FROG: A Fast and Reliable Crowdsourcing Framework

Peng Cheng [#], Xiang Lian [*], Xun Jian [#], Lei Chen [#]

[#] *Hong Kong University of Science and Technology, Hong Kong, China*
{pchengaa, leichen}@cse.ust.hk, xjian@connect.ust.hk

[*] *Kent State University, Ohio, USA*
xiang.lian@utrgv.edu

ABSTRACT

For decades, the crowdsourcing has gained much attention from both academia and industry, which outsources a number of tasks to human workers. Typically, existing crowdsourcing platforms include CrowdFlower, *Amazon Mechanical Turk* (AMT), and so on, in which workers can autonomously select tasks to do. However, due to the unreliability of workers or the difficulties of tasks, workers may sometimes finish doing tasks either with incorrect/incomplete answers or with significant time delays, due to the lack of the background knowledge or task complexity/difficulty. While some prior works considered improving the task accuracy through voting or learning methods, they usually did not fully take into account reducing the latency of the task completion. This is especially critical, when a task requester posts a group of tasks (e.g., sentiment analysis), and one can only obtain answers of *all* tasks after the last task is accomplished. As a consequence, the time delay of even one task in this group could delay the next step of the task requester's work from minutes to days, which is quite undesirable for the task requester.

Inspired by the importance of the task accuracy and latency, in this paper, we will propose a novel crowdsourcing framework, namely *Fast and Reliable crOwdsourcinG framework* (FROG), which intelligently assigns tasks to workers, such that the latencies of tasks are reduced and the expected accuracies of tasks are met. Specifically, our FROG framework consists of two important components, *task scheduler* and *notification* modules. For the task scheduler module, we formalize a *FROG task scheduling* (FROG-TS) problem, in which the server actively assigns workers with tasks with high reliability and low latency. We prove that the FROG-TS problem is NP-hard. Thus, we design two heuristic approaches, request-based and batch-based scheduling. For the notification module, we define an *efficient worker notifying* (EWN) problem, which only sends task invitations to those workers with high probabilities of accepting the tasks. To tackle the EWN problem, we propose a *smooth kernel density estimation* approach to estimate the probability that a worker accepts the task invitation. Through extensive experiments, we demonstrate the effectiveness and efficiency of our proposed FROG platform on both real and synthetic data sets.

Table 1: An Example of Tasks in the Crowdsourcing System.

Task	Tweet
t_1	5 Companies Growing Faster Than Apple Inc.
t_2	@apple why doesn't your people explain precautions when buying an expensive laptop.
t_3	This #Yosemite update is stressing me out #macbookpro
t_4	Thank you @Apple
t_5	Thanks @apple for 'Keep Newer' when transferring files from a local hard drive to a server.

1. INTRODUCTION

Nowadays, the crowdsourcing has become a very useful and practical tool to process data in many real-world applications, such as the sentiment analysis [29], image labeling [36], and entity resolution [35]. Specifically, in these applications, we may encounter many tasks (e.g., identifying whether two photos have the same person in them), which may look very simple to humans, but not that trivial for the computer (i.e., being accurately computed by algorithms). Therefore, the crowdsourcing platform is used to outsource these so-called *human intelligent tasks* (HITs) to human workers, which has attracted much attention from both academia [14, 26, 25] and industry [19].

Existing crowdsourcing systems (e.g., CrowdFlower [2] or *Amazon Mechanical Turk* (AMT) [1]) usually wait for autonomous workers to select tasks. As a result, some difficult tasks may be ignored (due to lacking of the domain knowledge) and left with no workers for a long period of time (i.e., with high latency). What is worse, some high-latency (unreliable) workers may hold tasks, but do not accomplish them (or finish them carelessly), which would significantly delay the time (or reduce the quality) of completing tasks. Therefore, it is rather challenging to guarantee high accuracy and low latency of tasks in the crowdsourcing system, in the presence of unreliable and high-latency workers.

Consider an example of the sentiment analysis in real applications of reviews and social media, where a researcher wants to analyze the sentiment of 5 tweets crawled from the Twitter, $t_1 \sim t_5$ (as depicted in Table 1), about the Apple Incorporation. In other words, the researcher needs to label each tweet with an attitude (sentiment) towards the Apple company, such as "positive", "neutral", or "negative". Since no computer algorithms can accurately analyze the sentiment of tweets, the researcher posts these 5 tweets as 5 sentiment analysis tasks, respectively, on a crowdsourcing platform (e.g., AMT [1]).

Example 1 (Accuracy and Latency Problems in the Crowdsourcing System) *In the example of the sentiment analysis above, assume that 4 workers, $w_1 \sim w_4$, from the crowdsourcing system autonomously accept some or all of the 5 tasks, t_i (for $1 \leq i \leq 5$, as given in Table 1), posted by the researcher (task requester).*

Table 2 shows the answers (sentiment labels) and time delays of tasks conducted by workers, w_j ($1 \leq j \leq 4$), where the last column

Table 2: Answers of Tasks from Workers.

Worker	Task	Answer	Time Latency	Correctness
w_1	t_1	Positive	3 min	$\times$
w_1	t_2	Negative	5 min	$\surd$
w_1	t_3	Negative	2 min	$\surd$
w_2	t_1	Negative	8 min	$\times$
w_2	t_2	Positive	**1 day**	$\times$
w_2	t_3	Positive	10 min	$\times$
w_2	t_4	Neutral	9 min	$\times$
w_2	t_5	Neutral	5 min	$\times$
w_3	t_4	Positive	8 min	$\surd$
w_3	t_5	Positive	8 min	$\surd$
w_4	t_3	Negative	3 min	$\surd$

provides the correctness ("$\surd$" or "$\times$") of the sentiment answers against the ground truth. Due to the unreliability of workers and the difficulties of tasks, workers cannot always do the tasks correctly. That is, workers may be more confident to do specific categories of tasks (e.g., biology, cars, electronic devices, and/or sports), but not others. For example, in Table 2, worker w_2 tags all tweets (tasks), $t_1 \sim t_5$, with wrong labels. It is very likely that worker w_2 is not familiar with the Apple company. Thus, in this case, it is rather challenging to guarantee the accuracy/quality of sentiment (task) answers, in the presence of such unreliable workers in the crowdsourcing system.

Furthermore, from Table 2, we can see that, all the 5 tasks are completed by workers within 10 minutes, except for task t_2 which takes worker w_2 almost one day to finish (either due to the delay of autonomous worker w_2 or because of the difficulty of task t_2). Such a long latency is highly undesirable for the task requester, who wants to proceed with the sentiment results for the next step of the social media research. Therefore, with high-latency workers in the crowdsourcing system, it is also important, yet challenging, to achieve low latency of the task completion. ■

In the aforementioned example, we can see that existing crowdsourcing systems have the challenges/obstacles of guaranteeing high accuracy and low latency. Inspired by this, in this paper, we propose a novel and effective crowdsourcing framework, called *Fast and Reliable crOwdsourcinG* (FROG) framework, which is designed for the system to actively assign workers with tasks (rather than for workers to autonomously select tasks), achieving both high reliability (accuracy) and low latency of all tasks.

Specifically, the FROG framework contains two important components, *task scheduler* and *notification* modules. In the task scheduler module, our FROG framework aims to avoid the scenario of tasks autonomously selected by unreliable workers (with high latency and low accuracy). Thus, we adopt the *server-assigned-task* (SAT) mode [21], in which the server actively schedules tasks for workers, considering both accuracy and latency. In particular, we formalize a novel *FROG task scheduling* (FROG-TS) problem under the SAT mode, which finds "good" worker-and-task assignments that minimize the maximal latencies for all tasks and maximize the accuracies (quality) of task results. We prove that the FROG-TS problem is NP-hard, by reducing it from the *multiprocessor scheduling problem* [17]. As a result, FROG-TS is not tractable. Alternatively, we design two heuristic approaches, request-based and batch-based scheduling, to efficiently tackle the FROG-TS problem.

Note that, existing works on reducing the latency are usually designed for specific tasks (e.g., filtering or resolving entities) [30, 34] by increasing prices over time to encourage workers to accept tasks [16], which cannot be directly used for general-purpose tasks under the budget constraint (i.e., the settings in our FROG framework). Some other works [18, 14] removed low-accuracy or high-latency workers, which may lead to idleness of workers and low throughput of the system. In contrast, our task scheduler module takes into account both factors, accuracy and latency, and can wisely design a worker-and-task assignment strategy with high accuracy, low latency, high throughput, and under a budget constraint.

In existing crowdsourcing systems, workers can freely join or leave the system. However, in the case that the system lacks of active workers, there is no way to invite more offline workers to perform online tasks. To address this issue, the notification module in our FROG framework is designed to notify those offline workers via invitation messages (e.g., by mobile phones). However, in order to avoid sending spam messages, we propose an *efficient worker notifying* (EWN) problem, which only sends task invitations to those workers with high probabilities of accepting the tasks. To tackle the EWN problem, we present a novel *smooth kernel density estimation* approach to efficiently compute the probability that a worker accepts the task invitation.

To summarize, in this paper, we have made the following contributions.

- We propose a new FROG framework for crowdsourcing, which consists of two important task scheduler and notification modules in Section 2.
- We formalize and tackle a novel worker-and-task scheduling problem in crowdsourcing, namely FROG-TS, which assigns tasks to suitable workers, with high reliability and low latency in Section 3.
- We propose a *smooth kernel density model* to estimate the probabilities that workers can accept task invitations for the EWN problem in the notification module in Section 4.
- We conduct extensive experiments to verify the effectiveness and efficiency of our proposed FROG framework on both real and synthetic data sets in Section 5.

Section 6 reviews previous works on the crowdsourcing. Section 7 concludes this paper.

2. PROBLEM DEFINITION

2.1 The FROG Framework

Figure 1 illustrates our *fast and reliable crowdsourcing* (FROG) framework, which consists of *worker profile manager*, *public worker pool*, *notification module*, *task scheduler module*, and *quality controller*.

Specifically, in our FROG framework, the worker profile manager keeps track of statistics for each worker in the system, including the response time (or in other words, the latency) and the accuracy of doing tasks in each category. These statistics are dynamically maintained, and can be used to guide the task scheduling process (in the task scheduler module).

