The Thesis Committee for Sukanya Thapa
certifies that this is the approved version of the following thesis:

Enhancing Worker Management and Supporting External Tasks in Crowdsourced Data Labeling

SUPERVISING COMMITTEE:

Matthew Lease, Supervisor

Ying Ding, Second Reader

Enhancing Worker Management and Supporting External Tasks in Crowdsourced Data Labeling

by

Sukanya Thapa

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Information Science

The University of Texas at Austin

December 2023

Abstract

Enhancing Worker Management and Supporting External Tasks in Crowdsourced Data Labeling

Sukanya Thapa, MSIS
The University of Texas at Austin, 2023

SUPERVISOR: Matthew Lease

Human data labeling is key to training supervised machine learning (ML) models. We propose a new software infrastructure layer to augment capabilities of Amazon's SageMaker Ground Truth (GT) data labeling platform. Whereas crowdsourced annotation via Amazon Mechanical Turk (MTurk) is well-established, Amazon's more recent GT platform is less known but specifically designed to support ML annotation. Differentiating features include a curated "public crowd" sourced from MTurk, and integrating human labeling into Amazon's broader SageMaker ML tool suite, which provides an end-to-end pipeline for training and deploying ML services.

Key features of our software layer include: 1) continuous worker performance monitoring wrt. Requester gold labels; 2) automatically restricting task access when performance standards are not met; 3) geographic-based restriction of task access to US-based workers; and 4) the ability to conduct external tasks off-platform while sourcing workers from GT and continuing to use GT's payment system. Our design seeks to streamline Requester experience with minimal changes, and to utilize a sustainable software design to ease long-term management, extension, and maintenance. More generally, design goals center on promoting efficient, user-friendly, and quality-focused data labeling with crowdsourced annotators.

Table of Contents

Chapter 1: Introduction

Human annotation of data is fundamental to training supervised machine learning models. With the release of Amazon's Mechanical Turk (MTurk)[1] platform nearly two decades ago, crowdsourced annotation swiftly became popular due to its efficiency, cost-effectiveness, and the ability to engage a global workforce. More recently, Amazon's SageMaker Ground Truth (GT)[2] platform was launched to focus more specifically on data annotation for machine learning (ML), with a variety of differentiating features in comparison to MTurk. For example, one can choose between three types of human workforces for annotation: public crowdsourced workers (curated from MTurk), third-party vendors (sourced from a marketplace[3]), or private (i.e., "bring your own") workers. GT is also integrated with the broader Amazon SageMaker[4] suite for machine learning, thereby integrating data annotation into an end-to-end pipeline for training and deploying ML services. For further discussion of differences between GT vs. MTurk, see Chapter 2.5 and Table 2.1.

Most germane to our work, GT strives to simplify the *Requester* experience by exposing fewer controls to the Requester. While this motivation is laudable, some Requesters may prefer to retrain access to some of the controls provided by MTurk while also taking advantage of other different features of GT (e.g., integration with SageMaker), in order to have the best of both worlds. Of specific interest here, GT lacks Requester controls to: 1) further curate its public workforce (i.e., by screening specific workers in or out); 2) to restrict tasks to specific geographic regions (e.g., the US); and 3) to conduct external tasks off-platform while sourcing and paying workers via GT. In regard to external tasks, it is common with MTurk to execute more flexible

[1]`https://www.mturk.com/`
[2]`https://aws.amazon.com/sagemaker/groundtruth/`
[3]`https://aws.amazon.com/marketplace/`
[4]`https://docs.aws.amazon.com/sagemaker/`

task designs via one's own webserver, or to use third-party survey platforms (such as Qualtrics or SurveyMonkey) to preserve a constant task design across workers sourced from different workforce providers.

In this work, we seek to bridge the gap between the strengths of each respective platform, MTurk and GT. To achieve this, we authored a novel *Worker Quality Control Engine* atop GT that augments it with a set of MTurk-like features, including:

- **Continuous worker performance monitoring**. This is achieved via Requester-provided 1) task gold labels; and 2) a task-specific accuracy function (which allows for flexible comparison between gold labels and worker responses than simple exact match).

- **Automatic enforcement of accuracy standards.** When Requester-specified performance standards are not met, task access is immediately blocked. This prevents access both to further data instances within a current, ongoing task and to future tasks (from any Requester using our system) having equal or stricter accuracy requirements. This functions in the background to limit task access to both historically and currently poorly-performing workers while minimizing effort for the Requester to take advantage of this workforce management functionality.

