Proceedings of the Tenth AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2022)

# TaskLint: Automated Detection of Ambiguities in Task Instructions

**V. K. Chaithanya Manam, Joseph Divyan Thomas, Alexander J. Quinn**

Purdue University, West Lafayette, Indiana, USA
{vmanam, thoma900, aq}@purdue.edu

## Abstract

Clear instructions are a necessity for obtaining accurate results from crowd workers. Even small ambiguities can force workers to choose an interpretation arbitrarily, resulting in errors and inconsistency. Crisp instructions require significant time to design, test, and iterate. Recent approaches have engaged workers to detect and correct ambiguities. However, this process increases the time and money required to obtain accurate, consistent results. We present TaskLint, a system to automatically detect problems with task instructions. Leveraging a diverse set of existing NLP tools, TaskLint identifies words and sentences that *might* foretell worker confusion. This is analogous to static analysis tools for code ("linters"), which detect possible features in code that might indicate the presence of bugs. Our evaluation of TaskLint using task instructions created by novices confirms the potential for static tools to improve task clarity and the accuracy of results, while also highlighting several challenges.

## Introduction

Crowdsourcing facilitates ease of delegation and completion of tasks using platforms such as Amazon Mechanical Turk and Upwork. Instruction clarity is paramount for optimum results. Designing effective instructions requires significant time and attention. Several iterations may be required, especially for tasks with intricate requirements. When the goal of delegating work is to save the requester time from having to do it themselves, spending time iterating on the task design naturally offsets that value.

Requesters often have a clear idea about the task responses they expect to receive. However, conveying that understanding to workers is complicated, as workers may lack prior knowledge of the task prerequisites. Ambiguities in instructions can lead to situations where different workers read the same set of instructions and interpret them differently. To address these ambiguities, researchers have proposed several methods, which can be categorized into two categories: proactive and reactive.

*Proactive approaches* seek to reduce flaws before the tasks are posted through enhancements to the task authoring interface. For example, guided interfaces help novice workers create better tasks by providing guidelines and recommendations to the requester while they create the task (Gutheim and Hartmann 2012). However, existing systems do not detect ambiguities, such as contradictory statements and ambiguous named entity resolution.

The downside of guided interfaces is that they inherently constrain the range of possible tasks that can be expressed.

*Reactive approaches* address flaws after they have been posted to workers, typically using feedback from workers to improve the task design. This may be done by posting a small batch and asking workers to do the task and provide feedback. A full batch is posted later (Gaikwad et al. 2017).

To reduce the time needed to obtain results, a requester may opt to post all of the tasks at once, and then solicit feedback from workers as they do the tasks. Feedback may come in the form of comments, questions, or potential edits (Manam and Quinn 2018). The downside of this approach is that the requester must remain available to receive and respond to the feedback. That requirement increases the burden on the requester.

To reduce the task turnaround time, our approach is to analyze the text of task instructions and provide feedback to the requester, allowing them to revise the instructions—before the task is posted. We draw inspiration from *lint* tools from software engineering, which statically analyze code to detect possible flaws before the code ever executes; as well as grammar checkers and other computer-assisted writing tools.

*TaskLint* is our system for helping task authors to identify ambiguities before the task is posted (i.e., proactive). Like lint tools for code, it identifies possible problems during the process of authoring the instructions. Like grammar checkers and other computer-assisted writing tools, it uses a wide range of NLP methods, drawing on many existing tools.

The key contributions of this paper are as follows:

- TaskLint is a system that gives feedback to requesters about potential ambiguities.
- We detail a taxonomy of ambiguity types that affect task design for crowd work, building on our prior work (Manam and Quinn 2018), and map ambiguity types to NLP tools, where possible.
- Our evaluation found that tasks authored and revised using feedback from TaskLint can elicit more accurate results, relative to tasks authored without such feedback.

## Related Work

TaskLint is related to prior work on AI-based writing assistants, linguistic studies of ambiguity, instruction quality, task design by novice requesters, and factors affecting the quality of results in crowdsourcing.

### Tools for Writing Assistance

Development of automated tools to assist writers has been ongoing since at least 1983 (Levin, Boruta, and Vasconcellos 1983). The market for AI-based writing assistants is growing rapidly, from a global valuation of $359 million in 2020 to a projection of $1 billion by 2028 (Growth Market Reports 2022). Grammarly (Grammarly 2015), a popular commercial service, detects mistakes in spelling, punctuation, and grammar; offers tips for improving clarity. Jasper (Jasper 2012) and Ink Editor (INK 2016) are services that help authors of advertising copy and blog posts to optimize for search engine rankings.

### Studies of Ambiguity in Linguistic

Linguists have studied ambiguity in the context of semantics (Lyons 1977), computational linguistics (Hirst 1992; Allen 1995), and philosophy (Levinson 1983; Walton 2013). Most ambiguities can be classified as syntactic, semantic, or pragmatic (Hausser 2014).

> **Syntactic ambiguity** exists when words can be interpreted according to multiple grammatical structures (parse trees) having different meanings. These can typically be resolved by rearranging the words. For example, "blue toy box" could be a blue box of toys, or a box of blue toys.
>
> Syntactic ambiguities encompass analytical, attachment, coordination, and elliptical ambiguities.
>
> **Semantic ambiguity** exists when a syntactically unambiguous sentence can have multiple meanings, depending on context. These can be resolved using knowledge of the context. For example, "Everyone painted a picture," could mean that each person painted a separate picture, or that everyone painted one picture together. Semantic ambiguity may be caused by scope ambiguity (e.g., what a quantifier such as "every" refers to) or lexical ambiguity (i.e., hyponyms, hypernyms, homonyms, and polynyms).
>
> **Pragmatic ambiguity** exists when the context in which a sentence is expressed suggests alternate interpretations. Pragmatic ambiguity exists only relative to a particular context. For example, "Click the large button," is only ambiguous if there are two (or more) large buttons.

Many tools have been developed to detect and sometimes resolve such ambiguities (Khezri 2017; Gleich, Creighton, and Kof 2010; Yang et al. 2010). The approaches can be categorized as rule-based, statistical (or probabilistic), or hybrid (or transformation-based) (Spoustová et al. 2007). Rule-based approaches are arduous as it requires that the rules be updated frequently and cover all possibilities, including exceptions. Contextual knowledge is also necessary to obtain a solution. Statistical approaches require a large set of training data. Hybrid approaches combine the features of rule-based and statistical approaches (Spoustová et al. 2007).

**Uncertainty** is also an important linguistic phenomenon that causes text ambiguity. It can be interpreted as a product of a lack of information; uncertainty arises due to propositions whose truth value cannot be explicitly determined. Consider the following examples:

1. It is cloudy.
2. It is not cloudy.
3. It is probably cloudy.

While all these sentences contain the word 'cloudy', their meaning is different. Only the first sentence explicitly states the presence of clouds (i.e., the proposition is true). The second sentence negates it (i.e., the proposition 'It is cloudy' is false), and the third sentence is uncertain about whether the proposition is true or not. The third case is an instance of uncertainty. Uncertainty is a complex phenomenon that can only be understood when the syntactic, semantic, and pragmatic aspects are considered simultaneously. Negbio (Peng et al. 2018), LUCI (Vincze 2014; Meyers 2017) are some of the tools used to detect uncertainty.