Moreover, the public worker pool contains the information of online workers who are currently available for doing tasks. Different from the existing work [18] with exclusive retainer pool, we use a shared public retainer pool, which shares workers for different tasks. It can improve the global efficiency of the platform, and benefit workers with more rewards by assigning with multiple tasks (rather than one exclusive task for the exclusive pool).

When the number of online workers in the public worker pool is small, the notification module will send messages to offline workers (e.g., via mobile devices), and invite them to join the platform. Since offline workers do not want to receive too many (spam) messages, in this paper, we will propose a novel *smooth kernel density model* to estimate the probabilities that offline workers will accept

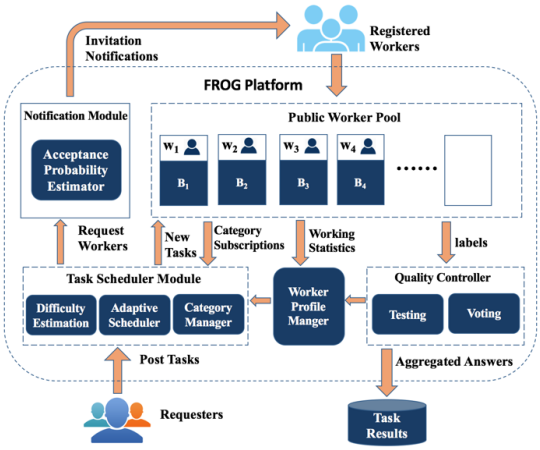


Figure 1: An Illustration of the FROG Framework.

the invitations, especially when the number of historical samples is small. This way, we will only send task invitations to those offline workers with high probabilities of accepting the tasks.

Most importantly, in the FROG framework, the task scheduler module wisely assigns tasks to suitable workers with the goals of reducing the latency and enhancing the accuracy for tasks. In this module, we formalize a novel *FROG task scheduling* (FROG-TS) problem, which finds the best worker-and-task assignments such that the maximal task latencies are minimized and the accuracies (quality) of task results are maximized. Due to the NP-hardness of this FROG-TS problem (proved later in Section 3.3), we design two approximation approaches, request-based and batch-based scheduling approaches.

Finally, the quality controller is in charge of the quality management during the entire process of the FROG framework. In particular, before workers are assigned to do tasks, we require that each worker need to register/subscribe one's expertise categories of tasks. To verify the correctness of subscriptions, the quality controller provides workers with some qualification tests, which include several sample tasks (with ground truth already known). Then, the system will later assign them with tasks in their qualified categories. Furthermore, after workers submit their answers of tasks to the system, the quality controller will check whether each task has received enough answers. If the answer is yes, then it will aggregate answers of each task (e.g., via voting methods), and then return final results.

In this paper, we will focus on general functions of two important modules, task scheduler and notification modules, in the FROG framework (as depicted in Figure 1), which will be formally defined in the next two subsections. In addition, we implement our framework and use WeChat [3] as its client to send tasks to workers and receive answers from workers.

2.2 The Task Scheduler Module

The task scheduler module focuses on finding a good worker-and-task assignment strategy with low latency (i.e., minimizing the maximum latency of tasks) and high reliability (i.e., satisfying the required quality levels of tasks).

Tasks and Workers. We first give the definitions for tasks and workers in the FROG framework. Specifically, since our framework is designed for general crowdsourcing platforms, we predefine a set, C, of r categories for tasks, that is, $C = \{c_1, c_2, ..., c_r\}$, where each task belongs to one category $c_l \in C$ $(1 \leq l \leq r)$. Here, each category can be the subject of tasks, such as cars, food, aerospace, or politics.

Table 3: Symbols and descriptions.

Symbol	Description
C	a set of task categories c_l
T	a set of tasks t_i
W	a set of workers w_j
$t_i.c$	the category of task t_i
q_i	a specific quality value of task t_i
s_k	the start time of a group of tasks T_k
e_k	the finish time of a group of tasks T_k
α_{jl}	the category accuracy of worker w_j on tasks in category c_l
r_{jl}	the response time of worker w_j on tasks in category c_l

Definition 1. *(Tasks) Let $T = \{t_1, t_2, ..., t_m\}$ be a set of m tasks in the crowdsourcing platform, where each task t_i $(1 \leq i \leq m)$ belongs to a task category, denoted by $t_i.c \in C$, and arrives at the system at the starting time s_i. Moreover, each task t_i is associated with a user-specified quality threshold q_i, which is the expected probability that the final result for task t_i is correct.* ■

Assume that task t_i is accomplished at the completion time e_i. Then, the latency, l_i, of task t_i can be given by: $l_i = e_i - s_i$, where s_i is the starting time (defined in Definition 1). Intuitively, the smaller the latency l_i is, the better the performance of the crowdsourcing platform is.

Definition 2. *(Workers) Let $W = \{w_1, w_2, ..., w_n\}$ be a set of n workers. For tasks in category c_l, each worker w_j $(1 \leq j \leq n)$ is associated with an accuracy, α_{jl}, that w_j do tasks in category c_l, and a response time, r_{jl}.* ■

As given in Definition 2, the category accuracy α_{jl} is the probability that worker w_j can correctly accomplish tasks in category c_l. Here, the response time r_{jl} measures the period length from the timestamp that worker w_j receives a task t_i (in category c_l) to the time point that he/she submits t_i's answer to the server.

In order to tackle the intrinsic error rate (unreliability) of workers, many crowdsourcing systems use various voting methods (e.g., majority voting [9]) to report high-accuracy results. For the ease of presentation, in this paper, we assume that each task is a binary task with YES/NO choices. The case of tasks with more than two choices can be easily transformed to a sequence of binary tasks. Thus, our proposed techniques can be extended to tasks with more than two choices, and we would like to leave it as our future work.

In the literature, there are some existing voting methods for result aggregations in the crowdsourcing system, such as the majority voting [9], weighted majority voting [14], half voting[28], and Bayesian voting [26]. In this paper, we use the majority voting for the result aggregation, which has been well accepted in many crowdsourcing works [9]. Assuming that the count of answering task t_i is odd, if the majority workers (not less than $\lceil \frac{k}{2} \rceil$ workers) vote for a same answer (e.g., Yes), then we take this answer as the final result of task t_i.

Denote W_i as the set of workers that do task t_i, and c_l as the category that task t_i belongs to. Then, we have the expected accuracy of task t_i as follows:

$$\Pr(W_i, c_l) = \sum_{x=\lceil \frac{k}{2} \rceil}^{k} \sum_{W_{i,x}} \Big(\prod_{w_j \in W_{i,x}} \alpha_{jl} \prod_{w_j \in W_i - W_{i,x}} (1 - \alpha_{jl}) \Big), \quad (1)$$

where $W_{i,x}$ is a subset of W_i with x elements.

Specifically, the expected task accuracy, $\Pr(W_i, c_l)$, calculated with Eq. (1) is the probability that more than half of the workers in W_i can answer t_i correctly. In the case of voting with multiple choices (other than 2 choices, like YES/NO), please refer to Appendix A of our technical report [11] for the equation of the expected accuracy of task t_i (with majority voting or other voting methods).

Table 3 summarizes the commonly used symbols.

The FROG Task Scheduling Problem. In the task scheduler module, one important problem is on how to route tasks to workers in the retainer pool with the guaranteed low latency and high accuracy. Next, we will formally define the problem of *FROG Task Scheduling* (FROG-TS) below.

Definition 3. (*FROG Task Scheduling Problem*) *Given a set T of m crowdsourcing tasks, and n workers in W, the problem of FROG task scheduling (FROG-TS) is to assign workers $w_j \in W$ to tasks $t_i \in T$, such that:*

1. *the accuracy $\Pr(W_i, c_l)$ (given in Eq. (1)) of task t_i is not lower than the required accuracy threshold q_i, and*
2. *the maximum latency $max(l_i)$ of tasks in T is minimized,*

where $l_i = e_i - s_i$ is the latency of task t_i, that is, the duration from the time s_i task t_i is posted in the system to the time, e_i, task t_i is completed. ■

We will later prove that the FROG-TS problem is NP-hard (in Section 3.3), and propose two effective approaches, request-based and batch-based scheduling, to solve this problem in Section 3.4.

2.3 The Notification Module

The notification module is in charge of sending notifications to those offline workers with high probabilities of being available and accepting the invitations (when the retainer pool needs more workers). In general, some workers may join the retainer pool autonomously, but it cannot guarantee that the retainer pool will be fulfilled quickly. Thus, the notification module will invite more offline workers to improve the fulfilling speed of the retainer pool.

Specifically, in our FROG framework, the server side maintains a public worker pool to support all tasks from the requesters. When *autonomous workers* join the system with a low rate, the system needs to invite more workers to fulfill the worker pool, and guarantees high task processing speed of the platform. One straightforward method is to notify all the offline workers. However, this broadcast method may disturb workers, when they are busy with other jobs (i.e., the probabilities that they accept invitations may be low). For example, assume that the system has 10,000 registered workers, and only 100 workers may potentially accept the invitations. With the broadcast method, all 10,000 workers will receive the notification message, which is inefficient and may potentially damage the user experience. A better strategy is to send notifications only to those workers who are very likely to join the worker pool. Moreover, we want to invite workers with high-accuracy and low-latency. Therefore, we formalize this problem as the *efficient worker notifying* (EWN) problem.

Definition 4. (*Efficient Worker Notifying Problem*) *Given the current timestamp ts, a set W of n offline workers, the historical online records $E_j = \{e_1, e_2, ..., e_n\}$ of each worker w_j, and the number, u, of workers that we need to recruit for the public worker pool, the problem of efficient worker notifying (EWN) is to select a subset of workers in W with high accuracies and low latencies to send invitation messages, such that:*

1. *the expected number, $E(P_{ts}(W))$, of workers who accept the invitations is greater than u, and*
2. *the number of workers in W, to whom we send notifications, is minimized,*

where $P_{ts}(.)$ is the probability of workers to accept invitations and join the worker pool at timestamp ts. ■

In Definition 4, it is not trivial to estimate the probability, $P_{ts}(W)$, that a worker prefers to join the platform at a given timestamp, especially when we are lacking of his/her historical records. However, if we notify too many workers, it will disturb them, and in the worst case drive them away from our platform forever. To solve the EWN problem, we propose an effective model to efficiently do the estimation in Section 4, with which we select workers with high acceptance probabilities, $P_{ts}(.)$, to send invitation messages such that the worker pool can be fulfilled quickly.

