Enabling Human Computation through Text-based Conversational Agents

Owen Huang

Web Information Systems
Data Science & Technology, EEMCS
Delft University of Technology

TU Delft

Enabling Human Computation through Text-based Conversational Agents

by

Owen Huang

to obtain the degree of Master of Science in Computer Science
at the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS),
Delft University of Technology,
to be defended publicly on Monday December 17, 2018 at 3:30 PM.

Student number: 4317459

Thesis committee:

Chair:	Prof. dr. ir. G.J.P.M. Houben, Faculty EEMCS, TU Delft
University supervisor:	Prof. dr. ir. A. Bozzon, Faculty EEMCS, TU Delft
University supervisor:	Dr. ir. P. Mavridis, Faculty EEMCS, TU Delft
Committee member:	Prof. dr. ir. J. Urbano, Faculty EEMCS, TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

Abstract

Human Computation (HC) has established itself to be a powerful tool for carrying out certain simple and repetitive tasks in the form of microtasks, which to this day are still difficult for a machine to automate. With the latest increase in interest in machine learning, HC has similarly gotten more attention as a popular way to acquire training data in large quantities. Traditional microtask crowdsourcing platforms, such as Amazon Mechanical Turk (AMT) or Figure Eight, are typically built using web-based interfaces. However, the speed and quality of data acquired via the crowd are naturally limited by the number of available workers and their skill set.

We perceive a grand opportunity in expanding the crowd by exploring alternative means to the traditional microtask crowdsourcing platforms that are reliant on the web-based interface. More specifically, as popular messaging services such as Telegram, WhatsApp and Facebook Messenger are used on a daily basis by millions of people across the world, we propose to perform HC activities inside these services through a *text-based conversational agent* (or *chatbot*). We foresee new opportunities arising in conducting HC inside the chatbot, that could leverage the access to a potentially larger and more diverse crowd.

In this thesis, we set the first step towards a new alternative to the typical web-based interface used in HC. As a result, we set out to investigate the viability of facilitating microtask crowdsourcing inside chatbots. To this end, we design and implement a chatbot that acts as a medium for the execution of microtask crowdsourcing activities, which is then used for conducting several pilot experiments. In addition, we propose a mapping from *Web* to *Chatbot* tasks for several commonly found User Interface (UI) elements inside worker interfaces. Thereafter, we conduct an elaborate experimental campaign to gauge the feasibility and interest of crowd workers to use the chatbot as a new medium for performing generic microtasks. We designed, implemented and executed six common microtask crowdsourcing types; *Information Finding*, *human OCR* (CAPTCHA), *Sentiment Analysis*, *Object Labelling*, *Image Annotation*, and *Speech Transcription*. For each task type, we implemented a microtask in both a web-based and conversational interface. By measuring the execution time, quality of answers and surveying workers' satisfaction of a total of 316 distinct workers recruited via Figure Eight, we show that chatbots can be effectively used as an alternative to the web-based interface to perform microtask crowd work. We report that out of all workers who participated in the chatbot tasks, 98.3% of the workers indicated a positive experience and were satisfied with their interaction with the chatbot, while performance in terms of task execution time and output quality was in general comparable.

Preface

In front of you lies the Master's thesis "*Enabling Human Computation through Text-based Conversational Agents*", that concludes my end of two years of study in Computer Science at the Delft University of Technology. This document has been written in partial fulfilment of the graduation requirements of the MSc. Computer Science programme following the Data Science & Technology Track.

The opportunity to work on such a large project is something I will be ever grateful for. My journey was of course not without challenge, which in the end helped me grow as a person and taught me a great deal by experiencing the process of conducting scientific research. This work is something that would not have been possible without the guidance and support of my supervisors.

I would therefore like to express my sincere gratitude to Alessandro Bozzon and Panagiotis Mavridis. You have always stood ready to help me, whenever I needed advice or to answer my questions. I would also like to thank Sihang Qiu for providing invaluable support throughout this project, from the very beginning to the end. Moreover, I would like to thank the other the thesis committee members, Geert-Jan Houben and Julián Urbano for the taking time to attend my thesis defence. I wish to also thank everyone with whom I had spent time during these two years in Delft. Lastly, I must express special thanks to my friends and family who have supported me throughout my time in Delft.

Owen Huang
Delft, the Netherlands
November 2018

Contents

List of Figures

List of Tables

1 Introduction

In this day and age, computational problems, such as complex differential equations, regression analysis, can often be easily solved by computers within seconds. In the past, these problems would take human computers months to solve.

While current day computers have proven to be powerful and vital to society, we are yet to arrive at that point where computers are able to perform complex cognitive and perceptual tasks. If one were to ask a machine if it finds a painting "beautiful", surely the machine would not be able to truly comprehend, interpret and appreciate it in a similar way as a human would.

Human Computation (HC) focuses on "*solving problems that computers cannot yet solve, through the use of human processing power*". One of the most common examples of HC found today is the well known CAPTCHA [68] (see Figure 1.1), which is commonly used as a type of test to tell apart human from machine. The computational process given to the crowd is often constructed in such a way that they can be completed within a reasonable amount of time and require a low level of skill. These tasks are referred to as *microtasks*, while *macrotasks* include the class of complex tasks requiring much more time and sometimes expert knowledge [11].

Figure 1.1: An example of a common CAPTCHA.

Modern technology has allowed us to connect to each other regardless of geographical location with increasing ease. People increasingly use social media (e.g. Facebook[1]) and messaging services (e.g. Facebook Messenger[2], WhatsApp[3], Telegram[4]) for interpersonal communication and networking. As globalization so ever increases, so does our opportunity to borrow the "wisdom of the crowd" for solving problems where machine still fails.

With the recent interest in artificial intelligence, *text-based conversational agents* or *chatbots* have gotten a surge of attention in both the industry and research. Chatbots allow humans to interact with a conversational partner in natural language through text. With the current opportunity to create chatbots inside

[1] https://www.facebook.com/
[2] https://www.messenger.com/
[3] https://www.whatsapp.com/
[4] https://telegram.org/

popular messaging services, the potential to reach out to larger crowds with HC piques our interest. With access to a larger worker base for microtask crowdsourcing, we may gain potential benefits such as further democratization of crowd work. This may result in an increase in worker diversity, in terms of demographics, skill sets and knowledge, which could improve the digital experimental environment for e.g. psychological research [2].

But provisioning of alternative means of microtask crowdsourcing may also have a societal impact because people at "the bottom of the pyramid" could even perform retributed digital work [52]. For many people performing crowd work has even become a necessity to make ends meet [57], while for some it may even be one of their only ways to earn wages due to e.g. medical or personal circumstances [3].

Other potential benefits as a result of a larger worker base, may include better real-time support for crowd-powered systems. In low-latency crowdsourcing, workers often are kept in retainment pools in order to have workers rather wait for incoming tasks than vice-versa [23, 25, 27]. In addition, situational and spatial crowdsourcing (i.e. during a commute or at a certain location) may similarly benefit from more available workers to increase the odds of finding qualified workers [35].

The benefits and motivation of why people use chatbots have been previously investigated, which predominantly indicated that chatbots increased "productivity" [8]. While the use of chatbots is already widespread in real-world businesses[5], they are mostly only able to operate well in pre-defined settings and consequently fail to serve its users when faced with unexpected requests. This is because chatbots work generally well as long as the chatbot understands what the user tries to achieve, knows with what it wants to respond, and holds *all* the required information to form the response. Moreover, gathering of training data and building a rich knowledge base is challenging in itself. Public data sources may not contain the information that is required and manually creating high-quality training data sets may be difficult and expensive. As a result, the extent of knowledge the chatbot contains is static; it merely stretches as far to what it has access to. This means that expanding chatbot functionality to deal with unforeseen scenarios becomes difficult.

In an effort of increasing the flexibility and capabilities of the system, it spawned the idea to use humans to enhance the traditional approach that relied purely on what the machine alone is capable of. The combination of the traditional chatbot system with HC (interchangeably; chatbot assisted by human-aid, Human-in-the-loop (HITL) chatbot, crowd-powered or crowd-based chatbot) is what gave birth to a *hybrid chatbot* [4, 6, 12, 23–25, 27, 40, 44]. Naturally following, hybrid chatbots may also capitalize on the possibility to learn from past conversations with users. Thus with the aid of humans, these chatbots are able to serve more complex user requests over time. In addition, hybrid chatbots may even improve the quality of its responses by getting a better grasp on distinct contexts of conversations.

While these chatbots already included the crowd to enhance their own capabilities, we consider the possibility to use the crowd in a different way. We propose to leverage the crowd residing in popular messaging services as a potential pool of workers for the crowdsourcing of microtasks. Although microtask crowdsourcing has typically been done through web-based interfaces via popular services such as AMT or Figure Eight, alternative interfaces remain unexplored. As a result, with the opportunity to reach out to more workers through the chatbot, we thereby set out to take the first step into a potential new field within HC; *conversational Human Computation*.

1.1. Problem Definition

Our work focuses on furthering our understanding of the viability of executing generic microtask crowdsourcing through text-based conversational agents. Previous work has shown that performing microtasks in text-based conversational agents is technically feasible for specific application scenarios [6]. However, the design and execution of generic microtasks through conversational interfaces raises questions on work speed and quality that have not been addressed in the literature. As our work focuses on *microtasks*, we use in the remainder of this thesis the term *task* to refer in general to microtasks unless we specifically state otherwise.

Our primary challenge lies in bridging the gap between HC and the conversational interface. However, with no medium to perform microtask crowdsourcing in a chatbot at our disposal, we ought to first raise the question to what extent we must adapt current chatbots to facilitate HC processes. While previous work has involved HC with chatbots, this rather focused on the execution of microtasks in traditional web-based

[5] E.g.: Meekan, a scheduling assistant,
Instalocate, a personal flight assistant, and
Emma, the personal fashion shopping assistant

interfaces. Our work focuses on the embedding of HC in the chatbot to perform *conversational Human Computation*.

Because of both the novelty of bringing HC into the chatbot and conversational HC (with the latter to our knowledge, remaining to be explored), we aim to direct our efforts to take the very first steps into uncharted territory within the field of HC. Firstly, we require to adapt current chatbot architecture to allow the inclusion of HC. This also means as we aim to develop in essence a conversational HC platform, that we must also take into account how we wish to model various data transactions taking place in the crowdsourcing of microtasks. Similarly, common HC processes such as task assignment, worker recruitment, and execution monitoring, need to be accounted for as well in our design. For example, how should tasks be modelled to allow seamless distribution to an arbitrary amount of workers all at once? Other questions concerning e.g. storing worker answers and aggregation of answers also require to be addressed.

In addition, with web-based microwork platforms being a well-established medium for the crowdsourcing of microtasks, the question of how to run similar microtasks in a conversational interface remains open. Because the chatbot is a conversational partner, workers will have the opportunity to ask questions as well instead of only providing answers to the chatbot's questions. Evidently, maintaining a certain flow of conversation throughout microtasks in the chatbot is similarly something left to be explored.

While controls and visual cues as *page scrolling, editing of answers, input feedback* (e.g. clicking a radio button) are common in web-based microtask crowdsourcing interfaces, chatbots do not naturally possess capabilities for similar navigation and visualization. Since the chatbot holds a conversation which is a dynamic process rather than a working in a static web-page, it implies that if a worker wants to perform a certain navigable action, the worker has to let the chatbot somehow know what the desired course of action is.

1.2. Research Focus

To investigate what the effects of executing microtasks in chatbots are—in terms of work speed and quality—we must first design and implement a chatbot that allows for conducting microtask crowdsourcing activities. We first define the main focus of this project by aiming to answer the following main research question:

Main RQ: To what extent can text-based conversational agents support the execution of microtask crowdsourcing activities?

Towards answering this question, we specify the following research sub-questions:

RQ 1: How do we build a text-based conversational agent that facilitates the execution of microtask crowdsourcing activities?
Before we are able to venture forth to deepen our understanding of the viability of doing microtask crowdsourcing through chatbots, we must facilitate the means necessary to run our experiments. To inspire and drive the design of the chatbot, we first look into previous work found in the literature. We aim to understand what the architecture of existing chatbots looks like. Furthermore, what common issues HC faces that we similarly need to overcome and what challenges lie ahead in bringing HC into the chatbot. Following all this, we apply what we learn from extant literature and propose our chatbot design and proceed to implement a prototype chatbot system that is able to function as a microwork platform.

RQ 2: How do we map web-based user interface elements to chatbot user interface elements for microtask crowdsourcing?
In order to test the viability of microtask crowdsourcing in the chatbot, we require to be able to perform the same commonly found microtasks in web-based microtask crowdsourcing platforms. The challenge here is to provide a mapping from web-based to chatbot UI input controls. We provide such a mapping and implement for several commonly found microtasks both a web-based and chatbot variant.

RQ 3: How do different types of user interface input elements for the conversational interface affect the execution time and output quality of microtasks?
With the mapping from web-based to chatbot input controls, we wish to explore the opportunities that the chatbot holds for housing microtask crowdsourcing activities in further detail. While web-based interfaces possess standardized input controls, which most people are familiar with (e.g. textboxes, radio buttons, and checkboxes), the UI representation in the chatbot is certainly something that remains to be explored. Our aim is to look for opportunities the chatbot holds for input controls and take a closer look what this would mean in terms of task execution time and output quality.

1.3. Contributions

Towards the realization of a conversational HC platform for the crowdsourcing of microtasks, we lay the foundation for many future opportunities in this thesis. We propose a design and implementation of a chatbot taking into account the possible continuation of this project for future work. Using this chatbot, we conduct pilot experiments to test the viability of the platform as an alternative to existing web-based microwork platforms. We summarize the contributions of our work in the following:

C1: To aid us in the design of our chatbot, we review chatbot and HC literature. We look into the challenges of implementing a chatbot, and what HC has already meant for the chatbot so far. As a result, we investigate what the main challenges in HC and crowd-based systems are. We aim to identify what technological gaps we require to address when combining chatbots with HC.

C2: We design and implement the chatbot, that allows for the crowdsourcing of microtasks. The chatbot is implemented to be ready for testing the execution of microtasks commonly found in existing web-based microtask platforms. Furthermore, for facilitating future experiments on chatbots and conversational crowdsourcing, we propose a modular architecture that allows the chatbot function both as a common conversational partner and as a conversational microwork platform for housing microtasks.

C3: We design and conduct an extensive experimental campaign for testing the viability of conversational HC. We implemented six common microtask types both in a web-based (Figure Eight) and conversational interface (Telegram). By recruiting a total of 316 distinct workers through Figure Eight, we show that chatbots as a conversational interface can be an effective alternative to web-based interfaces. We show that the task execution time and output quality of the six microtask types are generally comparable between the web- and conversational interface. Furthermore, we gauged the workers' satisfaction in using the chatbot as a new medium for microtask execution and found unanimous praise and interest for the chatbot.

1.4. Thesis Outline

This thesis consists of seven chapters. In Chapter 2, we discuss related work that will aid and inspire us in the design and implementation of the chatbot. We identify what challenges current chatbots address and look into the intricacies of HC. Following in Chapter 3, we elaborate on our proposed system architecture and design of the chatbot to facilitate the execution of microtasks. We follow in Chapter 4 by detailing the implementation of our design and show how the resulting chatbot operates. Thereafter in Chapter 5, we detail the setup of our experiments to evaluate the viability of performing microtask crowdsourcing. Afterwards in Chapter 6, we present and discuss the results of these experiments. Finally in Chapter 7, we conclude and provide avenues for future work.

2

Related Work

In this chapter, we aim to gain a deeper understanding of the developments in chatbot technology and Human Computation over the years. In order to envision the chatbot as a medium for HC, we study both concepts separately in Section 2.1 and Section 2.2 respectively. Moreover, we note the recent developments in fusing the chatbot together with the crowd to enhance its own capabilities. While we merely use the chatbot as a conversational interface, we also wish to inspire ourselves with possible future directions the chatbot may follow. As such, we study the applications of the crowd in chatbots themselves by detailing the intricacies of the recently proposed crowd-powered chatbots in Section 2.3.

Finally, we take a brief look into the field of mobile crowdsourcing in Section 2.4 and summarize our findings Section 2.5. We pay extra attention to proposals of mobile crowdsourcing platforms, as these works could help us further our understanding of interface design for mobile devices.

2.1. Conversational Agents

Conversational agents date all the way back to the 1960s with ELIZA [73], as one of the first text-based conversational agents or *chatbots*. With ELIZA (shown in Figure 2.1), a precedent had been set for the fascination we share in natural language communication between human and machine.

The term *conversational agent* has been widely adopted to refer to a class of conversational interfaces consisting of multiple modalities. Within this class, we draw a clear distinction between *text-based* and *speech-based* conversational agents. The former which we refer to as *traditional chatbots*, while the latter as Spoken Dialogue Systems (SDSs). We primarily focus on literature surrounding text-based conversational agents. However, we note that many techniques may be shared between chatbots and SDSs.

Chatbots typically share several key components [1, 20, 34] as shown in Figure 2.2, we discuss each component in detail:

- **Natural Language Understanding**: Everything a user utters towards the chatbot will first pass a Natural Language Understanding (NLU) unit, which tries to process the utterance. The goal of NLU is to map the utterance to something that the chatbot would be able to interpret and act upon.

  The most straightforward approach to perform NLU is through *rule-based pattern matching*. The simplest case would be to simply map entire utterances to some output category. For example, given the utterance '`hello`' this may map to an output category '`greeting`'. This would let a chatbot "understand" that such utterances are meant as a greeting. To take it a step further, this mapping could also be done on parts of the utterance or instead match based on a predefined set of keywords. This strategy of pattern matching was similarly adopted by ELIZA [73].

  With the introduction of the chatbot ALICE (or A.L.I.C.E.), Wallace [69] introduced an Extensible Markup Language (XML) dialect called Artificial Intelligence Markup Language (AIML) which improved the ease of creating chatbots. AIML supports advanced pattern matching (including different levels of wildcard tokens), and a powerful tool often[2] referred to as *symbolic reduction* or *symbolic recursion* which allows for complex substitution and creation of conditionals.

[1] `https://www.masswerk.at/elizabot/eliza.html`

[2] In [69] it is simply called the '<srai>' operator. Furthermore, Wallace notes that there is no official meaning of the acronym.

```
  Welcome to
                    EEEEEE  LL      IIII  ZZZZZZZ   AAAAA
                    EE      LL       II       ZZ   AA   AA
                    EEEEE   LL       II     ZZZ    AAAAAAA
                    EE      LL       II    ZZ      AA   AA
                    EEEEEE  LLLLLL  IIII  ZZZZZZZ  AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Maybe, do I look troubled to you?
ELIZA: You don't seem quite certain.
YOU:   No, I'm not.
ELIZA: Why not ?
YOU:   I don't know.
ELIZA: Don't you really know ?
YOU:   Maybe I do, but I cannot think of anything right now.
ELIZA: Why the uncertain tone ?
YOU:
```

Figure 2.1: ELIZA[1], one the earliest text-based conversational agent.

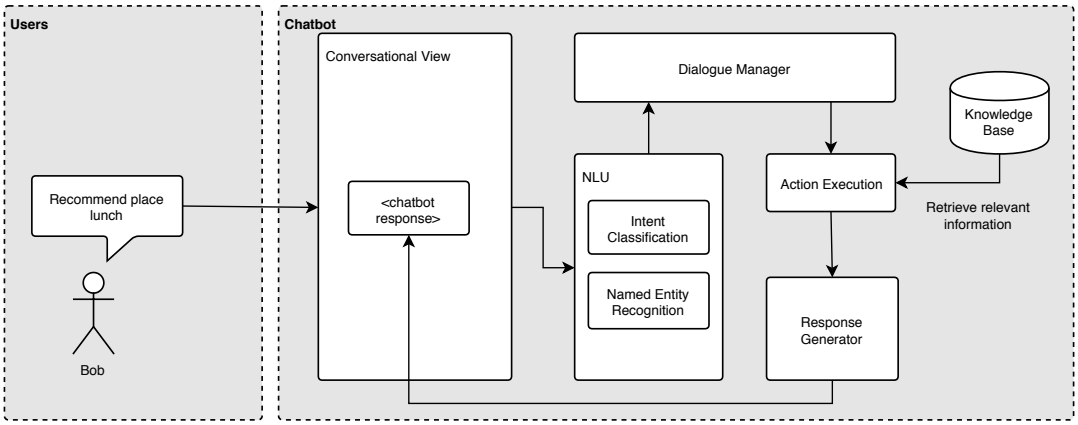


Figure 2.2: High-level concept of a data-driven traditional chatbot.

The alternative to rule-based pattern matching is to adopt a data-driven NLU model. With early work proposing to use an encoded forest of decision trees [36], machine learning had already made its way into NLU. However, the more modern approach in these data-driven NLU models seems to contain two key steps: *intent classification* and *entity extraction*. Intent classification holds the goal to map utterances to a single intent (which is comparable to the rule-based approach mapping to an output category). Entity extraction is then used to identify what important key-value pairs (i.e. entities) the utterance contains. For example, the utterance '`Hello, I am 50 years old.`' could result in the intent classification `greeting` and extracted entities '`age : 50`'.

- **Dialogue Management**: Throughout a conversation with the user, the chatbot will need to keep track of its dialogue state. The chatbot will require to know at any point if an utterance relates to anything stated previously in the conversation.

  Bui denotes that the simplest form of dialogue management is the finite state model, which essentially models the entire conversation as a state transition network. While this approach provides the most control over a conversation (due to its simplicity), it also means it would result in rigid responses by the chatbot because everything has been predetermined. The alternative is the frame-based model, which

extends the finite state model, using the idea filling a predetermined number of form elements or *slots* with information.

Other approaches are information state-based, which uses lets the dialogue flow based on the information state [64]. This approach also allows for the development of multimodal dialogue systems.

A different approach to dialogue management is using probabilistic models, such as Markov Decision Processs (MDPs) or Partially Observable Markov Decision Processs (POMDPs). These models in turn also inspired the use of reinforcement learning models [61].

More modern approaches often use a combination of a frame-based approach and probabilistic approach, using the idea of slot-filling in combination backed by a trained model [3].

- **Action Execution & Information Retrieval**: Before a chatbot is able to form a proper response, it generally requires to fetch some information from either the chatbot's internal knowledge base or some external source. In other cases, the chatbot may be required to perform some action that heeds the user's request. For example, booking a hotel room may be perhaps done by automatically sending the hotel a message detailing the reservation. For integration with external data sources, the chatbot could make use of services such as public web APIs to fetch information.

- **Response Generator**: Right before the chatbot is ready to send a user some response, the chatbot generally requires to formulate the response in natural language. Using a template-based approach such as with AIML, it is possible to directly map the input of a user to some fixed response (or again using rule-based pattern matching to have more flexibility, by using response templates).

  Similarly to NLU and dialogue management, it is also possible to use another model for selecting responses and even use a hybrid approach including response templates [63].

  A different approach would be to use machine translation to generate novel responses [56], which carries the opportunity to generalize better to multiple languages.

2.2. Human Computation

The term Human Computation (HC) has established its roots as a separate modern field of study after the dissertation from Von Ahn in 2005 [65]. While the term *Human Computation* is often used in context with *crowdsourcing*, it should not be used interchangeably. Quinn and Bederson [54] have thoroughly investigated the nuance between the terms HC and crowdsourcing and concluded that HC generally refers to "the general paradigm of computation", and that "human participation is directed by the computational system or process". The former also suggests that in the future what may be difficult to solve for computers may change.

2.2.1. Macro- and Microtasks

In crowdsourcing systems, the nature of tasks solvable by humans ranges from arbitrarily complex processes to asking simple dichotomous questions. While the boundary delineating such *macro-* and *microtasks* is not set in stone, tasks that take "a significant amount of time" and possibly require more expert knowledge [11, 62], compared to a possible split version of the task, are generally considered to be of macro-level. For example, the general task of transcribing a one-minute audio fragment containing dozens of sentences could be considered macro-task, when its counterpart microtask could be considered by splitting the audio fragment to transcribe each sentence individually. Similarly, macrotasks could also take hours to complete [21]. Among these types of macrotasks, some could also be difficult to decompose as a result of the goals of the task. For example, creating a single animated movie from a prompt [55].

Kittur et al. [33] have proposed *CrowdForge*. A general-purpose framework for performing complex macrotasks in microtask markets. By breaking down the macrotasks into smaller microtasks, these could be run with workflows in microtask markets.

Zacks et al. [75] suggest in a study of experimental psychology that humans perceive routinely tasks in segments, which raises the question of whether complex macrotasks can be perceived in a similar fashion. As a result, Cheng et al. [11] have recently studied the effects of breaking down of macrotasks to microtasks. Their experiments showed that for three tasks types—receipt arithmetic, sorting lists and speech transcription—workers preferred performing microtasks over macrotasks. In addition, microtasks were found to take longer to complete than macrotasks, but the output was considered of much higher quality.

[3] The currently popular Rasa Core is an example of such a hybrid approach: https://www.rasa.com/docs/core/

Breaking down macrotasks into a series of microtasks has also been studied in the context of designing *workflows*. These workflows define solving macrotasks as a series of microtasks. Kulkarni et al. [37] have proposed to crowdsource the creation of these workflows themselves, as finding effective workflows is a very challenging problem itself. Cai et al. [10] found that the order of crowdsourcing chains of microtasks affected task completion time depending on the complexity of the microtask. Dai et al. [13] have shown that introducing "micro-diversions", i.e. content that deliberately aims to break the flow of a worker to refresh cognitive abilities, between performing series of similar microtasks results in an increase in answering speed, while maintaining similar levels of output quality.

With a larger availability of lower-skilled than expert workers, microtasks have been studied in much more depth as a result of their better applicability to real-world crowdsourcing processes. Microtasks differ greatly in terms of goals, expected input and output types and required actions. In an effort to map common microtask types, Gadiraju et al. [19] have proposed a two-level categorization scheme of microtasks based on the goals of commonly found tasks. They also highlight the importance of realizing that tasks may also share similar goals, resulting in tasks belonging in multiple task type categories.

2.2.2. Challenges in Human Computation

HC faces many diverse challenges, which many are shared with crowdsourcing. We view different types of HC challenges from a *systems* and *human* perspective (i.e. *workers* and *requesters*). All challenges in HC could be traced back to three fundamental key aspects relating to the *latency*, *quality* and *cost* of the computational process as depicted in Figure 2.3.

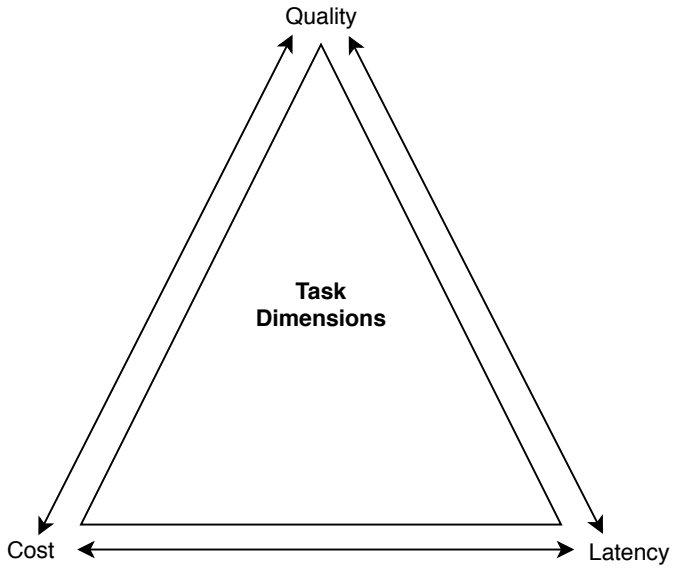


Figure 2.3: The three major factors in Human Computation. Each aspect is in direct relation to the other two.

From a systems perspective, the following challenges are evident [30, 34, 43]:

- **Cost and worker scalability**: The crowd is generally not free and limited in size, while the number of tasks to perform could easily become larger than the crowd is able to handle. This means that in terms of *scalability*, HC based systems may face high cost and latency if it were to rely too often and heavily on the crowd.

  To limit HC cost, different techniques may be employed. The most straightforward approach is to simply limit the number of tasks to crowdsource [70]. Other approaches include answer deduction and generalization of crowd answers. This is achieved by logically inferring properties from e.g. assertions or negations, and extrapolating crowd answers respectively [71, 72].

- **Worker recruitment & retainment**: To recruit workers, HC systems may adopt a *pull-* or *push-based* task routing or assignment model [42]. The former model allows workers to pick tasks themselves given a subset of available tasks. Contrary to pull-based task assignment, the push-based model lets the system handle to whom to "push" or assign tasks. Both models may benefit greatly from *user modelling*.

For example, if an HC system needs to assign a difficult task that requires expert knowledge, it becomes important to know which user possesses the necessary skills to perform that task [7].

While recruiting workers is a very challenging aspect of HC in itself, maintaining worker interest throughout tasks or *retaining* workers is equally important to minimize task completion time [15].

- **Data quality management**: As the crowd is human, the quality of answers provided by the crowd can greatly differ among each other. In an effort of managing the answer quality, a common practice is to assign the same task to multiple workers and aggregate the answers to reach some consensus [28, 42]. However, as the amount of tasks increases, so does the amount of workers. This causes again the issue of scalability. Moreover, some tasks such as *sentiment analysis* possess a certain degree of subjectivity, making judging answers in an object manner challenging [45].

  Other approaches to managing answer quality may involve indirect approaches, such as targeted crowdsourcing by modelling workers and the elimination of low-quality workers. To obtain the information for such worker models, the simplest approach would be to use ground-truth data—i.e. labelled data—which may be used to create questions to test if workers are answering earnestly [47]. The application of such tests may be directly used to block workers from (further) participation if they score insufficiently[4]. However, it may also be possible to perform other more advanced techniques such as an Expectation Maximization algorithm to compute certain parameters of the worker model [31, 76], or modelling workers as a graph and using graph theory to compute model parameters [32, 46].

- **Real-time support**: In the context of Question-Answer (QA) systems, real-time support would be desirable as long wait times may affect the user experience. For example, a user might wish to know where to find a nearby café that would not be crowded at the moment. While the crowd is busy finding a place that fulfils those requirements, the user will have to wait. Meanwhile, the user might have moved on and twenty minutes later the crowd might suggest a place that would not be nearby anymore.

  Huang et al. [26] have shown the viability of performing real-time *entity extraction*. Similarly, Savenkov et al. [60] attempted to perform real-time QA by pressuring workers to answer within small time limits and showed that answer quality was comparable to non-pressured workers. VizWiz [5] was one of the first real-time crowdsourcing systems, which aimed to answer visual questions which could aid blind people. Similarly, Legion [39] used the crowd to control user interfaces of several applications directly. With Legion:Scribe [41], Lasecki et al. demonstrated the application of real-time captioning.

As HC involves humans, who may wish to contribute by performing tasks or are creating tasks themselves (indirectly or directly), other types of challenges arise:

- **Worker fatigue**: Opposed to machines, humans may get tired after working for extended periods of time. As a result, workers who are fatigued may take longer to perform tasks and produce worse quality answers [9, 18]. Rzeszotarski et al. [13, 58] studied the impact of introducing *micro-breaks* in between microtasks to reduce fatigue and overcome boredom, which showed an increase in task completion time and maintain similar output quality.