In natural language processing (NLP), researchers have used word sense disambiguation (WSD) to identify and resolve text ambiguity. Many words have multiple meanings and WSD helps identify which 'meaning' is being employed in a particular context. The problem of WSD has been described as AI-complete (Mallery 1988). A WSD task has two variants: a lexical sample and an all-words task. The lexical sample task consists of disambiguating the occurrences of a small sample of target words that were previously selected. An all-words task comprises text that needs disambiguation.

*Semeval*[1] is an international word sense disambiguation competition held with the objective of performing a comparative evaluation of WSD systems in different kinds of tasks, including all-words and lexical sample tasks. In Semeval-2007, the best systems for coarse-grained English lexical sample and coarse-grained English all-words sample attained an $88.70\%$ accuracy and $82.50\%$ accuracy, respectively (Navigli 2009).

A small percentage of ambiguity in task instructions could drastically decrease the quality of work. Hence, none of these systems alone can be used to resolve ambiguity in task specifications.

### Instructions Quality

Gadiraju et al. studied crowd task instructions and workers' coping strategies and found that task clarity is a local property influenced by tasks and requesters, not a macro-property of a crowdsourcing ecosystem (Gadiraju, Yang, and Bozzon 2017). They introduced a model that predicts task clarity, using features such as the number of images and standard metrics of text readability, which they found to be *correlated* with high subjective assessments of task clarity by workers.

In our prior work, we have found that workers' subjective assessments of clarity do not necessarily predict the accu-

[1] http://www.senseval.org/

racy of the results or even workers' willingness to accept a task (which can affect job completion time) (Wu and Quinn 2017).

A task description can be clear—easy and comfortable to read—yet impossible to follow. For example, "Find the latitude and longitude of Washington," may sound clear, but without knowing which Washington is of interest (Washington DC or Washington State, USA) and what portion of the region to find the latitude and longitude for, the results may be inconsistent and unreliable.

Grady and Lease identified the importance of wording and terminology in task design (Grady and Lease 2010). Alonso and Baeza-Yates highlighted the importance of instruction quality in task design (Alonso and Baeza-Yates 2011). Wu et al. studied how the length of the task impacts the task turnaround time (Wu and Quinn 2017). Their results show that as task length increases, turnaround time also increases. Sampath et al. studied the effects of cognitive features such as visual saliency of the target field and working memory requirements of the task on the quality of work (Alagarai Sampath, Rajeshuni, and Indurkhya 2014). CrowdForge (Kittur et al. 2011), Turkomatic (Kulkarni, Can, and Hartmann 2012), and Crowd4u (Ikeda et al. 2016) help in decomposing a complex task into small tasks for crowdsourcing platforms. Other studies also emphasize the impact of task instructions and design on result quality (Marshall and Shipman 2013; Berg 2016; Kittur, Chi, and Suh 2008).

Daemo (Gaikwad, Whiting et al. 2017), a proactive mechanism that improves instruction quality. Requesters can post a few instances of the task to workers and receive feedback. Based on the feedback, the requester then improves the task instructions. After using Daemo, improved task instructions enabled workers to produce task results of superior quality. Similar results are observed in survey data, showing a positive relationship between the clarity of survey questions and the quality of obtained results (Fowler Jr 1992).

Sprout (Bragg and Weld 2018) and TaskMate (Manam et al. 2019) introduced a workflow that allows requesters to write minimum instructions for a task, relying on crowd workers to create clear and detailed instructions. However, workers creating detailed instructions would increase the cost and turnaround time for the requester.

WingIt (Manam and Quinn 2018) is a reactive mechanism where the instruction quality is improved as the workers work on the task. When workers encounter a problem with the task instructions, WingIt allows workers to either edit the instructions or ask a question with a guessed answer. The requester can then trust the workers' guess and allow them to continue working on the task based on their guess, or, the requester can review the workers' guess and reply with an acknowledgment containing the correct answer. However, trusting the workers' guesses could lead to wrong results. Reviewing and acknowledging workers' guesses would require requesters to be available until all tasks are completed.

Revolt (Chang, Amershi, and Kamar 2017), a collaborative system that addresses unclear or under-specified instructions for image-labeling tasks. Revolt strives to improve the quality of results without improving the instruction quality. This is done by allowing multiple workers to label an image with specific instructions. If there is a mismatch in the labeling, workers relabel the image based on a description provided by other workers.

## Task Design by Novice Requesters

To understand the strategies used by novice requesters, an experiment was conducted with 19 participants. Participants were recruited by an email list at a large university. All were students having limited experience with crowdsourcing (Papoutsaki et al. 2015). Students were asked to gather information regarding the academic careers of over $2,000$ professors from $50$ top Computer Science departments in the U.S. using Amazon Mechanical Turk. The authors then classified the strategies used by the students into six categories and compared the results of these strategies against those obtained by workers. The study found that some of the strategies (such as communicating with workers) chosen by the novice requesters were consistent with the recommended best practices.

Fantasktic was designed to help novice requesters (Gutheim and Hartmann 2012) author tasks using three task design techniques: a guided task specification interface, a preview interface, and a worker tutorial. The guided task specification interface provides task-specific guidelines and recommendations to the requester as they create the task. The preview interface displays the task as it would be visible to a worker. The worker tutorial is then automatically generated based on the sample answers provided by the requester. Workers' response quality showed significant improvement when the requester utilized the suggestions made by the guided task specification interface during task creation. However, task previews and worker tutorials had no noticeable impact on workers' response quality. Despite the clear benefit, guided task specification does not address instruction ambiguities. It is also impossible to create guided interfaces for every possible task that could be assigned on crowd platforms.

## Factors Affecting the Quality of Results

Clarity of instructions is not the only factor that can influence the quality of the results.

Requesters' choice of incentive amount and structure can also influence workers' performance (Kazai, Kamps, and Milic-Frayling 2013; Finnerty et al. 2013).

Workers' individual characteristics affect result quality by way of cognitive factors (Alagarai Sampath, Rajeshuni, and Indurkhya 2014), attention (Göritz, Borchert, and Hirth 2021; Rothwell et al. 2016), mood (Gadiraju and Demartini 2019; Zhuang and Gadiraju 2019; Xu, Zhou, and Gadiraju 2019; Qiu, Gadiraju, and Bozzon 2020), familiarity of the task (Kazai, Kamps, and Milic-Frayling 2013), task-dependent bias or errors (Kamar, Kapoor, and Horvitz 2015), payment (Kazai, Kamps, and Milic-Frayling 2013; Finnerty et al. 2013), perceived task difficulty (Kazai, Kamps, and Milic-Frayling 2013), busyness, fatigue and presence of companions (Ikeda and Hoashi 2017), worker reading/ not reading instructions (Rothwell et al. 2016), cognitive Biases (Eickhoff 2018; Saab et al. 2019; Hube, Fetahu, and Gadiraju 2019).

## TaskLint

We present TaskLint, a system for automatically detecting ambiguities during the task authoring process.

### Process

For requesters, using TaskLint is a four-stage process.

**Stage 1: Design Task Template** Requester uses a web-based rich text editor to create a draft of their task description. Parameters are specified by entering a text placeholder. For example, in a task that engages workers to find and enter information about courses, the requester would enter placeholders like "{{semester}}" and "{{university}}".