Moreover, since we want to invite workers with high acceptance probabilities, low response times, and high accuracies, we define the worker dominance below to select good worker candidates.

Definition 5. *(Worker Dominance) Given two worker candidates w_x and w_y, we say worker w_x dominates worker w_y, if it holds that: (1) $P_{ts}(w_x) > P_{ts}(w_y)$, (2) $\alpha_x \geq \alpha_y$, and (3) $r_x \geq r_y$, where $P_{ts}(w_j)$ is the probability that worker w_j is available, and α_x and r_x are the average accuracy and response time of worker w_x on his/her subscribed categories, respectively.*

Then, our notification module will invite those offline workers, w_j, with high ranking scores (i.e., defined as the number of workers dominated by worker w_j [37]). We will discuss the details of the ranking later in Section 4.

3. THE TASK SCHEDULER MODULE

The task scheduler module actively routes tasks to workers, such that tasks can be completed with small latency and the quality requirement of each task is satisfied. In order to improve the throughput of the FROG platform, in this section, we will estimate the difficulties of tasks and response times (and accuracies as well) of workers, based on records of recent answering. In particular, we will first present effective approaches to estimate worker and task profiles, and then tackle the *fast and reliable crowdsourcing task scheduling* (FROG-TS) problem, by designing two efficient heuristic-based approaches (due to its NP-hardness).

3.1 Worker Profile Estimation

In this subsection, we first present the methods to estimate the category accuracy and the response time of a worker, which can be used for finding good worker-and-task assignments in the FROG-TS problem.

The Estimation of the Category Accuracy. In the FROG framework, before each worker w_j joins the system, he/she needs to subscribe some task categories, c_l, he/she would like to contribute to. Then, worker w_j will complete a set of qualification testing tasks $T_c = \{t_1, t_2, ..., t_m\}$, by returning his/her answers, $A_j = \{a_{j1}, a_{j2}, ..., a_{jm}\}$, respectively. Here, the system has the ground truth of the testing tasks in T_c, denoted as $G = \{g_1, g_2, ..., g_m\}$.

Note that, at the beginning, we do not know the difficulties of the qualification testing tasks. Therefore, we initially treat all testing tasks with equal difficulty (i.e., 1). Next, we estimate the category accuracy, $\bar{\alpha_{jl}}$, of worker w_j on category c_l as follows:

$$\bar{\alpha_{jl}} = \frac{\sum_{i=1}^{|T_c|} \mathbb{1}(a_{ji} = g_i)}{|T_c|} \quad (2)$$

where $\mathbb{1}(v)$ is an indicator function (i.e., if $v = true$, we have $\mathbb{1}(v) = 1$; otherwise, $\mathbb{1}(v) = 0$), and $|T_c|$ is the number of qualification testing tasks.

Intuitively, Eq. (2) calculates the percentage of the correctly answered tasks (i.e., $a_{ji} = g_i$) by worker w_j (among all testing tasks).

In practice, the difficulties of testing tasks can be different. Intuitively, if more workers provide wrong answers for a task, then this task is more difficult; similarly, if a high-accuracy worker fails to answer a task, then this task is more likely to be difficult.

Based on the intuitions above, we can estimate the difficulty of a testing task as follows. Assume that we have a set, W_c, of workers w_j (with the current category accuracies α_{jl}) who have passed the

qualification test. Then, we give the definition of the difficulty β_i of a testing task t_i below:

$$\beta_i = \frac{\sum_{j=1}^{|W_c|} \left(\mathbb{1}(a_{ji} \neq g_i) \cdot \bar{\alpha_{jl}}\right)}{\sum_{j=1}^{|W_c|} \bar{\alpha_{jl}}} \quad (3)$$

where $\mathbb{1}(v)$ is an indicator function, and $|W_c|$ is the number of workers who passed the qualification test.

In Eq. (3), the numerator (i.e., $\sum_{j=1}^{|W_c|} \left(\mathbb{1}(a_{ji} \neq g_i) \cdot \bar{\alpha_{jl}}\right)$) computes the weighted count of wrong answers by workers in W_c for a testing task t_i. The denominator (i.e., $\sum_{j=1}^{|W_c|} \bar{\alpha_{jl}}$) is used to normalize the weighted count, such that the difficulty β_i of task t_i is within the interval $[0, 1]$. The higher β_i is, the more difficult t_i is.

In turn, we can treat the difficulty β_i of task t_i as a weight factor, and rewrite the category accuracy, $\bar{\alpha_{jl}}$, of worker w_j on category c_l in Eq. (2) as:

$$\bar{\alpha_{jl}} = \frac{\sum_{i=1}^{|T_c|} \left(\mathbb{1}(a_{ji} = g_i) \cdot \beta_i\right)}{\sum_{i=1}^{|T_c|} \beta_i}. \quad (4)$$

The Update of the Category Accuracy. The category accuracy of a worker may vary over time. For example, on one hand, the worker may achieve more and more accurate results, as he/she is more experienced in doing specific tasks. On the other hand, the worker may become less accurate, since he/she is tired after a long working day.

After worker w_j passes the qualification test of task category c_l, he/she will be assigned with tasks in that category, which would generate historical data on one's performance of conducting a set, T_r, of k real tasks in category c_l. We can update the category accuracy of a worker w_j based on such historical data.

Assume that the ground truth for k real tasks in T_r is $\{g_1, g_2, ..., g_k\}$, and answers provided by w_j are $\{a_{j1}, a_{j2}, ..., a_{jk}\}$, respectively. Then, we can update the category accuracy α_{jl} of worker w_j on category c_l as follows:

$$\alpha_{jl} = \theta_j \cdot \bar{\alpha_{jl}} + (1 - \theta_j) \cdot \frac{\sum_{i=1}^{k} \mathbb{1}(a_{ji} = g_i)}{k}, \quad (5)$$

where $\theta_j = \frac{|W_c|}{|W_c|+k}$ is a balance parameter to combine the performance of each worker in testing tasks and real tasks.

The Estimation of the Category Response Time. In reality, since different workers may have different abilities, skills, and speeds, their response times could be different, where the response time is defined as the length of the period from the timestamp that the task is posted to the time point that the worker submits the answer of the task to the server.

Furthermore, the response time of each worker may change temporally (i.e., with temporal correlations). To estimate the response time, we utilize the latest γ response records of worker w_j for answering tasks in category c_l, and apply the *least-squares method* [24] to predict the response time, r_{jl}, of worker w_j in a future timestamp. The least-squares method can minimize the summation of the squared residuals, where the residuals are the differences between the recent γ historical values and the fitted values provided by the model. We use the fitted line to estimate the category response time in a future timestamp.

3.2 Task Profile Estimation

In this subsection, we discuss the task difficulty, which may affect the latency of accomplishing tasks.

The Task Difficulty. Some tasks in the crowdsourcing system are in fact more difficult than others. In AMT [1], autonomous workers pick tasks by themselves. As a consequence, difficult tasks will be left without workers to conduct. In contrast, in our FROG platform, the server can designedly assign/push difficult tasks to reliable and low-latency workers to achieve the task quality and reduce the time delays.

For a given task t_i in category c_l with R possible answer choices ($R = 2$, in the case of YES/NO tasks), assume that $|W|$ workers accepted this task. Since some workers may skip the task (without completing the task), we denote γ_i as the number of workers who skipped task t_i, and Ω as the set of received answers, where $|\Omega| + \gamma_i = |W|$. Then, we can estimate the difficulty d_i of task t_i as follows:

$$d_i = \frac{\gamma_i}{|W|} + \frac{|\Omega|}{|W|} \cdot \frac{Entropy(t_i, \Omega)}{MaxEntropy(R)}. \quad (6)$$

Here, in Eq. (6), we have:

$$Entropy(t_i, \Omega) = \sum_{r=1, W_r \neq \emptyset}^{R} -\log\left(\frac{\sum_{w_j \in W_r} \alpha_{jl} + \epsilon}{\sum_{w_j \in W} \alpha_{jl} + R \cdot \epsilon}\right), \quad (7)$$

$$MaxEntropy(R) = R \cdot \left(-\frac{1}{R} \cdot \log\left(\frac{1}{R}\right)\right) = \log(R), \quad (8)$$

where W_r is the set of workers who select the r-th possible choice of task t_i and $|\Omega|$ is the number of received answers.

Discussions on the Task Difficulty. The task difficulty d_i in Eq. (6) is estimated based on workers' performance on doing tasks. Those workers who skipped the task treat tasks as being the most difficult (i.e., with difficulty equal to 1), whereas for those who did the task, we use the normalized entropy (or the diversity) of their answers to measure the task difficulty.

Specifically, the first term (i.e., $\frac{\gamma_i}{|W|}$) in Eq. (6) indicates the percentage of workers who skipped task t_i, or in other words considered the task difficulty is 1. Intuitively, when a task is skipped by more percentage of workers, it is more difficult.

The second term in Eq. (6) is to measure the task difficulty based on answers from those $\frac{|\Omega|}{|W|}$ percent of workers (who did the task). Our observation is as follows. When the answers of workers are spread more evenly (i.e., more diversely), it indicates that it is harder to obtain a final convincing answer of the task with high confidence. In this paper, to measure the diversity of answers from workers, we use the entropy, $Entropy(t_i, \Omega)$ (as given in Eq. (7)), of answers, with respect to the accuracies of workers. Intuitively, when a task is difficult to complete, workers will get confused, and eventually select diverse answers, which leads to high entropy value. Therefore, larger entropy implies higher task difficulty. To avoid the divide-by-zero error, when no answers are received, we add a small value ϵ (e.g., $\epsilon = 0.01$) to each possible choice of task t_i in Eq. (7). Moreover, we also normalize this entropy in Eq. (6), that is, dividing it by the maximum possible entropy value, $MaxEntropy(R)$ (as given in Eq. (8)).