- **Geographic-based restriction to tasks.** Specifically, we provide support for restrict tasks to US-based workers, though our approach, based on an open timezone library, could be easily extended to support restriction to other geographic regions.

- **External task support.** Using our system, Requesters can conduct tasks outside of GT while still sourcing and paying workers via GT. Specifically, we implement support for conducting Qualtrics surveys with GT workers. Our implementation could also be further generalized in future work to support other survey platforms.

The remainder of this thesis is organized as follows. In Chapter 2, we provide an introduction to the GT platform, discussing its capabilities and limitations. Chapter 3 outlines the design goals for our proposed enhanced system, highlighting key implementation challenges and potential resolutions. In Chapter 4, we delve into the architecture of the enhanced system and its data workflows. Chapter 5 focuses on overcoming limitations in GT to accommodate external Human Intelligence Tasks (HITs) and explores the integration of Qualtrics surveys onto the GT platform. In Chapter 6.1, we reflect on our work, its limitations, and implications for future work. Finally, we conclude in Chapter 6.2.

Chapter 2: Sagemaker Ground Truth: Capabilities and Limitations

Chapter 1 provided a brief introduction to Amazon's SageMaker Ground Truth (GT) labeling platform and some of its differentiating features vs. the better known Mechanical Turk (MTurk) platform. In this chapter, we introduce and discuss the GT platform in greater detail, including its capabilities and limitations. This lays the groundwork for understanding the novel *Worker Quality Control Engine* presented in this thesis as a layer atop GT. We briefly note in passing here that GT also provides a library of sample task UIs [1] to ease task creation, as well as a video [2] for getting started with use of the GT platform.

2.1 Overview of Existing System

Figure 2.1 illustrates a typical GT workflow. After presenting this workflow, we precede in Section 2.2 to further describe system concepts and vocabulary.

As shown at the left of the figure (#1), a Requester initiates a labeling job by configuring essential parameters. This involves providing the S3 location for the input manifest, specifying a Worker Task Template written in Crowd HTML and Liquid Template language, and implementing a Pre-Labeling Lambda Function to process data before dispatching it to the public workforce. Furthermore, a Post-Labeling Lambda Function manages the processing of labeled data upon job completion.

Once configured, the job is launched; see "Launch Labeling Job" (#2) in **Figure 2.1**. This dispatches it to MTurk, where Crowd Workers (#3) specifically curated for GT accept and complete the task (#4), receiving compensation upon successful completion. The labeled data then returns to GT from MTurk (#5). Subsequently,

[1]`https://github.com/aws-samples/amazon-sagemaker-ground-truth-task-uis`

[2]`https://www.youtube.com/watch?v=_FPI6KjDlCI`

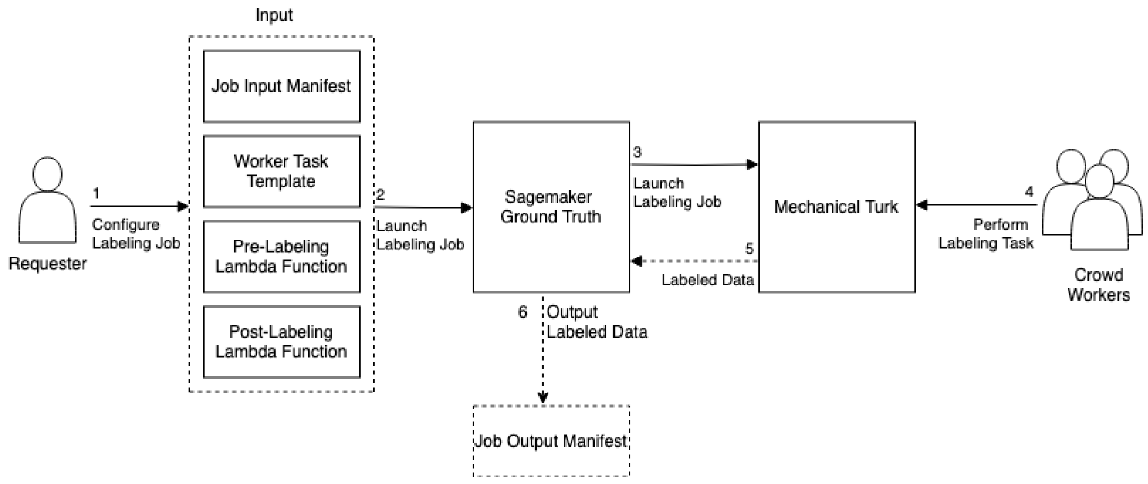


Figure 2.1: Overview of Existing System

GT processes the data further using the Post-Labeling Lambda Function to generate an output manifest (#6).