**Stage 2: Upload Input Values** Requester enters or uploads a text file containing values for the parameters, for each instance of the task to be created. The file must be formatted as tab-separated text (TSV). For example, in the course information example, the text file could contain a list of semesters and university names. The system will post one task for each row in this file.

**Stage 3: Preview Tasks** TaskLint combines the template (Stage 1) and input values (Stage 2) to create a set of task descriptions. Each row of the table of input values results in one task (e.g., find a HCI course offered in spring 2023 at MIT). Requester views the task previews.

**Stage 4: Respond to Feedback** Requesters will see the ambiguities in the task design and can modify the task design based on the feedback provided by the tool (Figure 1).

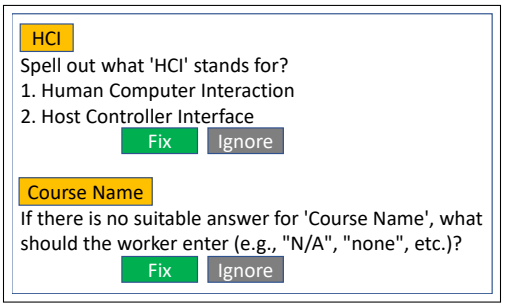


Figure 1: Analyze

## Ambiguity Types

The emphasis is on ambiguity types that might adversely affect the quality of results obtained from workers. This is a subset of the ambiguities described in *Related Work*.

The feedback messages provided by TaskLint are organized according to a taxonomy of 25 ambiguity types specific to task design for crowd work (Manam and Quinn 2018).

This section will explain each of the ambiguity types. Using a diverse set of preexisting modules and data sets (described in *Implementation*), TaskLint is able to detect 19 of the 25 ambiguity types in this taxonomy.

The ambiguity types are organized according to the component of the task design that is affected.

- **Input ambiguities** relate to the parameter values that are inserted in place of the placeholders in the task template to generate the task instances.
- **Process ambiguities** are in the descriptive text in the task template that tells the worker how to perform the task.
- **Output ambiguities** relate to the form fields the worker will enter data into, together with their associate labels. These are the outputs a requester cares about.

### Input Ambiguities

*Input Entity (IE)* ambiguities relate to a named entity inserted into the task template (i.e., in place of a placeholder) when task instances are generated.

**IE1: Abbreviations** Acronyms, even if known, may have multiple possible resolutions.

Example: A task asks workers to find data about courses on specified topics, including "HCI."

TaskLint detects that "HCI" is an acronym with multiple possible referents (human-computer interaction, host controller interface, etc.).

> Spell out what 'HCI' stands for?

**IE2: Vocabulary** Words that are likely to be unfamiliar to workers can cause confusion. Workers may guess the meaning (sometimes incorrectly).

Example: A task asks workers to find specified metrics about specified researchers, including finding the "h-index" for "Don Norman".

TaskLint analyzes the vocabulary level and determines that "h-index" (parameter value for "metric_name" placeholder) is an uncommon word.

> Workers may not know what 'h-index' means. Simplify?

**IE3: Person Name** First and last name may not uniquely identify the person about whom a task instance refers to.

Example: A task asks workers to search for information about specified computer scientists, including "Jimmy Lin."

TaskLint uses entity detection and classification to recognize "Jimmy Lin" as the name of a person. It prompts the requester to consider clarifying, especially if there might be more than one computer scientist by that name.

> Is there only one such person named 'Jimmy Lin'?

**IE4: Location Name** A place name may correspond to multiple geographic places.

Example: A task directs workers to search for information about each specified American city, including "Columbus."

TaskLint recognizes "Columbus" as a place name with multiple possible referents (Columbus, Ohio; Columbus, Indiana; Columbus, Georgia).

> Is that the only location named 'Columbus'?

**IE5: Organization Name** Organization references may refer to multiple real-world entities.

Example: A task asks workers to find data about specified consumer brands, including "Delta".

TaskLint detects "Delta" as a named entity reference for an organization that could refer to Delta Airlines, Delta Faucets, or Delta Dental (insurance company).

| Is that the only organization named 'Delta'? |
|---|

**IE6: Entity Type** Some named entity references can be interpreted relative to more than one entity type (i.e., person, place, organization, etc.), leading to different meanings.

Example: A task asks workers to find the postal code for "Dupont," which could refer to the chemical company (DuPont de Nemours, Inc.) or a city (Dupont, Indiana).

TaskLint uses named entity classification to detect named entity references with multiple possible entity types.

| What is 'Dupont' (e.g., organization, location, etc.)? |
|---|

*Input Syntax (IS)* ambiguities occur when spelling or punctuation mistakes result in multiple interpretations.

**IS1: Possible Misspelling** Misspellings or punctuation errors can force workers to guess what the requester meant. Different guesses can result in inconsistent results.

Example: A task asks workers to search for data about specified companies, including "Sisco." This could mean "Cisco" (technology company) or "Sysco" (food company).

TaskLint detects the misspelling using a spell check engine that includes proper nouns.

| Is the spelling of 'Sisco' correct? |
|---|

*Input Wrong (IW)* ambiguities occur when parameter values include an entity that exists and is not ambiguous, but is not what the requester intended.

**IW1: Wrong Entity** If an entity name is unambiguous, but not what the requester intended—and does not make sense—workers may try to guess what was intended.

Example: A task asks workers to find the largest university in a specified American city, including "Lafayette, Indiana." Lafayette, Indiana is a real city, but does not contain any universities. Did the requester mean "West Lafayette, Indiana" (with Purdue University) or "Lafayette, Louisiana" (with the University of Louisiana at Lafayette)?

TaskLint is currently unable to detect these ambiguities.

**IW2: Invalid Combination** If the requester combines two (or more) valid, unambiguous entity names in a way that does not make sense, workers may try to guess.

Example: A task asks workers to find the weight of the latest "Samsung iPhone." While Samsung is a valid electronics brand and the iPhone is a valid product line, Samsung does not produce the iPhone. Thus, workers might guess that the requester intended either Apple iPhone or Samsung Galaxy.

TaskLint is currently unable to detect these ambiguities.

*Input Units (IU)* ambiguities occur when the units for parameter values are missing.

**IU1: Missing Units** When parameter values include numbers without units, workers may make assumptions.

Example: A task asks workers to find a city in South America with an average temperature of "30 to 35" in May.

TaskLint detects quantities 30 and 35 as missing units.

| What are the units of '30' and '35'? |
|---|

*Input Format (IF) ambiguities occur when parameter values leaves room for multiple interpretations.*

**IF1: Date Format** A given date string may be interpreted differently depending on the locale.

Example: A task asks workers to find the name of a championship-winning sports player who was born on a specified day, including "$3/5/1980$."

TaskLint detects that $3/5/1980$ is a date that could be interpreted as either month/day/year or day/month/year.

| By what format should 3/5/1980 be interpreted? |
|---|

**IF2: Name of Person** Depending on the locale, a person's full name could be ordered first-last or last-first.

Example: A task asks workers to find a movie in which each actor appeared, including "Michael Blake." This could be interpreted to mean Michael (given name) Blake (surname), or Blake (given name) Michael (surname). There are movie actors with both names.

TaskLint recognizes "Michael Blake" as a person's name.

| Is "Michael Blake" in (first, last) or (last, first) order? |
|---|

## Process Ambiguities

*Process Steps (PS)* ambiguities occur when the task does not specify *how* to perform the task.