- **Motivation & participation incentive**: A large challenge in HC is motivating workers to perform tasks, which become often repetitive and boring [54].

  Quinn and Bederson note that one of the motivators could be altruism; simply do good. Mao et al. looked at comparing volunteers and paid workers [51]. They found that the performance of volunteers and paid workers were comparable within a single scenario and noted that their results may not be applicable to other types of tasks.

  A different motivator may also be the enjoyment of tasks, which goes hand-in-hand with gamifying the computational process. These type of games are referred to as Games with a purpose (GWAP).

  One of the most famous HC games is the ESP game [66], which was used for labelling of images. Players would be shown an image and must guess (with no means of communication between players) what the other player would label the image as.

[4] Popular crowdsourcing platforms such as Amazon Mechanical Turk (`https://www.mturk.com/`) and Figure Eight (`https://www.figure-eight.com/`) allow requesters to create quizzes that workers are forced to take. Workers that score below a certain score will be blocked from entering tasks. Furthermore, it is also possible to put these test questions inside the task to verify that workers are still participating without ill intention after passing the quiz.

Feyisetan et al. [17] used the ESP game as inspiration to design *Wordsmith*, to similarly label images. They showed that monetary rewards are not a necessary incentive for improving task acceptance and completion rates. They also explored with "furtherance rewards", which notified workers that they are given additional rewards if they choose to stay and perform more image labelling. These furtherance rewards showed an increase in the amount of work accepted and performed.

Liu et al. [49] performed a small case study with a gamified mobile crowdsourcing platform they proposed in [48], which showed reliable task completion time and task completion rate for QA type of tasks related to local events.

Ipeirotis and Gabrilovich [29] proposed a quiz system and used targeted advertisements to identify (volunteer) expert workers. They found that both task execution time and quality of answers from paid workers was significantly worse than non-paid workers and accredited this phenomenon to paid worker's lack of domain knowledge.

- **Worker diversity, bias, skill & knowledge**: Workers may possess distinct characteristics in terms of demographics, personality traits, skill set and knowledge. Evidently, these characteristics may end up influencing worker performance when tasks require expert knowledge or are of a subjective nature. For example, asking a youth who keeps up with the latest societal trends if something is "cool and hip" may trigger different responses than asking an elderly person. Similarly, if expert knowledge is required to perform some task, it is obvious that someone without that knowledge would not be qualified to participate.

  However, whenever there is money involved, there will naturally be people who try to maximize their earnings.

  Faltings et al. [16] noted several issues that cause *bias* in HC:

  1. People tend to use "heuristics" to solve problems. This means that people who wish to maximize profit, try to minimize their effort spent per task to perform as many tasks as they can in as little time as possible. In extreme cases, this could lead to *spam behaviour*, in which workers show disinterest in undertaking the task by providing random answers or answers following a fixed pattern [77].
  2. When workers who perform the same task multiple times, while the supposed answer is repeatedly the same, workers become more biased to answer similarly.
  3. Provided examples or information may instil bias itself, or in other words, it may damage the independence of the worker's assessment [50].

2.3. Crowd-powered Chatbots

The integration of microwork platforms with text-messaging and chatbot systems has primarily concerned enhancing the capabilities of the individual components of the conversational agent—e.g. collection of training data for intent recognition, acquisition and enrichment of knowledge—or to entirely substitute artificial intelligence for conversation management purposes.

One of the earliest examples of a chat-based crowdsourcing system is *Chorus* [25, 40]. Chorus allowed end-users to chat with a seemingly single conversational partner, while in reality it was backed by a group of crowd workers. For each conversation with an end-user, Chorus would recruit a small group of crowd workers who would be able to propose and vote on candidate responses. From the candidate responses, the highest voted response would be selected and sent to the user. The workers were able to cast votes on candidate responses via a web-based conversational interface, which resembled an online chat room. This interface showed the on-going conversation with the end-user and additionally provided the chat history from previously recruited crowd workers with the end-user. The interface provided buttons next to each candidate response to *upvote* or *downvote* it.

Another early example of a semi-automated chat system is *Guardian* [23]. Guardian was originally developed to extend SDS with broader capabilities for information retrieval through Web-APIs. However, Guardian would also be directly adaptable for chatbots. Guardian performs its dialogue management and response generation through the crowd. Guardian followed a two-phase approach; the first phase consisted of populating a QA collection based on some given API. The goal was to generate a set of QA pairs accompanied by a set of API parameters that would be relevant to those pairs. The second phase, contrary to the first, ran in

real-time. This phase would request workers to extract from on-going user requests a set of query parameters. With these query parameters, the API is queried for a result from which workers are then asked to extract the relevant information and form a natural response. Furthermore, Guardian also had workers manage the dialogue state by letting workers vote for follow-up inquiries in case Guardian required additional information for executing the query.

With the continuation of Chorus [25, 40], Lasecki et al. recently proposed *Evorus* [27]. Evorus is an evolution of Chorus where conversation automation is obtained by adding, learning, and improving automated responses using past information gained from the crowd. Evorus differs greatly from Chorus by introducing a novel framework in which candidate responses are still proposed by the crowd, but in addition, Evorus also allows external bots to generate candidate responses. Workers would be presented with a list of candidate responses similar to Chorus. However, in Evorus this list would also contain the candidate responses of the external bots in addition to worker generated responses.

In a vastly different application domain, *Calender.help* [12] was proposed to be an email-based personal assistant. Calender.help possessed the ability to automatically schedule meetings at a time which fits all the participants. At that time[5], Calender.help would assist in scheduling meetings when it was addressed in the cc or was among the recipients of the email. The system uses a simple three-step approach for the scheduling of meetings. First, Calender.help would attempt to use automated means to achieve a satisfactory result. In case this failed, in-house *non-skilled* workers are recruited to attempt the task. If this still would not produce the desired result, *expert* workers would be then asked to complete the task.

Liang et al. [44] propose CI-Bot, an early prototype of a conversational agent as question and answering system, that made use of the conversational interface for microtask crowdsourcing. The authors conducted a pilot experiment and reported good performance for image labelling tasks.

InstructableCrowd [24] is a conversational agent that can crowdsource "trigger-action" rules for IF-THEN constructs, e.g. to set an alarm or an event in a calendar application. Because InstructableCrowd was developed as a mobile application, it allowed the use of the device's sensors to create these trigger rules. Moreover, InstructableCrowd included several simple actions such as sending device notifications and text messages. Workers used a web-based interface similar to the chat room proposed in [25, 27]. Aside from a chat log shown to the worker, the creation of the IF-THEN rules was done based on the use of simple checkboxes, dropdown menus and text boxes.

Bradeško et al. proposed *Curious Cat*, a context-aware conversational knowledge acquisition system [6]. Curious Cat was similarly to [24] a mobile application. Curious Cat uses a logic-driven QA mechanism to answer user questions. Instead of mapping predefined questions to a set of answers via rule-based pattern matching, Curious Cat maps the questions (posed in natural language) to a set of logical predicates. Using a logic inference engine, the logical predicates are then translated back to natural language to form the appropriate response. Curious Cat also relied on workers to supplement information, in case automated means failed to resolve to find the necessary information to answer users. Unlike [12, 24, 25, 27, 40], Curious Cat made full use of the opportunity to view other users of the application as potential workers. As such, microtasks were executed inside the mobile interface of Curious Cat itself. The conversational interface was similar to that of a common chat room as in [24, 25, 40], but also showed suggested answer options in the form of "autocomplete suggestions" above the text box in which workers would type their answer. To target which workers to ask and what questions, Curious Cat made use of contextual information extracted from mobile sensors to model spatial information of users.

These systems demonstrated the technical feasibility of application-specific microtask execution through chatbots. Our work has a broader scope, as it addresses the execution of different classes of microtask crowdsourcing, with a principled comparison with traditional Web interfaces aimed at evaluating chatbots as a generic medium for crowd work.

2.4. Microtask Crowdsourcing through Mobile Interfaces

In previous work, mobile interfaces have been used as a means for addressing the problem of ubiquitous and opportunistic microtask crowdsourcing in either in a humanitarian[6] or academic [35, 38, 52, 74] setting.

Yan et al. [74] showed an early example in the use of mobile interfaces through their platform *mCrowd*, which was able to perform crowd sensing tasks through a native mobile application. mCrowd was a small

[5]The personal assistant Calender.help was initially called *Cal*, but became later part of Microsoft's personal assistant *Cortana*. The system would also be able to recognize being addressed within the body of an email. More information is available on Microsoft's Calender.help web-page: `https://calendar.help`

[6]e.g. Ushahidi: `https://www.ushahidi.com/`

demonstration to show the technical feasibility to perform mobile crowdsourcing and supported four different task types; image annotation, image collection, text-based QA, and QA based on location. The design of the interface is simple; three different "tabs" for 1) listing available tasks as a worker, 2) creating a task as a requester, and 3) collecting and viewing of task results.

Samdara et al. [59] experiment with different mobile interfaces to perform crowdsourcing on multimedia microtasks. They tested three different input controls making use of the physical number pad and other navigational buttons commonly found in non-touch mobile devices.

Narula et al. [52] introduced *MobileWorks*, which is a light-weight mobile crowdsourcing platform designed for web browsers of lower-end mobile phones. Therefore also enabling the execution of crowdsourcing tasks by people with limited connectivity. It uses a very simple and minimalistic design of web-pages to limit the amount of data transfer required to perform microtasks. The authors also showed in a small pilot experiment involving ten workers, that the workers were positive about their experience on the platform and would also be likely to recommend it to others.

In a similar spirit, Kumar et al. [38] push further the initiative of mobile crowdsourcing in the context of developing countries. They implement and test both a native mobile application (*Wallah*) that supports generic crowdsourcing tasks and also a system that can handle tasks with simple Short Messaging Service (SMS) exchange. To evaluate the system they measure the impact of different screen sizes into the ease of use of their interface as well as the task execution time and quality of different types of tasks. They found that among 59 workers, there were 18 different screen sizes and 48 different phone models. However, they note that larger screen sizes did not translate to fast task completion times nor improved output quality. Furthermore, they found a correlation between screen size and quality of work, especially for tasks such as video annotation, human OCR and translation. Image annotation tasks were the highest performing.

In [14], Della Mea et al. set up an experiment with four different crowdsourcing platforms (Figure Eight, formerly known as CrowdFlower, was not included) in order to check the difficulty and execution time of commonly performed tasks and input controls. They experienced technical and usability difficulties with straightforward mapping from Web user interfaces to mobile ones, and therefore propose a number of adaptations for their experts when it came to the evaluation (e.g. avoiding long descriptions, minimising scrolling).

2.5. Summary

The idea of enhancing chatbots with HC has only erupted recently. With many propositions on how to incorporate the crowd inside the chatbot in an attempt to improve performance, the feasibility and practicability of doing so on a large scale are still often overlooked.

While crowd-powered chatbots face similar issues as crowd-based systems, such as quality control, they require stricter conditions in terms of response latency. Higher response times from the crowd could cause an increase in wait times for users to receive their replies. But in order to reduce response latency, it may be required to make concessions in quality or cost.

With much work being poured into the improvement of chatbot systems, we recognize that to our knowledge the idea of using the chatbot system as an alternative interface for the facilitation of microtask crowdsourcing activities remains to be explored. Both as a worker and requester, the use of a conversational interface bears many future opportunities in pushing forward mobile crowdsourcing as a field of study, as a result of the potential benefit of conducting experiments in environments where workers could be aplenty.

3

Chatbot System Design

To give our chatbot implementation shape in Chapter 4, we elaborate in this chapter on all the decisions impacting the design of the chatbot. With the literature we have outlined in Chapter 2, we address the complications of HC and chatbots and the combination thereof in our design. We start by discussing the main considerations that have impacted the design of the chatbot in Section 3.1. Next, we discuss the dual nature of the chatbot as a *base chatbot* in Section 3.2 and as a *microtask crowdsourcing platform* in Section 3.3. We then discuss the conversational interaction design from the perspective of crowd workers in Section 3.4. Thereafter, we discuss the extent to which tasks may be designed and their setup in Section 3.5. Finally, we elaborate upon the complete system design in Section 3.6.

3.1. Design Principles

The design of the chatbot system is based on four main overarching goals and considerations. In addition, these guiding principles aid us in the understanding of the strengths and limitations of the chatbot system. We discuss these four design principles in Sections 3.1.1 to 3.1.4.

3.1.1. Adaptive & Reactive Human Computation

An important part during the execution of microtasks is to be able to monitor progress on both the task and workers. Tasks that take too long, thus expire as a result of exceeding their time limit need to be stopped. Tasks where workers unanimously answer in unexpected and unforeseen ways may need to be adjusted. Workers who seem to answer frivolously or are caught spamming answers may need to be blocked from further execution of the task. Consequently, the added value of employing task and worker monitoring capabilities with reactive mechanisms is apparent.

To allow the system to rapidly respond to any event during task execution, we introduce the abstract notion of task execution policies. These policies steer on a high level the overall execution of the task. The idea is that these policies would be controllable on a per task basis to allow for more control of every individual task's execution. These policies may then even be constructed in such a way, that it could adopt dynamic task execution strategies based on the status of the task (Figure 3.1). For example, a policy may automatically reject workers who repeatedly finish tasks in an uncharacteristically short amount of time. Alternatively, we may wish to adjust pricing of *judgments* (i.e. an item for which we request worker assessment through questions), depending on the level of agreement between worker answers. Another policy may consider the case where a trade-off may take place between higher task cost in favour of task completion speed and output quality (see also Figure 2.3). As recruiting more skilled workers may achieve higher output quality, the pay must naturally complement the difficulty of the task. However, limiting the selection of workers by their respective skill set may negatively impact the overall task completion time, as the requirements for task participation rises.

3.1.2. Modular System Design

We view the overall chatbot system as a combination of a *base chatbot* and a *HC* component that is responsible for the creation and execution of HC processes.

Rather than designing our chatbot for some specific application domain, we opt to frame the base chatbot as an *abstract logical unit* that merely provides the necessary means for the communication between different

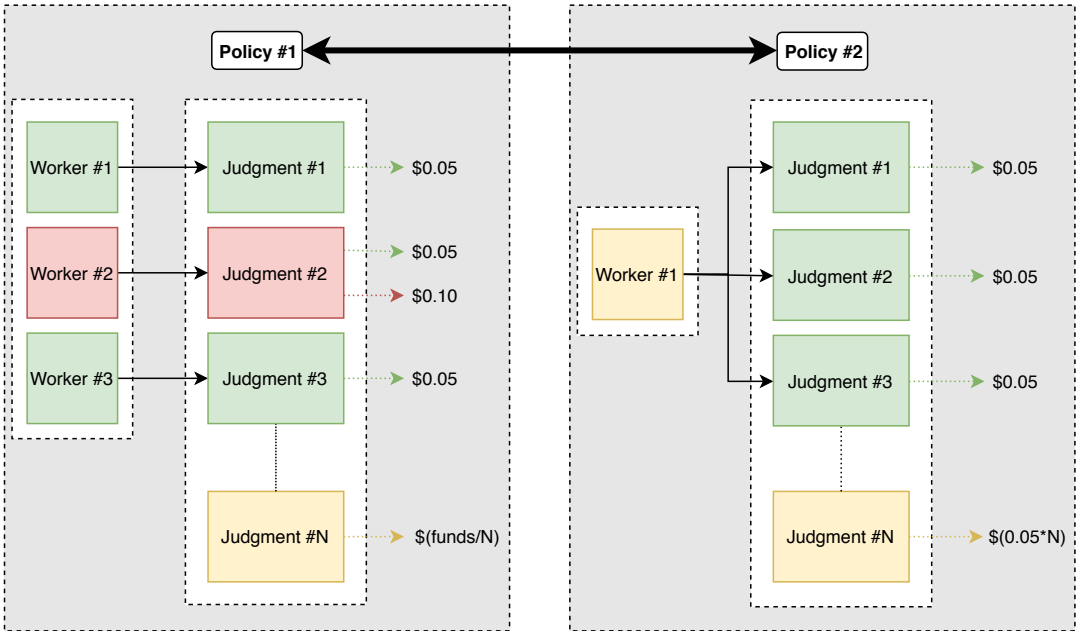


Figure 3.1: Adoption of two task execution policies switching dynamically based on e.g amount of workers available. In this example, the first policy allows each worker to procure a single judgment. Worker #2 is a spammer, which causes the system to double the payout for the next worker who answers earnestly. Pricing for remaining judgments is determined by the available budget. Switching to *policy #2* happens when very few workers are available. Here we only have one worker available and consequently that worker is tasked to provide multiple judgments. Moreover, the pricing is kept fixed contrasting dynamic pricing in *policy #1*.

system components. The base chatbot contains all components that the system requires to function as a conversational partner, while the HC component is an extension built upon the base chatbot.

As a result, we separate the user interface of the chatbot from the logical unit. This provides us with the benefit of allowing the chatbot to integrate with multiple third-party messaging services while keeping the possibility to design our own user interface.

Moreover, we let ourselves be inspired by the concept of a *crowd-powered chatbot*, which we have surveyed in Section 2.3. While the primary focus of our work lies with designing a *microwork platform* inside the chatbot, we acknowledge the opportunity that our proposed platform can be used to extend the functionality of the base chatbot to eventually become a crowd-powered chatbot.

3.1.3. Isolated Task Environment

Naturally following from the modular system design, we uphold the clear distinction between the goals and uses of the chatbot (see Figure 3.2). We acknowledge the duality in the use of the chatbot as a *traditional conversational partner* and as *microwork platform*. With the former, we refer to typical uses of the traditional chatbot; to engage in conversation for informational, transactional or conversational needs. While with the latter use, we refer to the engagement in conversation and interaction with the chatbot to achieve completion of microtask processes.

A major reason for this separation is that we wish to contain the execution of microtasks in a highly controlled environment. Our rationale is that we are more tolerant of potential mishaps during use of the chatbot as a traditional conversational partner than during the execution of microtasks. This is because we value serving a consistent work environment in which workers can grow accustomed to the conversational flow determined per design of the task. Moreover, we consider also the possibility of serving external requesters—i.e. tasks that do not originate from the chatbot itself as a requester—who may wish to have more control over the execution of microtasks in the chatbot. In contrast, we value flexibility for activities that involve the chatbot as a traditional conversational partner to endorse more free and natural interaction similar to talking to a human.

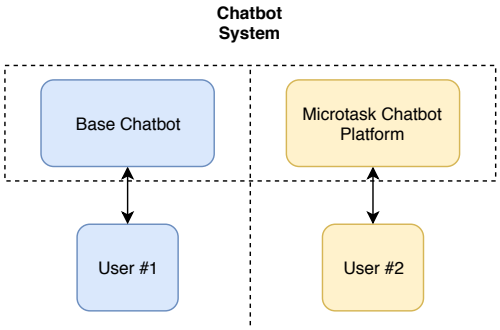


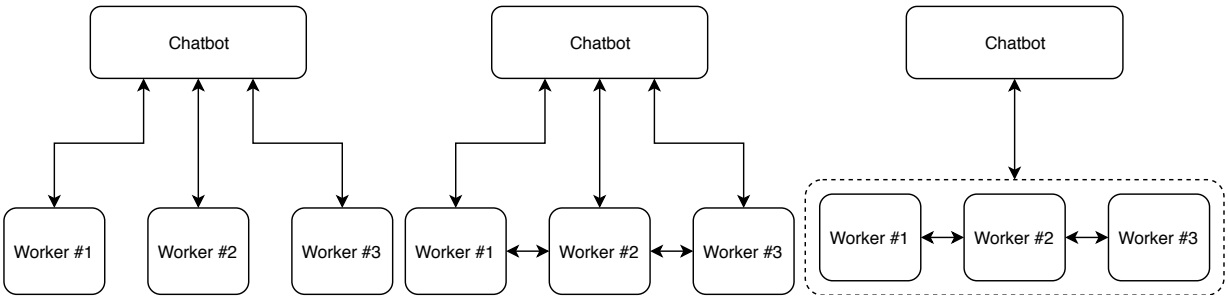
Figure 3.2: The chatbot as a traditional conversational partner and as microwork platform.

3.1.4. Non-Collaborative Dyadic Conversational Environment

We consider in our chatbot only the execution of microtasks through dyadic conversation. This is also mainly for the purpose of running controlled experiments in Chapter 5. Furthermore, we do not actively facilitate communication between workers for collaboration on microtasks of similar nature.

This means that answers from a single microtask judgment will always originate from a single worker. By this design, we endorse that individual workers solve problems using their own skill set rather than potentially relying on others to solve the microtask for them [22, 53].

On a high-level overview in Figure 3.3, we show that a worker is shown to communicate exclusively through the chatbot system during task execution. The chatbot does not allow a single task to be performed through a *group conversation*.



(a) Single Communication Channel. (b) Inter-Communication Channel. (c) Group-Communication Channel.

Figure 3.3: Overview of the worker environment. The system does not actively endorse communication between workers as shown in a), but still supplements the capability to do so as shown in b). We note the possibility for an additional communication channel in c), in which the chatbot converses with a group of crowd workers simultaneously, rather than on an individual level.

3.2. Base Chatbot

To shape our base chatbot system, we detail in this section the main components of the chatbot. All user interaction with the chatbot is powered through these components. Furthermore, we elaborate upon the functionality of all individual components of the chatbot system.

At the core of the chatbot, we have the *Natural Language Understanding*, *Dialogue Manager*, *Action Execution & Knowledge Base*, and *Response Generator*. Each request sequentially passes through these four main components, with each component depending on the input of their predecessors. We depict the relationships between these components in Figure 3.4.

We detail all the system components of the *base chatbot*, by discussing their responsibilities and functionality in Sections 3.2.1 to 3.2.4.

3.2.1. Natural Language Understanding

As soon as a user sends a message to the chatbot, the first step is processing the message to allow the interpretation of the message by the chatbot. The understanding of natural language follows a split design for both the chatbot and the HC component of the system. On the one hand, the base chatbot relies on a combination of a rule-based approach and data-driven models for message processing. While on the other hand,

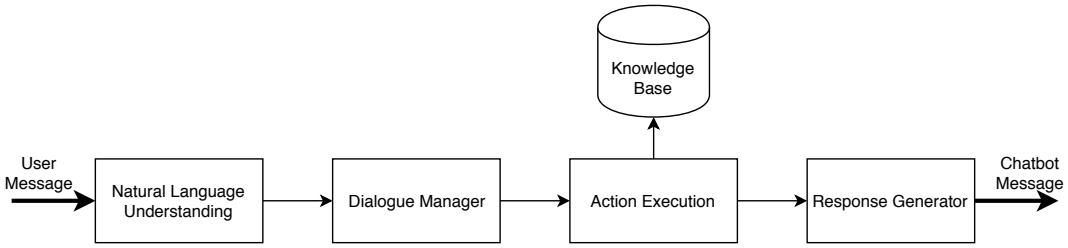


Figure 3.4: High-level schematic of the main components of the base chatbot system.

the HC side of the system exclusively uses a more rigid rule-based approach, which is less flexible than the data-driven models, but in return provides more control over the HC processes.

To setup the base chatbot for NLU, two models are used in parallel: a model to *classify intent* and a model to *extract key entities* of a message. We propose to use supervised machine-learning models for these two objectives. The goal of the two models is to identify what the *meaning and overall goal* behind a message are and to *find key information* that is used for other processes in the chatbot. To illustrate the idea of processing a message through these two models, imagine the following procedure:

1. Bob starts chatting with the chatbot and at some point sends the message "Hey, do you think it's going to start raining in 30 minutes?"
2. The message first gets processed and the `intent` is found to be `weather_inquiry`.
3. Similarly, the entities found are `forecast` and `time` with their respective values `raining` and `30 minutes`.

On a high-level view, the intent classifier model requires to be able to *map* the message Bob sent to a set of pre-determined intents. While the entity extractor should be able to find important concepts within a message.

For the HC side of the system, we wish to maintain full control over the conversations taking place during the execution of microtasks. We do this by stepping away from using any type of predictive model, but we note that it is certainly possible to adopt such an approach. We go back to the idea of QA systems; we propose to perform in essence direct *mapping* from input to output responses or actions. Consequently, this would allow for the opportunity to perform extensive parameterization of the conversational interaction on a system and even microtask level.

3.2.2. Dialogue Management

Using the information processed from the NLU component, the chatbot will have to determine its next course of action. To do so, the chatbot requires to realize that it is engaged in an active conversation leading up to the current state of affairs. Because we assume conversations to be dyadic, we wish to take advantage of the opportunity that allows the chatbot to make inquiries as well for e.g. clarification and asking for additional information. For example, if Bob would ask the chatbot the question "Do you know a good restaurant nearby?", it is unclear what Bob would find *good*, because the chatbot may not possess the knowledge about Bob's dietary preferences and potential allergies. Moreover, perhaps the chatbot is also not aware of Bob's current location, hence does not know what *nearby* would refer to. These things could be clarified by simply asking several questions to follow up on Bob's inquiry.

Similar to NLU, we view dialogue management for the base chatbot and the HC separately. For the base chatbot, we propose to use a mixed approach of using handcrafted rules and a data-driven model. This is a mix of the frame-based approach to match information obtained through the intent classification and entity extraction with the dialogue samples. Then using a model to determine what the exact conversational state the chatbot finds itself in and what action to undertake. We feed the model a set of multi-turn example conversations that specifies "trigger-action" conditions that use the processed output of the NLU as its input.

The HC side uses a pure finite-state approach. We illustrate the key interactions on a high level in Figure 3.5.

We note that it is possible that workers may quit halfway-through a task and start a new task at some later point in time. To account for this, we prompt workers if they wish to continue with a previously unfinished task. This situation will only occur if workers are attempting to start a new task, while the previous task

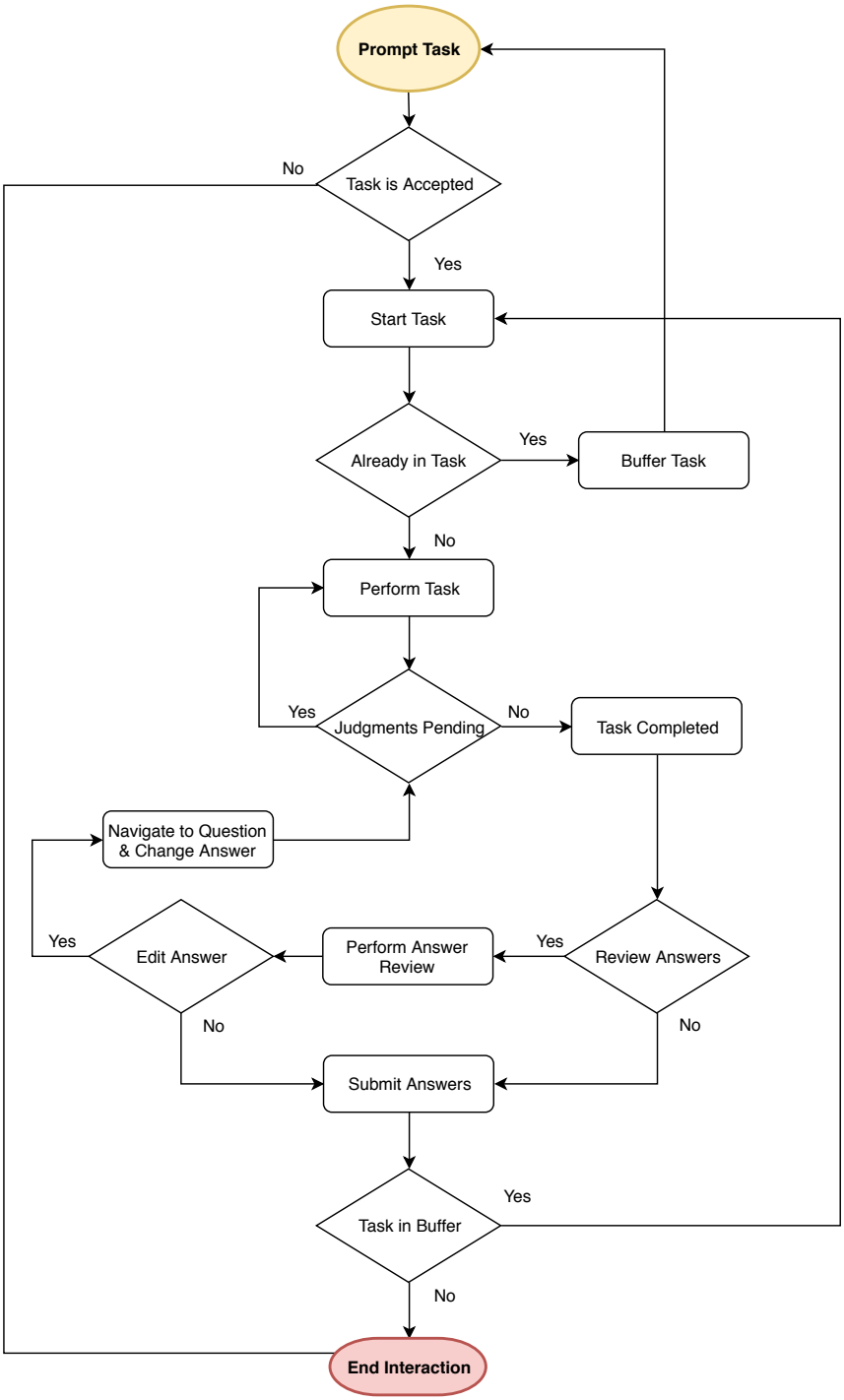


Figure 3.5: High level activity diagram of the dialogue management of the HC side of the system.

was left unfinished and has not expired. In the case that a worker were to accept and finish a previously unfinished task, we prompt after completion if the worker wishes to also perform the new task. We achieve this by *buffering* the new task, while the chatbot waits for the previously unfinished task to be completed.

3.2.3. Action Execution & Knowledge Base

Domain knowledge of the chatbot is stored in a database. The dialogue manager which decides upon which action to take may query the knowledge base for information it requires to serve user requests. To keep the *base chatbot* as general and flexible as possible, we choose to use a *document-based* database as our

knowledge base. To setup querying for information, the knowledge base uses *schema-based matching*. By using the entities extracted from the NLU, we match them with the respective *schema*. Each schema requires to be defined beforehand depending on the use and needs of the chatbot application and its domain. Aside from data records that may be directly queried by their *primary key*, we also choose to allow for querying on attribute (or *key*) level. All actions that require information from the knowledge base follow a pre-defined set of query specifications. To embrace the modular system setup, the communication between the system and the knowledge base goes through an Application Programming Interface (API). This also allows for the future possibility to provide (partial) access to the entire knowledge base of the chatbot for third-party services.

3.2.4. Response Generation

While the possibility exists for generative approaches for response generation, the system is built with the simple frame-based approach involving response templates. The system uses a set of template sentences that hold *slots* or *variables* that require to be filled with information, which allows for reasonably flexible response sentences. While this provides a lot of control in how the chatbot will respond, it however requires a larger amount of (template) samples than the generative approach, in order to simulate a wide vocabulary.