3.3 Hardness of the FROG-TS Problem

In our FROG-TS problem (as given in Definition 3), since we need to guarantee that the expected accuracy of each task t_i is not lower than its specified quality threshold q_i, we assume that each task needs to be answered by h different workers, such that the final result (via majority voting) is accurate enough. With m tasks and n workers, in the worst case, there are an exponential number of possible task-and-worker assignment strategies, which incurs high time complexity (i.e., $O(\frac{n!}{(n-h)!}^m)$).

Below, we prove that the FROG-TS problem is NP-hard, by reducing it from the *multiprocessor scheduling problem* (MSP) [17].

Theorem 3.1. *(Hardness of the FROG-TS Problem) The problem of FROG Task Scheduling (FROG-TS) is a NP-hard problem.*

PROOF. Please refer to Appendix B of our technical report [11]. □

The FROG-TS problem focuses on completing multiple tasks that satisfy the required quality thresholds, which requires that each task be answered by multiple workers. Thus, we cannot directly use the existing approximation algorithms for the MSP problem (or its variants) to solve the FROG-TS problem.

Algorithm 1: GreedyRequest(W, T)

Input: A worker w_j requesting for his/her next task and a set $T = \{t_1, t_2, ..., t_v\}$ of v uncompleted tasks
Output: Returned task t_i

1 **foreach** *task* t_i ***in*** T **do**
2 calculate the delay possibility value of t_i with Eq. (9);
3 select one task t_i with the highest delay probability value;
4 **if** *the expected accuracy of* t_i *is higher than* q_i **then**
5 Remove t_i from T;
6 **return** t_i;

Due to the NP-hardness of our FROG-TS problem, in the next subsection, we will introduce an adaptive task routing approach with two worker-and-task scheduling algorithms, *request-based* and *batch-based scheduling* approaches to efficiently retrieve the FROG-TS answers.

3.4 Adaptive Scheduling Approaches

In this subsection, we propose adaptive scheduling strategies to iteratively conduct as many tasks as possible, with the highest delay possibilities. We first order the unfinished tasks with their delay probabilities. The higher the delay probability is, the more likely the task will be delayed. Then, we iteratively assign the least number of workers to the task with the highest delay probability, such that the throughput of the platform (i.e., the number of the completed tasks) can be maximized.

3.4.1 The Delay Probability

As mentioned in the second criterion of the FROG-TS problem (i.e., in Definition 3, we want to minimize the maximum latency of tasks in T. In order to achieve this goal, we will first calculate the delay probability, $L(t_i)$, of task t_i in T, and then assign workers to those tasks with high delay probabilities first, such that the maximum latency of tasks can be greedily minimized.

We define the delay probability of task t_i in the logistic function [4] as follows:

$$L(t_i) = \frac{1}{1 + e^{-\epsilon_i \cdot d_i}}, \tag{9}$$

where ϵ_i is the time lapse of task t_i, d_i is the difficulty of task t_i given by Eq. (6), and e is the natural logarithm base (i.e., $e = 2.71828...$). The logistic function is widely used in a range of fields, including artificial neural networks, linguistics, and statistics. Here, we use logistic function to capture the increase of delay probability is exponential when the difficulty of task increases a little and then slow down. Note that, other methods can also be used to estimate the delay probability of tasks.

Since we use the entropy-based equation to measure the difficulties of tasks, the difficulty d_i of task t_i will be in a range of $[0, +\infty)$. Further, ϵ_i indicates the steepness of the logistic function. Then, the delay probability $L(t_i)$ will be in the range of $[0.5, 1)$.

3.4.2 Request-based Scheduling (RBS) Approach

With the estimation of the delay probabilities of tasks, we propose a *request-based scheduling* (RBS) approach. In this approach, when a worker becomes available, he/she will send a request for the next task to the server. Then, the server calculates the delay probabilities of the on-going tasks on the platform, and greedily return the task with the highest delay probability to the worker.

The pseudo code of our request-based scheduling approach, namely GreedyRequest, is shown in Algorithm 1. It first calculates the delay probability of each uncompleted task in T (lines 1-2). Then, it selects a suitable task t_i with the highest delay probability (line 3). If we find the expected accuracy of task t_i (given in Eq. (1)) is higher than the quality threshold q_i, then we will remove task t_i from T. Finally, we return/assign task t_i to worker w_i.

The Time Complexity of RBS. We next analyze the time complexity of the request-based scheduling approach, GreedyRequest, in Algorithm 1. For each task t_i, to compute its delay probability, the time complexity is $O(1)$. Thus, the time complexity of computing delay probabilities for all v uncompleted tasks is given by $O(v)$ (lines 1-2). Next, the cost of selecting the task t_i with the highest delay probability is also $O(v)$ (line 3). The cost of checking the completeness for task t_i and removing it from T is given by $O(1)$. As a result, the time complexity of our request-based scheduling approach is given by $O(v)$.

3.4.3 Batch-based Scheduling (BBS) Approach

Although the RBS approach can easily and quickly respond to each worker's request, it in fact does not have the control on workers in this request-and-answer style. Next, we will propose an orthogonal *batch-based scheduling* (BBS) approach, which assigns each worker with a list of suitable tasks in a batch, where the length of the list is determined by his/her response speed.

The intuition of our BBS approach is as follows. If we can assign high-accuracy workers to difficult and urgent tasks and low-accuracy workers with easy and not that urgent tasks, then the worker labor will be more efficient and the throughput of the platform will increase.

Specifically, in each round, our BBS approach iteratively picks a task with the highest delay probability (among all the remaining tasks in the system), and then greedily selects a minimum set of workers to complete this task. Algorithm 2 shows the pseudo code of the BBS algorithm, namely GreedyBatch. In particular, since no worker-and-task pair is assigned at the beginning, we initialize the assignment set $\mathbb{A}$ as an empty set (line 1). Then, we calculate the delay probability of each unfinished task (given in Eq. (9)) (lines 2-3). Thereafter, we iteratively assign workers for the next task t_i with the highest delay probability (lines 4-6). Next, we invoke Algorithm MinWorkerSetSelection, which selects a minimum set, W_o, of workers who satisfy the required accuracy threshold of task t_i (line 7). If W_o is not empty, then we insert task-and-worker pairs, $\langle t_i, w_j \rangle$, into set $\mathbb{A}$ (lines 8-10). If each worker w_i cannot be assigned with more tasks, then we remove him/her from W (lines 11-12). Here, we decide whether a worker w_j can be assigned with more tasks, according to his/her response times on categories, his/her assigned tasks, and the round interval of the BBS approach. That is, if the summation of response times of the assigned tasks is larger than the round interval, then the worker cannot be assigned with more tasks; otherwise, we can still assign more tasks to him/her. After all tasks have been processed, we return the set, $\mathbb{A}$, of task-and-worker pairs (line 13).

Minimum Worker Set Selection. In line 7 of Algorithm 2 above, we mentioned a MinWorkerSetSelection algorithm, which selects a minimum set of workers satisfying the constraint of the quality threshold q_i for task t_i. We will discuss the algorithm in detail, and prove its correctness below.

Before we provide the algorithm, we first present one property of the expected accuracy of a task.

Lemma 3.1. *Given a set of workers, W_i, assigned to task t_i in category c_l, the expected accuracy of task t_i can be calculated as follows:*

$$\begin{aligned} \Pr(W_i, c_l) &= \Pr(W_i - \{w_j\}, c_l) \\ &+ \alpha_{jl}\Big(\sum_{U}\big(\prod_{w_o \in U} \alpha_{ol} \prod_{w_o \in W_i - U - \{w_j\}} (1 - \alpha_{ol})\big)\Big) \end{aligned} \tag{10}$$

where $U = W_{i,\lceil \frac{k}{2} \rceil} - \{w_j\}$ and $\Pr(W_i, c_l)$ is defined in Eq. (1).

Algorithm 2: GreedyBatch(W, T)

Input: A set, $T = \{t_1, t_2, ..., t_m\}$, of m unfinished tasks and a set, $W = \{w_1, w_2, ..., w_n\}$, of n workers
Output: Assignment $\mathbb{A} = \{\langle t_i, w_j \rangle\}$

1 $\mathbb{A} \leftarrow \emptyset$;
2 **foreach** *task* t_i ***in*** T **do**
3 calculate the delay possibility value of t_i with Eq. (9);
4 **while** $T \neq \emptyset$ ***and*** $W \neq \emptyset$ **do**
5 select task t_i with the highest delay possibility value;
6 remove t_i from T;
7 $W_o \leftarrow$ MinWorkerSetSelection(t_i, W, W_i);
8 **if** $W_o \neq \emptyset$ **then**
9 **foreach** $w_j \in W_o$ **do**
10 Insert $\langle t_i, w_j \rangle$ into $\mathbb{A}$;
11 **if** w_j *cannot be assigned with more tasks* **then**
12 Remove w_j from W;
13 **return** $\mathbb{A}$;

PROOF. For a task t_i in category c_l, assume a set of k workers W_i are assigned to it. As the definition of the expected accuracy of task t_i in Eq. (1) shows, for any subset $V_i' \subseteq W_i$ and $|V_i| \geq \lceil \frac{k}{2} \rceil$, when worker $w_j \in W_i$ is not in V_i, we can find an addend A of

$$\Big(\prod_{w_o \in V_i} \alpha_{ol} \prod_{w_o \in W_i - V_i - w_j} (1 - \alpha_{ol}) \Big) \cdot (1 - \alpha_{jl})$$

in Eq. (1). As the Eq. (1) enumerates all the possible subsets of W_i with more than $\lceil \frac{k}{2} \rceil$ elements, we can find a subset $V_i' = V_i + \{w_j\}$, which represents another addend A' of

$$\alpha_{jl} \Big(\prod_{w_o \in V_i' - \{w_j\}} \alpha_{ol} \prod_{w_o \in W_i - V_i'} (1 - \alpha_{ol}) \Big)$$

in Eq. (1). Then, we have:

$$A + A' = \prod_{w_o \in V_i} \alpha_{ol} \prod_{w_o \in W_i - V_i - w_j} (1 - \alpha_{ol}).$$

After we combine all these kind of pairs of addends of worker w_j, we can obtain:

$$\begin{aligned} \Pr(W_i, c_l) &= \sum_{x=\lceil \frac{k}{2} \rceil - 1}^{k-1} \sum_{W_{i,x}'} \Big(\prod_{w_o \in W_{i,x}'} \alpha_{ol} \prod_{w_o \in W_i' - W_{i,x}'} (1 - \alpha_{ol}) \Big) \\ &\quad + \alpha_{jl} \Big(\sum_{U} \big(\prod_{w_o \in U} \alpha_{ol} \prod_{w_o \in W_i - U - \{w_j\}} (1 - \alpha_{ol}) \big) \Big) \\ &= \Pr(W_i', c_l) \\ &\quad + \alpha_{jl} \Big(\sum_{U} \big(\prod_{w_o \in U} \alpha_{ol} \prod_{w_o \in W_i - U - \{w_j\}} (1 - \alpha_{ol}) \big) \Big), \end{aligned} \tag{11}$$

where $W_i' = W_i - \{w_j\}$ and $W_{i,x}'$ is a subset of W_i' with x elements, and $U = W_{i, \lceil \frac{k}{2} \rceil} - \{w_j\}$. Eq. (11) holds as k is always odd to ensure the majority voting can get a final result. □

We can derive two corollaries below.