**PS1: Unspecified Steps** If the requester does not specify *how* to perform part of a task that requires some skill or process that workers are unfamiliar with, workers may attempt to perform it in the wrong way, leading to inconsistency.

Example: A task asks workers to shorten a paragraph of text that is presumed to be verbose, down to a specified word count. The task does not specify whether to reword the sentence to be more verbose, or simply remove words.

TaskLint is unable to detect this ambiguity type.

*Process Wrong (PW)* ambiguities occur when steps are specified but wrong and may not lead to the intended result.

**PW1: Incorrect Steps** Even if the steps have been fully specified, any incorrect information within them can confuse workers, leading to results that are incorrect or inconsistent.

Example: A task asks workers to manipulate spreadsheets using Microsoft Excel and directs them to use the "PivotFilter" feature, but no such feature exists in that application.

TaskLint is unable to detect this ambiguity type. Inferring whether or not steps are likely to be correct requires deep knowledge and reasoning about the world, and the ability to imagine what would happen if the steps were followed. We are unaware of any AI solutions that can do this to the level of generality that would be needed to detect Incorrect Steps (PW1) ambiguities for TaskLint.

*Process wOrds (PO)* ambiguities occur when the vocabulary is unfamiliar or has multiple meanings.

**PO1: Unquantified Quantities** Imprecisely worded quantities (e.g, *as lot*, *nearby*, etc.) inherently require interpretation. Since different workers may interpret such terms differently, results may be inconsistent or just wrong.

Example: A task asks workers to read articles about the music industry and annotate the names of artist who have sold "a lot" of records.

TaskLint uses uncertainty detection (Vincze 2014) to detect all vaguely worded quantities in the task description.

> Can you be more specific than 'a lot'?

**PO2: Specialized Vocabulary, PO3: Loaded Terms** When the process description uses specialized vocabulary that many workers would not know (PO2) or words with homonyms that would also make sense in the context (PO3), workers may spend time looking up the terms or trying to infer which meaning was intended. Gadiraju et al found that $18\%$ of workers have used dictionaries or other tools to understand over $50\%$ of the tasks they completed (Gadiraju, Yang, and Bozzon 2017).

Example: A task asks workers to find sources for purchasing cookies made with "sustainable organic palm oil."

TaskLint recognizes "sustainable organic" as a term that may not be understood by some workers.

> Workers may not know what 'sustainable organic' means. Simplify or define?

*Process Contradiction (PC)* ambiguities occur when the description of steps contains contradictory information.

**PC1: Contradictory Steps** When instructions contain contradictory information, workers are left to guess or try to reconcile the contradiction.

Example: A task asks workers to view images of receipts and annotate entries for purchases of fruits and vegetables. Later, the same instructions state, "Do not annotate fruit." Some workers might guess that both fruits and vegetables are desired (and the later statement is incorrect), while others might opt to omit the fruit.

TaskLint detects and reports the contradiction.

> The following may be contradictory: "In the receipt below, annotate each entry for purchases of fruits and vegetables." and "Do not annotate fruit."

## Output Ambiguities

*Output Entity (OE)* ambiguities occur when an entry calls for an entity, but there are multiple possibilities.

**OE1: Multiple Entities** By following the task description, workers might end up with more than one possible result to enter. Having more than one possible result completely depends on the tasks. We cannot predict whether or not a given task has more than one possible result. Hence, TaskLint is unable to detect this ambiguity type.

Example: A task asks workers to name the department where discrete math, theory of computation, and machine learning are taught at each specified university.

TaskLint cannot detect this type of ambiguity because it cannot infer the association between separate references to the same entry field.

*Output eXception (OX)* ambiguities occur when there is no valid answer for some task instances.

**OX1: No Answer** If, for some inputs (parameter values), there is no valid entry (answer for a worker to enter), the requester should specify what the worker should do. Otherwise, workers may guess.

Example: A task asks workers to enter the first adjective found in a given sentence.

TaskLint detects that no instructions were given for what to do if it contains no adjectives (e.g., "Eat your kale.").

> Specify what to do if there is no "first adjective."

*Output Units (OU)* ambiguities relate to the units with which an entry should be expressed.

**OU1: Units Unspecified** When an entry form field requires a quantity, but units are unspecified, workers might follow their locale or guess based on context of the task.

Example: A task instructs workers to find the height of each specified building but does not specify units (e.g., feet).

TaskLint detects that a quantity is called for, and that no units are mentioned.

> In what units should *height* be entered?

*Output Format (OF)* ambiguities occur when there are multiple possible ways to format output entry.

**OF1: Time Format** When the requester asks workers to submit any information related to the time and does not specify the format in which to submit the time information, it will be ambiguous for the workers to predict the correct format of the time that the requester is asking for.

Example: A task directs the worker to find the time when Neil Armstrong landed on the moon, but does not specify which format the time should be expressed with.

TaskLint detects that an entry field refers to a time and does not include such a specification.

> In what format should 'time' be expressed?

**OF2: Phone Number Format** When a task calls for a phone number to be entered, if the format of the phone number is not specified, differences in country code or area code can lead to ambiguities that would make it impossible for someone to call the number.

Example: A task directs the worker to search for the direct office phone number of each named marketing manager from various companies around the world, but does not specify in what format it should be entered.

TaskLint detects that a phone number is called for and that no format has been specified.

| In what format should 'phone number' be expressed? |
|---|

*Output Precision (OP)* ambiguities occur when asking for a quantity, task does not specify how precise the answer should be.

**OP1: Precision** In the task description, the requester may use some words, such as nearby, closely, faraway, etc., to describe quantities, and workers might not understand how to interpret these quantities precisely. For example, if a user asks a worker to find a movie theater close to Chicago (ORD) airport terminal 5, workers do not know whether 'close' refers to the walking distance or driving distance. To detect this, we parse the task description and check if the requester has used any hedge words (Wormer 2018) and show a warning message to the user:

| How precise should 'close' be? |
|---|

*Output Wrong (OW)* ambiguities occur when entity requested is not what requester intended.

**OW1: Information Requested Conflicts with the Form Fields** If the descriptive text referring to a form field conflicts with the label next to that form field, workers might choose one or the other arbitrarily.

Example: A task directs workers to search for the weight of a Lenovo G50 laptop, but then provides a form field labeled "screen size."

TaskLint is unable to detect this ambiguity type.

## Implementation

TaskLint is implemented as a web application using the Python Flask framework (Grinberg 2018). The rich text editor used by requesters to author draft task templates is based on the TinyMCE rich text editor component. Placeholders for task parameters are entered as plain text (e.g, "{{city_name}}").

To analyze a task design, TaskLint parses the HTML to extract text, as well as the labels and structure of the input form fields (i.e., text entries, check boxes, etc.). Together with the parameter values entered separately by the requester, a task design comprises three components: (1) explanatory text, (2) form fields, and (3) parameter values.