This approach is used for the response generation in both the base chatbot and the HC component. By marking parts of the templates as slots, the system will recognize that certain parts of the template require substitution. To select an appropriate response, the system is able to employ various policies. For example, given the following two templates for a greeting:

```
Template 1: Hello {{first_name}}.
Template 2: What's up {{first_name}}.
Template 3: Hello {{first_name}} {{last_name}}.
```

Template 1 and 2 both contain the single `{{first_name}}` slot, while template 3 contains the two slots `{{first_name}}` and `{{last_name}}`. A greedy slot filling policy may be adopted, that prefers to select template 3 as when the information for both slots is available. Other policies may also include arbitrarily complex conditions, such as ranking candidate responses by the amount of times templates have been previously used, or simply selecting randomly.

3.3. Chatbot Microwork Platform

The HC component in the system allows for the creation and planning, execution and monitoring, and collection of the results of microtasks. As the chatbot is built as an alternative to the web-based microwork platform, we similarly direct our focus to the microtask processes in our system design.

The HC module contains the following main components: the *Task Planner*, *Worker Selector*, *Task Assigner*, *Execution Controller*, and *Result Aggregator*. Each component only activates whenever the processes of the previous components have completed. To illustrate this concept; to select workers, we must first have planned how many workers the task requires. Similarly, before we are able to assign workers, we must of course first have determined which workers should be involved. To execute a task we must first have both the task designed and know which workers to send it to. We depict these sequential relations between these components in Figure 3.4.

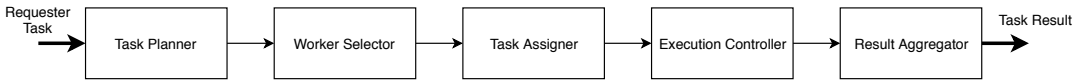


Figure 3.6: High level schematic of the main components of the Human Computation part of the system.

We detail all the system components of the HC module, similarly to the *base chatbot* in Sections 3.3.1 to 3.3.5.

3.3.1. Task Planning

At the core and start of an HC process, is the creation and planning of tasks. The component responsible for the planning of the task is also what is most exposed to an external requester. This means that tasks are to be highly parameterized to allow for better control of the chatbot system for every task's goals. The planning of tasks is only at a microtask level. The system does not support explicitly support macrotasks, nor does it have the ability for automated break-down into a set of microtasks. Consequently, the system does not support setting task execution to follow specified workflows to execute a series of microtasks in succession.

3.3.2. Worker Selection

When a task is created and ready for execution, the number of workers specified by the task must first be recruited. This system component tackles the problem of selecting what workers to include (and exclude) for a specific task. The selection of workers may be specified within the task itself, which may use attributes identifying workers from each other as a filter option. The selection of workers may be kept as simple as permitting anyone, to more complex strategies involving selection through user modelling. For our system, we opt to keep things basic (while keeping the possibility for expanding) through allowing workers to participate as long as they abide by all exclusion rules set on task level, such as preventing workers performing similar task types. We note that this is a more reactive than proactive approach, targeting distinct challenges. In the case, where the system requires to actively seek out candidate workers, the availability of a worker must be considered and is not a given. In consideration for our original goals and experimental purposes in Chapter 5, we leave addressing effective strategies for this challenge as a separate future matter to investigate.

3.3.3. Task Assignment

The system follows the push-based task assignment approach. This approach would allow for more control over managing the number of tasks each worker is able to perform. This means that workers may be less prone to fatigue as a result of enforcing shorter "work sessions". Workers would not be able to pick multiple tasks from a list of available tasks by their own initiative. As the system notifies workers individually with a prompt to request their participation in a microtask, the worker would be also allowed to decline an assigned microtask. In the case a worker would decline a task, this system component would need to find a replacing worker to carry out the task.

Even though the system directly assigns workers to tasks, we note that there is also the possibility to combine the worker selection process together with the assignment of tasks to construct a hybrid approach between the pull- and push-based approach. For example, the worker selection system component would first find *candidate workers* based on some policy. Thereafter, the task assignment component may rank tasks by e.g. urgency or difficulty, and select a subset of tasks to present to the candidate workers. The workers may then pick from a list of possible tasks.

3.3.4. Task Execution

To send and execute the task, this component would be responsible for managing and ensuring that a task gets executed as it is supposed to. Furthermore, the monitoring of the task falls upon this system component. As soon as a task is distributed among a group of workers, answers submitted by the workers are saved to the database of the system. We setup the system to only submit answers to the database as soon as all assigned judgments of a task are completed. This is done to push the full completion of an assigned task, after which the worker is compensated with some remuneration. A possible alternative is to allow payment per judgment, which would allow workers to still get paid even if a microtask were not fully completed. While this may be interesting to have when the system is fully put in production, we constrained the system in order for better experimental control.

3.3.5. Result Aggregation

After a task has finalized, the answers from workers must be aggregated in some way to reach consensus on what is considered the "final" answer. The way answers may be aggregated holds a variety of possibilities. With closed-ended questions—consisting of a fixed set of answer options—standard approaches such as (weighted) majority voting may be directly applied. However, because typically microtasks also use of open-ended questions, other approaches may be used such as computing some metric on the answers and clustering answers based on the computed metric.

As there are many possible strategies for aggregation depending on the circumstances of the task, the system allows for designing dynamic aggregation strategies. This means that rather than being fixed to one aggregation strategy, we wish to have the chatbot system be capable of changing the aggregation strategy whenever we want to.

3.4. Worker Conversational Flow

A large part of building our alternative HC approach to the web-based interface is designing the flow of conversation with a worker. In this section, we touch upon the decisions that have been made for the conversational flow during microtask execution from the worker's point of view. More specifically, we discuss how

microtasks follow a structured conversation which is complemented with access to a set of navigational controls. We discuss the different navigational controls in Section 3.4.1. In addition, we introduce the notion of feedback in Section 3.4.2.

3.4.1. Navigational Controls

By taking a look at a simple web-based interface used for microtask execution (Figure 3.7), typically standard Hypertext Markup Language (HTML) form elements are used.

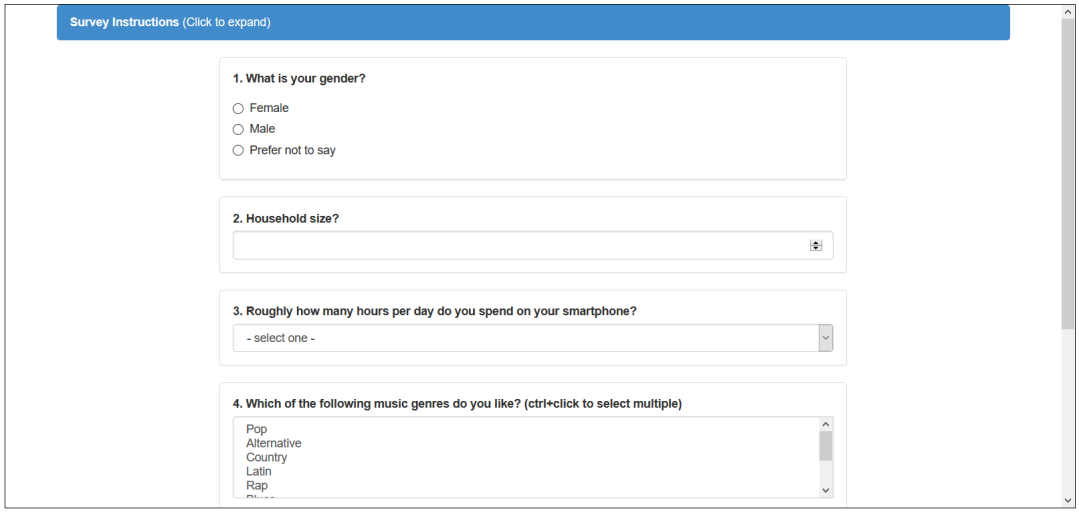


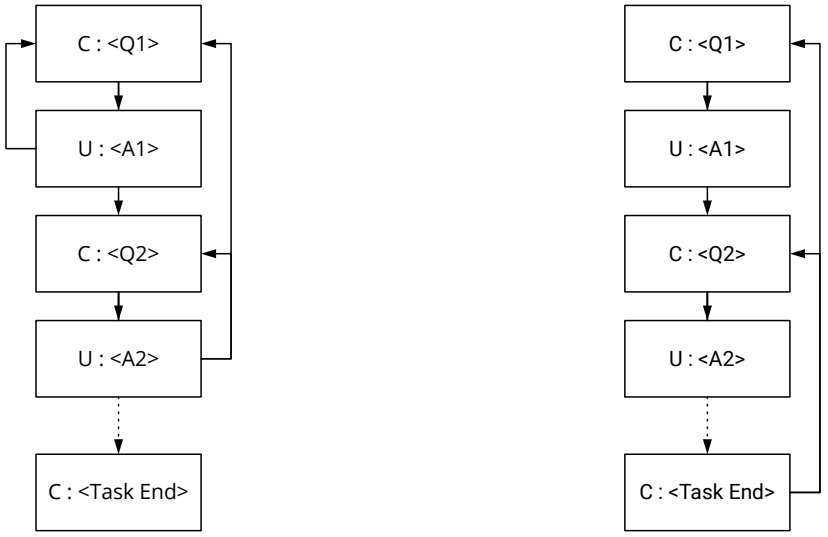
Figure 3.7: Example survey template in the web-based interface from AMT.

Many customary navigational controls, e.g. page scrolling and editing previous form input, are by default provided by the web-browser, these are not present inside the chatbot. While it is possible to design a web-based interface, containing something resembling an online chat-room as a conversational interface, we wish to avoid that the chatbot is only able to function inside the web-browser. We choose to have a generalized conversational interface, which is independent of the environment wherein it is provided. This means that we favour to provide all controls through conversation, rather than depend on the environment to provide them.

Because conversations are structured in chronological order—meaning that it is non-trivial to "go back" to some point in the conversation—we propose several key navigational controls:

1. **Answer editing**: In a web-based interface, it is trivial to edit some answer in a form element as long as it is not submitted. A worker would merely have to overwrite the previous answer by changing the previous input inside the form element. Through conversation, we distinguish two main ways to handle editing of answers as shown in Figure 3.8. We combine both approaches and allow the editing of answers during the execution of the tasks, which only allows navigation to previously answered questions. After all judgments are completed, we allow navigation to all answered questions. To indicate which answer to edit, questions would be numbered in the order they have been sent to the user.

2. **(Re-)viewing task instructions**: As the conversational interface primarily consists of a *chat log*, which depending on the device may show only a limited amount of messages, it becomes important to be able to decompose long task instructions. While in a web-based interface, workers may be accustomed to scrolling through long instructions, we wish to avoid forcing workers to do so in the conversational interface. We consider a conversation as a highly engaging activity, which focuses on the continuous interaction between the user and chatbot. We aim to take full advantage of the capability to actively steer the execution of the microtask as we see fit. We split the task instructions into three commonly included elements: an *overview*, *example* and a list of *steps* workers must follow to complete the task. The overview shows a short description of the overall task. While the example shows what kind of questions will be asked to the worker. Lastly, the steps summarize what is required from the worker to solve the task.



(a) Inter-execution answer editing.

(b) Post-execution answer editing.

Figure 3.8: In total two questions are posed by the chatbot (C), to which the worker (U) answers directly after. (a) When the worker is able to answer, there is also the option to navigate to a previous question. (b) After all questions have been answered, only then the worker is allowed to navigate to any previous question.

Because task instructions are typically displayed before the start of the task (and will be pushed to the back of the chat log at some point), the system allows workers to also request to review task instructions during and after task execution similar to the approach taken in the editing of previously given answers.

3. **Reviewing answers**: In the web-based interface, it is fairly easy to scroll back up the web-page to review previously given answers. In a similar spirit to avoid back-traversal in the chat logs, we include an option to review answers post-execution of the task. This ability would summarize previously given answers and repeats the necessary navigational instructions for editing answers.

4. **Submitting answers**: To finalize answers and complete a microtask, the system must prompt for confirmation. Since submitting answers fully terminates the task, the chatbot will naturally only allow this post-execution.

3.4.2. Feedback Control

With every action a worker takes, the chatbot must respond accordingly. Since interaction between worker and chatbot takes place through the conversation, we setup the chatbot to send acknowledgements through its messages.

In web-based interfaces, closed-ended questions typically use HTML elements—as radio buttons, checkboxes and drop-down menus—that naturally include visual feedback of the selected answers. In the chatbot, we do not possess such standardized interface elements. Therefore, in the chatbot, we choose to send a separate message as an acknowledgement. The main benefit of this approach is that it may be implemented in any kind of conversational interface regardless of its specific application environment. During any microtask, the chatbot is set to repeat any given answer to closed-ended questions through a separate message. For example, if a question were to ask the worker to state what their opinion is on a painting; with two options “like” and “dislike”, picking “like” would make the chatbot acknowledge the worker’s answer by repeating “like” in a separate message.

A big benefit of the conversational interface is that any event may be easily communicated to the user by simply sending a message. As a result, event and error handling are done by communicating this to the user through messages. For example, if a worker were to spend too much time on a task and exceeds its time limit, the worker is blocked from further participation. The chatbot is able to easily communicate this to the worker by explaining the situation through a message. In addition, the chatbot may even take on a didactic role by providing further instructions on how to avoid such future events, what current options the worker is left with, what the consequences (if any) of repeated cases are, etc.

3.5. Requester Task Design

As each task has disparate goals, the creation of tasks should be highly adjustable leaving much up to the exact needs of each task. We discuss in Section 3.5.1, how each task is structured and what parameters are desirable to include in the process of designing them. We also delineate the manner in which answer validation is performed in Section 3.5.2.

3.5.1. Task Structure & Parameters

Because we wish to achieve flexible customization of a task's execution, we setup a list of key parameters for each task. We take note of adjustable task parameters found in popular web-based microtask platforms as AMT and Figure Eight. We list all key task attributes in Table 3.1.

Table 3.1: List of the key task attributes with a brief description on their use.

Task Attributes	**Description**
Title	The title of the task. This is displayed to the worker at the start of the task.
Instruction	A set of instructions to help the worker understand the goals of the task and with directions to complete it; structured through an *overview*, *example* and *steps*.
Deadline	The maximum amount of time in seconds that is allowed to pass before a worker fails the task.
Judgments	The amount of workers requested per data item.
Reward	The payment amount in cents that a worker receives when the worker completes the task.
Items Per Assignment	The amount of data items to include per task assigned to each worker.
Maximum Judgments Per Worker	The maximum amount of judgments that may be collected per worker.

Each task is structured to contain *data items*, over which a set of *questions* may be asked. The set of data items attached to each task is comparable to a database table. Each data item would analogous to a table row, containing a number of attributes that may be used for composing the task questions.

Each data item is to be judged by a number of workers set via the *judgments* parameter. Consequently, increasing the value of this parameter may yield more answers (but at a higher cost in due payments).

3.5.2. Answer Validation

To setup basic quality control of answers, the task can employ answer validation. In the web-based interface, for typical form elements, client-side scripting may be used to validate inputs. In the chatbot, every answer would be passed server-side in this case. This allows the system to check all incoming answers against a predefined set of input rules. We distinguish two sets of validators based on open- and closed-ended questions, and open- and closed-input types:

1. **Open-input validator**: All open-input validators behave by comparing the worker's answer (which is a naturally a string) to a set of rules. For example, when the worker answers by typing out a short sentence, a validator may enforce a character limit or check that the sentence adheres to some regular expression.

2. **Closed-input validator**: All closed-input validators relate to checking of higher level rules. These validators enforce that the worker's answer e.g. covers a minimum or maximum amount of options or that some option may not be chosen in combination with another.

3.6. Modelling the Chatbot System

In Appendix A, we show the entire system modelled from a data perspective. The model contains both entities used for the base chatbot and for the chatbot as a microtask crowdsourcing platform.

In our system, we do not distinguish between worker and regular chatbot users. Instead, we embrace the opportunity to have regular users become workers as well. As a result, we model the users of our chatbot system as a single *User* entity. To model user interaction, we propose to represent all user (and chatbot) messages as *Utterances*. As a result, a *Conversation* as an abstraction (which bears resemblance to a chatlog) is introduced as a collection of sequentially sent *Utterances*. To drive these conversations, the chatbot uses a conversational *Model*. This model is then powered by a set of response templates (represented as a *Templates* entity), that may rely on the abstraction of chatbot *Knowledge*.

Since we also want to be able to extend our range to recruit workers from external sources outside the chatbot, we model the worker's access point for communication through a separate *Platform* entity. Consequently, we model our tasks to allow them to concurrently run on different platforms as soon as it gets published.

Naturally, tasks that are published require to be assigned to workers (in our model *Users*). We model our tasks, in such a way that they contain *Data Items*, which in turn contain a collection of *Question* entities. By modelling each *Data Item* with a set of questions, we gain the benefit of having a distinct set of questions per*Data Item*. This also allows us to model more complex tasks, with different types of questions per *Data Item*. These *Data Items* are then assigned to workers alongside the set of questions. To keep track of which worker is providing an assessment on which *Data Items*, we introduce a *Task Instance* entity. This entity models all task session-specific parameters, such as *task start and completion date*. Finally, the entire task *Results* are collected as a separate entity by aggregation of all answers. To embrace the notion of the *crowd-powered chatbot*, we allow these results to be used for updating the chatbot *Knowledge*.

4

Chatbot Implementation

In this chapter, we elaborate upon the implemented chatbot system. This system follows the modular design as specified in the design. Following a similar structure as in Chapter 3, we begin by discussing the base chatbot in Section 4.1, then follow by the HC component in Section 4.2. Finally, we detail the deployment of the chatbot in Section 4.3.

4.1. Conversational Interface

The entire chatbot system has been implemented using Python v2.7 to avoid compatibility issues with older third-party Python modules. The chatbot consists of the four main components: we discuss the NLU in Section 4.1.1, dialogue management in Section 4.1.2, the setup of the knowledge base in Section 4.1.3 and finally the generation of responses in Section 4.1.4.

4.1.1. Natural Language Processing

Every user message first passes the NLU component in the chatbot, before any further action is taken:

For the base chatbot, we used the open-source library *Rasa NLU*[1] to process all user messages. For each user message, Rasa NLU performs two main activities: 1) *classification of user intent* and 2) *entity extraction.*

The classification of user intent is done via the training of a simple linear Support Vector Machine (SVM), which Rasa NLU carries out through the popular machine-learning library *scikit-learn*[2]. For the purpose of demonstrating the capability to perform the NLU in a data-driven way, we trained a linear SVM using the default settings in Rasa NLU on a set of sample sentences within the restaurant domain.

The extraction of entities is performed by Rasa NLU, by first processing the user message through the *spaCy*[3] library, which provides tools for Natural Language Processing (NLP). We note that it is possible to use different NLP tools for this process as well. Since spaCy provides simple packaged models—which includes a tokenizer, part-of-speech tagger, parsing, named entity recognition, and pre-trained *word vectors*—for a variety of languages, we chose to use their simple English model in our implementation. As a result, our chatbot only covers NLU for conversations carried out in English.

For the NLU during execution of microtasks, we use a more rigid approach. Instead of relying on the data-driven models, which performance highly depends on the quality of the training data, we use AIML. AIML matches the user message to a set of sentence templates, which may contain certain keywords or patterns. In other words, we perform rule-based pattern matching to map each user message to a specific output. We use AIML to detect the instruction of the navigational controls as shown in Table 4.1.

Whenever a user message contains—at any position in the message regardless of capitalization—one of the exact keywords listed in Table 4.1, we perform the corresponding action.

[1] https://rasa.com/docs/nlu/
[2] http://scikit-learn.org/stable/
[3] https://spacy.io/

Table 4.1: All implemented navigational controls.

Navigational Control	AIML Keyword
View Task Instructions	TASK INSTRUCTION(S)/HELP
View Task Example	EXAMPLE
View Task Steps	STEPS
Indicate Ready To Start Task	CONTINUE, GO, READY, BEGIN
Edit Answer	EDIT/CHANGE (QUESTION) <question nr>
Perform Answer Review	REVIEW

4.1.2. Dialogue Management

With the information obtained from the NLU component, the dialogue management determines the next course of action to undertake. For the base chatbot, we use *Rasa Core*[4] which integrates well with Rasa NLU. Rasa Core similar to Rasa NLU operates on a trained model, which similarly benefits from higher flexibility in dialogue management, but may be less robust than rule-based approaches.

For the HC component, we use AIML, which allows us to map user messages to any type of output. However, this approach means that we require to handle each mapping individually. Consequently, this means that to power the interaction between the worker and the chatbot, the chatbot has been implemented using several simple states. We show the interaction for performing microtasks on a high-level in Figure 4.1.

Whenever a worker accepts a task, the chatbot first sends the details of the task to the worker. Each task contains at least one judgment, which is then sent to the worker accordingly. Each answer of the worker passes a validation state, which is determined by the specified validator per question. Finally, the task may be finalized when all assigned judgments have been completed by the worker.

4.1.3. Knowledge Base

At the core of the chatbot stands the data storage. We refer to the entire system's data storage as its knowledge base. We separate the knowledge base into two parts: 1) for use of the general knowledge of the chatbot and 2) the storage of all data involved in the HC processes.

Since the system is based on separate system modules, the knowledge base is no exception. This also allows us to develop and deploy the knowledge base separately from the chatbot. To facilitate the communication between the chatbot system and its database, we implement an API. The system is then able to communicate with the knowledge base via simple HyperText Transfer Protocol (HTTP) requests. We support in the system the basic `GET`, `POST`, and `DELETE` HTTP requests. In addition, to update any knowledge base *document* (analogous to a single data record) we use `PATCH` requests.

The knowledge base is implemented using MongoDB v4.0.2, which was at the time the latest available stable version. As a result, all data is stored in a JavaScript Object Notation (JSON) format.

4.1.4. Response Generation

To have the chatbot form its responses, we implement a frame-based slot filling mechanism, using handcrafted information to fill the slots. The system uses handcrafted sentence templates, which contains *slots*, which may be filled using a pre-defined set of options or using information obtained during the interaction with the user.

The chatbot also uses *emojis* as certain slots in its response templates. For ease of use, the chatbot relies on the *emoji*[5] library to perform the mapping from *emoji codes* to the Unicode representation determined by the Unicode Consortium[6]. To use the *emojis* in a more flexible way, we categorize the *emojis* we included in our set of responses by their goals and expression as follows in Table 4.2:

Each emote expression may then be used as a separate slot inside a response template. The chatbot then randomly selects from the set of emote codes within the emote expression category. For example, by concatenating multiple slots together such as:

```
TEMPLATE 1: {greeting} {emoji_greeting}!
```

[4] https://www.rasa.com/docs/core/
[5] https://github.com/carpedm20/emoji/
[6] http://www.unicode.org/emoji/charts/full-emoji-list.html

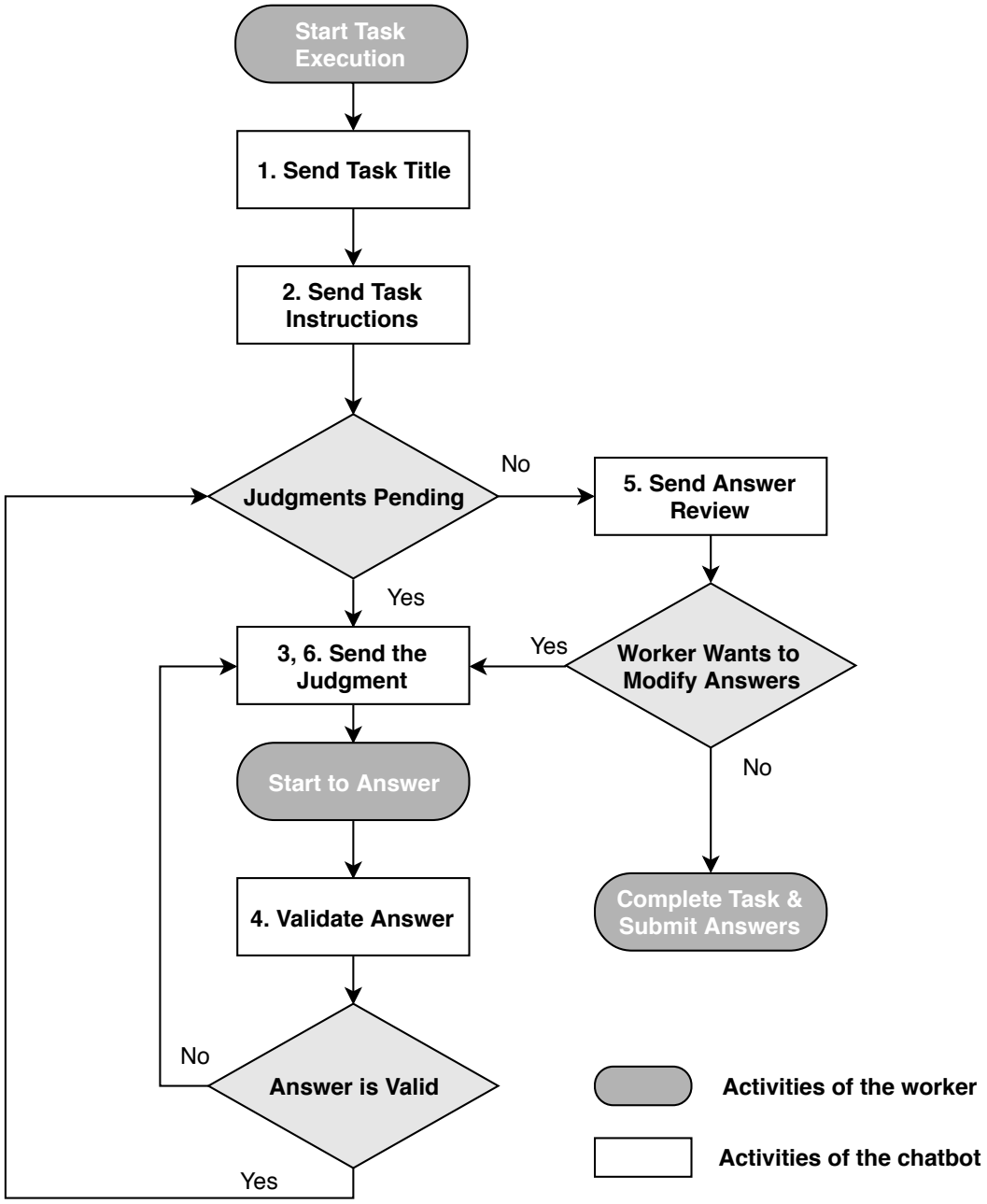


Figure 4.1: High level overview of the conversation management logic of conducting a microtask.

We are then able to construct entire response sentences, by randomly selecting the slot values from within their pre-defined values. In the previous example, we sequentially select a value among the `greeting` set, then do the same for the `emoji_greeting` set and finally perform a substitution to build the final response.

4.1.5. User Interface

The logic of the chatbot is implemented as a singular module, which is separated from the UI. For our UI, we choose Telegram[7] as a platform to host the chatbot. Because the UI is loosely coupled from the chatbot logic, we may easily extend our chatbot implementation to reach out to more users by supporting other messaging services such as Facebook Messenger[8] or Slack[9].

[7] https://telegram.org/
[8] https://www.messenger.com/
[9] https://slack.com/

Table 4.2: All emote codes used in the responses by the chatbot.

Goal/Expression	Emote Code
Wondering	:face_with_monocle:, :thinking_face:, :face_with_raised_eyebrow:
Cheeky	:smirking_face:, :face_with_monocle:, :face_with_steam_from_nose:, :speak_no_evil:
Confusion	:confused:, :persevere:
Happy	:heart:, :face_blowing_a_kiss:, :hugging_face:, :smiling_face_with_smiling_eyes:, :kissing_face_with_closed_eyes:
Anger	:angry:

Telegram provides an extensive API[10] for hosting and building the chatbot interface. Similar to other messaging services, Telegram provides two main ways of interaction for chatbots: *text-* and *keyboard-based* interaction. All text-based interaction is represented via *Message* objects, which may contain multimedia content such as a message, videos, audio. Each Message object may also contain a custom *Keyboard object*. We specifically make use of *inline keyboards*, which are rendered as custom *buttons* within an ongoing chat with the user.

In addition, Telegram also provides native support for rendering *inline audio players* whenever an audio file is sent to the user as shown in Figure 4.2.

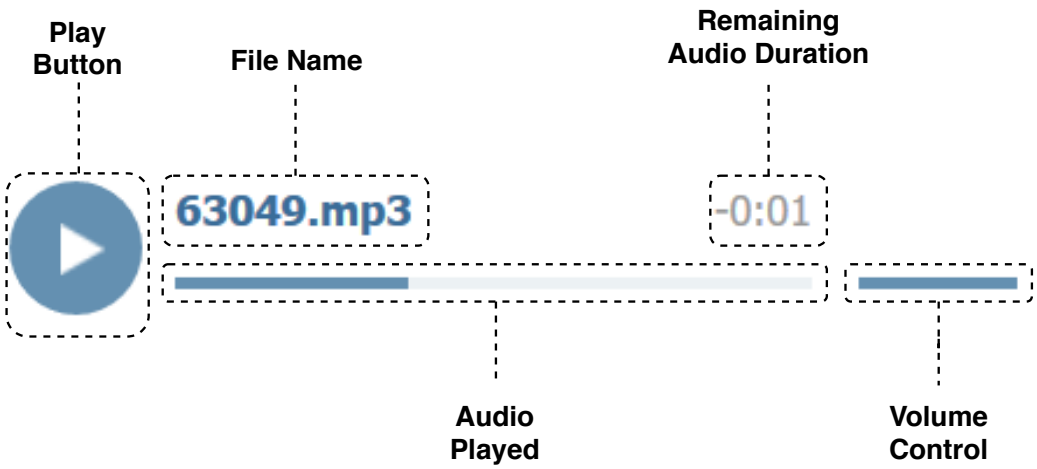


Figure 4.2: Inline audio player in the Telegram Web Client.

4.2. Microtask Execution

In this section, we upon a few intricacies on the execution of microtasks inside the chatbot. We first discuss the challenges arising in the assignment of tasks on a technical level in Section 4.2.1. We then follow-up, by showing the possibility to extend the chatbot as a microtask crowdsourcing platform to function with external third-party platforms in Section 4.2.2.

4.2.1. Concurrency in Task Assignment

As we can imagine, the distribution of tasks to collect judgments has to deal with issues such as keeping track of how many workers are assigned to which tasks. While we perform task assignment on a policy-level, the system still requires to know which judgments are collected from whom to prevent assignment of tasks on pending judgments, which occurs while workers are still performing their task.

[10] https://core.telegram.org/bots/api

We introduce a *lock* mechanism to prevent assignment to pending judgments, inspired by *semaphores*. We update per *data item* (which may collect multiple judgments), a counter which keeps track of which workers are currently working on that particular data item. This counter is incremented for each new worker that starts a separate task instance. Similar, we decrement the counter, whenever a task is completed or has failed due to exceeding the task deadline.