Corollary 3.1. *For a task t_i in category c_l with a set of k assigned workers W_i, if the category accuracy α_{jl} of any worker $w_j \in W_i$ increases, the expected accuracy $\Pr(W_i, c_l)$ of task t_i will increase.*

PROOF. In Eq. (10), when the accuracy α_{jl} of worker w_j increases, the first factor $\Pr(W_i - \{w_j\}, c_l)$ will not be affected, and the second factor will increase. Thus, the corollary is proved. □

Corollary 3.2. *For a task t_i in category c_l with a set of k assigned workers W_i, if we assign a new worker w_j to task t_i, the expected accuracy of task t_i will increase.*

PROOF. With Lemma 3.1, we can see that when adding one more worker to a task t_i, the expected accuracy of task t_i will increase. □

Theorem 3.2. *To finish a task satisfying its required expected accuracy threshold, it needs less workers with high accuracies rather than workers with low accuracies.*

PROOF. With Corollaries 3.1 and 3.2, to increase the expected accuracy of a task t_i, we can use workers with higher category accuracies or assign more workers to task t_i. When the required expected accuracy is given, we can finish task t_i with a smaller number of high-accuracy workers. □

In Theorem 3.2, we can see that high-accuracy workers are more efficient in finishing tasks satisfying the expected accuracies. This indicates that, we can use fewer workers with higher accuracies to finish a task. To accomplish as many tasks as possible, we aim to greedily pick the least number of workers to finish each task iteratively.

Algorithm 3: MinWorkerSetSelection(t_i, W, W_i)

Input: A set $W = \{w_1, w_2, ..., w_n\}$ of available workers, a task t_i in category c_l with a set of already assigned workers W_i
Output: A minimum set of workers assigned to task t_i

1 $W_o \leftarrow W_i$;
2 **while** $Pr(W_o, c_l) < q_i$ *and* $|W - W_o| > 0$ **do**
3 choose a new worker w_j with the highest accuracy α_{jl};
 W.remove(w_j);
4 W_o.add(w_j);
5 **if** $Pr(W_o, c_l) \geq q_i$ **then**
6 **return** $W_o - W_i$;
7 **else**
8 **return** $\emptyset$;

Algorithm 3 exactly shows the procedure of MinWorkerSetSelection, which selects a minimum set, W_o, of workers to conduct task t_i. In each iteration, we greedily select a worker w_j (who has not been assigned to task t_i) with the highest accuracy in the category of task t_i, and assign workers to task t_i (lines 2-4). If such a minimum worker set exists, we return the newly assigned worker set; otherwise, we return an empty set (lines 5-8). The correctness of Algorithm 3 is shown in Lemma 3.2.

Lemma 3.2. *The number of workers in the set W_o returned by Algorithm 3 is minimum, if W_o exists.*

PROOF. Let set W_o be the returned by Algorithm 3 to satisfy the quality threshold q_i and worker w_j is the last one added to set W_o. Assume there is a subset of workers $W' \subseteq W$ such that $|W'| = |W_o| - 1$ and $Pr(W', c_l) \geq q_i$.

Since each worker in W_o is greedily picked with the highest current accuracy in each iteration of lines 2-4 in Algorithm 3, for any worker, $w_k \in W_o$ will have higher accuracy than any worker $w_k' \in W'$. As $|W'| = |W_o - \{w_j\}|$, according to Corollary 3.1, we have $Pr(W_o - \{w_j\}, c_l) > Pr(W', c_l)$. However, as w_j is added to W_o, which means $Pr(W_o - \{w_j\}, c_l) < q_i$. It conflicts with the assumption that $Pr(W', c_l) \geq q_i$. Thus, set W' cannot exist. □

The Time Complexity of BBS. To analyze the time complexity of the batch-based scheduling (BBS) approach, called GreedyBatch, as shown in Algorithm 2, we assume that each task t_i needs to be answered by h workers. The time complexity of calculating the delay probability of a task is given by $O(m)$ (lines 2-3). Since each iteration solves one task, there are at most m iterations (lines 4-13). In each iteration, selecting one task t_i with the highest delay probability requires $O(m)$ cost (line 5). The time complexity of

the MinWorkerSetSelection procedure is given by $O(h)$ (line 7). The time complexity of assigning workers to the selected task t_i is $O(h)$ (lines 8-12). Thus, the overall time complexity of the BBS approach is given by $max(O(m^2), O(m \cdot h))$.

4. THE NOTIFICATION MODULE

In this section, we introduce the detailed model of the notification module in our PROG framework (as mentioned in Section 2), which is in charge of sending invitation notifications to offline workers in order to maintain enough online workers doing tasks. Since it is not a good idea to broadcast to all offline workers, our notification module only sends notifications to those workers with high probabilities of accepting invitations.

4.1 Kernel Density Estimation for the Workers' Availability

In this subsection, we will model the availability of those (offline) workers from historical records. The intuition is that, for each worker, the patten of availability on each day is relatively similar. For example, a worker may have the spare time to do tasks, when he/she is on the bus to the school (or company) at about 7 am every morning. Thus, we may obtain their historical data about the timestamps they conducted tasks.

However, the number of historical records (i.e., sample size) for each worker might be small. In order to accurately estimate the probability of any timestamp that a worker is available, in this paper, we use a non-parametric approach, called *kernel density estimation* (KDE) [31], based on random samples (i.e., historical timestamps that the worker is available).

Specifically, for a worker w_j, let $E_j = \{e_1, e_2, ..., e_n\}$ be a set of n active records that worker w_j did some tasks, where event e_i ($1 \leq i \leq n$) occurs at timestamp ts_i. Then, we can use the following KDE estimator to compute the probability that worker w_j is available at timestamp ts:

$$f(ts|E_j, h) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{ts - ts_i}{h}\right),$$

where e is the event that worker w_j is available and will accept the invitation at a given timestamp ts, $K(\cdot)$ is a kernel function (here, we use Gaussian kernel function $K(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2}$), and h is a scalar bandwidth parameter for all events in E. The bandwidth of the kernel is a free parameter and exhibits a strong influence on the estimation. For simplicity, we set the bandwidth following a rule-of-thumb [33] as follows:

$$h = \left(\frac{4\hat{\sigma}^5}{3n}\right)^{\frac{1}{5}} = 1.06\hat{\sigma}n^{-1/5}, \tag{12}$$

where $\hat{\sigma}$ is the standard deviation of the samples. The rule works well when density is close to being normal, which is however not true for estimating the probability of workers at a given timestamp ts. Nonetheless, adapting the kernel bandwidth h_i to each data sample e_i may overcome this issue [8].

Inspired by this idea, we select k nearest neighbors of event e_i (here, we consider neighbors by using time as measure, instead of distance), and calculate the adaptive bandwidth h_i of event e_i with $(k+1)$ samples using Eq. (12), where k is set to $\beta \cdot n$ (β is a ratio parameter). Afterwards, we can define the adaptive bandwidth KDE as follows:

$$f(ts|E_j) = \frac{1}{n} \sum_{i=1}^{n} K\left(\frac{ts - ts_i}{h_i}\right). \tag{13}$$

4.2 Smooth Estimator

Up to now, we have discussed the adaptive kernel density approach to estimate the probability that a worker is available, based on one's historical records (samples). However, some workers may just register or rarely accomplish tasks, such that his/her historical events are not available or enough to make accurate estimations, which is the "cold-start" problem that often happens in the recommendation system [32].

Inspired by techniques [32] used to solve such a cold-start problem in recommendation systems and the influence among friends [12] (i.e., friends tend to have similar behavior patterns, such as the online time periods), we propose a *smooth KDE model* (SKDE), which combines the individual's kernel density estimator with related scale models. That is, for each worker, we can use historical data of his/her friends to supplement/predict his/her behaviors.

Here, our FROG platform is assumed to have the access to the friendship network of each worker, according to his/her social networks (such as Facebook, Twitter, and WeChat). In our experiments of this paper, our FROG platform used data from the WeChat network.

Specifically, we define a smooth kernel density estimation model as follows:

$$P_{SKDE}(ts|E_j, E) = \sum_{s=1}^{S} \alpha_s f(ts|E^s) \tag{14}$$

where $\alpha_1, \alpha_1, ..., \alpha_s$ are non-negative *smoothing factors* with the property of $\sum_{s=1}^{S} \alpha_s = 1$, E is the entire historical events of all the workers, and $f(ts|E^s)$ is the s-th *scaling density estimator* calculated on the subset events E^s.

For a smooth KDE model with S (> 2) scaling density estimators, the first scaling density estimator can be the basic individual kernel density estimator with $E^1 = E_j$ and the S-th scaling density estimator can be the entire population density estimator with $E^s = E$. Moreover, since our FROG platform can obtain the friendship network of each worker (e.g., Facebook, Twitter, and WeChat), after one registers with social media accounts, we can find each worker's k-step friends. This way, for the intermediate scaling density estimators $s = 2, ..., S-1$, we can use different friendship scales, such as the records of the 1-step friends, 2-step friends, ..., $(S-2)$-step friends of worker w_i.

According to the famous Six degrees of separation theory [5], S is not larger than 6.