**NLP Tools Used** Our current implementation detects 19 of the 25 ambiguity types in our taxonomy using 12 tools.

- Regular expressions – OX1, OF1, OF2
- Acronym DB (Acronym DB 2008) detects acronyms and suggests possible resolutions for IE1.
- Bing Spell Check (Microsoft 2017) detects possible misspellings for IS1.
- Azure Text Analytics (Microsoft 2014) detects dates, names, and phone numbers for IF1 and IF2.
- GeoNames (Geonames 2020) detects potentially ambiguous place names for IE4 and IE5.
- Hedge word detection (Wormer 2018) detects imprecise adjectives and adverbs for OP1.
- Numeric Fused-Head (NFH) (Elazar and Goldberg 2019) detects quantities without specified units for IU1.
- Quantulum3 (Mündler 2018) detects units for OU1.
- RoBERTa (Liu et al. 2019) – detects textual contradictions for PC1.
- Spacy NER (Honnibal and Montani 2017) – named entity recognition and classification; for IE3, IE6
- Textstat (Bansal and Aggarwal 2020) – for IE2, PO2, PO3
- Uncertainty detection (Vincze 2014) – for PO1

Each tool is invoked via a module, which runs in parallel to ensure a responsive experience for requesters. The output of each is interpreted and used to generate feedback messages that are designed to be understandable by requesters. The messages refer to the specific text or entity in the task design where the potential ambiguity was found.

# Study Design

We conducted an evaluation to test whether providing automated feedback using TaskLint can improve outcomes versus creating and posting task design feedback.

The study had two stages. In Phase 1, a novice requester created instructions for several task scenarios (naïve), used our tool to detect ambiguities in the task instructions, and modified the instructions based on the feedback from the tool (TaskLint). In Phase 2, we posted their instructions to Amazon Mechanical Turk (AMT) and measured the performance of tasks created (naïve, TaskLint). These phases will be described in more detail under *Method* below.

## Research Questions and Hypotheses

Our study sought to answer two central research questions, each of which gives rise to a set of hypotheses.

**RQ1:** Can TaskLint improve performance?

- **H1** Accuracy of results will be higher for TaskLint than Naïve.
- **H2** Workers' completion time will be less for TaskLint.

**RQ2:** Can TaskLint detect nocuous ambiguities?

- **H3** Requesters perceive Tasklint feedback (overall) as "useful".
- **H4** Requesters perceive Tasklint feedback for Input ambiguities (overall) as "useful".
- **H5** Requesters perceive Tasklint feedback for Process ambiguities (overall) as "useful".
- **H6** Requesters perceive Tasklint feedback for Output ambiguities (overall) as "useful".

## Method

**Phase 1: Requesters Design Task in Online Settings** Participants were recruited via emails and invited for an online 2-hour experiment in which they posed as novice requesters and designed tasks with the aid of TaskLint.

Each participant received a $20 gift card to compensate for their time. The study was approved by the Institutional Research Board at Purdue University. All participants signed a consent form.

There were 20 participants having no prior experience with microtask task design. Each completed a questionnaire

about their background. No participants had prior experience designing crowd tasks or working in crowd work markets. Nineteen (95%) reported experience designing fewer than 10 written instructions of other types. One participant (5%) reported having experience in designing more than 10 instructions.

**Procedure** Participants authored tasks using a web-based editor based on the TinyMCE[2] rich text editor component. We customized the editor to support the creation of HTML forms containing input fields, including text fields, multiline text entry, checkboxes, and radio buttons.

Before creating any tasks, each participant viewed a video tutorial about how to author task designs using our task design editor. To confirm their understanding, they created a sample task design.

Each participant created three task designs, each directing workers to search for information on the web in support of three fictional scenarios that we provided: (1) mobile phone shopping, (2) Turing award winners, and (3) professors.

For each scenario, we provided a brief description of the situation in which the information might be needed, and a document with the desired output (i.e., the *answers*). The participant then assumed the role of a novice requester and authored a task design directing workers to find that information. Participants were instructed to write the instructions so that workers' entries (output) would match the desired output that we had provided.

**Rationale for *Jeopardy* Evaluation Method** Our evaluation method is unusual—and unlike real-life task design use cases—because instead of starting with an objective and designing a task to elicit the needed data, the participant starts out with the end result they are trying to get to, and then works backward, designing a task that they believe should elicit the same data.

We call this method the *Jeopardy* method, because like the American TV game show by that name, the participants are given the answers and are tasked with providing questions that might lead to those answers.

While perhaps counter-intuitive, the Jeopardy method allows us to *implicitly* communicate to the participant the requirements—i.e., what sort of data workers should enter—in a form that would not bias the words the participants used in their task design.

In contrast, a naïve evaluation method would have been to simply tell the participant what kind of information the worker should enter, and how to get it. For example, we could tell the participant, "Write instructions directing the worker to search on the ACM website for the name of each Turing award winner and enter the year of the award and summary of achievements in your input form." However, that would bias the participant's task design. They might write instructions like, "Please search on the ACM website for the name of each Turing award winner and enter the year of the award and a summary of achievements in the form below." There would be little opportunity for organically occurring ambiguities in the participant's task design, thus limiting our ability to evaluate the effectiveness of TaskLint to help detect such ambiguities.

The Jeopardy method also establishes an objective means of assessing the correctness of the workers' entries (output). To measure correctness, we can simply compare the text of the workers' entries with the desired output we provided to the participants who authored the task designs.

Naïve approaches to assessing the accuracy of workers' entries (output) would have been to assess ourselves or have the participant come back and assess the accuracy. Unless we had established a comprehensive specification of correctness in advance that both the participant and researchers fully understood, we would not be in a position to assess correctness. Correctness is determined not by whether the workers followed the instructions, but whether they did *what the requester intended*. Even if we had the participant (requester) come back and evaluate correctness, their evaluation would be arbitrary, since the scenario was fictional. Real-life accuracy depends on whether the data meet the needs of the situation. In a fictional scenario, that cannot be determined.

**Naïve Condition** The initial version of each design was saved for use in our evaluation. It represents the state of the task design without the benefit of the feedback from TaskLint. We call this version the *naïve condition* for purposes of our analysis.

**TaskLint Condition** After creating instructions for all three tasks, participants viewed feedback from TaskLint about each task design and used the feedback to revise each one. Revisions were made after all of the tasks had been authored to avoid learning effects (i.e., learning to avoid making the kinds of ambiguities that TaskLint reports).

Before using TaskLint, each participant viewed a tutorial video about how to use it. Then, for each task design, TaskLint displayed a list of possible ambiguities. The participant reported each feedback item as helpful or not helpful using a button in the interface. This is analogous to the use of common spelling and grammar checkers, which prompt users to fix or ignore each suggestion, though TaskLint does not automatically modify the task design. As participants annotated the feedback items, they revised the task design using the same interface with which they authored it. We call the resulting task design—after revising based on feedback from TaskLint—the *TaskLint condition* for purposes of our analysis.

**Phase 2: Post Tasks to MTurk** In the second part of the study, we posted these instructions to MTurk and collected the results from workers.

As described above, there were two experimental conditions: the naïve condition (before utilizing TaskLint feedback) and the TaskLint condition (after utilizing the feedback). Each worker received task designs from either the naïve condition or the TaskLint condition—but not both.

A total of 439 distinct workers participated. Participation was restricted to workers with a minimum approval rate of 90%. There were no other qualification criteria.

[2]https://www.tiny.cloud/

**Payment** We initially paid $0.30 USD per completed HIT. That resulted in an effective hourly rate of $5.78 per hour, which is below the US federal minimum wage ($7.25). To raise the effective hourly rate to $8.00 per hour, we paid an additional $0.12 per HIT as a bonus to workers.