4.2.2. Integration with Third-party Resources

To allow our chatbot to function within popular messaging services, we adopt an event-driven model. This means that our chatbot acts reactively, upon subscribing to a service that sends events (which may be triggered by e.g. a user sending a message) to our chatbot system. We achieve this by attaching *webhooks* to indicate such a subscription to our system. In our implementation, we attach a *webhook* to our system from within the Telegram services. As a result, our chatbot may also incorporate multiple webhooks from varying messaging services. This allows our chatbot to function simultaneously in multiple messaging services e.g. Telegram and Facebook Messenger.

Furthermore, to provide integration with third-party microwork platforms, such as Figure Eight or AMT, we may use deep linking of the chatbot to redirect workers to start interacting directly with the chatbot. This is something that is also natively supported by Telegram's data model to represent conversations. We similarly model in our system conversations between users as a separate entity. This may allow our system to be extended for the future possibility to perform deep linking on a separately developed messaging service.

In order to verify that externally recruited workers (e.g through Figure Eight) have successfully performed microtasks inside the chatbot, we developed a validation mechanism based on the distribution of randomly generated validation tokens. We summarize the core concept in Figure 4.3.

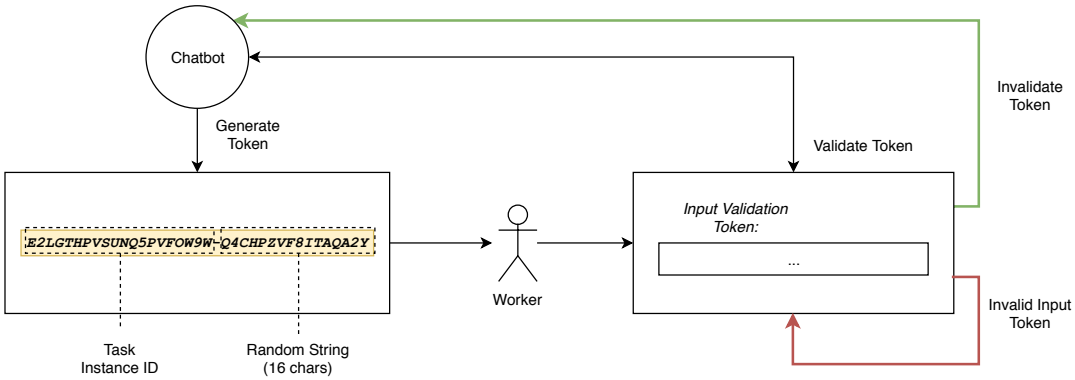


Figure 4.3: Validation Token Mechanism.

Whenever a worker completes a task, the chatbot generates using the task instance ID a random validation token. We include in the token also a random string of 16 characters long to ensure that the token is at least of reasonable size to minimize odds of accidental token collision. This token is saved as a separate attribute for that specific task instance. The worker puts this token somewhere inside the external platform (e.g. as a question in the task), the platform then runs a validation request to our chatbot system. Our chatbot then verifies that token is both previously *unused* and *matches the validation token found in the database*. Afterwards, if the process is successful, the chatbot invalidates the token to prevent it from being used again.

4.3. System Deployment

In this section, we elaborate on the deployment of the chatbot system. First, we detail the even-driven model used in Telegram in Section 4.3.1. Then we discuss the configuration of the web server to host the chatbot system in Section 4.3.2.

4.3.1. Event-Driven Model

Since messages are sent based on certain events, the quantity of events depends on what events the messaging service considers. In our case, for Telegram we receive events based on *user messages*, *button interactions*, but also from webhooks attached to tasks running in external microtask crowdsourcing platforms.

Since every user message is captured as an event in Telegram, naturally we may also have to deal with the issue of *spam*. To combat a potential case of an overflood of events by single users, we limit the events by filtering them based on their timestamps. More precisely, we ignore events that contain similar payload to previous events within a set time-frame. We note that our approach is a very simple, but still effective approach for distinguishing individual spammers. However, this does not protect us from potentially more harmful attacks targeting the entire system such as Distributed Denial-of-Service (DDoS) attacks.

4.3.2. Web Server Configuration

To conduct the experiments with real-world crowd workers, we deploy our chatbot system on a production server. For this we use a simple Virtual Machine (VM)—Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz, 4 processors, running Ubuntu 16.04.4 LTS—hosted by the Dutch SURF[11].

We used *Flask*[12] as our web framework, with *Gunicorn*[13] as our Web Server Gateway Interface (WSGI) HTTP server. Furthermore, we used *Nginx*[14] as a reverse proxy for forwarding incoming requests to our running services.

We also hosted our own database and the API to facilitate database communication between our services on the same VM.

To setup the webhook for receiving events from Telegram, our services required to comply with the HyperText Transfer Protocol Secure (HTTPS) protocol. To receive a valid SSL/TLS certificate, we used *certbot*[15].

Since we do not expect too heavy load, we run Gunicorn for the API with four separate *worker processes*. For our chatbot system, we run a single worker process. We note that we also enabled multi-threading.

[11] https://www.surf.nl/
[12] http://flask.pocoo.org/
[13] https://gunicorn.org/
[14] https://www.nginx.com/
[15] https://certbot.eff.org/

5

Viability of Chatbot Microtask Crowdsourcing

To answer RQ2 and RQ3, we design and run three separate experiments, which together consisted of a total of 24 separately run microtasks. We first discuss the overall design of the tasks in Section 5.1. We finally discuss the execution of each experiment in Section 5.2.

5.1. Experimental Design

In this section, we touch upon the experimental design to setup the tasks. We discuss first the overarching goals of the experiments in Section 5.1.1. Then discuss the entire setup of the tasks and recruitment of workers in Sections 5.1.2 to 5.1.5. To evaluate the experiments, we state what our dependent variables are in Section 5.1.6. Finally, we discuss what the execution flow of the web and chatbot tasks are in Section 5.1.7 and how we implement these tasks in Sections 5.1.8 and 5.1.9.

5.1.1. Experiment Goals

To answer RQ3, we are interested in comparing the performance (task execution time and output quality) of different tasks in the web-based interface and the conversational interface. Moreover, we wish to find how different types of input control may influence the performance. We aim to find out what task types—which differ in data and input control type—are more suitable for the web-based interface and conversational interface.

In the *first experiment*, we compose six basic and common task types and execute them in both the web-based and conversational interface. The main goal of this first experiment is to look if there are any discrepancies in terms of work speed and quality between executing generic microtask types within the web-based and conversational interface. To this end, we compare for all tasks the task execution time and output quality between each task type for the web-based and conversational interface.

In the *second experiment*, we investigate what the implications are of using different input control types for closed-ended questions (i.e. questions to which a list of answer options are provided). For this experiment, we therefore only require to design tasks that exclusively contain closed-ended questions.

In the *third and final experiment*, we wish to find out if initially hiding task instructions in the chatbot impacts the task execution time. Furthermore, we raise the question of how often workers truly view task instructions.

Lastly, as part of the main RQ, we want to know how workers would feel about doing microtasks through a conversational interface. Hence, we survey the workers on their demographics and experience with the chatbot.

5.1.2. Selecting Task Types

To first find out what web UI elements are typically used in tasks, we take a look at the task templates provided by the popular web-based microtask crowdsourcing platforms Amazon Mechanical Turk (AMT) and Figure Eight. For the full list of task templates from both services see Appendix C.

We distinguish and categorize in Table 5.1 the most prominent *task types* in terms of *data type* and *input type element*. For each task type, we map the web UI elements to the possible UI representations in the chatbot. In addition, we introduce four task type categories based on this mapping from input type element of the web to the chatbot interface.

Table 5.1: Categorization of key task types based on the templates from AMT and Figure Eight.

Categorization	**Task Type**	**Data Type**	**Input Type Elements**	
			Web	**Chatbot**
C1	Image Annotation	Image	Multiple Choice	Text, Buttons
	Data Collection	Text	Multiple Choice, Text	Text, Buttons
	Search Relevance	Image	Multiple Choice	Text, Buttons
	Data Categorization	Image, Text	Multiple Choice	Text, Buttons
	Sentiment Analysis	Text	Multiple Choice	Text, Buttons
	Data Validation	Image, Text	Multiple Choice	Text, Buttons
C2	Speech Transcription	Audio	Text	Text
	Character Recognition (human OCR)	Image	Text	Text
C3	Object Labelling	Image	Drawing (Bounding Box, Line, Markers)	HTML Game
C4	Survey	None	Different types-several questions	Text, Buttons, HTML Game

From the task types in Table 5.1, we aim to include all unique triplets of *data type* and *input type elements*. As a result, from category *C1*, we select the *Image Annotation*, *Data Collection*, and *Sentiment Analysis* task types. We exclude the *Data Categorization* and *Data Validation* task because they draw very close similarities to the Image Annotation and Sentiment Analysis tasks in terms of the data type and input type elements.

We fully include the *C2* and *C3* categories in our experiments; picking *Speech Transcription*, *human OCR* and *Object Labelling*. We finally do not include the *Survey* task type, because we already cover the input type elements through the aforementioned task types.

Because we are ultimately interested in mapping the web interface UI elements to representative chatbot UI elements, we identify and categorize in Table 5.2 the following UI elements in both web and the chatbot interface. Through this mapping of the web to the chatbot interface, we set the focus on four distinct types of UI elements:

1. **Free Text**: All open-ended questions are answered through freely typed text that is input by the worker through a keyboard. Among the included task types, the *Data Collection*, *Speech Transcription* and *human OCR* task types require the worker to type out free text answers.

2. **Single Selection**: All closed-ended questions require the worker to select their answer from a list of provided answer options. The single selection UI element allows for only a single option to be selected. Among the included task types, the *Sentiment Analysis* uses the single selection element.

3. **Multiple Selection**: Closely related to the single selection UI element, this instead allows for multiple options to be selected. Among the included task types, *Image Annotation* makes use of this element.

4. **Image Segmentation**: To identify local objects in an image, the image may be "split" through various means. We focus only on the drawing of *bounding boxes* because from an informational point of view we gain in most cases more information than drawing *lines* and *markers*. This is because we would also

be able to estimate or derive desirable properties gained from drawing lines and markers—as object centre and surface—through the results achieved from drawing bounding boxes. Among the included task types, the *Object Labelling* task type uses this UI element.

Table 5.2: Summary of considered UI elements and their implementation, in both Web and Chatbot interface.

UI Element	**Web**	**Chatbot**
Free Text	Single/Multi-Line Text	Message
Single Selection	Radio Buttons	Single Button
Multiple Selection	Checkbox(es)	Multiple Buttons
Image Segmentation	Bounding Box	Select-Grid

5.1.3. Worker Interface

To run the tasks in a web interface, we use Figure Eight[1] as the microtask crowdsourcing platform. Figure Eight provides a standardized way to build simple web pages suiting microtask execution. The web page is constructed using a combination of HTML and Custom Markup Language (CML)—which is claimed to be a superset of HTML[2]—Cascading Style Sheets (CSS) and JavaScript (JS). While the customization options for building the web page is extremely potent, for our experiment we use only standard interface elements to minimize odds of spurious experimental results due to unfamiliar or highly customized UI elements.

For the *Free Text* UI element, Figure Eight is able to provide simple *text boxes* and *text areas* for inputting a single text sentence or larger paragraphs respectively. We depict these two UI elements in Figure 5.1. Workers may type out any answer by first gaining the input focus of the text box or area, which is determined by the web browser of the worker.

(a) Text box in Figure Eight.

(b) Text area in Figure Eight.

Figure 5.1: Standard HTML text box and area rendered through Figure Eight's web interface.

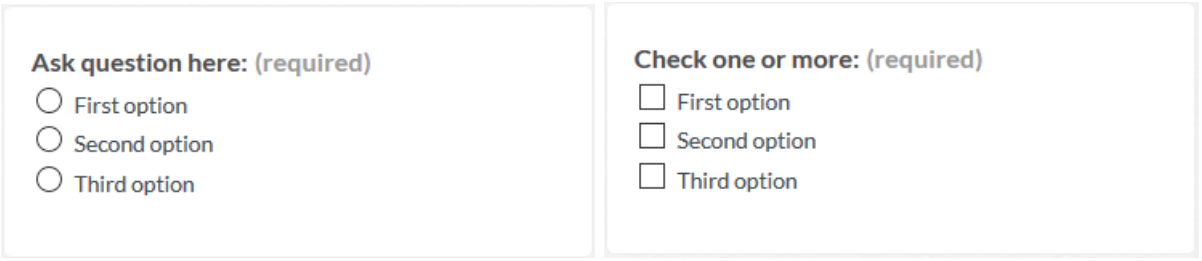
For *Single* and *Multiple Selection*, radio buttons and checkboxes may be used (shown in Figure 5.2), which are analogous to the standard HTML elements of similar name. As a result, the standard functionality of radio buttons allows us to mimic the behaviour of enforcing single input answers. Similarly, the checkboxes allow workers to tick multiple options at once to indicate multiple selected answers.

For *Image Segmentation*, Figure Eight provides a custom "Bounding Box" tool as seen in Figure 5.3. This allows workers to freely "draw" boxes of rectangular shapes—setting a starting point of any of the box's corner and dragging it alongside its diagonal to indicate the ending point—to enclose objects in the image within the bounds of any of the drawn shapes. Workers are allowed to draw as many shapes as they deem necessary, so it is up to the task instructions of the task to clarify how many shapes and how they should be drawn.

In general, tasks run in the web interface of Figure Eight show from top-to-bottom order the following task elements as shown in Figure 5.4:

1. **Task title**: A single text sentence indicating the title of the task. This sentence is displayed at the very top of the web page.

[1] `https://www.figure-eight.com/`
[2] `http://crowdflower.github.io/CML/#/?id=what-is-it`



(a) Radio Buttons in Figure Eight.

(b) Checkboxes in Figure Eight.

Figure 5.2: Standard HTML radio buttons and checkboxes rendered through Figure Eight's web interface.

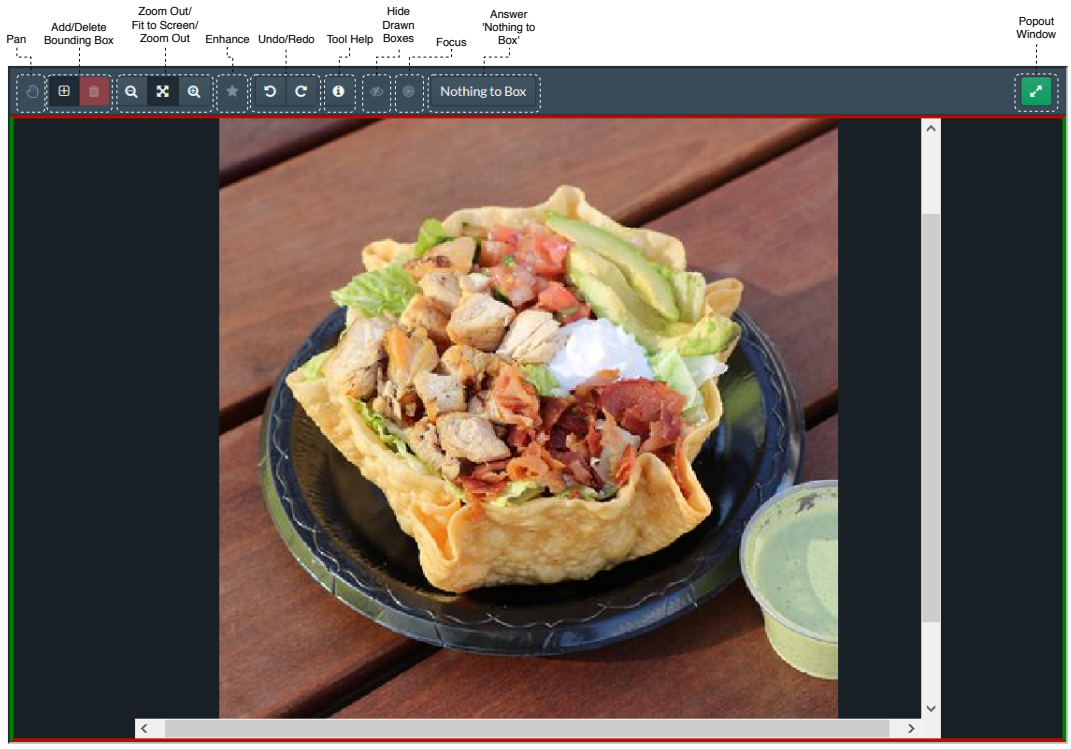


Figure 5.3: The custom Bounding Box tool provided in Figure Eight. The *Enhance* feature allows for certain images to increase contrast. The *Focus* feature allows hiding all drawn boxes except for the one that is focused on.

2. **Hide/Show Instruction button**: A simple button that toggles hiding or showing the entire section containing the task instructions.

3. **Instructions**: The task instruction content, which may be as long and thorough as a requester wishes it to be. The instructions may contain basic instructional text, as well as visual aids such as images and videos to describe anything that is necessary to successfully complete the task per definition of the requester. If the web page is not able to display the entirety of the instructions all at once on screen, the worker will have to use page scrolling to navigate up and down the web page.

4. **Judgment**: Each judgment consists of data content (if present) and a set of questions. The data content is typically displayed before the questions, but may also be shown after. Any question which is posed above the web UI element also shows if the question is *required* (i.e. input validation on empty inputs) or *optional* in the case it does not show the `(required)` tag.

5. **Submission button**: A simple button that ends the task and submits all the provided answers to all judgments.

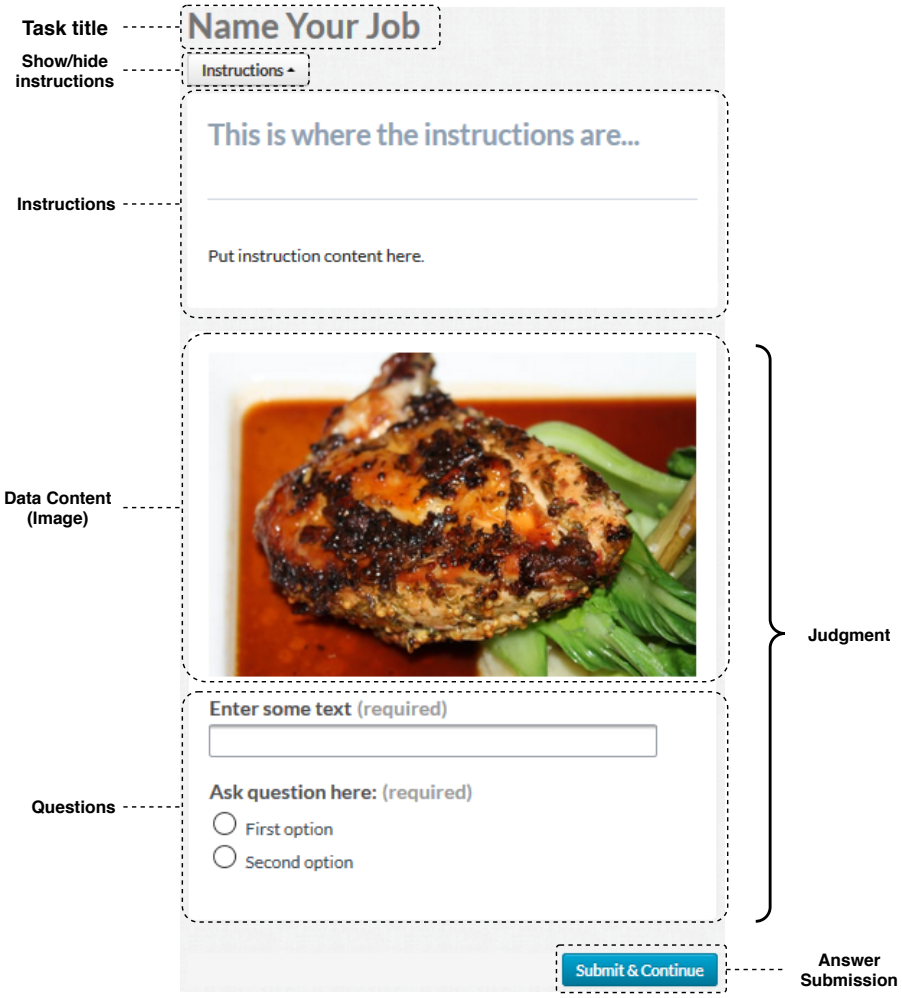


Figure 5.4: The task structure in the web interface rendered through Figure Eight.

When a task contains more than a single question (including repeating questions when multiple judgments were asked), all content items and their respective input elements are presented from top-to-bottom within the same web page.

For the *conversational interface*, there is currently no standardized implementation for UI elements. Therefore, we use the standard UI elements in our chatbot implementation in Telegram to mimic the typical work environment found in the web interface. Since microtask crowdsourcing UIs typically emphasize minimalism and ease of use, we similarly share this design principle in the creation of the *conversational flow*, driving the execution of the task. Based on the typical elements from the web interface, we introduce them to the conversational interface in a similar manner. The entire execution of a task is represented by the six simple states in Figure 4.1. Moreover, we show a basic example of the conversational flow in the conversational interface in Figure 5.5.

We summarize the interaction between the worker and chatbot within a microtask as follows:

1. **Starting the Task**: To start the task, the worker has to begin a conversation with the chatbot by pressing a *Start* button as shown in Figure 5.6.

2. **Sending Task title**: Similar to the web interface, the chatbot sends in natural language a single message in which the title of the task is specified.

3. **Sending Instructions**: The chatbot next sends a brief overview of the task per definition of the requester and allows the worker to inquire for a more elaborate explanation of the task. Workers may pick to view a simple example and also a list of steps to successfully finish the task. Furthermore, the chatbot also includes an option to view additional instructions on how to edit previously given answers. Finally,

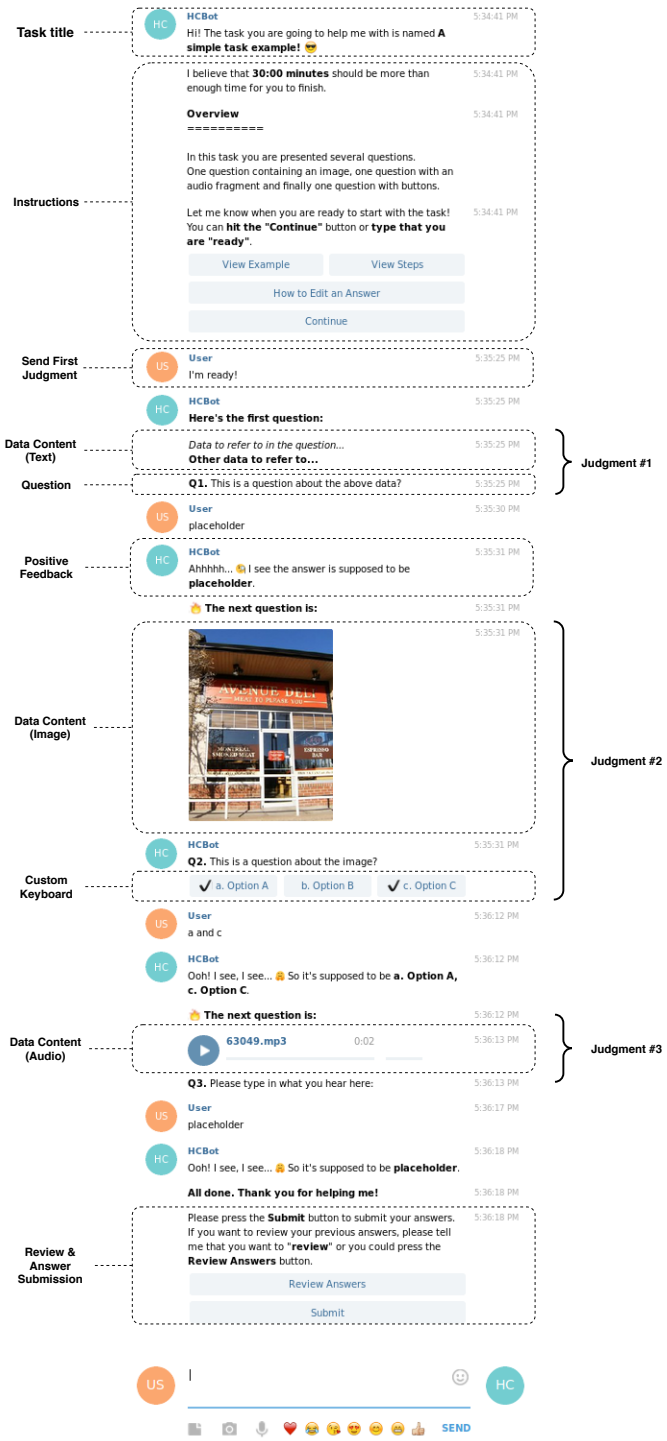


Figure 5.5: An example of the Conversational Work Interface developed for the experiment.

the chatbot waits for the worker to indicate that they are ready to start the task before sending the first question.

4. **Sending Judgment**: The chatbot keeps sending judgments one by one until there the worker has done all judgments it had been assigned to. Before a new judgment is sent, the chatbot awaits the answer of the worker each time and validates the answers if any validation rules have been provided. The judgment is repeatedly sent as long as an answer does not successfully pass validation. Similar to the web interface, judgments may consist of a data item and a set of questions. This means that it is also

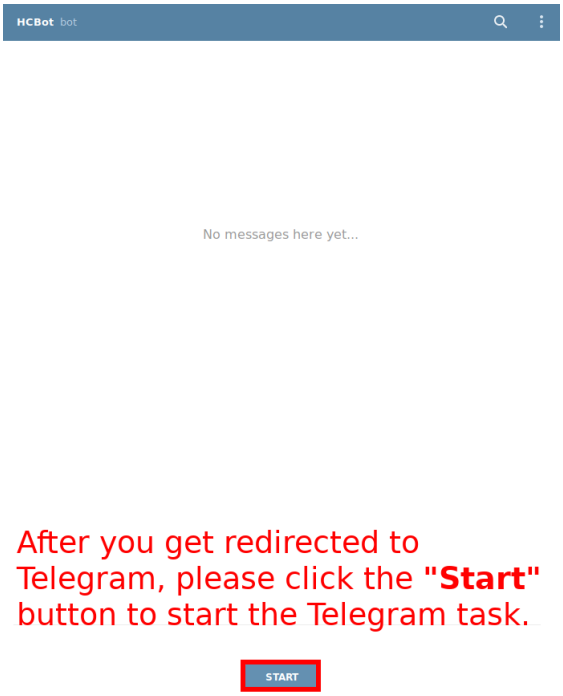


Figure 5.6: Start button for starting a task in the chatbot.

possible that multiple questions are asked per data item. Workers must provide their answer through either the *Message* UI element (i.e. the area in which the worker may type their answer) or through any interactive elements provided by the chatbot.

5. **Review Answers & Submit**: After all judgments in the assigned task have been completed by the worker, there are two options provided; 1) *Review Answers* and 2) *Submit*. The former option summarizes all given answers to each question and repeats instructions on how to edit previously given answers. The latter ends the task and submits all the given answers.

For the single- and multiple selection UI elements in the chatbot, we have designed four different ways—we name *Custom Keyboards*—to provide an answer. The first two are based on the *Free Text* UI element (see Figure 5.7). We provide a worker with a list of options in *Text* and ask them to:

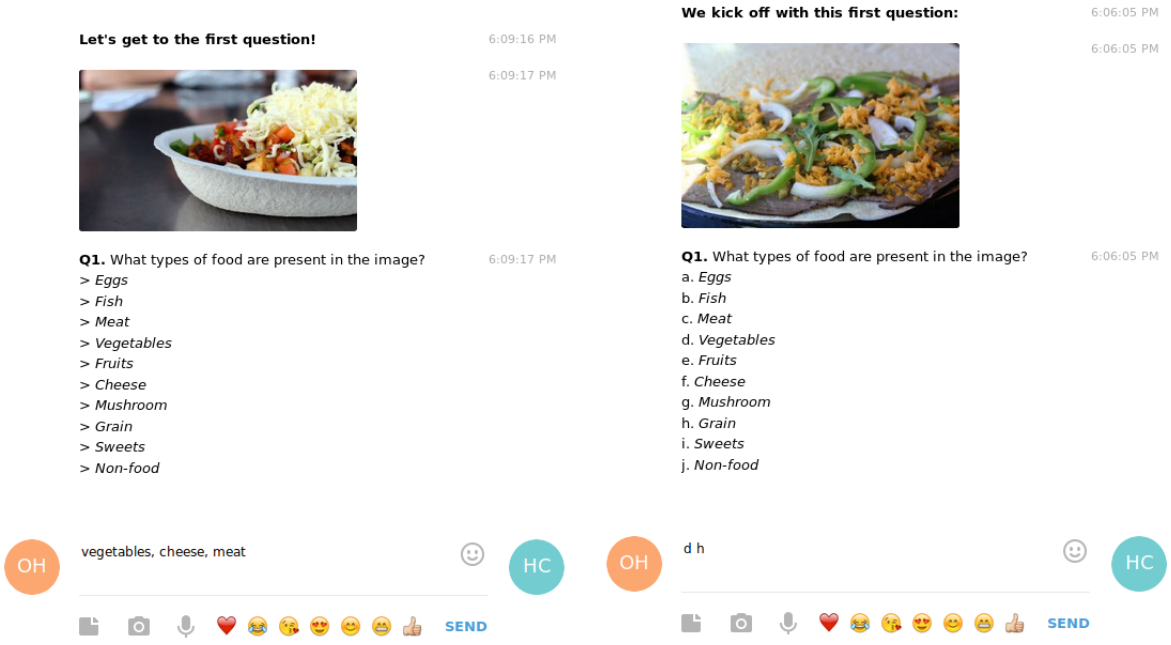
1. Answer through *Text-Only* input. This requires the worker to type out the entire option text. For the selected task types (*Sentiment Analysis* and *Image Annotation*), capitalization is not strictly required, thus not validated upon. This allows for more flexible inputs and prevents the need for having to capitalize all input answers in case of multiple selection (which follows our design accounting for workers' ease of use).

2. Answer through *Code-Only* input. Contrary to the *Text-Only* input, herein each option is prefixed with a code (followed by a *period* symbol, which is not part of the code but merely used to enhance readability). We supply in our case an alphabetic code by following each option in the order wherein they are presented. Workers must answer by typing out these codes.

For both of these *Custom Keyboards*, in the case of multiple selection, answers may be provided (regardless of order) by separating them with a *whitespace* character or *commas*.

The two other *Custom Keyboards* are based on the *Single-* and *Multiple Selection* UI elements (see Figure 5.8). We provide workers with *Buttons* and ask them to:

1. Answer through *Button-Only* input. This requires the worker to select a button that contains their preferred answer. Thereafter, workers must indicate their final selection by selecting the *Next* button. This button confirms their selection and lets them proceed further with the task.



(a) Text-Only Custom Keyboard.

(b) Coded-Only Custom Keyboard.

Figure 5.7: Text-Only and Code-Only Custom Keyboards. Note that in both cases, *whitespace* or *commas* may be used to separate answers. In these examples, (a) uses *commas*, while (b) uses *whitespace*.