To train the SKDE model, we need to set proper values for smoothing factors α_s. We use the latest event records as validation data E^v (here $|E^v| = n$), and other history records as the training data E^h. Specifically, for each event e_k in E^v, we have the estimated probability as follows:

$$P(ts_k|E^h, \alpha) = P_{SKDE}(ts_k|E^h, \alpha) = \sum_{s=1}^{S} \alpha_s f(ts_k|E^s)$$

where S is the number of scaling density estimators. Then, to tune the smoothing factors, we use the Maximum Likelihood Estimation (MLE) with log-likelihood as follows:

$$\hat{\alpha} = argmax_{\alpha} \log\big(\prod_{k=1}^{n} P(ts_k|E^h, \alpha)\big) \tag{15}$$

However, Eq. (15) is not trivial to solve, thus, we use EM algorithm to calculate its approximate result.

We initialize the smoothing factors as $\alpha_s = 1/S$ for $s = 1, 2, ..., S$. Next, we repeat Expectation-step and Maximization-step, until the smoothing factors converge.

Expectation Step. We add a latent parameter $Z = \{z_1, z_2, ..., z_S\}$, and its distribution on ts_k is $Q_k(Z)$, then we can estimate $Q_k(Z)$ as follows:

$$\begin{aligned} Q_k^t(z_s|\alpha^t) &= P(z_s|ts_k, E^h, \alpha^t) \\ &= \frac{\alpha_s^t \cdot f(ts_k|E^s)}{\sum_{i=1}^{S} \alpha_i^t \cdot f(ts_k|E^i)}, \end{aligned}$$

where $f(\cdot)$ is calculated with Eq. (13).

Maximization Step. Based on the expectation result of the latent parameter Z, we can calculate the next smoothing factor values α^{t+1} with the maximum likelihood estimation as follows:

$$\alpha^{t+1} = argmax_{\alpha} \sum_{k=1}^{n} \sum_{s=1}^{S} Q_k^t(z_s) \log \left(\frac{P(st_k, z_s|\alpha^t)}{Q_k^t(z_s)} \right)$$
$$= argmax_{\alpha} \sum_{k=1}^{n} \sum_{s=1}^{S} Q_k^t(z_s) \log \left(\frac{\alpha_s^t \cdot f(ts_k|E^s)}{Q_k^t(z_s)} \right),$$

where $f(\cdot)$ is calculated with Eq. (13).

4.3 Processing of the Efficient Worker Notifying Problem

As given in Definition 4, our EWN problem is to select a minimum set of workers with high probabilities to accept invitations, to whom we will send notifications.

Formally, given a trained smooth KDE model and a timestamp ts, assume that we want to recruit u more workers for the FROG platform. In the EWN problem (in Definition 4), the acceptance probability $P_{ts}(w_j)$ of worker w_j can be estimated by Eq. (14).

Next, with Definition 5, we can sort workers, w_j, based on their ranking scores $R(w_j)$ (e.g., the number of workers dominated by each worker) [37]. Thereafter, we will notify top-v workers with the highest ranking scores.

The pseudo code of selecting worker candidates is shown in Algorithm 4. We first initialize the selected worker set, W_n, with an empty set (line 1). Next, we calculate the ranking scores of each worker (e.g., the number of other workers can be dominated with the Definition 5) (lines 2-3). Then, we iteratively pick workers with the highest ranking scores until the selected workers are enough or all workers have been selected (lines 4-8). Finally, we return the selected worker candidates to send invitation notifications (line 9).

Algorithm 4: WorkerNotify(W, T)

Input: A set, $W = \{w_1, w_2, ..., w_n\}$, of offline workers, the expected number, u, of acceptance workers, and the current timestamp ts

Output: A set, W_n, of workers to be invited

1 $W_n = \emptyset$;
2 **foreach** *worker* w_j ***in*** W **do**
3 calculate the ranking score $R(w_j)$ of w_j;
4 **while** $u > 0$ ***and*** $|W| > 0$ **do**
5 select one worker w_j with the highest ranking score in W;
6 $W = W - \{w_j\}$;
7 W_n.add(w_j);
8 $u = u - P_{ts}(w_j)$
9 **return** W_n;

The Time Complexity. To compute the ranking scores, we need to compare every two workers, whose time complexity is $O(n^2)$. In each iteration, we select one candidate, and there are at most n iterations. Assuming that n workers are sorted by their ranking scores, lines 4-8 have the time complexity $O(n \cdot \log(n))$. Thus, the time complexity of Algorithm 4 is given by $O(n^2)$.

Discussions on Improving the EWN Efficiency. To improve the efficiency of calculating the ranking scores of workers, we may utilize a 3D grid index to accelerate the computation, where 3D includes the acceptance probability, response time, and accuracy. Each worker is in fact a point in a 3D space w.r.t. these 3 dimensions. If a worker w_j dominates a grid cell gc_x, then all workers in cell gc_x are dominated by w_j. Similarly, if worker w_j is dominated by the cell gc_x, then all the workers in gc_x cannot be dominated by w_j. Then, we can compute the lower/upper bounds of the ranking score for each worker, and utilize them to enable fast pruning [37].

Table 4: Experimental Settings.

Parameters	Values
the number of categories l	5, 10, **20**, 30, 40
the number of tasks m	1000, 2000, **3000**, 4000, 5000
the number of workers n	100, 200, **300**, 400, 500
the range of quality threshold $[q^-, q^+]$	[0.8, 0.85], **[0.85, 0.9]**, [0.9, 0.95], [0.95, 0.97]

5. EXPERIMENTAL STUDY

5.1 Experimental Methodology

Data Sets for Experiments on Task Scheduler Module. We use both real and synthetic data to test our task scheduler module. We first conduct a set of comparison experiments on the real-world crowdsourcing platform, gMission [10], and evaluate our task scheduler module on 5 data sets. Tasks in each data set belong to the same category. For each experiment, we use 16 tasks for each data set (category). We manually label the ground truth of tasks. To subscribe one category, each worker is required to take a qualification test consisting of 5 testing questions. We uniformly generate quality threshold for each task, within the range $[0.85, 0.9]$. Below, we give brief descriptions of the 5 real data sets.

1) *Disaster Events Detection (DED)*: DED contains a set of tasks, which ask workers to determine whether a tweet describes a disaster event. For example, a task can be "Just happened a terrible car crash" and workers are required to select "Disaster Event" or "Not Disaster Event".

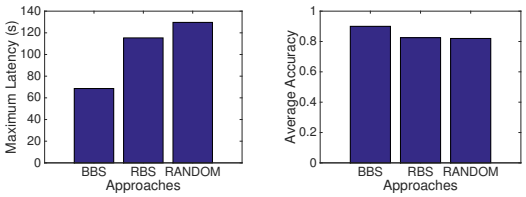
2) *Climate Warming Detection (CWD)*: CWD is to determine whether a tweet considers the existence of global warming or climate change or not. The possible answers are "Yes", if the tweet suggests global warming is occurring, and "No" if the tweet suggests global warming is not occurring. One tweet example is "Global warming. Clearly.", and workers are expected to answer "Yes".

3) *App Search Match (ASM)*: In ASM, workers are required to view a variety of searches for mobile Apps, and determine if the intents of those searches are matched. For example, one short query is "music player"; the other one is a longer one like "I would like to download an App that plays the music on the phone from multiple sources like Spotify and Pandora and my library." If the two searches have the same intent, workers should select "Yes"; otherwise, they are expected to select "No".

4) *Body Parts Relationship Verification (BPRV)*: In BPRV, workers should point out if certain body parts are part of other parts. Questions were phrased like: "[Part 1] is a part of [part 2]". For example, "Nose is a part of spine" or "Ear is a part of head." Workers should say "Yes" or "No" for this statement.

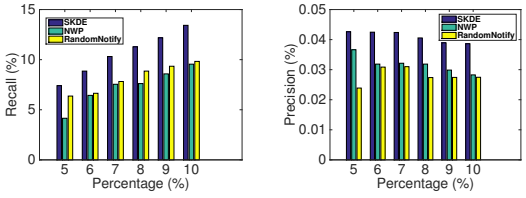
5) *Sentiment Analysis on Apple Incorporation (SAA)*: Workers are required to analyze the sentiment about Apple, based on tweets containing "#AAPL, @apple, etc". In each task, workers are given a tweet about Apple, and asked whether the user is positive, negative, or neutral about Apple. We used records with positive or negative attitude about Apple, and asked workers to select "positive" or "negative" for each tweet.

For synthetic data, we simulate crowd workers based on the observations from real platform experiments. Specifically, in experiments on the real platform, we measure the average response time, r_{jl}, of worker w_j on category c_l, the variance of the response time σ_{jl}^2, and the category accuracy α_{jl}. Then, to generate a worker, we uniformly produce the response speed on category c_l within a range of $[r_l^-, r_l^+]$, where r_l^- and r_l^+ are the minimum and maximum response times, respectively, measured from the real platform. Similarly, we can generate the variance of the response time, and the category accuracy of each worker.



(a) Recall (b) Precision

Figure 2: The Performance of the Task Scheduler Module on Real Data.



(a) Recall (b) Precision

Figure 3: The Performance of the Notification Module on Real Data.

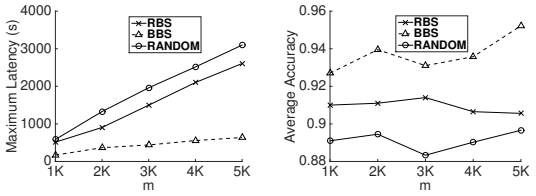
Table 4 depicts the parameter settings in our experiments, where default values of parameters are in bold font. In each set of experiments, we vary one parameter, while setting other parameters to their default values. For each experiment, we report the maximum latency and the average accuracy of tasks.

Data Sets for Experiments on Notification Module. To test our notification module in the FROG framework, we utilize *Higgs Twitter Dataset* [13]. The Higgs Twitter Dataset is collected for monitoring the spreading process on the Twitter, before, during, and after the announcement of the discovery of a new particle with features of the elusive Higgs boson on July 4th, 2012. The messages posted on the Twitter about this discovery between July 1st and 7th, 2012 are recorded. There are 456,626 user nodes and 14,855,842 edges between them. In addition, the data set contains 563,069 activities. Each activity happens between two users and can be retweet, mention, or reply. We initialize the registered workers on our platform with users in the Higgs Twitter Dataset (and their relationship on the Twitter). What is more, the activities in the data set is treated as online records of workers on the platform.