## Results

In Phase 1, the 20 novice requester participants created a total of 60 task designs (three task designs per participant). TaskLint generated 330 reports of possible ambiguities, of which the participants labeled 154 as useful. Table 1 shows the further classification of these ambiguity detections corresponding to each ambiguity type.

In Phase 2, the 439 workers on Mechanical Turk performed the tasks, including the Naïve and TaskLint-improved versions.

### H1: Accuracy of Results Will Be Higher for TaskLint than Naïve

We compare workers' results with the results (ground truth) provided to the participants and label them as correct or incorrect accordingly. To understand whether TaskLint performs better in terms of accuracy compared to Naïve, we conducted a chi-square test for the results of the task with TaskLint and Naïve. There was a significant association between the TaskLint and the accuracy of results, $\chi^2(1) = 214.00$, $p < 0.001$. Based on the odds ratio, TaskLint produced 1.58 times more accurate results compared with Naïve. In other words, modifications to the original task instructions based on the feedback from TaskLint had a positive impact on the final outcome.

### H2: Workers' Completion Time Will Be Less for TaskLint

We hypothesized that facilitating improvements to the instructions would decrease completion time because workers would spend less time trying to resolve ambiguities. Thus, we expected lower completion times with the instructions revised using feedback from TaskLint, versus the Naïve case.

To compare the completion time with Naïve and TaskLint, we performed the Wilcoxon rank sum test. Student's t-test was not used due to non-homogeneous variance.

The Wilcoxon rank sum test showed that there is a significant difference in the completion time with Naïve and TaskLint. The completion time of TaskLint ($Median = 110.72$) was significantly less than Naïve ($Median = 117.73$), $W = 691433, p < 0.05, r = -0.055$.

### H3: Requesters Perceive Tasklint Feedback (Overall) as "Useful"

Recall that TaskLint prompts requesters to judge whether the feedback provided by the tool is useful or not. In our experiment, our tool detected a total of 330 problems with all the task instructions designed by 20 participants. Of the 330 problems detected, 154 were labeled as "useful" by the participants, and 176 were labeled as "not useful".

Table 1 shows further classification of these ambiguities detection corresponding to each ambiguity type. Ambiguities not detected in the study are omitted from that table.

To understand whether TaskLint's overall feedback is useful or not, we conducted a chi-square test. There was no significant association between the TaskLint feedback and the "usefulness", $\chi^2(1) = 1.47, p > 0.05$.

| Ambiguity | | Detected | Useful | Not useful | Chi-squared test |
|---|---|---|---|---|---|
| **Input Entity (IE)** | 1 | 24 | 15 | 9 | $\chi^2(1) = 1.5, p > 0.05$ |
| | 3 | 41 | 11 | 30 | $\chi^2(1) = 8.80, p < 0.01$ |
| | 4 | 12 | 1 | 11 | $\chi^2(1) = 8.33, p < 0.01$ |
| | 5 | 32 | 18 | 14 | $\chi^2(1) = 0.5, p > 0.05$ |
| | 6 | 44 | 17 | 27 | $\chi^2(1) = 2.27, p > 0.05$ |
| **Input Format (IF)** | 2 | 40 | 23 | 17 | $\chi^2(1) = 0.9, p > 0.05$ |
| **Input Syntax (IS)** | 1 | 20 | 10 | 10 | $\chi^2(1) = 0, p > 0.05$ |
| **Process wOrds (PO)** | 2 | 4 | 2 | 2 | $\chi^2(1) = 0, p > 0.05$ |
| **Process Contradiction (PC)** | 1 | 14 | 1 | 13 | $\chi^2(1) = 10.27, p < 0.05$ |
| **Output eXception (OX)** | 1 | 76 | 41 | 35 | $\chi^2(1) = 0.47, p > 0.05$ |
| **Output Units (OU)** | 1 | 15 | 12 | 3 | $\chi^2(1) = 5.4, p < 0.05$ |
| **Output Format (OF)** | 1 | 4 | 2 | 2 | $\chi^2(1) = 0, p > 0.05$ |
| **Output Precision (OP)** | 1 | 4 | 1 | 3 | $\chi^2(1) = 1, p > 0.05$ |

Table 1: Ambiguities detected

### H4: Requesters Perceive Tasklint Feedback for Input Ambiguities (Overall) as "Useful"

To understand whether TaskLint feedback for Input ambiguities is useful or not, we conducted a chi-square test. There was no significant association between the TaskLint feedback and the "usefulness", $\chi^2(1) = 2.48, p > 0.05$.

### H5: Requesters Perceive Tasklint Feedback for Process Ambiguities (Overall) as "Useful"

To understand whether TaskLint feedback for Process ambiguities is useful or not, we conducted a chi-square test. There was a significant association between the TaskLint feedback and the "usefulness", $\chi^2(1) = 8, p < 0.05$. Based on the odds ratio, TaskLint feedback for Process ambiguities was five times more not useful (3 Useful vs. 15 Not useful).

### H6: Requesters Perceive Tasklint Feedback for Output Ambiguities (Overall) as "Useful"

To understand whether TaskLint feedback for Output ambiguities is useful or not, we conducted a chi-square test. There was no significant association between the TaskLint feedback and the "usefulness", $\chi^2(1) = 1.71, p > 0.05$.

### Observations and Additional Analysis

The following are some of the behaviors we observed from study participants' interactions with TaskLint.

- Participants responded to or dismissed every message presented by TaskLint, top to bottom, before clicking the check button again to check for new ambiguities.
- When a certain ambiguity type was detected (e.g., units missing), participants did not necessarily correct all related ambiguities in their task instructions.
- Participants reported TaskLint feedback for Input Entity IE3 ($\geq$ 2 instances by that name of person), IE4 ($\geq$ 2 instances by that name of location), and Process Contradiction PC1 (Contradicting Steps) as "not useful" and Output Units OU1 (Units not specified in the entry form) as "useful". We conducted a chi-square test (Table 1). There was a significant association between the TaskLint feedback for IE3, IE4, PC1, OU1, and the "usefulness".

## Discussion and Future Work

TaskLint is able to detect 19 of 25 ambiguity types described in this paper. Our implementation leverages several current-generation NLP tools to detect a varied set of ambiguity types in task instructions.

Results of our study demonstrated that TaskLint can detect problems with the instructions and further help requesters improve the quality of the instructions, thereby significantly enhancing the accuracy of the results.

### Challenge of Assessing Accuracy

When faced with ambiguities in a task, humans may intuit what the requester was expecting based on context, common sense, and prior experience with similar tasks. When we post tasks with ambiguities, we may get the correct result even without using any mechanism to disambiguate. Hence, it is very difficult to show that any new method to disambiguate will produce better accuracy of results significantly.

For a wrong result, there are two ways to analyze it: "foresight" and "hindsight" (Norman 2013). Foresight analysis means predicting the reasons that can lead to a wrong result in advance. Hindsight analysis will find the reason for the wrong result after we get the results.

With this effort, we have tested whether a lint tool for task instructions could detect some types of ambiguities. Our current implementation is a prototype, sufficient for evaluation, but it does not support the rest of the workflows needed for real-world deployment. In the future, we envision TaskLint being integrated as a preprocessing step into crowd work sites, or combined with other novel workflows from research (e.g., Daemo's Prototype tasks (Gaikwad et al. 2017), WingIt (Manam and Quinn 2018)) to improve the efficiency of ambiguity detection.