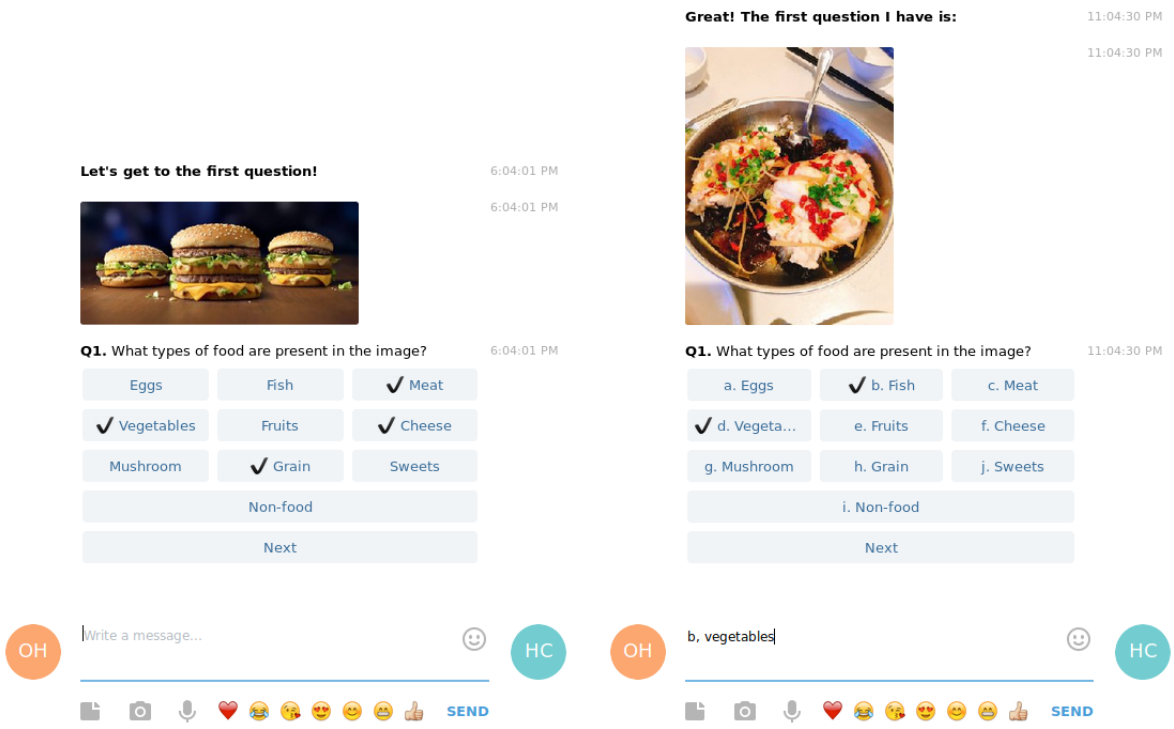
2. Answer through *Mixed* input. This is a combination of the *Text-Only*, *Code-Only* and *Button-Only Custom Keyboards*. This means that workers may pick whichever way they prefer to answer. We note that we allow within a single question, workers may mix their way of answering. As a result, workers may answer for a single question by typing out the full-text option alongside with using coded options. We give priority to *Free Text* input over *Button* input, which means mixing *Button* input with *Free Text* results in the *Free Text* input to override any previously selected *Button* input.

5.1.4. Worker Selection

To recruit workers for our experiments, we use the microtask crowdsourcing platform Figure Eight wherein we also have built our web-based task interface. For all tasks running in both the web and chatbot interface, we allow the participation of workers with a *contributor level 1* or higher. Figure Eight allows the selection of three contributor levels, where *level 1* is the lowest and *level 3* the highest. Setting a higher contributor level targets workers which are supposedly the most experienced among their peers. We choose to set this contributor level to the lowest possible level to allow for the largest possible crowd to participate in our tasks.

Since Figure Eight recruits their workers through other external *channels* as well, the task is also posted on other microtask crowdsourcing platforms aside from only Figure Eight. We make use of all channels that were included by default. For a complete list of the included channels, refer to Appendix B. In a similar spirit of recruiting the most possible diverse worker group, we do not limit recruitment of workers by geographical location nor by preferred language. Furthermore, we allow workers to perform each task we list on Figure Eight only once. We achieve this by setting the maximum number of judgments each worker could perform equal to the number of judgments we assign per task.

Similarly, we prevent workers from performing the same task in Telegram. In addition, we also enforce that workers are unable to perform tasks of similar type for all tasks running in Telegram. We note that this is only necessary for the chatbot tasks because all tasks run within the web-based interface are of distinct task type. To block worker participation inside Telegram, workers would be notified via a message the chatbot sends—stating that they are unable to start the task if they had previously done the same task or a task of similar type. This means that a worker would not be able to enter a chatbot task of similar type even if they were listed as two separate tasks in Figure Eight. However, we do allow workers who participated in a task that occurred in the web interface to join a chatbot task. Consequently, to minimize the possibility that workers who completed web tasks would end up performing chatbot tasks of similar type, we launch the chatbot tasks



(a) Button-Only Custom Keyboard.

(b) Mixed Custom Keyboard.

Figure 5.8: Button-Only and Mixed Custom Keyboards.

with a time gap after the web tasks had completed (see in Table 5.8). All these considerations are done in order to maximize worker diversity and reduce the odds for spurious results.

5.1.5. Task Design

Based on the categorization of tasks in Table 5.1 and the UI elements in Table 5.2, we consider a total of six different task types in our experiments:

1. **Information Finding**: This task is analogous to the aforementioned *Data Collection* task. We use the term *Information Finding* based on the taxonomy of Gadiraju et al. in [19]. In this task, workers are given a data source and are tasked to find specific relevant information. To simulate this process, we choose to present for each judgment a single text-based data record containing a set of business-related attributes, such as the *name* and *address*.

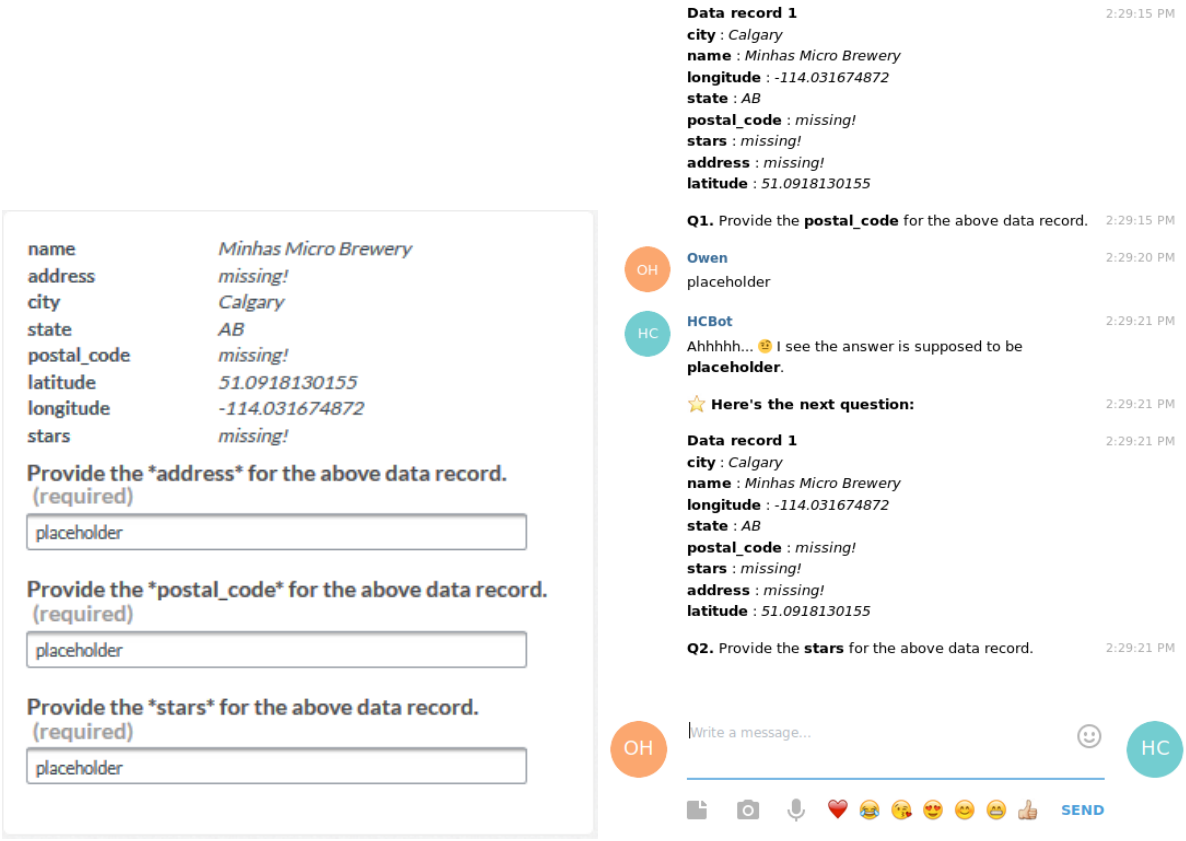
   Workers are then tasked to find missing attribute values from the data record, by scouring for information available on the web.

   For the data records, we use 17 business data records listed in the `Yelp`[3] dataset. The Yelp dataset contains a large set of data records on North American businesses, which also includes *Yelp user data*, *user reviews*, and a large number of *photos*. For this task, we only require and use the data records on the businesses. From the 17 records, we create 50 task objects by randomly removing a combination of three of the following fields per record: *name*, *address*, *city*, *state*, *postal code* and *stars* (i.e. the business rating). We then set the number of judgments to be equal to the number of data records, where all three questions correspond to one of the three removed data fields for each data record. Each worker is only assigned a single judgment to keep the task simple and short.

   To ensure that a worker is able to find information for a business, we ensure the identifiability of that business by always providing at least the *name* or *postal code*. These two attribute values are therefore never jointly removed from the same business record. In the instructions of this task, workers are requested to use any commercial search engine to find the three missing attribute values that have been

[3]Yelp dataset: `https://www.yelp.com/dataset`

randomly removed from the business record. Workers then have to provide their answers as free text in three separate input forms. For the web, this meant three separate *text boxes*, for the chatbot three separate *Messages* (see Figure 5.9).



(a) Web. (b) Chatbot.

Figure 5.9: Web and Chatbot Information Finding Task.

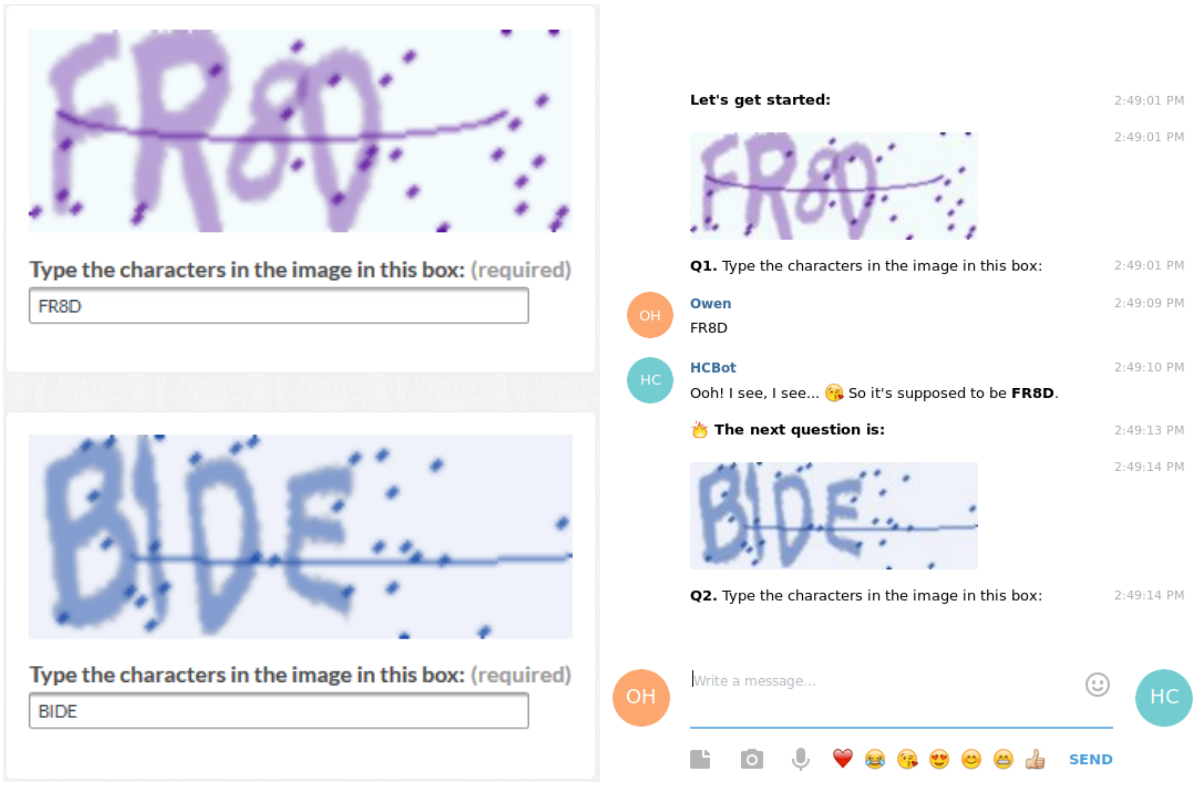
2. **Human OCR (CAPTCHA[4])**: The human OCR task is categorized as a media transcription task [19]. In this task, workers are presented with a CAPTCHA image and must transcribe the text the image contains. We use a CAPTCHA generator[5] to create 50 distinct images to use them as CAPTCHAs. Each CAPTCHA contains a sequence of four characters; which could be any digit or capitalized letter. We choose a length of four characters long to avoid over-complicating the CAPTCHAs. In a similar spirit, we exclude special characters and symbols such as punctuation marks, currency symbols, etc.

   Each CAPTCHA contains a random coloured image background with artificially generated *noise*. This noise arbitrarily places *dots* and *horizontal lines* inside the image. All character sequences possess a distinct colour shade compared to their background to ensure their visibility.

   In the web interface, workers are being requested to perform five judgments (i.e solve five CAPTCHAs). Each judgment shows an image containing a character sequence followed with a *text box* below the image to type the answer in. In the chatbot, workers are again required to answer through a *Message* (Figure 5.10). In the instructions of the task, workers are requested to mind their capitalization of their answers.

3. **Speech Transcription**: In this audio transcription task, workers are asked to transcribe recordings of English speech retrieved from Tatoeba[6]. Aside from English speech, this collection also contains many short translated phrases from different languages. We select 50 distinct recordings with each a play

[4] Originally coined by von Ahn et al. in 2000 as *Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA)* in `http://www.captcha.net/` and later further explained by the same authors in [67].
[5] CAPTCHA generator: `https://pypi.org/project/captcha/`
[6] Tatoeba dataset: `https://tatoeba.org/eng/audio/index`



(a) Web.

(b) Chatbot.

Figure 5.10: Web and Chatbot human OCR task.

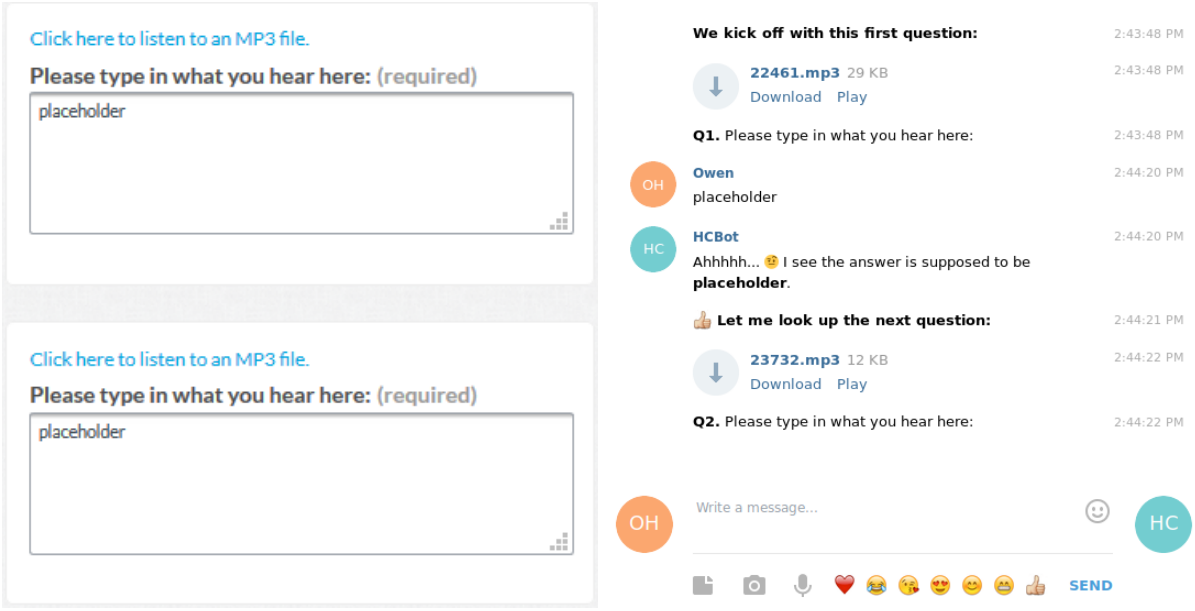
length ranging from 2 to 8 seconds. Each worker is required to perform three judgments, which corresponds to listening to three audio recordings and accordingly transcribing them.

In the web interface, to listen to a single audio recording, workers need to click a link that opened an external web-page which then automatically played that recording. Afterwards, workers were requested to type in what they heard in the recording in a *text area*. In the chatbot, workers are shown a message (by the chatbot) that contained the audio recording. Workers are then able to directly play the audio recording by clicking a *Play button* inside that message. To type in the transcribed audio fragment, workers must send a message to the chatbot with their answer typed out as shown in Figure 5.11. In both interfaces, workers would be free to play the recordings multiple times.

4. **Sentiment Analysis**: In this task, we ask workers to assess the sentiment of user reviews on businesses from the `Yelp` dataset. We select for this task a similar amount of data objects to the aforementioned tasks; 50 reviews. Consequently, we select the reviews from the `Yelp` dataset. To maintain sufficient diversity on the selected businesses, we select a maximum of three reviews per business. The length of the selected reviews varies, ranging from several sentences to whole paragraphs. Workers are asked to judge the *overall sentiment* of a review by picking from the following options; *Positive*, *Negative*, or *Neutral*. We also include an additional *Unsure* option, to indicate that they are uncertain what the sentiment would be.

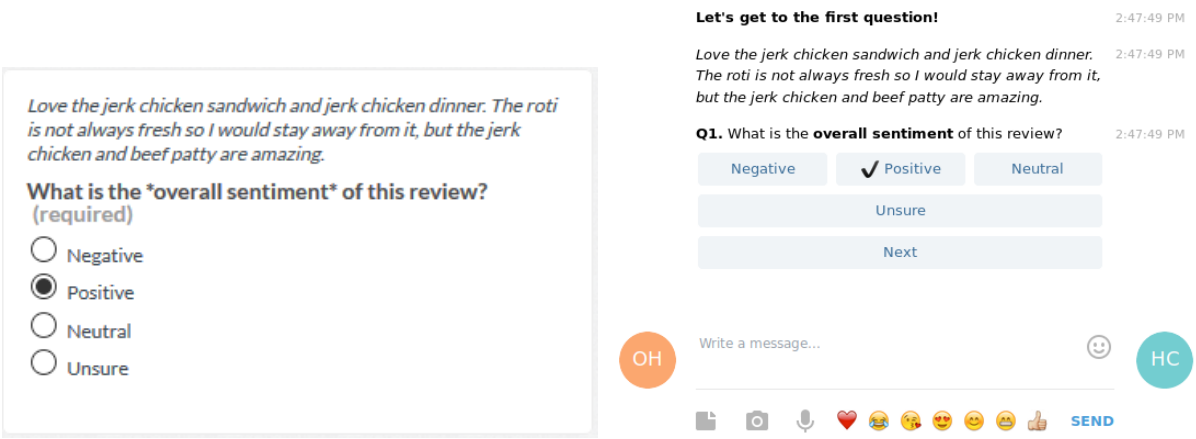
   As shown in Figure 5.12, we have that in the web interface, workers are able to pick from the options through radio buttons. In the chatbot, workers are able to pick a single answer (enforced through answer validation) through buttons.

5. **Image Annotation**: This data enhancement task [19] has the goal of determining the categories of the food items contained in an image. The worker would be able to choose from a set of pre-defined options: *Eggs*, *Fish*, *Meat*, *Vegetables*, *Fruits*, *Cheese*, *Mushroom*, *Grain*, and *Sweets*. In case the image did not contain any food category that was applicable, workers are requested to only select a *Non-food* option. Since the `Yelp` dataset also contained images, we use 50 distinct images, with a 50–50 split of 25 images containing food and 25 non-food objects.



(a) Web. (b) Chatbot.

Figure 5.11: Web and Chatbot Speech Transcription Task.



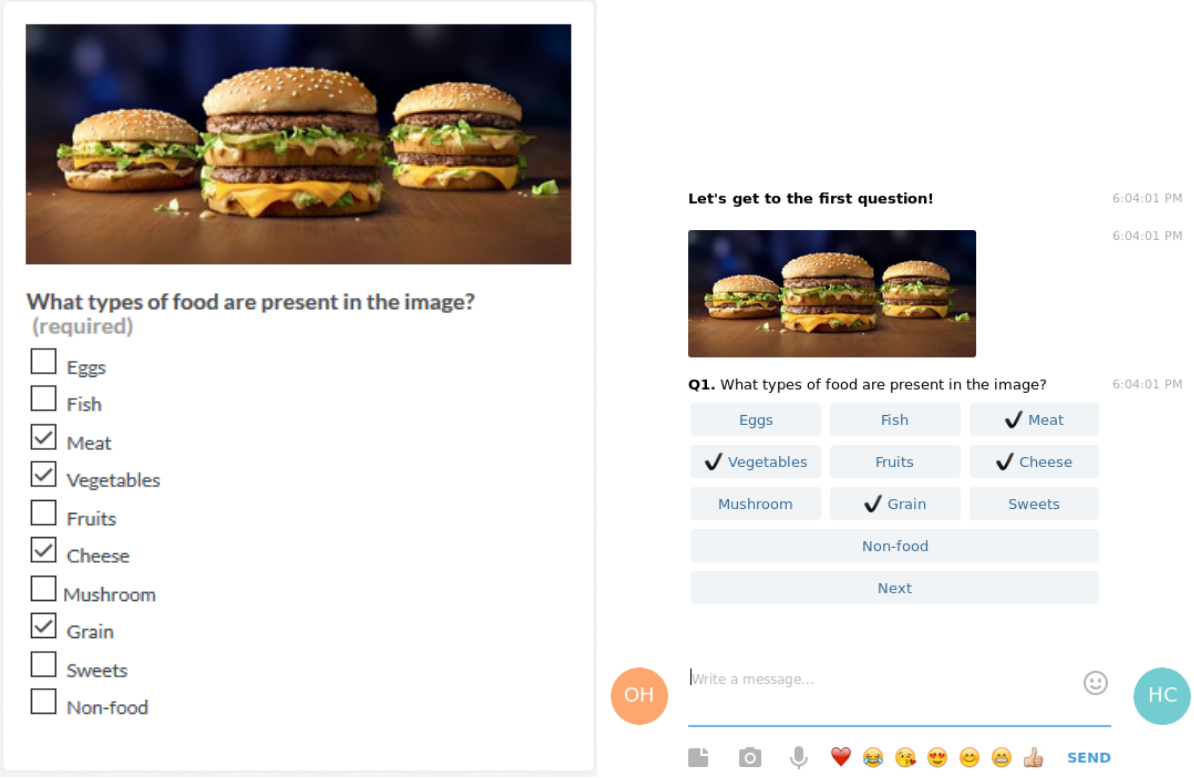
(a) Web. (b) Chatbot.

Figure 5.12: Web and Chatbot Sentiment Analysis Task.

In the web interface, we use checkboxes to allow multiple selection as shown in Figure 5.13. While in the chatbot, similar to the *Sentiment Analysis* task, we allow multiple buttons to be selected (through input validation).

6. **Object Labelling**: Similar to the *Image Annotation* task, the *Object Labelling* task is also categorized as a data enhancement task. We ask workers in this task to locate objects in an image. From a user interaction perspective, this could be seen as the most complex task among the six we consider. We select 50 arbitrary images from the `Yelp` dataset that depicted either *food*- or *non-food*-related objects, similar to the *Image Annotation* task. We maintain the same split of 25 images showing *food* and 25 *non-food*. Workers are first instructed to determine if an image contains any *food* related object.

   In the web interface, when any image contained any *food* related object, workers are tasked to draw for each *food*-related object a separate rectangular bounding box as tight as possible around the object. If no *food*-related object could be seen in the image, workers were instructed to use the *Nothing to Box* button in the provided tool. In the chatbot, workers are prompted to start a Telegram HTML game, which makes them having to select all the tiles from a grid in which the *food*-related object was located



(a) Web.

(b) Chatbot.

Figure 5.13: Web and Chatbot Image Annotation Task.

in (see Figure 5.14). In the case no *food*-related objects are present in the image, the workers may click the *Nothing to Select* button (shown in yellow).

5.1.6. Measurements & Metrics

In our experiments, we have the following three main dependent variables: *Execution Time*, *Answer Quality*, and *Workers' Satisfaction*.

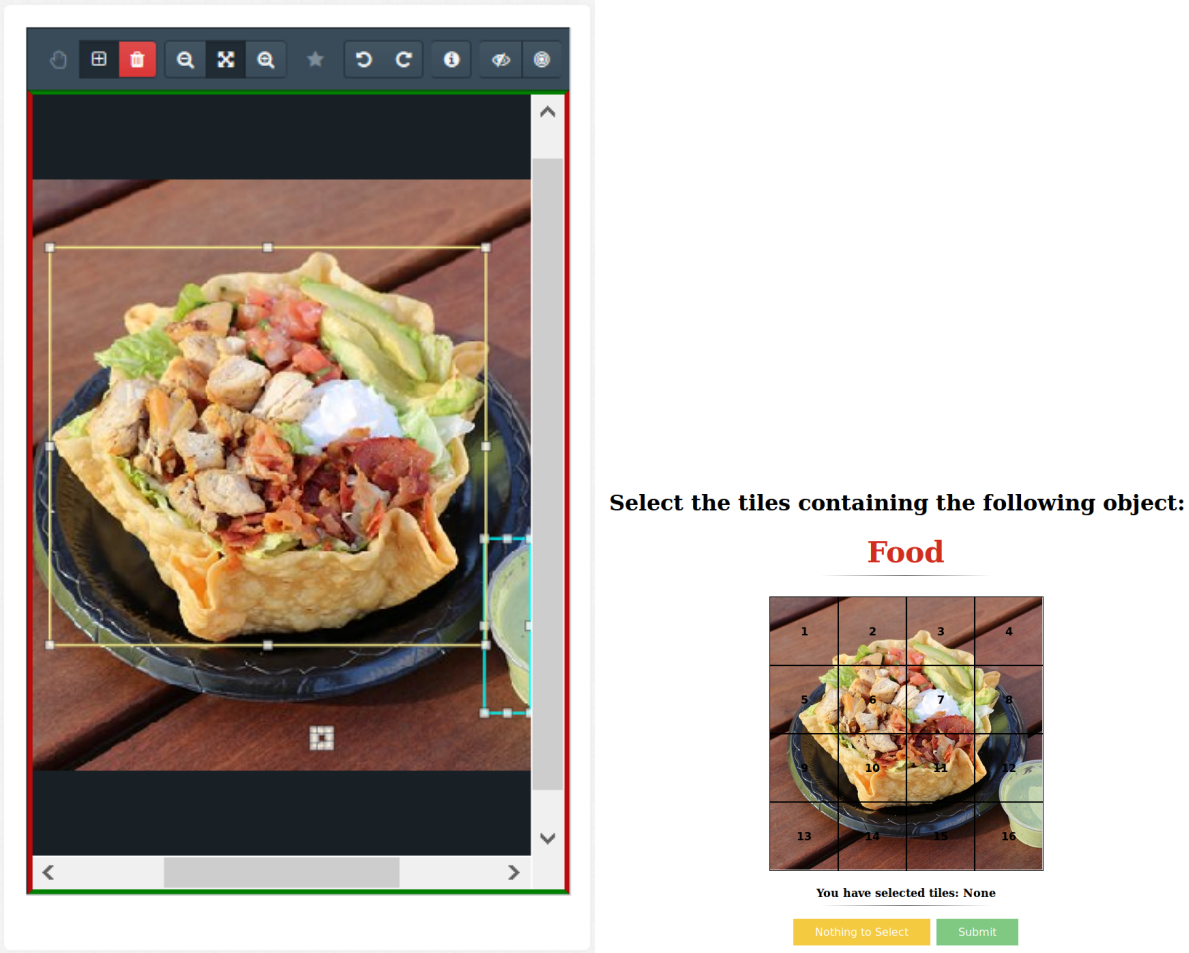
1. **Execution Time**: We measure the task execution time as the time (in seconds) between the start and the submission of a task from a single worker.

   In the web interface, the start of the task is analogous to the moment when the worker has selected to perform a task and is shown the worker interface. The moment the worker presses the *Submit & Continue* button (see Figure 5.4), the task ends. The time between these two moments will be used as the execution time for the tasks run in the web interface.

   In the chatbot interface, the start of the task is determined when the worker presses the *Start* button (see Figure 5.6). The end of the task is marked by the moment the worker presses the *Submit* button (see Figure 5.5). We then compute the execution time by taking the difference between these two moments in time.

2. **Answer Quality**: To measure the quality of worker answers, we compare them to a ground truth. Since each task will deal with different type of answers—which in some cases may be strictly equal to the ground truth—we outline what approach we take for comparing answers to ground truth for all the considered task types:

   Since we use the `Yelp` dataset, we do not possess ground truth data for the *Sentiment Analysis*, *Image Annotation*, and *Object Labelling* tasks. For the *Sentiment Analysis* task—due to the subjective nature of the task—we let all judgments be annotated by a minimum of three people using a majority vote. In the case of no majority, we request the opinion of a fourth person. Each answer is then directly compared to the ground truth label. For the *Image Annotation* and *Object Labelling* tasks, we similarly annotate



(a) Web.

(b) Chatbot.

Figure 5.14: Web and Chatbot Object Labelling Task.

the judgments and manually inspect if workers have answered correctly. For the *Image Annotation* task, we mark an answer as correct, as long as it contained at least one correct annotation and no more than two wrong annotations. Similarly, for *Object Labelling* in the web interface, answers are deemed correct, when the boxes are drawn within a marginal offset of our annotated ground truth. Conversely, for the chatbot, all answers are accepted when at least half of the selected tiles were also selected in the annotated ground truth.

For the *Information Finding* and *Speech Transcription* task, we have ground truth data by the task design. Since the worker's answers are provided in *Free Text*, we manually inspect the answers as well. We tolerate simple spelling and grammatical errors. However, we assume an answer to be incorrect, whenever it deviates too greatly from the ground truth.