Competitors and Measures. For the task scheduler module, we conduct experiments to test our two adaptive scheduling approaches, request-based (RBS) and batch-based scheduling (BBS) approaches. We compare them with a random method, namely RANDOM, which randomly routes tasks to workers.

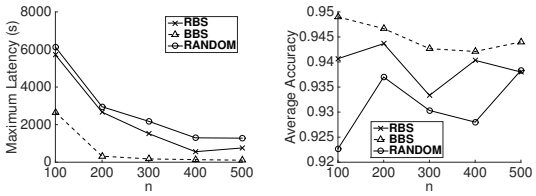
Recall that, RBS returns one task with the highest delay probability to a worker who is requesting his/her next task, such that the maximum latency of tasks can be greedily reduced. BBS routes tasks to workers in a batch style, which means in each round it iteratively picks the next task with the highest delay probability and assigns a minimum set of workers to this task, satisfying its quality requirement (until all tasks have been processed or no workers is available). To show the effectiveness of our adaptive scheduling approaches, we compare our RBS and BBS approaches with the random method, (RANDOM). We hire 15 workers from the WeChat platform to conduct this experiment.

For the notification module, we conduct experiments to compare our smooth KDE model with a random method, namely RandNotify, which randomly selects the same number of tasks as the smooth KDE approach. Moreover, we also compare our approach with a simple method, namely *Nearest Worker Priority* (NWP), which selects workers with the most number of historical records within the 15-minute period before or after the given timestamp in previous dates. For each predicted worker, if he/she has activities within the time period from the the target timestamp to 15 minutes later, we treat that it is a correct prediction. At timestamp ts, we denote $N_c(ts)$ as the number of correct predictions, $N_t(ts)$ as the number of total predictions and $N_a(ts)$ as the number of activities that really happened.



(a) Maximum Latency (b) Average Accuracy

Figure 4: Effect of the number of tasks m.



(a) Maximum Latency (b) Average Accuracy

Figure 5: Effect of the number of workers n.

For experiments on the task scheduler module, we report maximum latencies of tasks and average task accuracies, for both our approaches and competitor method. For experiments on the notification module, we present the precision ($= \frac{N_c(ts)}{N_t(ts)}$) and recall ($= \frac{N_c(ts)}{N_a(ts)}$) of all tested methods. Our experiments were run on an Intel Xeon X5675 CPU @3.07 GHZ with 32 GB RAM in Java.

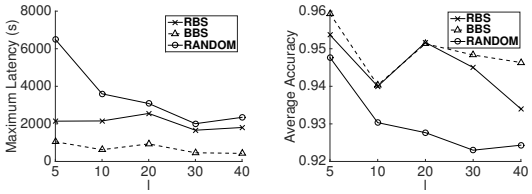
5.2 Experiments on Real Data

The Performance of the Task Scheduler Module on Real Data. Figure 2 shows the results of experiments on our real platform about the task scheduler module of our framework. For the maximum latencies shown in Figure 2(a), our two approaches can maintain lower latencies than the baseline approach, RANDOM. Specifically, BBS can achieve a much lower latency, which is just half of that of RANDOM. For the average accuracies shown in Figure 2(b), our two approaches achieve higher average accuracies than RANDOM. Moreover, the average accuracy of BBS is higher than that of RBS. The reason is that, BBS can complete the most urgent tasks with minimum sets of workers, achieving the highest category accuracies. In contrast, RBS is not concerned with the accuracy, and just routes available workers to tasks with the highest delay probabilities. Thus, RBS is not that effective, compared with BBS, to maintain a low latency.

The Performance of Notification Module on Real Data. To show the effectiveness of our smooth KDE model, we present the recall and precision of our model compared with NWP and RandomNotify, by varying the number of prediction samples from 5% to 10% of the entire population. As shown in Figure 3(a), our smooth KDE model can achieve higher recall scores than the other two baseline methods. In addition, when we predict with more samples, the advantage of our smooth KDE model is more obvious w.r.t. the recall scores. The reason is that our smooth KDE model can utilize the influence of the friends, which is more effective when we predict with more samples. Similarly, in Figure 3(b), smooth KDE model can obtain the highest precision scores among all tested methods.

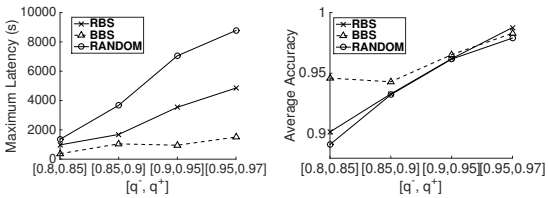
5.3 Experiments on Synthetic Data

Effect of the Number, m, of Tasks. Figure 4 shows the maximum latency and average accuracy of three approaches, RBS, BBS, and RANDOM, by varying the number, m, of tasks from $1K$ to $5K$, where other parameters are set to their default values. As shown in Figure 4(a), with more tasks (i.e., larger m values), all the three approaches achieve higher maximum task latency. This is because,



(a) Maximum Latency (b) Average Accuracy

Figure 6: Effect of the number of categories l.



(a) Maximum Latency (b) Average Accuracy

Figure 7: Effect of the specific quality value range $[q^-, q^+]$.

if there are more tasks, each task will have relatively fewer workers to assign, which prolongs the latencies of tasks.

RANDOM always has higher latency than our RBS approach, followed by BBS. Here, the maximum latency of BBS remains low, and only slightly increases with more tasks. The reason has been discussed in Section 5.2.

Figure 4(b) illustrates the average accuracies of three approaches, with different m values. Since BBS always chooses a minimum set of workers with the highest category accuracies, in most cases, the task accuracies of BBS are higher than the other two approaches. When the number, m, of tasks becomes large, RBS achieves higher accuracies of tasks than RANDOM. Nonetheless, from the figure, RBS and BBS approaches can achieve high task accuracies (i.e., $91\% \sim 95\%$).

Effect of the Number, n, of Workers. Figure 5 shows the experimental results, where the number, n, of workers changes from 100 to 500, and other parameters are set to their default values. For the maximum latencies shown in Figure 5(a), when the number, n, of worker increases, the maximum latencies of three algorithms decrease. This is because, with more workers, each task can be assigned with more workers (potentially with lower latencies). Since the quality thresholds of tasks are not changing, with more available workers, the maximum latencies thus decrease. Similarly, BBS can maintain a much lower maximum latency than the other two algorithms. The maximum latencies of RBS are higher than that of BBS, but lower than RANDOM. Both RBS and BBS are better than the RANDOM method. For the average accuracies in Figure 5(b), our RBS and BBS algorithms can achieve high average accuracies (i.e., $93.5\% \sim 95\%$). The average accuracies of BBS are slightly higher than that of RBS, and that of RANDOM is the lowest.

Effect of the Number, l, of Categories. Figure 6 varies the number, l, of categories from 5 to 40, where other parameters are set by default. From Figure 6(a), we can see that, our RBS and BBS approaches can both achieve low maximum latencies, with different l values. Similar to previous results, BBS can achieve the lowest maximum latencies among three approaches, and RBS is better than RANDOM. Moreover, in Figure 6(b), with different l values, the accuracies of BBS and RBS remain high (i.e., $93\% \sim 96\%$), and are better than that of the other two algorithms.

Effect of the Range of the Quality Threshold $[q^-, q^+]$. Figure 7 shows the performance of RBS, BBS, and RANDOM approaches, where the range, $[q^-, q^+]$, of quality thresholds, q_i, increases from $[0.8, 0.85]$ to $[0.95, 0.97]$, and other parameters are set to their default values. Specifically, as depicted in Figure 7(a), when the range of the quality threshold increases, the maximum latencies of the three tested algorithms also increase. The reason is that, with higher quality threshold q_i, each task needs more workers to be satisfied (as shown by Corollary 3.2). Similarly, BBS can achieve much lower maximum latencies than that of RBS and RANDOM. Further, RBS is better than RANDOM, w.r.t. the maximum latency.

In Figure 7(b), when the range of q_i increases, the average accuracies of all the three algorithms also increase. This is because, when q_i increases, each task needs more workers to satisfy its quality threshold (as shown by Corollary 3.2), which makes the average accuracies of tasks increase. Similar to previous results, our two approaches, BBS and RBS, can achieve higher average accuracies than RANDOM.

We also tested data sets with other parameters (e.g., different distributions of workers/tasks on categories), and omit the similar results due to the space limitation.

6. RELATED WORK

Crowdsourcing has been well studied by different research communities (e.g., the database community), and widely used to solve problems that are challenging for computer (algorithms), but easy for humans (e.g., sentiment analysis [29] and entity resolution [35]). In the databases area, CrowdDB [15] and Qurk [27] are designed as crowdsourcing incorporated databases; CDAS [26] and iCrowd [14] are systems proposed to achieve high quality results with crowds; gMission [10] and MediaQ [22] are general spatial crowdsourcing systems that extend crowdsourcing to the real world, where workers need to physically move to specific locations to conduct tasks. Due to intrinsic error rates of humans, crowdsourcing systems always focus on achieving high-quality results with minimum costs. To guarantee the quality of the results, each task can be answered by multiple workers, and the final result is aggregated from answers with voting [14, 9] or learning [26, 20] methods. Li et al. [25] provided a survey about the crowdsourcing and its variants in the databases area.

Due to the diversity of the workers and their autonomous participation style in existing crowdsourcing markets (e.g., Amazon Mechanical Turk (AMT) [1] and Crowdflower [2]), the quality and completion time of crowdsourcing tasks cannot always be guaranteed. For example, in AMT, the latency of finishing tasks may vary from minutes to days [15, 23]. Some difficult tasks are often ignored by workers, and left uncompleted for a long time. Recently, several works [16, 18, 30, 34] focused on reducing the completion time of tasks. In [30, 34], the authors designed algorithms to reduce the latencies of tasks for specific jobs, such as rating and filtering records, and resolve the entities with crowds. The proposed techniques for specific tasks, however, cannot be used for general crowdsourcing tasks, which is the target of our FROG framework.