### Limitations

The performance of TaskLint depends on the accuracy of the NLP tools we use in detecting ambiguities. Most of the NLP tools we used can detect ambiguities occurring only at the word level or at the sentence level. Our tool cannot detect ambiguities that would require exogenous information or contextual information. Currently, our tool cannot detect unspecified steps or incorrect steps. To detect these, we may require domain-specific knowledge. In the future, we plan to use domain-specific knowledge to detect these ambiguities.

## Conclusion

This paper introduced TaskLint, a lint-like tool for helping requesters clarify task instructions by identifying ambiguities. We described the design rationale and implementation, and shared results from our evaluation with twenty novice task designers, who each created three different tasks.

Our evaluation found that TaskLint feedback can lead to improvements in result quality and the time to obtain results.

## Acknowledgments

We are grateful to the anonymous crowd workers and lab study participants, without whom this work would not be possible. Dan Goldwasser, Matt Lease, Praveen Paritosh, Chris Welty, Lora Aroyo, and Ujwal Gadiraju have all shared their expertise at key junctions of this work.

## References

Acronym DB. 2008. Acronym DB. www.acronymdb.com. [Online; accessed June 7, 2022].

Alagarai Sampath, H.; Rajeshuni, R.; and Indurkhya, B. 2014. Cognitively inspired task design to improve user performance on crowdsourcing platforms. In *CHI '14: Proceedings of the 32nd CHI Conference on Human Factors in Computing Systems*, 3665–3674. New York, NY, USA: ACM.

Allen, J. 1995. *Natural language understanding*. New York, NY, USA: Pearson.

Alonso, O.; and Baeza-Yates, R. 2011. Design and implementation of relevance assessments using crowdsourcing. In *ECIR '11: Proceedings of the 33rd European Conference on Information Retrieval*, 153–164. New York, NY, USA: Springer.

Bansal, S.; and Aggarwal, C. 2020. Textstat. https://pypi.org/project/textstat/. [Online; accessed June 7, 2022].

Berg, J. 2016. Income security in the on-demand economy: findings and policy lessons from a survey of crowdworkers. *Comparative Labor Law and Policy Journal*, 37(3).

Bragg, J.; and Weld, D. S. 2018. Sprout: Crowd-Powered Task Design for Crowdsourcing. In *UIST '18: Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 165–176. New York, NY, USA: ACM.

Chang, J. C.; Amershi, S.; and Kamar, E. 2017. Revolt: Collaborative crowdsourcing for labeling machine learning datasets. In *CHI '17: Proceedings of the 35th CHI Conference on Human Factors in Computing Systems*, 2334–2346. New York, NY, USA: ACM.

Eickhoff, C. 2018. Cognitive biases in crowdsourcing. In *WSDM '18: Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, 162–170. New York, NY, USA: ACM.

Elazar, Y.; and Goldberg, Y. 2019. Where's my head? Definition, data set, and models for numeric fused-head identification and resolution. *Transactions of the Association for Computational Linguistics*, 7: 519–535.

Finnerty, A.; Kucherbaev, P.; Tranquillini, S.; and Convertino, G. 2013. Keep it simple: Reward and task design in crowdsourcing. In *CHItaly '13: Proceedings of the Biannual Conference of the Italian Chapter of SIGCHI*, 1–4. New York, NY, USA: ACM.

Fowler Jr, F. J. 1992. How unclear terms affect survey data. *Public Opinion Quarterly*, 56(2): 218–231.

Gadiraju, U.; and Demartini, G. 2019. Understanding worker moods and reactions to rejection in crowdsourcing. In *HT '19: Proceedings of the 30th ACM Conference on Hypertext and Social Media*, 211–220. New York, NY, USA: ACM.

Gadiraju, U.; Yang, J.; and Bozzon, A. 2017. Clarity is a Worthwhile Quality: On the Role of Task Clarity in Microtask Crowdsourcing. In *HT '17: Proceedings of the 28th ACM Conference on Hypertext and Social Media*, 5–14. New York, NY, USA: ACM.

Gaikwad, S.; Chhibber, N.; Sehgal, V.; Ballav, A.; Mullings, C.; Nasser, A.; Richmond-Fuller, A.; Gilbee, A.; Gamage, D.; Whiting, M.; et al. 2017. Prototype Tasks: Improving Crowdsourcing Results through Rapid, Iterative Task Design. *arXiv preprint*, arXiv:1707.05645.

Gaikwad, S. N. S.; Whiting, M. E.; et al. 2017. The Daemo Crowdsourcing Marketplace. In *CSCW '17: Proceedings of the 20th ACM Conference on Computer Supported Cooperative Work and Social Computing*, 1–4. New York, NY, USA: ACM.

Geonames. 2020. Geonames. http://www.geonames.org. [Online; accessed June 7, 2022].

Gleich, B.; Creighton, O.; and Kof, L. 2010. Ambiguity detection: Towards a tool explaining ambiguity sources. In *REFSQ '10: Proceedings of the 16th International Working Conference on Requirements Engineering: Foundation for Software Quality*, 218–232. New York, NY, USA: Springer.

Göritz, A. S.; Borchert, K.; and Hirth, M. 2021. Using attention testing to select crowdsourced workers and research participants. *Social Science Computer Review*, 39(1): 84–104.

Grady, C.; and Lease, M. 2010. Crowdsourcing Document Relevance Assessment with Mechanical Turk. In *CSLDAMT '10: Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 172–179. Stroudsburg, PA, USA: ACL.

Grammarly. 2015. Grammarly. www.grammarly.com. [Online; accessed June 7, 2022].

Grinberg, M. 2018. *Flask web development: developing web applications with python*. O'Reilly Media, Inc.

Growth Market Reports. 2022. AI Writing Assistant Software Market – Global Industry Analysis, Size, Share, Growth, Trends, and Forecast. https://growthmarketreports.com/report/ai-writing-assistant-software-market-global-industry-analysis. [Online; accessed June 7, 2022].

Gutheim, P.; and Hartmann, B. 2012. *Fantasktic: Improving Quality of Results for Novice Crowdsourcing Users*. Master's thesis, EECS Department, University of California, Berkeley. [Online; accessed June 7, 2022].

Hausser, R. 2014. *Foundations of computational linguistics, third edition*. Springer.

Hirst, G. 1992. *Semantic interpretation and the resolution of ambiguity*. Cambridge, UK: Cambridge University Press.

Honnibal, M.; and Montani, I. 2017. spacy 2: Natural language understanding with bloom embeddings. *convolutional neural networks and incremental parsing*, 7(1): 411–420.

Hube, C.; Fetahu, B.; and Gadiraju, U. 2019. Understanding and mitigating worker biases in the crowdsourced collection of subjective judgments. In *CHI '19: Proceedings of the 37th CHI Conference on Human Factors in Computing Systems*, 1–12. New York, NY, USA: ACM.

Ikeda, K.; and Hoashi, K. 2017. Crowdsourcing go: Effect of worker situation on mobile crowdsourcing performance. In *CHI '17: Proceedings of the 35th CHI Conference on Human Factors in Computing Systems*, 1142–1153. New York, NY, USA: ACM.

Ikeda, K.; Morishima, A.; Rahman, H.; Roy, S. B.; Thirumuruganathan, S.; Amer-Yahia, S.; and Das, G. 2016. Collaborative Crowdsourcing with Crowd4U. *Proceedings of the VLDB Endowment*, 9(13): 1497–1500.