For the *human OCR* task, since we generate the CAPTCHAs, we also have ground truth data. As a result, we compare the entire answer to the label of that CAPTCHA. While we request workers to mind the capitalization of their answers, we opt to disregard any errors made of this nature. This is done in consideration that this is not a large offence. We are primarily interested in excluding only truly malicious users, such as *spammers* who try to cheat by copying the same answer to multiple questions or answer randomly.

3. **Workers' Satisfaction**: To determine the workers' satisfaction, we survey the overall experience of workers who participated in the chatbot tasks. We note that by the implementation of the experiments, it would be difficult to survey the worker satisfaction in similar fashion for the web tasks. This is because adding additional survey questions inside each task running in the web interface (i.e. in Figure Eight) indirectly affects the measurement of the *Execution Time*.

The workers' satisfaction is measured as an aggregate of two survey questions that were asked through the web interface: 1) We ask if the worker "*would be interested in doing similar tasks again in the Telegram chatbot*" and 2) An optional question with a text area that allowed workers to leave any comments on their overall experience with the chatbot.

Aside from the three key dependent variables, we survey several items for the chatbot tasks through the web interface:

1. *Operating System (OS)*: We ask workers to indicate what operating system they are viewing the task on. We provide a list of options to select from, but also include an *Other* option to freely specify an operating system if it is not among the provided options.

2. *Age*: We provide several age ranges as options and ask workers to select which applies to them.

3. *Gender*: We politely inquire the worker what gender they would identify themselves most with. An option to not comment on their gender was also included.

In addition, we perform six additional measurements through the chatbot on the following statistics:

1. *Times Answer Edited*: Count of the number of times the chatbot was instructed to allow editing of a previously given answer.

2. *Times Instructions Asked*: Count of the number of times the chatbot was instructed to show the overview of the task with options to view an example, steps and instructions on editing answers.

3. *Times Example Viewed*: Count of the number of times the worker had chosen to view an example of a judgment.

4. *Times Steps Viewed*: Count of the number of times the worker had chosen to view a list of steps to solve a judgment.

5. *Times Instruction Edit Answer Viewed*: Count of the number of times the worker had chosen to view the explanation on how to instruct the chatbot to edit answers.

6. *Times Answers Reviewed*: Count of the number of times the worker had chosen to review their answers during and at the end of the task.

5.1.7. Task Execution Flow

Since we select Figure Eight as our microtask crowdsourcing market, tasks are listed one by one for workers showing their title. We distinguish between two types of tasks: *Web* and *Chatbot Tasks*. The *Chatbot Tasks* include in their task title the string `*|*Requires Telegram*|*` to suggest the presence of a technical requirement for their execution.

All the *Web Tasks* are fully executed within the Figure Eight platform, with the default Figure Eight workflow and task assignment policy. After a *Web Task* is published, workers who qualify for participation—i.e. at or above contributor level 1—may find the task in a list among other (unrelated) tasks. When a worker clicks a task, the worker is directly sent to the worker interface and immediately starts the task.

The *Chatbot Tasks* are split into two sections: one part in Figure Eight and one part within the chatbot in Telegram. The *Chatbot Tasks* are similarly published as *Web Tasks*. Workers could find the task listing in a similar fashion as the *Web Tasks*. When the worker selects the task, they start performing the Figure Eight part of the task. Workers are informed that in order to (participate and) finish the task, they need to be logged into Telegram with an account.

Within the body of the task instructions, we provide a link to an external web-page to show additional instructions on how to register a Telegram account (see Appendix E for the contents of this web-page). In addition, we provide instructions on how to *start* the task in Telegram. We also include several preview images to inform the worker about the nature of the task.

Workers are also explicitly informed that no personal information (e.g. name or phone numbers) would be stored. We also provide instructions to workers if they wished to withdraw from the experiment and that they would be allowed to do so at any time.

Below the task instructions, workers are required to consent to several of our participation terms by marking a checkbox in the web interface. These terms are explained in detail on an external web-page, of which the full web-page contents are included in Appendix D.

Only, if the worker checks this checkbox, the remainder of the Figure Eight task would be shown. This remainder includes a short survey that inquires about their currently used work environment.[7]

After the short survey, we provide an URL that would redirect workers to start a conversation with our chatbot in Telegram. Depending on the worker's work environment, the worker may 1) have been redirected to a Web client version of Telegram; or, if the worker had a native Telegram client installed, 2) to the native Telegram application; or, if the worker was viewing the task on mobile, 3) to the mobile Telegram application.

The task assignment is performed by the chatbot dynamically, using a round robin policy on the remaining judgments to be collected. Following the conversational flow depicted in Figure 4.1, after the final submission of the answers, the worker would be provided with a randomly generated validation token. The chatbot instructs the worker to input this token into a text box in Figure Eight to finalize completion of the task.

Workers are then presented with two final survey questions which asked them if they were interested to perform a similar task again in Telegram. Workers would be able to choose either a *Yes* or *No* option. Lastly, workers were free to optionally comment on their working experience in a text area.

5.1.8. Web Task Implementation

All tasks implemented in Figure Eight consist of the following three main components:

1. **HTML-based Instructions**: To provide the instructions at the beginning of each task, Figure Eight accepted simple HTML code. As a result, we accordingly implemented the instructions entirely in HTML. Each web task contained three headers (listed in the web-page from top-to-bottom); an *overview*, *example* and *steps*.

   All external resources—such as images or audio files—that were required for the data items and instructions, were hosted on our own server. These resources were then accessible through a Uniform Resource Identifier (URI). For example, to display an image inside the instructions, we simply use HTML and point to the URI as the source of the resource.

2. **CML-based Data Items & Questions**: In Figure Eight, we implemented the interface for displaying judgments to workers through CML. In Figure Eight, all tasks may contain data items over which questions are asked. The set of data items is structured as a simple table, with a set of column headers. To build a displayable judgment, we must refer to the column header—enclosed by double brackets—of the data item inside an HTML element. Then we follow with a corresponding UI input element, which in addition contains the question that the worker has to answer. For example, if we have a simple data item as shown in Table 5.3:

   Table 5.3: Example data row in Figure Eight.

Business_id	Photo	Name
1	URI_one	name_1
2	URI_two	name_2

   Now, to create a simple text box question that refers to a `name` and `photo` of the business, we first embed in HTML the column headers and enclose them by double brackets. Figure Eight then performs a substitution on these double bracket *variables* with the data item values. Thereafter, we display the question along with the text box UI input below the data content. We depict this process in Figure 5.15.

   Furthermore, we included in all *Web Task* questions a *required* validator. This keyword was included in the CML of the question, which forces workers to always provide an answer to the question. In other words, no empty inputs in the UI input element were allowed.

[7] We opted to not employ any fingerprinting techniques that may detect the digital work environment of the worker. This was a carefully made consideration, which otherwise would trade-off between the privacy of the worker and the benefit of gaining some additional information. Since we do not specifically address device and application-specific information in our research questions, this is left for potential future work.

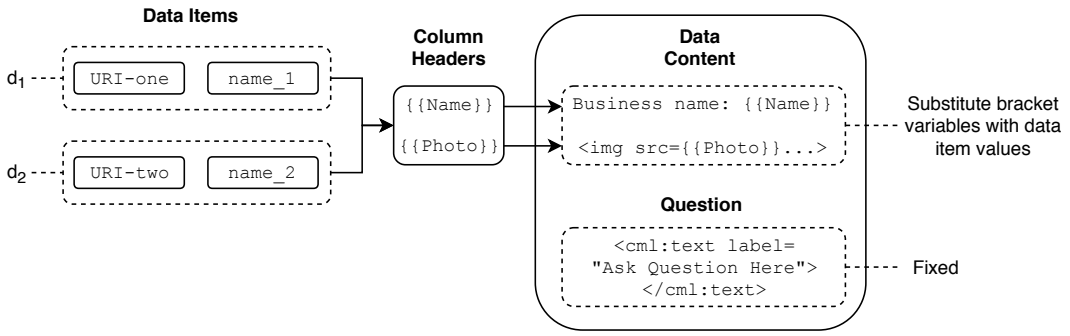


Figure 5.15: Process of creating the view of judgments in Figure Eight. In this example, we create a question that has a simple textbox input, showing the business name and an image using HTML above the question.

3. **Additional CSS and JS**: Since HTML and CML—which is a superset of HTML—was used, this also meant that Figure Eight supported custom CSS and JS to be included in the task implementation.

   For the *Web Tasks*, we only include custom JS for the *Information Finding* task. Since Figure Eight does not natively support hiding certain questions depending on the value of a data item attribute, we supplement the task with our own implemented script to hide specific questions. This was necessary to avoid asking workers the question to find attribute values that are not missing.

   For the survey in the *Chatbot Tasks*, we use some basic CSS to style the links to the consent form and to the Telegram chatbot. This was done to attract more attention to these important links.

5.1.9. Chatbot Task Implementation

The *Chatbot Tasks* run partly in Figure Eight and in Telegram. This means that we similarly implement the *Chatbot Tasks* using the same toolset, that we use to create the *Web Tasks*.

Since the actual microtask of the *Chatbot Task* takes place inside Telegram, there is also no need to have the actual data items over which we ask questions in Figure Eight. The assignment process of data items would be instead handled by the system infrastructure of the chatbot. This follows a standard round-robin policy for the assignment of judgments. Additionally, the chatbot also keeps track of which judgments are being performed and prevents repeated assignment of them if judgments are still pending.

The *Chatbot Task* title in Figure Eight is prefixed with the string `*|*Requires Telegram*|*`. The instructions of all the *Chatbot Tasks* in Figure Eight then contained the following three elements:

1. **Overview**: We announce that workers are required to have a *registered Telegram* account and provided additional instructions on an external web page in case they wished to register one which is fully shown in Appendix E. We also notify workers that any data we collected during their Telegram session would be stored anonymously and be used exclusively for academic purposes. We explicitly state that no personal identifying information, including mobile phone numbers, would be stored.

2. **Task Initiation Overview**: This short part of the instructions shows that the real microtask would take place in Telegram and includes Figure 5.6 to show how to proceed with initiating the task in Telegram.

3. **Telegram Preview**: For each task, we include several preview images of the microtask in question, that the worker would be performing in Telegram.

The part in Figure Eight contains a survey which initially prompts workers to provide consent by marking a checkbox as shown in Figure 5.16. Workers are urged to read up on the participation conditions of the task before agreeing, which are provided through an external web-page (see Appendix D for its contents).

After agreeing to the terms and conditions, the remainder of the survey is shown (Figure 5.17).

This survey supplies a link to redirect the worker to start the task in Telegram. Telegram provides support for deep linking, which allows us to directly send the worker to our chatbot.

To verify that a worker had truly completed a task in Telegram, they are prompted to input in the survey a randomly generated validation token, that they received after completing the Telegram task (see Figure 5.18).

This validation token included a part that matched a unique identifier of the task they performed in Telegram generated by our chatbot system. When a worker puts in the validation token in the survey in Figure

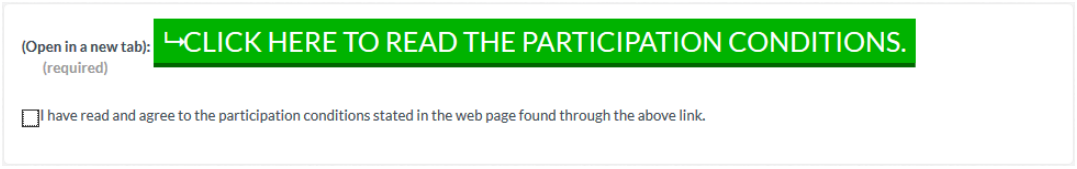


Figure 5.16: Agreement to participate in a Chatbot Task.

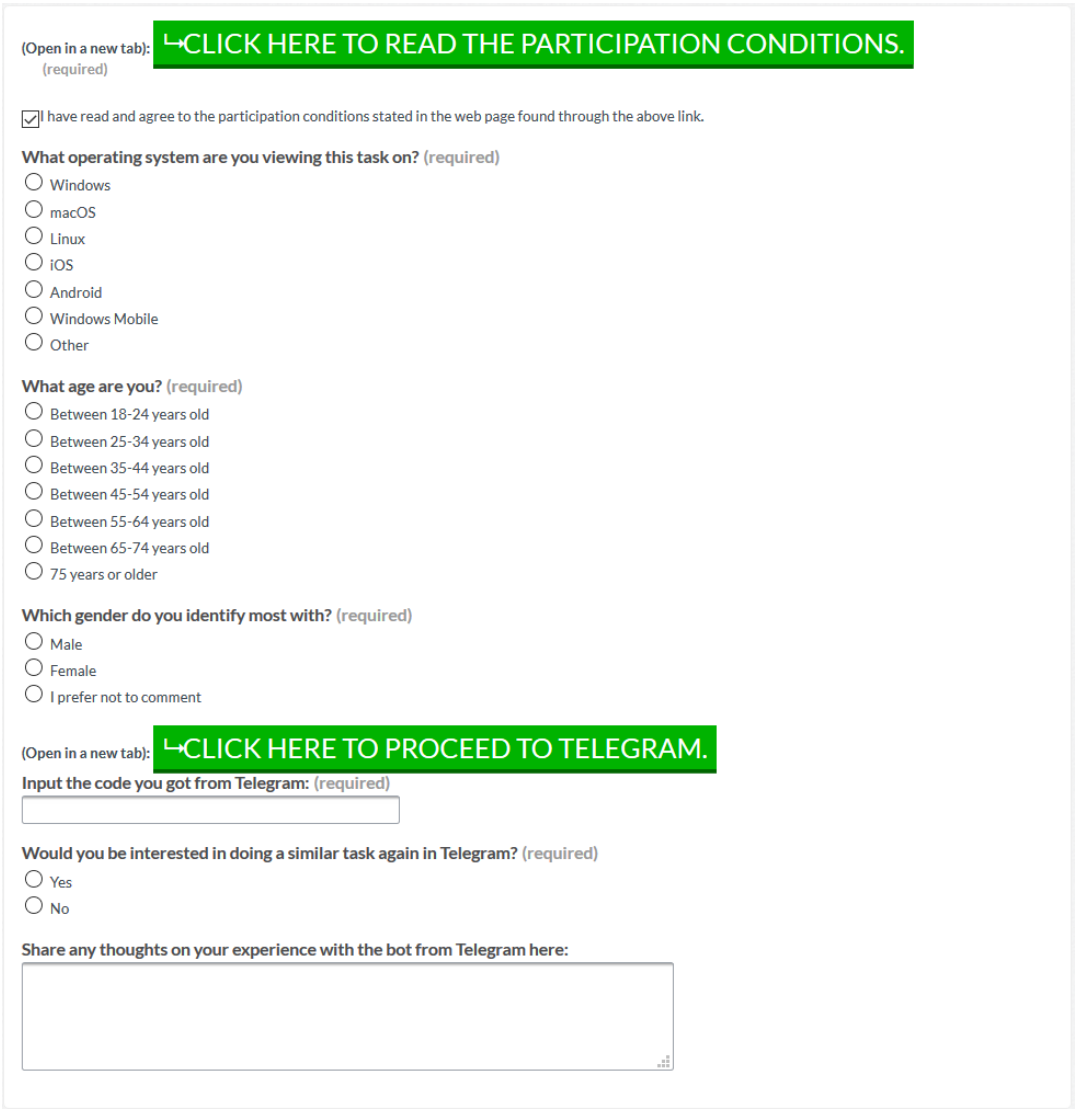


Figure 5.17: Chatbot Task Survey in Figure Eight.

Eight (using a custom validator in the CML), a simple HTTP request would be sent to our server to validate that the token had not been used before and is truly one generated by our chatbot.

Workers would not be able to complete and submit the task in Figure Eight, as long as this token would invalidate. We note that because we did not actively facilitate communication between different workers (and with task deadlines in place), we did not feel the necessity to validate on any worker identifier. This means that tokens obtained by a worker may be used (as long as they have not been used before) by other workers. However, we do validate on each separate task session of the worker. As a result, a worker may not use the validation token obtained in one task for another.

We note that for the specific case of the *Object Labelling* task, we implement a simple HTML "game" that may be accessed via a web-page. Inside Telegram, this would be viewable through an embedded web-page.

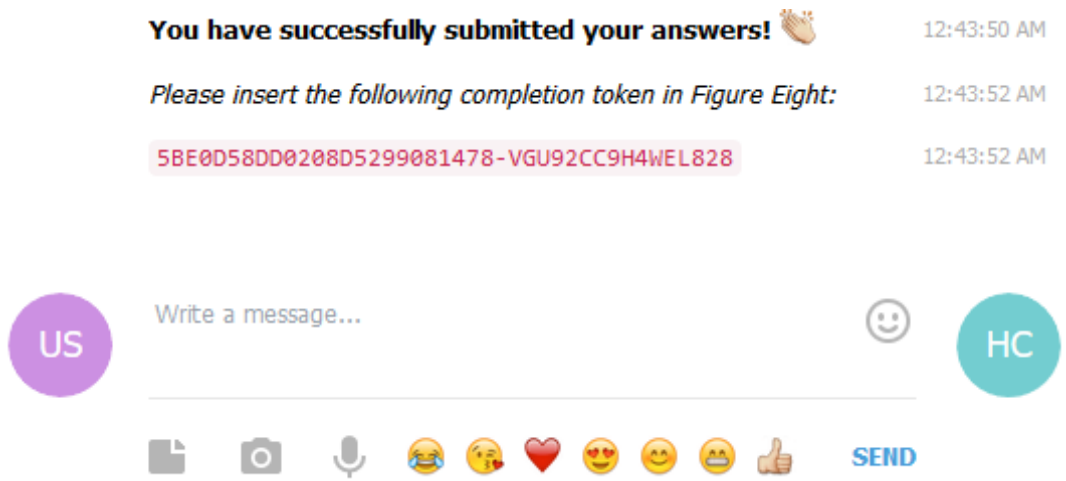


Figure 5.18: Chatbot Hands Out Validation Token After Task Completion.

This meant that workers would not have to be forced to open a web-browser manually. As a result, to generate the grid on top of the image, we implement our own simple algorithm (see Algorithm 1). We use the algorithm to generate a grid of 4x4 tile size to prevent tiles from being too small to select. This algorithm also takes into account that the aspect ratio of images may differ greatly. Consequently, we ensure that the tile *length* and *height* are always at *worst* in a $1:2$ (or $2:1$) ratio.

Algorithm 1 Grid Division Algorithm

INPUT:

- x and y are non-negative integers; where x and y are the input image *width* and *height* respectively.
- p and q are the grid tile *width* and *height* respectively, used in the current iteration for computing the eventual outputs.
- t is the maximum amount of tiles in the grid in both horizontal and vertical dimensions.
- s is the maximum tile size; p and q will not exceed s in size.

OUTPUT: p and q, which are computed *width* and *height* of the grid tiles such that the image is split equally with respect to the desired p and q in the *input*.

1: **function** GRID_DIVIDER(x, y, p, q, t, s)
2: **if** ($\frac{2x}{p} \leq t$ **and** $p > s$) **or** ($\frac{2y}{q} \leq t$ **and** $q > s$) **then**
3: **if** $p > s$ **then**
4: $p \leftarrow \frac{p}{2}$
5: **if** $q > s$ **then**
6: $q \leftarrow \frac{q}{2}$
7: $p, q \leftarrow GRID\_DIVIDER(x, y, p, q, t, s)$
8: **return** p, q

5.2. Experiment Execution

In this section, we discuss how the designed tasks with the setup from the previous section will be run. We begin by explaining the set configuration options that have been used for each task in Section 5.2.1. Thereafter, we elaborate upon our planning of the execution of the tasks in Section 5.2.2.

5.2.1. Task Settings

We summarize in Tables 5.4 to 5.6 the task settings for experiment 1, 2, and 3 respectively. Each task is designed to have 50 workers perform the work, for which we pay 15¢ per task assignment. For all tasks except *Information Finding* and *human OCR*, workers have to perform three judgments. For *Information Finding*, we request only a single judgment, because per judgment we already ask three questions. For *human OCR*, due to its very low difficulty, we choose to ask workers to perform five judgments.

All tasks did not require any entry requirement based on some pre-task quiz. Furthermore, tasks do not contain any test questions for assessing worker quality during task execution. This was done in consideration of measuring the "true" task execution time per single task assignment.

All workers were also able to perform a single task as listed in Figure Eight only once to account for potentially higher worker diversity. All 24 tasks had a time limit set of 30 minutes in total, which after initial test runs was found to be more than sufficient. After exceeding this time limit, workers will be automatically removed from their participation of the task. In such cases, workers may re-enter the task but may end up with different *Data Items* assigned to them.

Table 5.4: Task settings for experiment 1.

Task ID	Task	Independent Variables			Input Form	#Workers (#Judgments per Worker)	#Data Items (#Judgments per Data Item)
		Platform	Data Type	Input Type			
1	Information Finding	Figure Eight	Text	Free Text	Single-Line Text	50(1)	17(3)
2	Information Finding	Telegram	Text	Free Text	Message	50(1)	17(3)
3	Sentiment Analysis	Figure Eight	Text	Single-Selection	Radio Buttons	50(3)	50(3)
4	Sentiment Analysis	Telegram	Text	Single-Selection	Single Button	50(3)	50(3)
5	Human OCR	Figure Eight	Image	Free Text	Single-Line Text	50(5)	50(5)
6	Human OCR	Telegram	Image	Free Text	Message	50(5)	50(5)
7	Image Annotation	Figure Eight	Image	Multiple-Selection	Checkboxes	50(3)	50(3)
8	Image Annotation	Telegram	Image	Multiple-Selection	Multiple Buttons	50(3)	50(3)
9	Object Labelling	Figure Eight	Image	Image Segmentation	Bounding Box	50(3)	50(3)
10	Object Labelling	Telegram	Image	Image Segmentation	Select-Grid	50(3)	50(3)
11	Speech Transcription	Figure Eight	Audio	Free Text	Multi-Line Text	50(3)	50(3)
12	Speech Transcription	Telegram	Audio	Free Text	Message	50(3)	50(3)

5.2.2. Execution Schedule

To lower the odds of workers performing multiple task types (e.g. by participating in an *Information Finding* task in experiment 1 and 3), we run the experiments according to a set schedule. We first perform a test run for all experiments to resolve any potential issues that may arise, following the schedule in Table 5.7. These test runs are run with five workers per task each. Thereafter, we proceed with the execution of the full experiment as shown in Table 5.8. We note that we deliberately executed all tasks within weekdays while avoiding the weekend. This was done to maintain consistency in terms of task execution settings across all tasks. We run after the test run of the *Web Tasks*, the full run of the *Web Tasks*. The *Web Tasks* are executed separately from the *Chatbot Tasks*, with a few days in between the task execution. For the *Chatbot Tasks*, we run them separately in batches based on the experiment to which they belong. This means that the second experiment is launched after the first has completed, while the third is launched after the second has been completed. Because we expected the test runs to finish quickly, we planned to perform all testing of the *Chatbot Tasks* within a single day.

Table 5.5: Task settings for experiment 2.

Task ID	Task	Independent Variables			Input Form	#Workers (#Judgments per Worker)	#Data Items (#Judgments per Data Item)
		Platform	Data Type	Input Type			
13	Sentiment Analysis	Telegram	Text	Free Text (Text-Only)	Message	50(3)	50(3)
14	Sentiment Analysis	Telegram	Text	Free Text (Code-Only)	Message	50(3)	50(3)
15	Sentiment Analysis	Telegram	Text	Free Text (Text and Code), Single-Selection	Message, Single Button	50(3)	50(3)
16	Image Annotation	Telegram	Image	Free Text (Text-Only)	Message	50(3)	50(3)
17	Image Annotation	Telegram	Image	Free Text (Code-Only)	Message	50(3)	50(3)
18	Image Annotation	Telegram	Image	Free Text (Text and Code), Single-Selection	Message, Single Button	50(3)	50(3)

Table 5.6: Task settings for experiment 3.

Task ID	Task	Independent Variables				Input Form	#Workers (#Judgments per worker)	#Data Items (#Judgments per Data Item)
		Platform	Instructions	Data Type	Input Type			
19	Information Finding	Telegram	Hidden	Text	Free Text	Message	50(1)	17(3)
20	Sentiment Analysis	Telegram	Hidden	Text	Single-Selection	Single Button	50(3)	50(3)
21	Human OCR	Telegram	Hidden	Image	Free Text	Message	50(5)	50(5)
22	Image Annotation	Telegram	Hidden	Image	Multiple-Selection	Multiple Buttons	50(3)	50(3)
23	Object Labelling	Telegram	Hidden	Image	Image Segmentation	Select-Grid	50(3)	50(3)
24	Speech Transcription	Telegram	Hidden	Audio	Free Text	Message	50(3)	50(3)

Table 5.7: Task execution schedule of all the test runs of all experiments. We note that the numbering of batches is sorted by the launch dates. The second batch is the full run of the *Web Tasks*.

Batch ID	Launch Date	Experiment	Task ID	Platform
Batch 1 (Test)	24–09–2018	Experiment 1	1	Figure Eight
			3	Figure Eight
			5	Figure Eight
			7	Figure Eight
			9	Figure Eight
			11	Figure Eight
Batch 3 (Test)	26–09–2018	Experiment 1	2	Telegram
			4	Telegram
			6	Telegram
			8	Telegram
			10	Telegram
			12	Telegram
Batch 4 (Test)	26–09–2018	Experiment 2	13	Telegram
			14	Telegram
			15	Telegram
			16	Telegram
			17	Telegram
			18	Telegram
Batch 5 (Test)	26–09–2018	Experiment 3	19	Telegram
			20	Telegram
			21	Telegram
			22	Telegram
			23	Telegram
			24	Telegram

Table 5.8: Task execution schedule of all the full runs of all experiments.

Batch ID	Launch Date	Experiment	Task ID	Platform
Batch 2 (Full)	25–09–2018	Experiment 1	1	Figure Eight
			3	Figure Eight
			5	Figure Eight
			7	Figure Eight
			9	Figure Eight
			11	Figure Eight
Batch 3 (Full)	27–09–2018	Experiment 1	2	Telegram
			4	Telegram
			6	Telegram
			8	Telegram
			10	Telegram
			12	Telegram
Batch 4 (Full)	28–09–2018	Experiment 2	13	Telegram
			14	Telegram
			15	Telegram
			16	Telegram
			17	Telegram
			18	Telegram
Batch 5 (Full)	01–10–2018	Experiment 3	19	Telegram
			20	Telegram
			21	Telegram
			22	Telegram
			23	Telegram
			24	Telegram

6

Results and Discussion

In this chapter, we discuss the results obtained from running the three experiments, that we designed in the previous chapter. A total of 24 tasks have been run, recruiting workers from the internal network of Figure Eight and many external microtask crowdsourcing platforms. As the main objective of our work is to gain a deeper understanding of the viability of the chatbot as an alternative work interface for microtask crowdsourcing, workers were neither conditioned on their participation to pre-existing quality levels, nor subjected to any type of qualification test. We perform analysis on all collected results from the experiments. Therefore, we provide a complete reflection of the performance achieved by the Figure Eight population. We note that this potentially includes malicious individuals. We begin by discussing the differences we found between the web and chatbot interfaces in Section 6.1. Thereafter, we look into the use of different *Custom Keyboards* for single- and multiple-selection tasks in Section 6.2. Following in Section 6.3, we report on several statistics on the workers. Finally, we close with a discussion of the results in Section 6.4.

6.1. Web vs. Chatbot Work Interface

To study the difference between the work interface for the *Web* and the *Chatbot*, we have run in the first experiment a total of six tasks in both interfaces. For each task, we measured both the *Execution Time* and the *Answer Quality*.

- **Execution Time**: For the execution time, we have aggregated the execution times through averaging across all participating workers for the *Web* and *Chatbot* tasks in Table 6.3. We depict additional insights into the distribution of the execution time and the differences between web and chatbot in Figure 6.1. Since we are interested if there exists a statistically significant difference in *Execution Time* between the *Web* and *Chatbot Tasks*, we perform a Mann-Whitney-Wilcoxon pair-wise significance test between the *Web* and *Chatbot Tasks*. We note that the distributions of the execution time of all task types show exponential relationships. Moreover, by manual inspection into the outliers, we observe that these are not caused by *spammers*. We rather notice that the tasks are performed earnestly by the workers, without any attempt in tricking the system. However, we observe that these workers had accepted multiple tasks within a short range of each other, but completed them much later on. We speculate that this may be the result of workers trying to *claim* multiple tasks at once to guarantee their spot for participation—as the maximum number of participants for each of our listed tasks was only 50. We compute the two-sided p-values as shown in Table 6.1:

  From the p-values, we observe that for $\alpha = 0.05$ the execution time distributions for the four task types—*Information Finding*, *Sentiment Analysis*, *Object Labelling*, and *Speech Transcription*—both chatbot with- and without initially showing instructions have no statistically significant difference ($p > 0.05$). However, we note that there is still a considerable difference present for *Sentiment Analysis* when comparing to the *Information Finding*, *Object Labelling* and *Speech Transcription* tasks.

  A statistically significant difference is present for $\alpha = 0.05$ for the *Human OCR* task *with instructions* with a $p = 0.0010$. In addition, the *Image Annotation* task *without instructions* shows a statistically significant difference of $p = 0.029$.

Table 6.1: p-values of Mann-Whitney-Wilcoxon test on the *Web* and *Chatbot Tasks*. The *Between Web* column shows the pair-wise test between *Chatbot Tasks* and the *Web Tasks*. The *Between Chatbot* column is the pair-wise test between *Chatbot Tasks* with- and without instructions.

Task Type	**Between Web**		**Between Chatbot**
	With Instructions	**Without Instructions**	
Information Finding	0.4819	0.5122	0.2224
Human OCR	**0.0010**	0.9377	**0.0010**
Sentiment Analysis	0.1766	0.1350	0.9560
Object Labelling	0.7854	0.7290	0.8632
Image Annotation	0.8066	**0.0292**	**0.0274**
Speech Transcription	0.4380	0.3920	0.7801

We note that the *Speech Transcription* tasks show a slightly longer execution time in the *Web* than in the *Chatbot* as seen in Table 6.3. We attribute this result to the UI design of the *Web Task*. In the *Web Interface*, workers were forced to open a separate web-browser tab in order to play the audio fragment. In addition, workers would naturally have to navigate back to the web-browser tab to input the answer. We speculate that these inconveniences might have caused delays in the process of completing the task.

Furthermore, the *Object Labelling* tasks were completed on average faster in the *Chatbot Interface* than in the *Web Interface*. This might be possibly due to the intuitiveness and lower complexity in using the *Image Segmentation* control tool in the chatbot (*tile selection* vs. *drawing bounding boxes* in the *Web Interface*).

The difference in execution time with the *Sentiment Analysis* tasks and the statistically significant difference ($p = 0.0010$) with the *Human OCR* task (with instructions) might be explained by the presence of long textual instructions at the beginning of the chatbot interface. Contrary to the *Web Interface*, these instructions could not be naturally hidden. We note that this hypothesis is also supported by the results in Table 6.3. We observe that by configuring the chatbot to hide instructions from the worker's initial view, the task execution time is in general lower.

