).

[13] Jingyu Tang et al. *Dark Patterns Meet GUI Agents: LLM Agent Susceptibility to Manipulative Interfaces and the Role of Human Oversight.* 2025. arXiv: `2509.10723 [cs.HC]`. URL: `https://arxiv.org/abs/2509.10723` (cit. on p. 8).

[14] Justin D. Weisz, Jessica He, Michael Muller, Gabriela Hoefer, Rachel Miles, and Werner Geyer. «Design Principles for Generative AI Applications». In: *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems.* CHI '24. Honolulu, HI, USA: Association for Computing Machinery, 2024. ISBN: 9798400703300. DOI: `10.1145/3613904.3642466`. URL: `https://doi.org/10.1145/3613904.3642466` (cit. on pp. 9, 24, 25, 27).

[15] Nazli Cila. «Designing Human-Agent Collaborations: Commitment, responsiveness, and support». In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems.* CHI '22. New Orleans, LA, USA: Association for Computing Machinery, 2022. ISBN: 9781450391573. DOI: `10.1145/3491102.3517500`. URL: `https://doi.org/10.1145/3491102.3517500` (cit. on pp. 10, 22).

[16] Q. Vera Liao, Daniel Gruen, and Sarah Miller. «Questioning the AI: Informing Design Practices for Explainable AI User Experiences». In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems.* CHI '20. Honolulu, HI, USA: Association for Computing Machinery, 2020, pp. 1–15. ISBN: 9781450367080. DOI: `10.1145/3313831.3376590`. URL: `https://doi.org/10.1145/3313831.3376590` (cit. on pp. 12, 25).

[17] Saleema Amershi et al. «Guidelines for Human-AI Interaction». In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems.* CHI '19. Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–13. ISBN: 9781450359702. DOI: `10.1145/3290605.3300233`. URL: `https://doi.org/10.1145/3290605.3300233` (cit. on pp. 12, 14, 77).

[18] Nielsen Norman Group. *10 Usability Heuristics for User Interface Design.* 1998. URL: `https://www.nngroup.com/articles/ten-usability-heuristics/` (cit. on pp. 21, 22, 26, 35, 78).

[19] Browserbase. *Browserbase: A web browser for AI agents applications.* 2024. URL: `https://www.browserbase.com/` (cit. on p. 33).

[20] Browserbase. *Stagehand: A browser automation SDK built for developers ...* 2024. URL: `https://www.stagehand.dev/` (cit. on p. 34).

[21] Google LLC. *Google*. 1997. URL: `https://www.google.com/` (cit. on p. 41).

[22] Meta. *React*. 2013. URL: `https://react.dev/` (cit. on p. 41).

[23] Microsoft. *TypeScript: JavaScript With Syntax For Types*. 2013. URL: `https://www.typescriptlang.org/` (cit. on p. 41).

[24] VoidZero Inc. *Vite | Next Generation Frontend Tooling*. 2020. URL: `https://vite.dev/` (cit. on p. 41).

[25] NextUI Inc. *HeroUI (Previously NextUI) - Beautiful, fast and modern React ...* 2025. URL: `https://heroui.com/` (cit. on p. 41).

[26] Tailwind Labs. *Tailwind CSS - Rapidly build modern websites without ever ...* 2019. URL: `https://tailwindcss.com/` (cit. on p. 41).

[27] GitHub Inc. *GitHub · Change is constant. GitHub keeps you ahead. · GitHub*. 2008. URL: `https://github.com/` (cit. on p. 41).

[28] MDN. *JavaScript*. 1995. URL: `https://developer.mozilla.org/it/docs/Web/JavaScript` (cit. on p. 48).

[29] Google. *Puppeteer*. 2017. URL: `https://pptr.dev/` (cit. on p. 79).

Acknowledgements

At the end of this journey I want to acknowledge all the people that have helped me through this.
First I want to thank my supervisors, professor Luigi De Russis and Tommaso Calò, for their constant support and guidance in this difficult, yet stimulating thesis work. Your advice helped me grow during this time and I am thankful for this.
I want to thank my parents, who have always supported me and made me the person that I am today. Thank you for your sacrifices and for your love. This achievement is also yours.
Thanks to all my relatives who showed support and cared for me since I was little.
I want to thank all my friends, from those who I met at school and even after all this time passing remained by my side, to the new friends I met here in Turin, who made this last years unforgettable.
Thank all of you for the memories shared together, for your love and support.