Gao et al. [16] leveraged the pricing model from prior works, and developed algorithms to minimize the total elapsed time with user-specified monetary constraint or to minimize the total monetary cost with user-specified deadline constraint. They utilized the decision theory (specifically, Markov decision processes) to dynamically modify the prices of tasks. Daniel et al. [18] proposed a system, called CLAMShell, to speed up crowds in order to achieve consistently low-latency data labeling. They analyzed the sources of labeling latency. To tackle the sources of latency, they designed several techniques (such as straggler mitigation to assign the delayed tasks to multiple workers, and pool maintenance) to improve the average worker speed and reduce the worker variance of the retainer pool.

Different from the existing works [16, 30, 34, 6, 18, 7], our FROG framework adopts the server-assigned-task (SAT) mode (instead of the worker-selected-task (WST) mode in prior works) for

general crowdsourcing tasks (rather than specific tasks), and focuses on both reducing the latencies of all tasks and improving the accuracy of tasks under the SAT mode (instead of either latency or accuracy under the WST mode). In our FROG framework, the task scheduler module actively assigns workers to tasks with high reliability and low latency, which takes into account response times and category accuracies of workers, as well as the difficulties of tasks (not fully considered in prior works). We also design two novel scheduling approaches, request-based and batch-based scheduling. Different from prior works [14, 18] that simply filtered out workers with low accuracies, our work utilizes all possible worker labors, by scheduling difficult/urgent tasks to high-accuracy/fast workers and routing easy and not urgent tasks to low-accuracy workers.

Moreover, Bernstein et al. [6] proposed the retainer model to hire a group of workers waiting for tasks, such that the latency of answering crowdsourcing tasks can be dramatically reduced. Bernstein et al. [7] also theoretically analyzed the optimal size of the retainer model using queueing theory for realtime crowdsourcing, where crowdsourcing tasks come individually. These models may either increase the system budget or encounter the scenario where online workers are indeed not enough for the assignment during some period. In contrast, with the help of smart devices, our FROG framework has the capability to invite offline workers to do tasks, which can enlarge the public worker pool, and enhance the throughput of the system. In particular, our notification module in FROG can contact workers who are not online via smart devices, and intelligently send invitation messages only to those available workers with high probabilities. Therefore, with the new model and different goals in our FROG framework, we cannot directly apply techniques in previous works to tackle our problems (e.g., FROG-TS and EWN).

7. CONCLUSION

The crowdsourcing has played an important role in many real applications that require the intelligence of human workers (and cannot be accurately accomplished by computers or algorithms), which has attracted much attention from both academia and industry. In this paper, inspired by the accuracy and latency problems of existing crowdsourcing systems, we propose a novel *fast and reliable crowdsourcing* (FROG) framework, which actively assigns workers to tasks with the expected high accuracy and low latency (rather than waiting for autonomous unreliable and high-latency workers to select tasks). We formalize the FROG task scheduling (FROG-TS) and efficient worker notifying (EWN) problems, and proposed effective and efficient approaches (e.g., request-based, batch-based scheduling, and smooth KDE) to enable the FROG framework. Through extensive experiments, we demonstrate the effectiveness and efficiency of our proposed FROG framework on both real and synthetic data sets.

8. REFERENCES

[1] Amazon mechanical turk [online]. Available: `https://www.mturk.com/mturk/welcome`.

[2] Crowdflower [online]. Available: `https://www.crowdflower.com`.

[3] Wechat [online]. Available: `http://www.wechat.com/en/`.

[4] A. Agresti and M. Kateri. *Categorical data analysis*. Springer, 2011.

[5] A.-L. Barabasi. Linked: How everything is connected to everything else and what it means. *Plume Editors*, 2002.

[6] M. S. Bernstein, J. Brandt, and R. C. Miller. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *UIST*, 2011.

[7] M. S. Bernstein, D. R. Karger, R. C. Miller, and J. Brandt. Analytic methods for optimizing realtime crowdsourcing. *arXiv preprint arXiv:1204.2995*, 2012.

[8] L. Breiman, W. Meisel, and E. Purcell. Variable kernel estimates of multivariate densities. *Technometrics*, 1977.

[9] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask?: jury selection for decision making tasks on micro-blog services. *PVLDB*, 2012.

[10] Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, Y. Tong, and C. J. Zhang. gmission: A general spatial crowdsourcing platform. *PVLDB*, 2014.

[11] P. Cheng, X. Lian, X. Jian, and L. Chen. Frog: A fast and reliable crowdsourcing framework [technical report]. Available:`http://haidaoxiaofei.github.io/d/frog.pdf`.

[12] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *SIGKDD*, 2011.

[13] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi. The anatomy of a scientific rumor. *Scientific reports*, 2013.

[14] J. Fan, G. Li, B. C. Ooi, K.-l. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, 2015.

[15] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[16] Y. Gao and A. Parameswaran. Finish them!: Pricing algorithms for human computation. *PVLDB*, 2014.

[17] M. R. Gary and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[18] D. Haas, J. Wang, E. Wu, and M. J. Franklin. Clamshell: speeding up crowds for low-latency data labeling. *PVLDB*, 2015.

[19] P. G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS: Crossroads*, 2010.

[20] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *Advances in neural information processing systems*, 2011.

[21] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *SIGGIS*, 2012.

[22] S. H. Kim, Y. Lu, G. Constantinou, C. Shahabi, G. Wang, and R. Zimmermann. Mediaq: mobile multimedia management system. In *MMSys*, 2014.

[23] A. Kittur, E. H. Chi, and B. Suh. Crowdsourcing user studies with mechanical turk. In *SIGCHI*, 2008.

[24] S. J. Leon. *Linear algebra with applications*. Macmillan New York, 1980.

[25] G. Li, J. Wang, Y. Zheng, and M. Franklin. Crowdsourced data management: A survey. 2016.

[26] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: a crowdsourcing data analytics system. *PVLDB*, 2012.

[27] A. Marcus, E. Wu, D. R. Karger, and S. Madden. Crowdsourced databases: Query processing with people. CIDR, 2011.

[28] L. Mo, R. Cheng, B. Kao, X. S. Yang, C. Ren, S. Lei, D. W. Cheung, and E. Lo. Optimizing plurality for human intelligence tasks. In *CIKM*, 2013.

[29] S. M. Mohammad and P. D. Turney. Crowdsourcing a word–emotion association lexicon. *Computational Intelligence*, 2013.

[30] A. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. *PVLDB*, 2014.

[31] M. Rosenblatt et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 1956.

[32] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR*, 2002.

[33] B. W. Silverman. *Density estimation for statistics and data analysis*. CRC press, 1986.

[34] V. Verroios and H. Garcia-Molina. Entity resolution with crowd errors. In *ICDE*, 2015.

[35] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 2012.

[36] P. Welinder and P. Perona. Online crowdsourcing: Rating annotators and obtaining cost-effective labels. In *CVPRW*, 2010.

[37] M. L. Yiu and N. Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *PVLDB*, 2007.

APPENDIX

A. EXPECTED ACCURACY OF MULTI-CHOICES TASK

Majority voting with multiple choices. Given a task t_i in category c_l and a set of workers W_i assigned to it, when we use majority voting with R choices, the expected accuracy of tasks t_i can be calculated as follows:

$$\Pr(W_i, c_l) = \sum_{x=\lceil \frac{k}{R} \rceil, |W_{i,x}| \text{is max}}^{k} \sum_{W_{i,x}} \Big(\prod_{w_j \in W_{i,x}} \alpha_{jl} \prod_{w_j \in W_i - W_{i,x}} (1-\alpha_{jl}) \Big),$$

where $W_{i,x}$ is a subset of W_i with x elements.

Weighted Majority voting with multiple choices. Given a task t_i in category c_l and a set of workers W_i assigned to it, when we use weighted majority voting with R choices, the expected accuracy of tasks t_i can be calculated as follows:

$$\Pr(W_i, c_l) = \sum_{x=\lceil \frac{k}{R} \rceil, Weight(W_{i,x}) \text{is max}}^{k} \sum_{W_{i,x}} \Big(\prod_{w_j \in W_{i,x}} \alpha_{jl} \prod_{w_j \in W_i - W_{i,x}} (1-\alpha_{jl}) \Big),$$

where $W_{i,x}$ is a subset of W_i with x elements, and $Weight(W)$ is the weight of a given worker set W.

Half voting with multiple choices. Half voting only return the results selected by more than half workers. Given a task t_i in category c_l and a set of workers W_i assigned to it, when we use half voting with R choices, the expected accuracy of tasks t_i can be calculated as follows:

$$\Pr(W_i, c_l) = \sum_{x=\lceil \frac{k}{2} \rceil}^{k} \sum_{W_{i,x}} \Big(\prod_{w_j \in W_{i,x}} \alpha_{jl} \prod_{w_j \in W_i - W_{i,x}} (1 - \alpha_{jl}) \Big),$$

where $W_{i,x}$ is a subset of W_i with x elements. Half voting is effective when there are more than two choices and the expected accuracy of each task is calculated by same equation same with that of majority voting with two choices.

B. HARDNESS OF THE FROG-TS PROBLEM

Theorem B.1. *(Hardness of the FROG-TS Problem) The problem of FROG Task Scheduling (FROG-TS) is a NP-hard problem.*

PROOF. We prove the lemma by a reduction from the multiprocessor scheduling problem (MSP). A multiprocessor scheduling problem can be described as follows: Given a set J of m jobs where job j_i has length l_i and a number of m processors, the multiprocessor scheduling problem is to schedule all jobs in J to m processors without overlapping such that the time of finishing all the jobs is minimized.

For a given multiprocessor scheduling problem, we can transform it to an instance of FROG-TS problem as follows: we give a set T of m tasks and each task t_i belongs to a different category c_i and the specified accuracy is lower than the lowest category accuracy of all the workers, which means each task just needs to be answered by one worker. For n workers, all the workers have the same response time $r_i = l_i$ for the tasks in category c_i, which leads to the processing time of any task t_i is always r_i no matter which worker it is assigned to.

As each task just needs to be assigned to one worker, this FROG-TS problem instance is to minimize the maximum completion time of task t_i in T, which is identical to minimize the time of finishing all the jobs in the given multiprocessor scheduling problem. With this mapping it is easy to show that the multiprocessor scheduling problem instance can be solved if and only if the transformed FROG-TS problem can be solved.

This way, we reduce MSP to the FROG-TS problem. Since MSP is known to be NP-hard [17], FROG-TS is also NP-hard, which completes our proof. □