For the *Image Annotation* task, we note that hiding the instructions show a statistically significant difference between the *Web Task* and with initially showing the instructions. We speculate that this might be caused by workers feeling the need to consult the task instructions. While we provide instructions on how to do so in the chatbot, we remark that workers may not be very familiar with the use of requesting the instructions through message interaction. Consequently, this might have caused delays in completing the task.

- **Answer Quality**. To measure the quality of answers provided by the workers, we compute the *precision* as shown in Equation (6.1) for all task types in both the *Web Tasks* and *Chatbot Tasks*:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{6.1}$$

We summarize in Table 6.2 the work performance evaluation expressed in the $Precision$ for the six considered task types. In general, we observe that all the task types show comparable performance. Furthermore, we note that the precision between tasks for the *Chatbot Interface* is on average slightly lower than in the *Web Interface*.

We note that by analysis of the results, we found an equal distribution of malicious and badly performing workers across the tasks. Users that were branded malicious—when copying their answers to multiple questions or simply answering randomly—are excluded from the analysis in answer quality. To explain why the rather trivial *Human OCR* task merely resulted in a precision from 0.70 to 0.80, we inspected the results manually. We observed that most errors were made due to the presence of ambiguous characters in our generated CAPTCHAs (e.g. a "D" character resembled both a capital "O" or a

Table 6.2: Workers output precision across tasks and platforms.

Task Type	**Web**	**Chatbot**	
		With Instructions	**Without Instructions**
Information Finding	0.86	0.88	0.82
Human OCR	0.72	0.80	0.76
Speech Transcription	0.71	0.69	0.68
Sentiment Analysis	0.89	0.83	0.81
Image Annotation	0.85	0.79	0.82
Object Labelling	0.77	0.80	0.77

"0" (zero), while a rotated "L" looked very similar to a "V"). Moreover, the *Object Labelling* task resulted in slightly higher precision in the *Chatbot Task* compared to the *Web Task*. We believe that the intuitiveness of the grid selection tool, and simplified task actions and controls required in the chatbot may be the cause of this result. We also note that in general the answer quality seems to be slightly lower for tasks *without* initially shown instructions compared to the tasks that do show them at the start. We attribute the slightly lower answer quality in the particular case of the *Information Finding* task (without showing instructions) due to the fact that several workers seemed to fail to understand what the distinction between some *attributes* are, such as *state* and *city*.

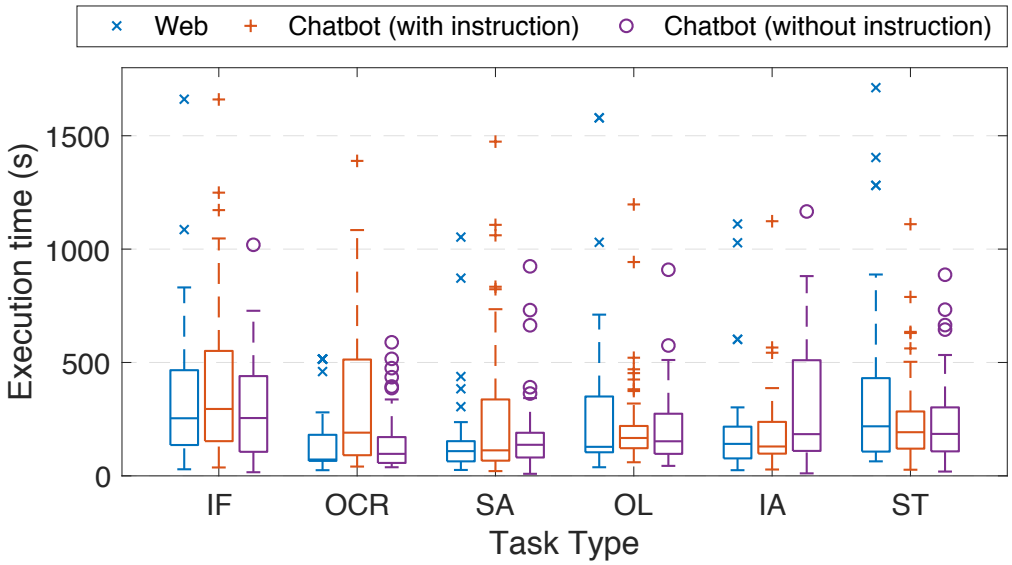


Figure 6.1: Task execution time (in seconds) for all six task types: Web vs. Chatbot with instructions vs. Chatbot without instructions.

6.2. Influence of UI Elements in the Chatbot Interface

Similar to the previous approach for studying the execution time, we list the various execution times for the four *Custom Keyboards* in Table 6.4. Furthermore, in Figure 6.2 we depict other basic statistics alongside the distribution of execution times for both *Sentiment Analysis* (single-selection) and *Image Annotation* (multiple-selection) tasks.

Similar to the previous approach, we observe that the distributions of execution time are exponential. Therefore, we again perform a Mann-Whitney-Wilcoxon pair-wise significance test between the *Web* and *Chatbot Tasks*. From the results in Table 6.4 and Table 6.5, we observe that the use of different custom keyboards leaves an impact on the task execution times. For both the single- and multiple-selection tasks, we have a statistically significant difference of $p < 0.01$ for $\alpha = 0.05$ with the text configuration (both *Sentiment Analysis* and *Image Annotation* tasks). Furthermore, for the code configuration in the *Sentiment Analysis* task,

Table 6.3: Execution time ($\mu \pm \sigma$: average and standard deviation, unit: seconds) in each Work Interface.

Task Type	**Web**	**Chatbot**	
		With Instructions	**Without Instructions**
Information Finding	339 ± 299	395 ± 346	299 ± 227
Human OCR	149 ± 144	339 ± 333	151 ± 141
Speech Transcription	346 ± 365	253 ± 204	248 ± 194
Sentiment Analysis	157 ± 188	272 ± 325	181 ± 175
Image Annotation	198 ± 215	192 ± 180	313 ± 265
Object Labelling	293 ± 345	223 ± 208	205 ± 157

Table 6.4: Execution time ($\mu \pm \sigma$: average and standard deviation, unit: seconds) in each chatbot interface.

Task Type	**Mixed**	**Button-only**	**Text-only**	**Code-only**
Sentiment Analysis	168 ± 157	272 ± 325	225 ± 160	257 ± 187
Image Annotation	185 ± 163	192 ± 180	332 ± 313	252 ± 276

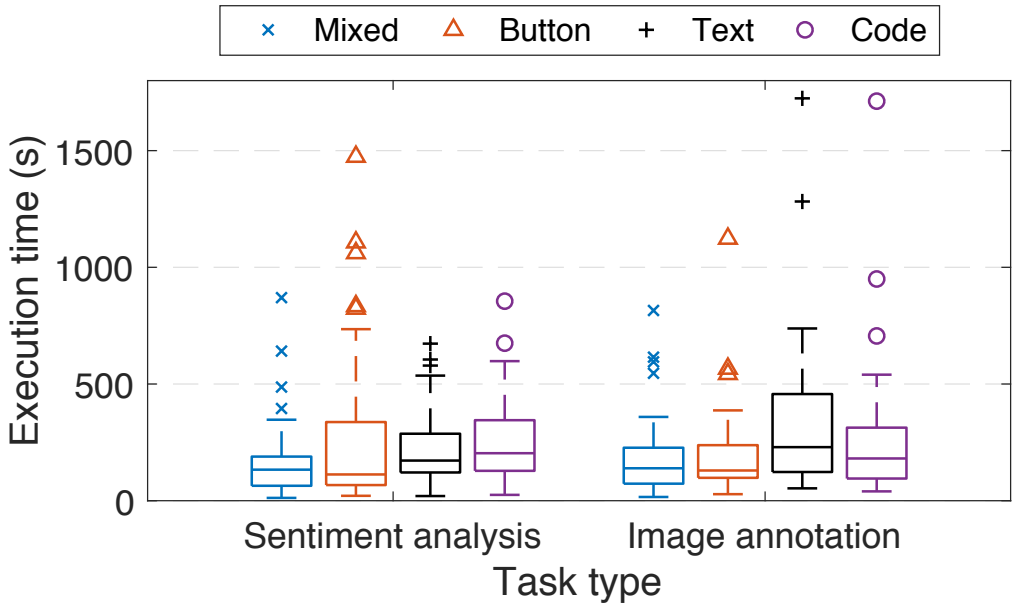


Figure 6.2: Task execution time for all Custom Keyboards; *mixed*, *button-only*, *text-only*, and *code-only*.

we have a similar significant difference of $p < 0.01$. Through observation in Table 6.4, we see that the availability of multiple input alternatives (*Mixed Custom Keyboard*) yields faster execution times for both single- and multiple-selection.

We note that while there is a clear best option, there is no clear total order of performance across the two tasks. The use of *button-based* interaction pulls ahead in terms of execution time in favour of the *text-based* approach. In addition, we note that both the *button-based* and *text-based* interaction may be applied without worry for a decrease in answer quality. This is because both approaches utilise a similar input validation mechanism, preventing from answers to be accepted that do not fall within the set of given selectable options.

We also recognize that in the case of the *Mixed Custom Keyboard* for both single- and multiple-selection, workers almost exclusively used the buttons as the input medium. We believe that the use of buttons communicates a quicker understanding of what is expected from the worker. We attribute this result to the intuitiveness of interactive buttons.

Table 6.5: p-values of Mann-Whitney-Wilcoxon test on the single- and multi-selection tasks for the *Web* and *Chatbot Tasks* using *Custom Keyboards*.

Task Type	**Between Web**			
	Mixed	**Button-only**	**Text-only**	**Code-only**
Sentiment Analysis	0.3098	0.1766	**0.0011**	**0.0003**
Image Annotation	0.9188	0.8066	**0.0036**	0.1842

6.3. General Statistics & Worker Demographics

In total across all three experiments, we have deployed 24 separate tasks (as *jobs*) in Figure Eight. Each task contained 50 task instances that have been assigned to 50 workers. This resulted in a combined total of 1200 task executions counting all experiments. Each worker was compensated 0.15¢ per task instance, resulting in a cost of approximately $180 U.S. dollars, accounting only for the payment for the tasks themselves (and discounting any additional fees).

Across all tasks, a total of 316 distinct workers have participated. These 316 workers completed at least one task, with an average of $\mu = 3.886$, a standard deviation of $\sigma = 2.4941$, and $median = 2$. Among these workers, 31 workers have performed both *Web* and *Chatbot Tasks*. From the 316 workers, a total of 167 workers participated in one of the experiments, while 93 workers were part of two experiments, and finally, 56 workers took part in all three experiments.

We report that within the use of our supplied navigational commands inside the chatbot, only 10 workers inquired to (re)view the task instructions. However, workers viewed the steps of task 38 times altogether. While counting across all tasks, the example of a task was viewed 112 times. Moreover, we noted 17 occasions where instructions on how to edit answers had been explicitly requested. Combined with the 46 cases where answers were reviewed, a total of 21 times answers had been edited.

Furthermore, we surveyed that a total of 12.2% of workers who performed *Chatbot* tasks had indicated that they performed the task on a mobile device as shown in Figure 6.3. As seen in Figure 6.3, the distribution of *OS* used shows a large bias towards the use of any version of the *Windows* OS. Surprisingly, only a very small number of workers use other popular OS, such as MacOS or any Linux distribution. We note that workers were also free to supplement any other OS that was not among the given list of options (through the *Other* option). However, no other OS was collected that deviated from the initially given list of options.

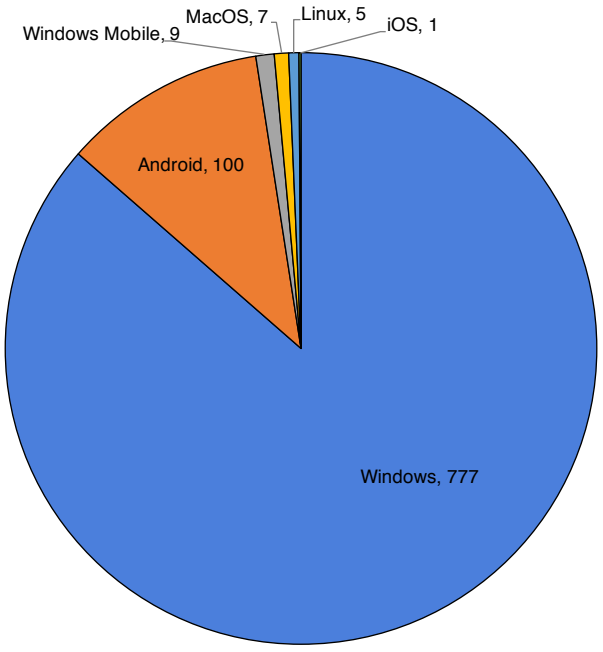


Figure 6.3: Operating System distribution among all participants who participated in the chatbot tasks.

We also surveyed workers to indicate within what age-range they fell. We depict the results in Figure 6.4. While we cannot determine if workers had answered truthfully, we observe that a clear majority is found in an age above 25 years old. Interestingly, a small group of 91 (approximately 10%) workers had indicated to be of 45 or above years old. Furthermore, the second largest (29%) group of workers was found to be in their early stages of adulthood (18-24).

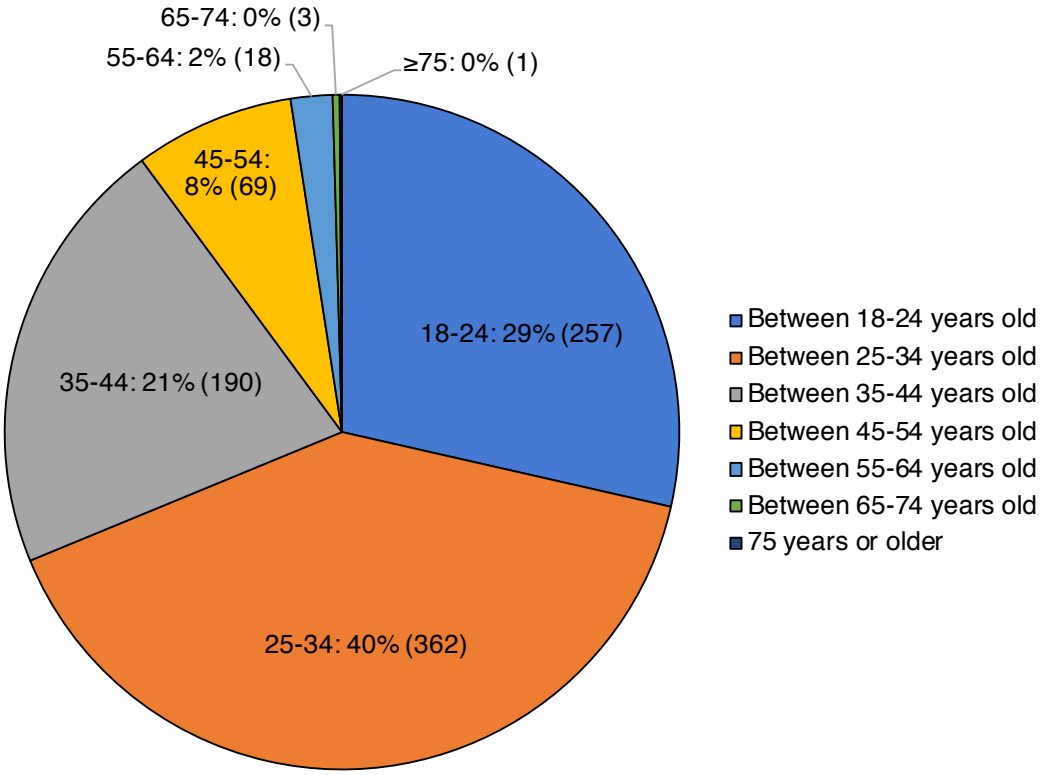


Figure 6.4: Age-range distribution among all participants who participated in the chatbot tasks.

The last item on our survey, related to the worker demographics, inquired to which gender they would identify themselves most with. By aggregating the results from all experiments, we found that 682 workers indicated that they consider themselves to be male. While 205 workers indicated to be female. The remaining 13 workers preferred not to comment on their gender.

Lastly, Figure Eight collects automatically for all the participants in our tasks their residing country. We note that these statistics are determined by the IP-address of each worker, meaning that the determined result may not always be accurate. For example, a worker could use online Virtual Private Network (VPN) services that may affect their IP-address. We show the collected results in Figure 6.5. Clearly, the majority of workers (53%) had taken up residence in Venezuela (VEN). The second largest group (8%) of workers originate from Egypt (EGY). The remaining 39% of the workers included a wide variety of countries; such as Turkey (TUR), Ukraine (UKR), Philippines (PHL), Mexico (MEX), Nigeria (NGA) and Algeria (DZA).

The last two items on the survey related to the *workers' satisfaction*. We required workers to indicate if they were willing to perform *similar* tasks again that they had performed within Telegram. Out of all 900 task executions in the chatbot, an overwhelming 98.3% of workers indicated a positive experience with the chatbot and stated to be willing to perform similar tasks again in Telegram. Moreover, we also allowed workers to optionally leave comments on their experience with the chatbot. We report that a total of 115 workers left comments, which in general ranged from a single sentence to several. By manual inspection of the comments, we found much praise for the intuitive user experience (e.g. *"Very easy to understand, and easy and fastest now we have buttons"*, *"very pleasant experience, i like the replays from the BOT, very interactive! Thx!"*, *"i loved this task, is so much different to the others, and i think is a excellent work it with telegram. nice"*, *"It was different, but i like it.."*, *"Yeah, i like this type of Task, is cool, a new feature is coming to us"*). Other workers remarked the enjoyable experience (*"This is fun and easy task I may try another task like this! Great!"*, *"Its fun!! best experience for first time using telegram haha"*).

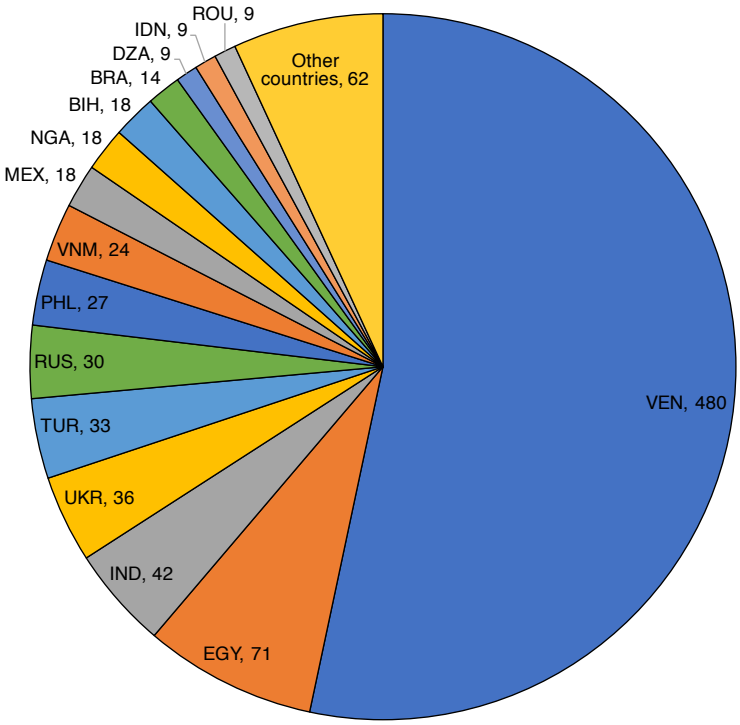


Figure 6.5: Country distribution among all participants who participated in the chatbot tasks.

We noticed that a worker also commented—"*Good easy. Why the quiz or select tile(s) not in telegram app, it's will more fun.*"—on the fact that they would prefer that the selection grid tool in the chatbot for the *Object Labelling* task were to be natively supported in Telegram.

In contrast, another worker commented within the *Object Labelling* task in Spanish: "*MEJORAR LAS INSTRUCCIONES*", which indicated that they found the instructions were insufficient.

6.4. Towards Conversational Human Computation

Through analysis of the results on all three experiments, we show that chatbots could be a suitable alternative to the current web-based microtask crowdsourcing platforms. While it may be too soon to fully embrace the chatbot as an alternative, our results hold for at least the six considered task types both in terms of execution time and quality.

We have received universal acclaim among the participants for our chatbot as an alternative interface for performing microtasks. Workers had expressed in particular their appreciation for the *Mixed Custom Keyboard* for single- and multiple-selection based tasks, and the selection-grid input for *Object Labelling*.

By comparing our work to previous work in mobile crowdsourcing [14, 35, 38], we follow in their footsteps and again highlight the importance of task and interaction design. Our results suggest that for common tasks like *Sentiment Analysis* and *Image Annotation*, the custom *Button-based* keyboard enables execution times comparable to web interfaces. The instructions and chatbot navigational commands also have an impact on the execution time. We find that this is especially apparent for domain specific tasks (e.g. labelling of food).

While our results may not be directly compared to previous work, due to unavailable datasets and code, we find that our results are similar to the ones obtained in previous studies with mobile user interfaces [14, 38]. However, we note that by comparing the answer quality of our *Human OCR* and *Image Annotation* task to the results found by Kumar et al. in [38], they were rather of comparable quality.

With the elaborate experimental design and execution plan in Chapter 5, we have accounted for many in- and extrinsic factors that could have influenced the results. However, we state the following potential threats to the validity of our results:

- **Representation of Worker Population**: The 316 workers who participated in our experiments still remains a small group among the vast population of crowd workers. As a result, the group of workers we recruited may not represent the entire population of crowd workers well. While Figure Eight is a popular platform, which also recruits from many other external worker channels, we need further experimentation to generalize our findings. Therefore we urge for the experimentation on other crowdsourcing platforms aside from Figure Eight, as well as exploring other messaging platforms for the chatbot.

- **User Interface Usability**: The tasks that we have implemented in the *Web Interface* were kept very standard and simple to minimize the risk of confusing workers with an unknown UI. Similarly, for the chatbot, we provided only the bare necessities for interaction to complete microtasks. We remark that there is a clear possibility that workers may not be familiar with the specific Telegram messaging service. However, we allowed workers to pick any Telegram client they could use. This also included a *Web-based* Telegram client (which is identical in functionality, and look and feel to the native desktop and mobile clients), which workers unfamiliar with Telegram may have found easier to work with.

- **Task Complexity**: In our experiments, we have not actively addressed the complexity of a task contrary to the works in [14, 38]. We also take note that as the task complexity rises, the instructions of the task also need to heed the issue of effective communication.

- **Limitations in Task & Input Types**: We acknowledge the potential to explore more different task types and UI element variations. As an example, the chatbot allows for interesting UI inputs such as voice-recorded messages, sending of video recordings, sharing images, etc. Aside from UI input elements, there are also other task types that remain to be explored such as annotating videos (done in [38, 59]).

7

Conclusion and Future Work

In this chapter, we provide an answer to our main research question: *"To what extent can text-based conversational agents support the execution of microtask crowdsourcing activities?"* in Section 7.1. We then provide an outlook on future opportunities in progressing conversational HC in Section 7.2.

7.1. Conclusion

As we are witnessing widespread interest and adoption of text-based conversational agents or chatbots in both industry and as a subject of study, we raised the question of the potential applicability of the chatbot as an alternative interface to the traditional web interface for performing HC activities.

In our thesis, we aimed to investigate the viability of the chatbot as a microtask crowdsourcing platform. Towards this goal, we started by dividing our main research question into the three separate research-sub questions:

RQ 1: How do we build a text-based conversational agent that facilitates the execution of microtask crowdsourcing activities?

Inspired by the literature, we have designed a full chatbot system with a dual nature; a chatbot as a conversational partner and as a microtask crowdsourcing platform. In our work, we have primarily directed our efforts towards the realization of the latter by laying the foundation for a conversational interface in which microtask crowdsourcing activities are fully supported. We have implemented and deployed the system to function in production, which has been directly used for running all the tasks from the experiments we designed. We adopted the principles of modular system design and setup the system such that individual system components may be extended for future development, including support for multiple messaging services.

RQ 2: How do we map web-based user interface elements to chatbot user interface elements for microtask crowdsourcing?

In order to perform a comparison between the *Web* and *Chatbot* interface, we have designed and implemented tasks in both interfaces. To be able to host tasks that we are currently able to create in the *Web* interface, we proposed an abstraction alongside a mapping to represent UI elements found in the *Web* to the *Chatbot*. With this abstraction, we implemented and incorporated the respective UI elements in our conducted experiments.

RQ 3: How do different types of user interface input elements for the conversational interface affect the execution time and output quality of microtasks?

In order to study the effects of different types of UI input elements, we conducted a thorough experimental campaign in which 316 distinct workers took part in. Thereafter, we performed a systematic analysis of six task types, in which we show that task execution times and output qualities in the chatbot are comparable

to the tasks that were run through a web-based interface. Among all participants, we found that they unanimously expressed interest in performing microtasks in the future through the chatbot as a work execution medium. With highlights found in previous work, we strengthened and stressed the importance of task-specific interaction design. Furthermore, we showed that the convenience of advanced text input interfaces currently available in messaging platforms like Telegram may affect the task execution time.

7.2. Future Work

With our work, we have set the first step in a potential new direction for HC. We have shown that conversational HC is feasible and that it has also sparked interest in crowd workers. As a result, our work provides inspiration for future research opportunities:

- **Device Constraints**: In our work, we did not address the impact of device specifications in our UI design. We imagine that given the opportunity to perform conversational HC in a mobile interface, it becomes of the essence to consider the possibility to prefer certain UI elements in favour of others, as a result of e.g. screen size and touch controls. We foresee challenges in balancing task complexity to fit appropriate UI elements, such as trading-off information gain in a tool for *Image Segmentation* in favour of user usability.

- **Mobile Crowdsourcing**: Because conversational interfaces are already readily available through widely popular messaging services such as Telegram and Facebook Messenger, we notice the new opportunities for performing crowdsourcing activities in a mobile fashion. We see an opportunity to use the conversational interface to perform situational and location-based crowdsourcing, where workers may perform tasks requiring to be physically present at some location. For example, tasks may include taking photos with a mobile phone or creating a recording at some specified location and time.

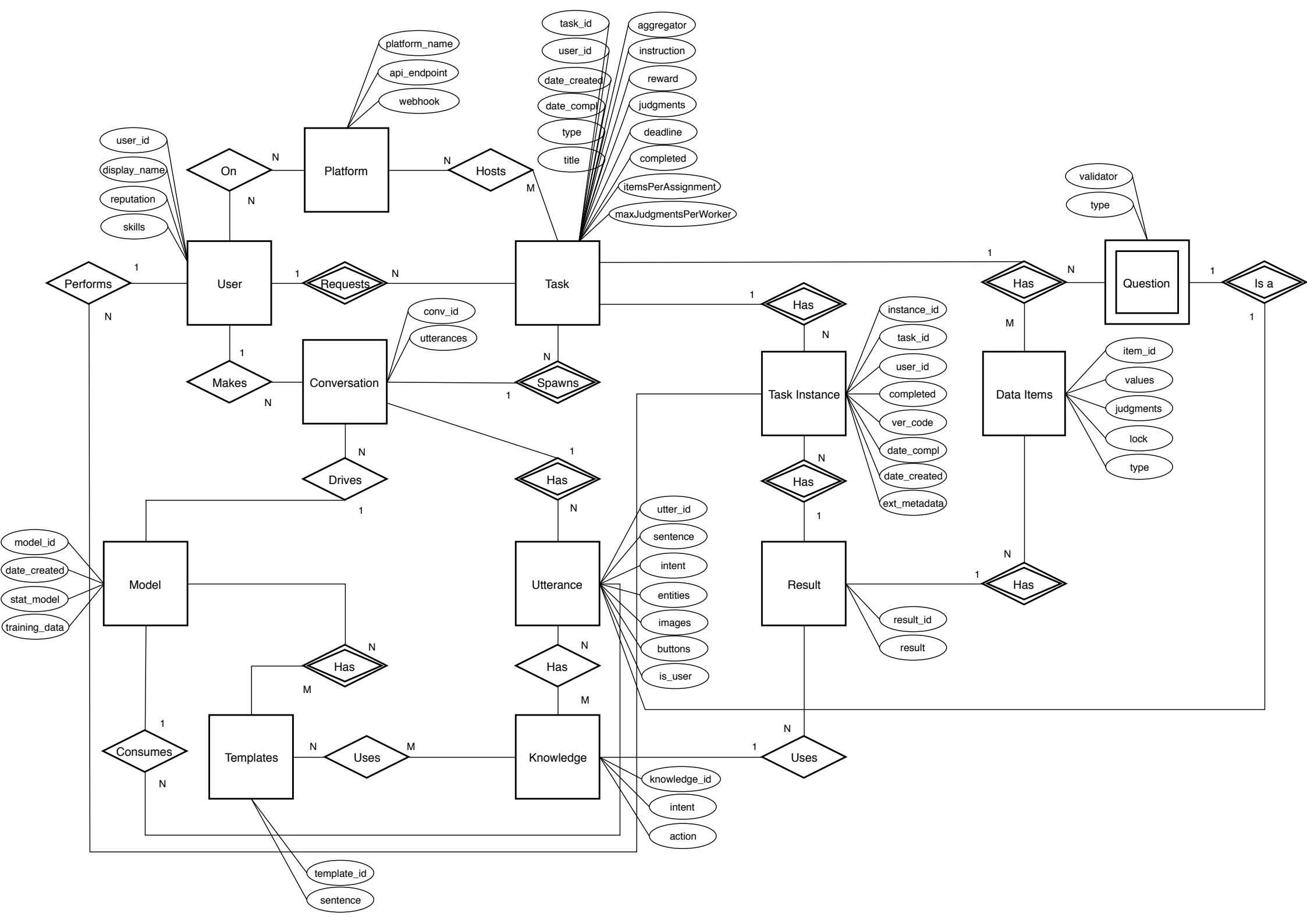
- **System Peak Performance**: While our proposed system is capable of facilitating a variety of microtasks, this was still carried out in a (partially) controlled environment. We raise therefore the question how well our system will function in a true real-world setting as high-traffic microtask crowdsourcing platform. Furthermore, we are in particular interested in what the peak performance is for different task and content types and different use cases (e.g. video content, microtask workflows, worker retainment). Moreover, we are curious how well a microtask crowdsourcing platform would function over a prolonged period of time within a popular messaging platform such as Telegram. We see opportunities to take advantage of strategies involving the notification system of messaging services to maintain higher worker retainment. We are also interested in the application of combining multiple worker sources—including workers from the messaging platform and other web-based platforms—to attempt sustaining near-real time crowdsourcing.

- **Conversational Requester Interface**: In our work, we only looked from the perspective of a worker. However, we raise the question if the conversational interface could also be applied effectively for designing tasks as a requester. As we have shown that the conversational interface is a viable alternative for microtask execution, we have yet to discern the challenges in the conversational interface for the entire task design and implementation process. To achieve the full realization of a conversational crowdsourcing platform, we require to facilitate the means for external task requesters. We foresee challenges in managing data for the task creation process and monitoring on-going tasks in the conversational interface.