INK. 2016. INK. www.inkforall.com. [Online; accessed June 7, 2022].

Jasper. 2012. Jasper. www.jasper.ai. [Online; accessed June 7, 2022].

Kamar, E.; Kapoor, A.; and Horvitz, E. 2015. Identifying and accounting for task-dependent bias in crowdsourcing. In *HCOMP '15: Proceedings of the 3rd AAAI Conference*

*on Human Computation and Crowdsourcing*, 92–101. Palo Alto, CA, USA: AAAI Press.

Kazai, G.; Kamps, J.; and Milic-Frayling, N. 2013. An analysis of human factors and label accuracy in crowdsourcing relevance judgments. *Information retrieval*, 16(2): 138–178.

Khezri, R. 2017. *Automated Detection of Syntactic Ambiguity Using Shallow Parsing and Web Data*. Master's thesis, Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg. [Online; accessed June 7, 2022].

Kittur, A.; Chi, E. H.; and Suh, B. 2008. Crowdsourcing User Studies with Mechanical Turk. In *CHI '08: Proceedings of the 26th CHI Conference on Human Factors in Computing Systems*, 453–456. New York, NY, USA: ACM.

Kittur, A.; Smus, B.; Khamkar, S.; and Kraut, R. E. 2011. CrowdForge: Crowdsourcing Complex Work. In *UIST '11: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 43–52. New York, NY, USA: ACM.

Kulkarni, A.; Can, M.; and Hartmann, B. 2012. Collaboratively Crowdsourcing Workflows with Turkomatic. In *CSCW '12: Proceedings of the 15th ACM Conference on Computer Supported Cooperative Work*, 1003–1012. New York, NY, USA: ACM.

Levin, J. A.; Boruta, M. J.; and Vasconcellos, M. T. 1983. Microcomputer-based environments for writing: A writer's assistant. *Classroom computers and cognitive science*, 219–232.

Levinson, S. C. 1983. *Pragmatics*. Cambridge, UK: Cambridge University Press.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint*, arXiv:1907.11692.

Lyons, J. 1977. Semantics I & II. Cambridge.

Mallery, J. C. 1988. *Thinking about foreign policy: Finding an appropriate role for artificially intelligent computers*. Master's thesis, Political Science Department, MIT.

Manam, V. C.; Jampani, D.; Zaim, M.; Wu, M.-H.; and J. Quinn, A. 2019. TaskMate: A Mechanism to Improve the Quality of Instructions in Crowdsourcing. In *WWW '19: Companion Proceedings of the 30th World Wide Web Conference*, 1121–1130. New York, NY, USA: ACM.

Manam, V. C.; and Quinn, A. J. 2018. WingIt: Efficient Refinement of Unclear Task Instructions. In *HCOMP '18: Proceedings of the 6th AAAI Conference on Human Computation and Crowdsourcing*, 108–116. Palo Alto, CA, USA: AAAI Press.

Marshall, C. C.; and Shipman, F. M. 2013. Experiences Surveying the Crowd: Reflections on Methods, Participation, and Reliability. In *WebSci '13: Proceedings of the 5th Annual ACM Web Science Conference*, 234–243. New York, NY, USA: ACM.

Meyers, B. 2017. LUCI: Linguistic Uncertainty Classifier Interface. https://github.com/meyersbs/uncertainty. [Online; accessed June 7, 2022].

Microsoft. 2014. Text Analytics API. https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/. [Online; accessed June 7, 2022].

Microsoft. 2017. Bing Spell Check API V7. https://www.microsoft.com/en-us/bing/apis/bing-spell-check-api. [Online; accessed June 7, 2022].

Mündler, N. 2018. Quantulum3. https://github.com/nielstron/quantulum3. [Online; accessed June 7, 2022].

Navigli, R. 2009. Word Sense Disambiguation: A Survey. *ACM Computer Survey*, 41(2): 10:1–10:69.

Norman, D. 2013. *The design of everyday things: Revised and expanded edition*. New York, NY, USA: Basic books.

Papoutsaki, A.; Guo, H.; Metaxa-Kakavouli, D.; Gramazio, C.; Rasley, J.; Xie, W.; Wang, G.; and Huang, J. 2015. Crowdsourcing from scratch: A pragmatic experiment in data collection by novice requesters. In *HCOMP '15: Proceedings of the 3rd AAAI Conference on Human Computation and Crowdsourcing*, 140–149. Palo Alto, CA, USA: AAAI Press.

Peng, Y.; Wang, X.; Lu, L.; Bagheri, M.; Summers, R.; and Lu, Z. 2018. Negbio: a high-performance tool for negation and uncertainty detection in radiology reports. *AMIA Summits on Translational Science Proceedings*, 2018: 188.

Qiu, S.; Gadiraju, U.; and Bozzon, A. 2020. Just the Right Mood for HIT! In *ICWE '20: Proceedings of the 20th International Conference on Web Engineering*, 381–396. New York, NY, USA: Springer.

Rothwell, S.; Carter, S.; Elshenawy, A.; and Braga, D. 2016. Job Complexity and User Attention in Crowdsourcing Microtasks. In *HCOMP '16: Proceedings of the 4th AAAI Conference on Human Computation and Crowdsourcing*, 20–25. Palo Alto, CA, USA: AAAI Press.

Saab, F.; Elhajj, I. H.; Kayssi, A.; and Chehab, A. 2019. Modelling cognitive bias in crowdsourcing systems. *Cognitive Systems Research*, 58: 1–18.

Spoustová, D. J.; Hajič, J.; Votrubec, J.; Krbec, P.; and Květoň, P. 2007. The Best of Two Worlds: Cooperation of Statistical and Rule-Based Taggers for Czech. In *BSNLP '07: Proceedings of the 6th Workshop on Balto-Slavonic Natural Language Processing*, 67–74. Prague, Czech Republic: Association for Computational Linguistics.

Vincze, V. 2014. *Uncertainty Detection in Natural Language Texts*. Ph.D. thesis, School in Computer Science, University of Szeged. [Online; accessed June 7, 2022].

Walton, D. 2013. *Fallacies arising from ambiguity*, volume 1. New York, NY, USA: Springer Science & Business Media.

Wormer, T. 2018. Hedge Words List. https://github.com/words/hedges/blob/main/data.txt. [Online; accessed June 7, 2022].

Wu, M.-H.; and Quinn, A. J. 2017. Confusing the Crowd: Task Instruction Quality on Amazon Mechanical Turk. In *HCOMP '17: Proceedings of the 5th AAAI Conference on Human Computation and Crowdsourcing*, 206–215. Palo Alto, CA, USA: AAAI Press.

Xu, L.; Zhou, X.; and Gadiraju, U. 2019. Revealing the role of user moods in struggling search tasks. In *SIGIR '19: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1249–1252. New York, NY, USA: ACM.

Yang, H.; Willis, A.; De Roeck, A.; and Nuseibeh, B. 2010. Automatic Detection of Nocuous Coordination Ambiguities in Natural Language Requirements. In *ASE '10: Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, 53–62. New York, NY, USA: ACM.

Zhuang, M.; and Gadiraju, U. 2019. In What Mood Are You Today? An Analysis of Crowd Workers' Mood, Performance and Engagement. In *WebSci '18: Proceedings of the 10th ACM Conference on Web Science*, 373–382. New York, NY, USA: ACM.