- **Exploring Conversation**: A very large aspect we did not address in our work, which of course is a large benefit of the conversational interface, is taking up the “art of conversation”. We contemplate that the field of cybernetics may provide interesting insights into the design of the chatbot as a fully fledged conversational partner during microtask activities. In our current implementation, we employed a simple feedback loop mechanism to acknowledge user interaction, but factors such as chatbot personality may potentially affect worker performance. Furthermore, we note the possibility of training workers to gain knowledge and learn new skills through repeated conversations with the chatbot. As a result, we also note that the chatbot may, in turn, learn new things from workers, which the chatbot may then teach to other workers. Furthermore, we are also interested in the dynamics of group-based conversation for the execution of microtasks.

A

Full Entity-Relationship Diagram


platform_name
api_endpoint
webhook
task_id
user_id
date_created
date_compl
type
title
aggregator
instruction
reward
judgments
deadline
completed
itemsPerAssignment
maxJudgmentsPerWorker
user_id
display_name
reputation
skills
On
Platform
Hosts
validator
type
Performs
User
Requests
Task
Has
Question
Is a
conv_id
utterances
Makes
Conversation
Spawns
Has
instance_id
task_id
user_id
completed
ver_code
date_compl
date_created
ext_metadata
Task Instance
Data Items
item_id
values
judgments
lock
type
Drives
Has
Has
model_id
date_created
stat_model
training_data
Model
Utterance
Result
utter_id
sentence
intent
entities
images
buttons
is_user
result_id
result
Has
Has
Has
Consumes
Templates
Uses
Knowledge
Uses
knowledge_id
intent
action
template_id
sentence

B

Included Figure Eight Contributor Channels

Table B.1: All default included contributor channels as selectable through the task settings in Figure Eight.

Channel Name	Included
Adept Technologies BPO	✓
BitcoinGet	✓
CapeStart	✓
Clickworker (DE)	✓
Clickworker (EN)	✓
ClixSense	✓
Coleman Data Solutions	✓
Content Runner	✓
Creative Solutions Network	✓
Crowd Guru	✗
CrowdWorks	✓
DDD (Kenya)	✓
Daproim	✓
Daproim (Digital Campus Connect)	✓
Data House	✓
DataPure	✓
Earnably	✓
EntropiaPartners (Second Life)	✓
Figure Eight Elite	✓
FusionCash	✓
FusionCash (Family Friendly)	✓
Get-Paid.com	✓
Gift Hunter Club	✓
GiftHulk	✓
GrabPoints	✓
Hiving	✓
Human in the Loop	✓
IndiVillage	✓
Infosource Bulgaria	✓
KayCaptions	✓
KeepRewarding	✓
Kinetic Business Solutions	✓
Listia	✓
Memolink	✓
NeoBux	✓
Opsify LLC	✓
Points to Shop	✓
PrizeRebel	✓
Proximo	✓
SmartOne	✓
SuperRewards	✓
Swagbucks (Prodege)	✓
Taqadam	✓
TeleNet Global	✓
TimeBucks Tasks	✓
Tremor Games	✓
TrueAccord	✓
Vivatic Tasks	✓
Wannads	✓
Zen3 Infosolutions	✓
iMerit India	✓
infosearchbpo	✓
instaGC	✓
oWorkers	✓
rProcess	✓
vCare	✗
vCare India	✓
vCare USA	✓

C

MTurk and Figure Eight Task Templates

Table C.1: Task templates which a requester is able to pick from in AMT and Figure Eight.

Task type		**Data Type**	**Input Type Elements**	**Task Goal**
Amazon Mechanical Turk	**Figure Eight**			
Sentiment Wizard; Sentiment Project; Sentiment of a Tweet	Sentiment Analysis	Text	Multiple Choice	Given a short text fragment, determine what the sentiment of the fragment is.
Choose image A or B	Search Relevance	Text, Image	Multiple Choice	Given a search query and its result, compare which one is more relevant.
Categorization Wizard; Categorization Project	Data Categorization	Text, Image	Multiple Choice	Given a piece of media content, determine what categories it belongs to.
Collect data; Collect data from a Website; Writing	Data Collection & Enrichment	Text	Text, Multiple Choice	Given a (partial) data record, find or complement a data attribute.
-	Data Validation	Text, Image	Text, Multiple Choice	Given a piece of media content, validate its correctness and completeness.
Tagging of an Image	Image Annotation	Image	Multiple Choice, Bounding Box, Draw Line, Mark Objects	Given an image annotate the image contents from a list of options, bound all objects in boxes, indicate the center of objects, draw the object body length.
Transcription from A/V; Transcription from a Receipt; Transcription from an Image	Transcription	Image, Audio	Text, Multiple Choice	Given an image, audio or video fragment, indicate the legibility, intelligibility or visibility and transcribe the contents.
Moderation of an Image	Content Moderation	Text, Image	Multiple Choice	Given an image or user comment or URL, determine if the contents comply with a set of given service rules.
Survey (Link)	-	None	Different types-several questions	A series of questions that may highly differ. May also be run on an external platform by providing an URL.

D

Consent Form

Research Participation

You are being invited to participate in a research study to evaluate a novel method to perform *Human Computation* in a conversational manner.
This study is being conducted by *Owen Huang, MSc student* from the *TU Delft*.

The purpose of this research study is to evaluate any interaction you have with a chatbot in the instant messaging service **Telegram**, while you perform a microtask, comparable to the tasks found in **Figure Eight**. Performing this microtask will take you approximately 5 minutes to complete.

Your participation in this study is entirely voluntary. You may choose not to participate by closing this task in Figure Eight and Telegram. By performing and fully completing this task in Figure Eight and the given microtask in Telegram (i.e. clicking the '*Submit*' button, at the end of the microtask in Telegram), you agree to participate in this research study. If you decide to participate in this study, you may withdraw your participation at any time.
If you decide not to participate in this study or decide to withdraw your participation at any time, you will not be penalized. If you do not complete this task in Figure Eight and did not click the '*Submit*' button in Telegram, this will be treated as the withdrawal of your participation in this study and any answers you have given thus far will not be used.

Participation Procedure

The procedure of performing the microtask in Telegram involves, if applicable:

- The registration of a Telegram account with a valid mobile phone number.
- The process of downloading a Telegram mobile or desktop client.

Thereafter, you will be redirected through clicking the URL in this Figure Eight task (after confirming you possess a Telegram account) to start a one-on-one conversation with a Telegram bot.
You will then perform a microtask by interacting with the Telegram bot, which involves reading and understanding any given instructions to you and answering a series of short and simple questions.
After the completion of the microtask you will be prompted to click the '*Submit*' button in order to confirm your participation in this study and submitting all your answers given to the Telegram bot.
By clicking the '*Submit*' button, you will be presented with a validation token that verifies the completion of your microtask and allows the completion of the task in Figure Eight.

Any information we store, as collected by the completion of this task in Figure Eight and the use of Telegram is done fully anonymous. We will not store any information that may personally identify you, such as IP adresses, names or mobile phone numbers. The results of this study will only be used for academic purposes, such as scholarly publications.

If you have any inquiries about this research study, please contact Owen Huang (O.Huang@student.tudelft.nl). This research has been reviewed by the TU Delft Human Research Ethics Committee.

ELECTRONIC CONSENT

By accepting and completing this task in Figure Eight and the given microtask in Telegram:

- You declare that you are over 18 years old.
- You voluntarily agree to participate.
- You have fully read and understood the above information.

If you do not wish to participate in this research study, please close this task in Figure Eight and if applicable any open conversations in Telegram incurred by the Figure Eight task.

E

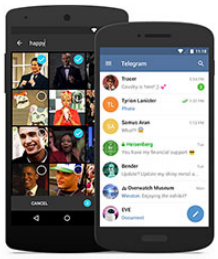
Telegram Registeration Instructions

How to register a Telegram account

Note: If you already have a registered Telegram account, you may skip these steps.
Please log into your Telegram account to continue on with the task.

Steps

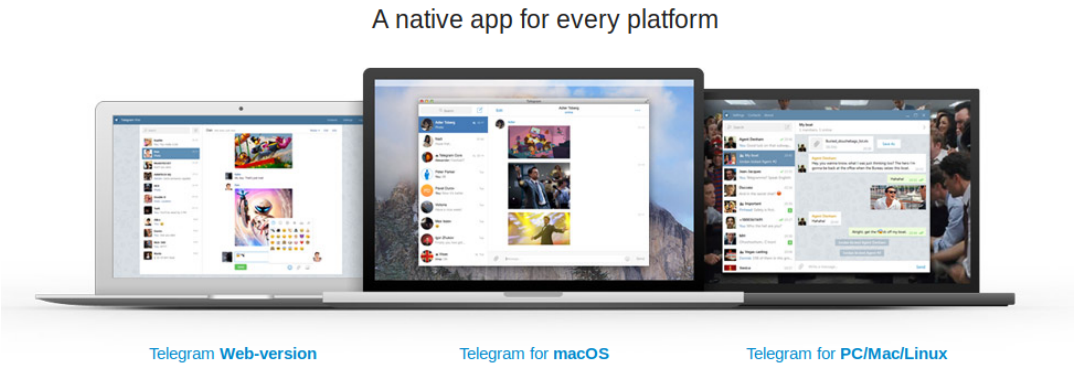
1. Download and install the mobile or desktop Telegram client here.



Telegram for **Android**

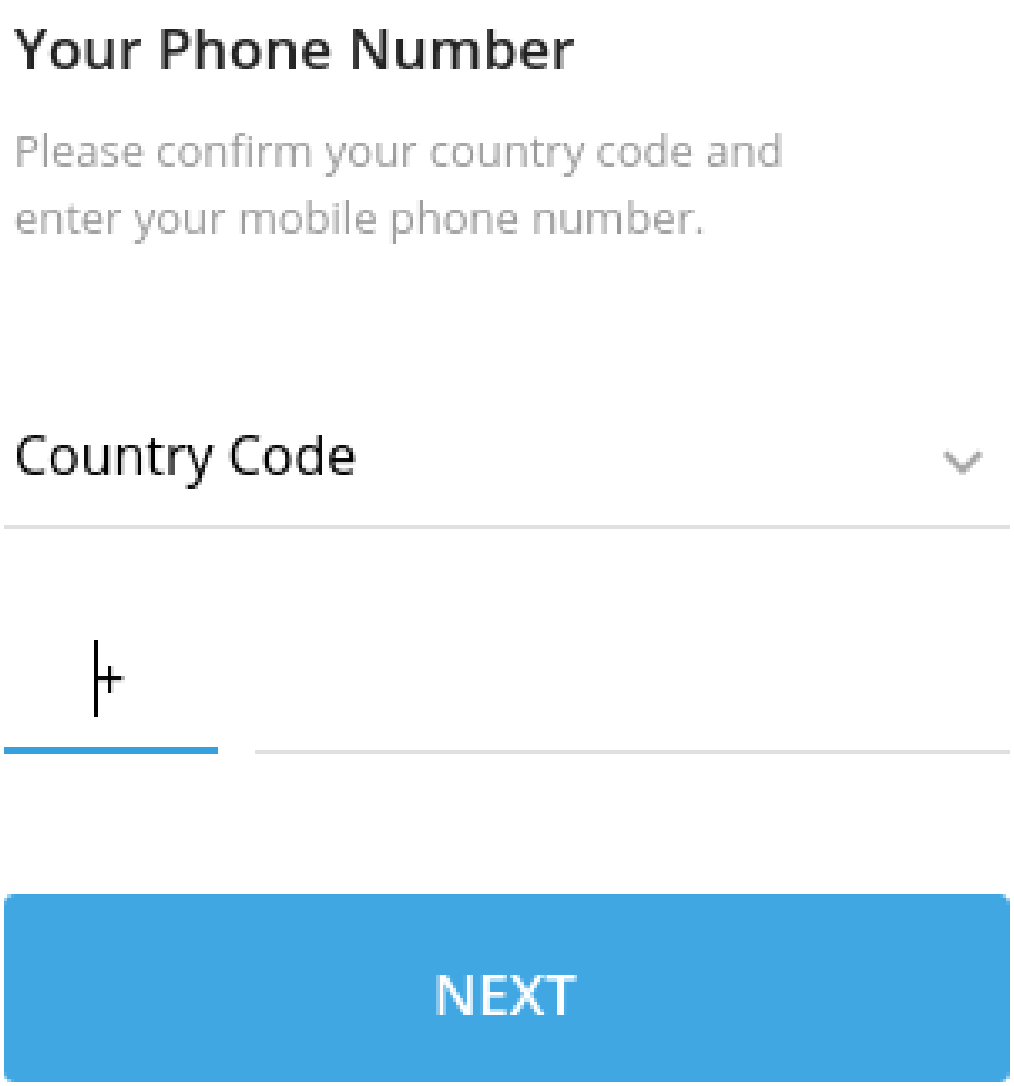
Telegram for **iPhone / iPad**

Telegram for **WP**

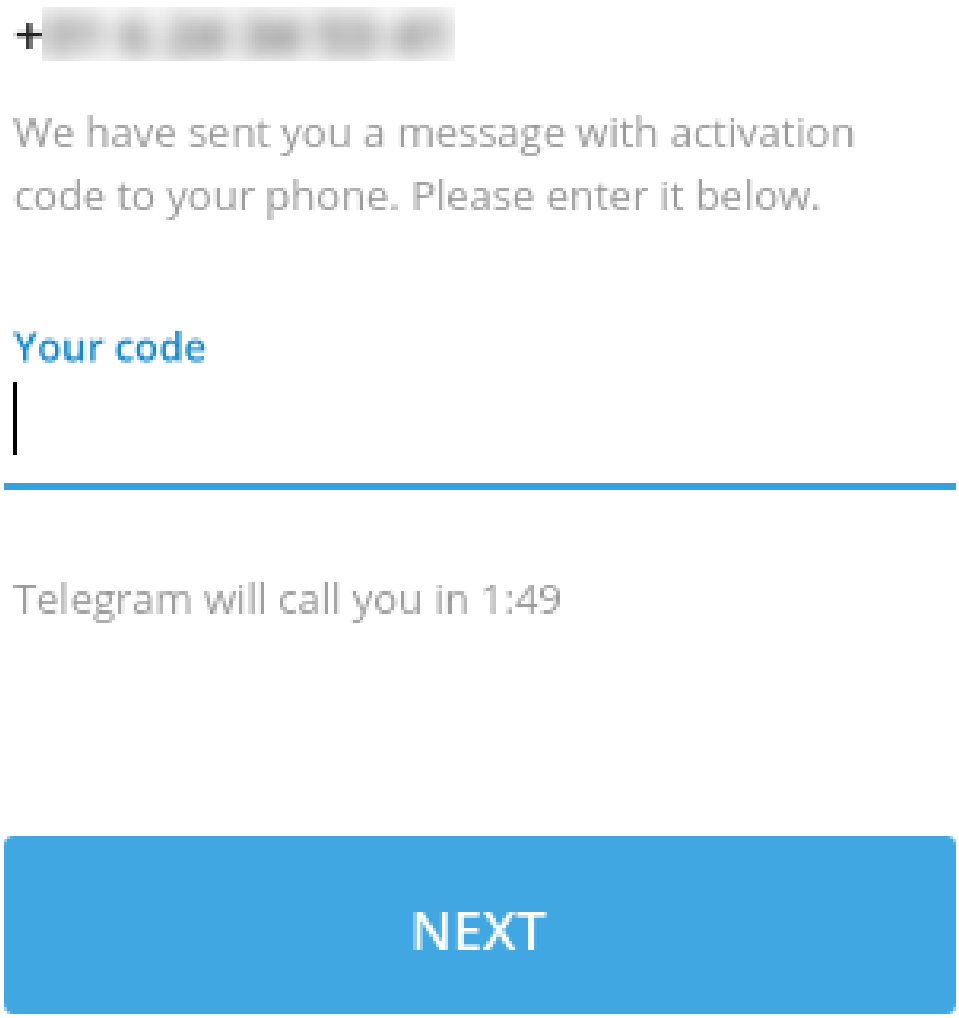


Telegram **Web-version** Telegram for **macOS** Telegram for **PC/Mac/Linux**

2. Open the mobile or desktop Telegram client and proceed until you are eventually asked to register with your phone number:



3. Register with a valid mobile phone number and activate your login by inputting the activation code received through either via SMS or phone call.



4. After putting in the activation code provided by Telegram, you have successfully registered and logged into a Telegram account and are ready for this task!

Bibliography

[1] Hadeel Al-Zubaide and Ayman A Issa. Ontbot: Ontology based chatbot. In *Fourth International Symposium on Innovation in Information & Communication Technology (ISIICT)*, pages 7–12. IEEE, 2011.

[2] Tara S Behrend, David J Sharek, Adam W Meade, and Eric N Wiebe. The viability of crowdsourcing for survey research. *Behavior research methods*, 43(3):800, 2011.

[3] Janine Berg. Income security in the on-demand economy: Findings and policy lessons from a survey of crowdworkers. *Comp. Lab. L. & Pol'y J.*, 37:543, 2015.

[4] Fumihiro Bessho, Tatsuya Harada, and Yasuo Kuniyoshi. Dialog system using real-time crowdsourcing and twitter large-scale corpus. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 227–231. Association for Computational Linguistics, 2012.

[5] Jeffrey P Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, et al. Vizwiz: nearly real-time answers to visual questions. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 333–342. ACM, 2010.

[6] Luka Bradeško, Michael Witbrock, Janez Starc, Zala Herga, Marko Grobelnik, and Dunja Mladenić. Curious cat–mobile, context-aware conversational crowdsourcing knowledge acquisition. *ACM Transactions on Information Systems (TOIS)*, 35(4):33, 2017.

[7] Jonathan Bragg, Andrey Kolobov, Mausam Mausam, and Daniel S Weld. Parallel task routing for crowdsourcing. In *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.

[8] Petter Bae Brandtzaeg and Asbjørn Følstad. Why people use chatbots. In *International Conference on Internet Science*, pages 377–392. Springer, 2017.

[9] Alec Burmania, Srinivas Parthasarathy, and Carlos Busso. Increasing the reliability of crowdsourcing evaluations using online quality assessment. *IEEE Transactions on Affective Computing*, 7(4):374–388, 2016.

[10] Carrie J. Cai, Shamsi T. Iqbal, and Jaime Teevan. Chain reactions: The impact of order on microtask chains. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3143–3154, New York, NY, USA, 2016. ACM.

[11] Justin Cheng, Jaime Teevan, Shamsi T. Iqbal, and Michael S. Bernstein. Break it down: A comparison of macro- and microtasks. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 4061–4064, New York, NY, USA, 2015. ACM.

[12] Justin Cranshaw, Emad Elwany, Todd Newman, Rafal Kocielnik, Bowen Yu, Sandeep Soni, Jaime Teevan, and Andrés Monroy-Hernández. Calendar. help: Designing a workflow-based scheduling agent with humans in the loop. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2382–2393. ACM, 2017.

[13] Peng Dai, Jeffrey M. Rzeszotarski, Praveen Paritosh, and Ed H. Chi. And now for something completely different: Improving crowdsourcing workflows with micro-diversions. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, pages 628–638, New York, NY, USA, 2015. ACM.

[14] Vincenzo Della Mea, Eddy Maddalena, and Stefano Mizzaro. Mobile crowdsourcing: Four experiments on platforms and tasks. *Distributed and Parallel Databases*, 33(1):123–141, 2015.

[15] Djellel Eddine Difallah, Michele Catasta, Gianluca Demartini, and Philippe Cudré-Mauroux. Scaling-up the crowd: Micro-task pricing schemes for worker retention and latency improvement. In *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.

[16] Boi Faltings, Radu Jurca, Pearl Pu, and Bao Duy Tran. Incentives to counter bias in human computation. In *Second AAAI conference on human computation and crowdsourcing*, 2014.

[17] Oluwaseyi Feyisetan, Elena Simperl, Max Van Kleek, and Nigel Shadbolt. Improving paid microtasks through gamification and adaptive furtherance incentives. In *Proceedings of the 24th International Conference on World Wide Web*, pages 333–343. International World Wide Web Conferences Steering Committee, 2015.

[18] Michael J Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM, 2011.

[19] Ujwal Gadiraju, Ricardo Kawase, and Stefan Dietze. A taxonomy of microtasks on the web. In *Proceedings of the 25th ACM Conference on Hypertext and Social Media*, pages 218–223, New York, NY, USA, 2014. ACM.

[20] Supratip Ghose and Jagat Joyti Barua. Toward the implementation of a topic specific dialogue based natural language chatbot as an undergraduate advisor. In *International Conference on Informatics, Electronics & Vision (ICIEV)*, pages 1–5. IEEE, 2013.

[21] Daniel Haas, Jason Ansel, Lydia Gu, and Adam Marcus. Argonaut: macrotask crowdsourcing for complex data processing. *Proceedings of the VLDB Endowment*, 8(12):1642–1653, 2015.

[22] David Hirshleifer and Siew Hong Teoh. Herd behaviour and cascading in capital markets: A review and synthesis. *European Financial Management*, 9(1):25–66, 2003.

[23] Ting-Hao Kenneth Huang, Walter S Lasecki, and Jeffrey P Bigham. Guardian: A crowd-powered spoken dialog system for web apis. In *Third AAAI conference on human computation and crowdsourcing*, 2015.

[24] Ting-Hao Kenneth Huang, Amos Azaria, and Jeffrey P Bigham. Instructablecrowd: Creating if-then rules via conversations with the crowd. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1555–1562. ACM, 2016.

[25] Ting-Hao Kenneth Huang, Walter S Lasecki, Amos Azaria, and Jeffrey P Bigham. "Is There Anything Else I Can Help You With?" Challenges in Deploying an On-Demand Crowd-Powered Conversational Agent. In *Fourth AAAI Conference on Human Computation and Crowdsourcing*, 2016.

[26] Ting-Hao'Kenneth' Huang, Yun-Nung Chen, and Jeffrey P Bigham. Real-time on-demand crowd-powered entity extraction. *arXiv preprint arXiv:1704.03627*, 2017.

[27] Ting-Hao'Kenneth' Huang, Joseph Chee Chang, and Jeffrey P Bigham. Evorus: A Crowd-powered Conversational Assistant Built to Automate Itself Over Time. *arXiv preprint arXiv:1801.02668*, 2018.

[28] Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Lam Ngoc Tran, and Karl Aberer. An evaluation of aggregation techniques in crowdsourcing. In *International Conference on Web Information Systems Engineering*, pages 1–15. Springer, 2013.

[29] Panagiotis G Ipeirotis and Evgeniy Gabrilovich. Quizz: targeted crowdsourcing with a billion (potential) users. In *Proceedings of the 23rd international conference on World wide web*, pages 143–154. ACM, 2014.

[30] Panagiotis G. Ipeirotis and Praveen K. Paritosh. Managing crowdsourced human computation: A tutorial. In *Proceedings of the 20th International Conference Companion on World Wide Web*, pages 287–288, New York, NY, USA, 2011. ACM.

[31] Panagiotis G Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 64–67. ACM, 2010.

[32] David R Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In *Advances in neural information processing systems*, pages 1953–1961, 2011.

[33] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E Kraut. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 43–52. ACM, 2011.

[34] P. Kucherbaev, A. Bozzon, and G. Houben. Human aided bots. *IEEE Internet Computing*, pages 1–1, 2018.

[35] Pavel Kucherbaev, Azad Abad, Stefano Tranquillini, Florian Daniel, Maurizio Marchese, and Fabio Casati. Crowdcafe-mobile crowdsourcing platform. *arXiv preprint arXiv:1607.01752*, 2016.

[36] Roland Kuhn and Renato De Mori. The application of semantic classification trees to natural language understanding. *IEEE transactions on pattern analysis and machine intelligence*, 17(5):449–460, 1995.

[37] Anand Kulkarni, Matthew Can, and Björn Hartmann. Collaboratively crowdsourcing workflows with turkomatic. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1003–1012. ACM, 2012.

[38] Abhishek Kumar, Kuldeep Yadav, Suhas Dev, Shailesh Vaya, and G. Michael Youngblood. Wallah: Design and evaluation of a task-centric mobile-based crowdsourcing platform. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 188–197. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[39] Walter S. Lasecki, Kyle I. Murray, Samuel White, Robert C. Miller, and Jeffrey P. Bigham. Real-time crowd control of existing interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 23–32, New York, NY, USA, 2011. ACM.

[40] Walter S. Lasecki, Rachel Wesley, Jeffrey Nichols, Anand Kulkarni, James F. Allen, and Jeffrey P. Bigham. Chorus: A crowd-powered conversational assistant. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, pages 151–162, New York, NY, USA, 2013. ACM.

[41] Walter S Lasecki, Raja Kushalnagar, and Jeffrey P Bigham. Legion scribe: real-time captioning by non-experts. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility*, pages 303–304. ACM, 2014.

[42] Edith Law and Luis von Ahn. *Human Computation*. Morgan & Claypool Publishers, 1st edition, 2011. ISBN 1608455165, 9781608455164.

[43] Guoliang Li, Jiannan Wang, Yudian Zheng, and Michael J Franklin. Crowdsourced data management: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2296–2319, 2016.

[44] Xulei Liang, Rong Ding, Mengxiang Lin, Lei Li, Xingchi Li, and Song Lu. CI-Bot: A Hybrid Chatbot Enhanced by Crowdsourcing. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, pages 195–203. Springer, 2017.

[45] Bing Liu. Sentiment analysis and subjectivity. *Handbook of natural language processing*, 2:627–666, 2010.

[46] Qiang Liu, Jian Peng, and Alexander T Ihler. Variational inference for crowdsourcing. In *Advances in neural information processing systems*, pages 692–700, 2012.

[47] Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. CDAS: A Crowdsourcing Data Analytics System. *Proceedings of the VLDB Endowment*, 5(10):1040–1051, 2012.

[48] Yefeng Liu, Vili Lehdonvirta, Mieke Kleppe, Todorka Alexandrova, Hiroaki Kimura, and Tatsuo Nakajima. A crowdsourcing based mobile image translation and knowledge sharing service. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, page 6. ACM, 2010.

[49] Yefeng Liu, Todorka Alexandrova, and Tatsuo Nakajima. Gamifying intelligent environments. In *Proceedings of the 2011 international ACM workshop on Ubiquitous meta user interfaces*, pages 7–12. ACM, 2011.

[50] Jan Lorenz, Heiko Rauhut, Frank Schweitzer, and Dirk Helbing. How social influence can undermine the wisdom of crowd effect. *Proceedings of the National Academy of Sciences*, 108(22):9020–9025, 2011.

[51] Andrew Mao, Ece Kamar, Yiling Chen, Eric Horvitz, Megan E Schwamb, Chris J Lintott, and Arfon M Smith. Volunteering versus work for pay: Incentives and tradeoffs in crowdsourcing. In *First AAAI conference on human computation and crowdsourcing*, 2013.

[52] Prayag Narula, Philipp Gutheim, David Rolnitzky, Anand Kulkarni, and Bjoern Hartmann. Mobileworks: A mobile crowdsourcing platform for workers at the bottom of the pyramid. *Human Computation*, 11 (11):45, 2011.

[53] André Orléan. Bayesian interactions and collective dynamics of opinion: Herd behavior and mimetic contagion. *Journal of Economic Behavior & Organization*, 28(2):257–274, 1995.

[54] Alexander J Quinn and Benjamin B Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1403–1412. ACM, 2011.

[55] Daniela Retelny, Sébastien Robaszkiewicz, Alexandra To, Walter S. Lasecki, Jay Patel, Negar Rahmati, Tulsee Doshi, Melissa Valentine, and Michael S. Bernstein. Expert Crowdsourcing with Flash Teams. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, pages 75–85, New York, NY, USA, 2014. ACM.

[56] Alan Ritter, Colin Cherry, and William B Dolan. Data-driven response generation in social media. In *Proceedings of the conference on empirical methods in natural language processing*, pages 583–593. Association for Computational Linguistics, 2011.

[57] Joel Ross, Lilly Irani, M Silberman, Andrew Zaldivar, and Bill Tomlinson. Who are the crowdworkers?: shifting demographics in mechanical turk. In *CHI'10 extended abstracts on Human factors in computing systems*, pages 2863–2872. ACM, 2010.

[58] Jeffrey M Rzeszotarski, Ed Chi, Praveen Paritosh, and Peng Dai. Inserting micro-breaks into crowdsourcing workflows. In *First AAAI Conference on Human Computation and Crowdsourcing*, 2013.

[59] Navkar Samdaria, Ajith Sowndararajan, Ramadevi Vennelakanti, and Sriganesh Madhvanath. Mobile interfaces for crowdsourced multimedia microtasks. In *Proceedings of the 7th International Conference on HCI, IndiaHCI 2015*, pages 62–67, New York, NY, USA, 2015. ACM.

[60] Denis Savenkov and Eugene Agichtein. CRQA: Crowd-Powered Real-Time Automatic Question Answering System. In *Fourth AAAI Conference on Human Computation and Crowdsourcing*, 2016.

[61] Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The knowledge engineering review*, 21(2):97–126, 2006.

[62] Heinz Schmitz and Ioanna Lykourentzou. Online sequencing of non-decomposable macrotasks in expert crowdsourcing. *ACM Transactions on Social Computing*, 1(1):1, 2018.

[63] Iulian V Serban, Chinnadhurai Sankar, Mathieu Germain, Saizheng Zhang, Zhouhan Lin, Sandeep Subramanian, Taesup Kim, Michael Pieper, Sarath Chandar, Nan Rosemary Ke, et al. A deep reinforcement learning chatbot. *arXiv preprint arXiv:1709.02349*, 2017.

[64] David R Traum and Staffan Larsson. The information state approach to dialogue management. In *Current and new directions in discourse and dialogue*, pages 325–353. Springer, 2003.

[65] Luis Von Ahn. *Human Computation.* PhD thesis, Pittsburgh, PA, USA, 2005. AAI3205378.

[66] Luis Von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006.

[67] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer, 2003.

[68] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer, 2003.

[69] Richard Wallace. The elements of aiml style. *Alice AI Foundation*, 2003.

[70] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494, 2012.

[71] Jiannan Wang, Sanjay Krishnan, Michael J Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 469–480. ACM, 2014.

[72] Sibo Wang, Xiaokui Xiao, and Chun-Hee Lee. Crowd-based deduplication: An adaptive approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1263–1277. ACM, 2015.

[73] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[74] Tingxin Yan, Matt Marzilli, Ryan Holmes, Deepak Ganesan, and Mark Corner. mcrowd: A platform for mobile crowdsourcing. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 347–348, New York, NY, USA, 2009. ACM.

[75] Jeffrey M Zacks, Barbara Tversky, and Gowri Iyer. Perceiving, remembering, and communicating structure in events. *Journal of Experimental Psychology: General*, 130(1):29, 2001.

[76] Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. Qasca: A quality-aware task assignment system for crowdsourcing applications. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1031–1046. ACM, 2015.

[77] Dongqing Zhu and Ben Carterette. An analysis of assessor behavior in crowdsourced preference judgments. In *SIGIR 2010 workshop on crowdsourcing for search evaluation*, pages 17–20, 2010.