2.2 SageMaker Ground Truth: Concepts and Vocabulary

1. **Crowd Workers**: While GT provides access to three different workforces (public, vendor, and private), we focus on the public workforce in our implementation. Public workers are sourced from MTurk and find and complete annotation tasks via the MTurk platform as usual. However, only a curated subset of MTurk's workforce receives qualifications to access GT tasks, which are posted on MTurk under the Requester handle `MLDataLabeler`.

2. **Requesters**: Individuals (e.g., researchers, practioners, etc.) who initiate labeling jobs. Requesters configure job parameters, such as input manifests and Worker Task Templates, to facilitate the labeling process.

3. **Labeling Job**: The process aspects of assigning labeling tasks to workers. A job includes the configuration, execution, and post-processing stages.

4. **Labeling Task**: A specific data instance to be labeled (or unit of work to be completed, if a task page contains multiple data instances to be annotated) within a larger labeling job. This corresponds to a Human Intelligence Task (HIT) in MTurk vocabulary [3].

5. **Assignment**. Whenever a worker accepts a labeling task (i.e., agrees to perform it), this is known as an *assignment* of the given worker to the given task. A labeling job may specify that each of its tasks be completed by multiple workers redundantly, e.g., for the purposes quality control. See MTurk vocabulary [4].

6. **AWS Lambda Functions** [5]: Serverless computing services that allow the execution of code without the need for provisioning or managing servers. In the context of SageMaker GT, they perform data processing before and after the labeling process.

   (a) **Pre-Labeling Lambda Function**: Responsible for pre-processing data before dispatching it to the workforce.

   (b) **Post-Labeling Lambda Function**: Manages processing after the completion of labeling tasks.

7. **S3 Bucket**[6]: Amazon's Simple Storage Service (S3) allows data storage and retrieval. In the context of SageMaker GT, S3 is used to store input manifests and other essential data for labeling jobs.

8. **Manifests**: in the context of GT, a manifest is a structured document that serves as a component in the configuration of labeling jobs. There are two primary types of manifests:

[3]`https://docs.aws.amazon.com/AWSMechTurk/latest/RequesterUI/mechanical-turk-concepts.html`

[4]`https://docs.aws.amazon.com/AWSMechTurk/latest/RequesterUI/mechanical-turk-concepts.html`

[5]`https://docs.aws.amazon.com/lambda/`

[6]`https://docs.aws.amazon.com/s3/`

   (a) **Input Manifest**: Configures the S3 location of input data for the labeling job. The input data provided to GT is subsequently dispatched to your workers for labeling.

   (b) **Output Manifest**: Generated post-labeling, providing metadata about the labeled dataset.

9. **Crowd HTML** [7]: A markup language used in Worker Task Templates. It providesa structured way to design the appearance and layout of tasks presented to crowd workers.

10. **Liquid Template Language** [8]: Liquid is a templating language used in conjunction with Crowd HTML. It allows for dynamic content generation and logic within the Worker Task Template, enabling flexibility in designing and customizing tasks. See GT documentation on adding automation with Liquid [9].

11. **Worker Task Template**: A structured template, authored in Crowd HTML and Liquid Template language, shaping the tasks assigned to crowd workers.

2.3 Additional AWS Building Blocks

1. **CloudWatch**[10]: a monitoring and observability service that offers real-time insights into the operational health of applications and resources. Our system uses CloudWatch to track and analyze logs of services involved in the lifecycle of a labeling job.

2. **DynamoDB**[11]: a NoSQL database service. Our system uses DynamoDB to store and manage performance history of workers.

[7] https://docs.aws.amazon.com/sagemaker/latest/dg/sms-ui-template-reference.html

[8] https://liquidjs.com/index.html

[9] https://docs.aws.amazon.com/sagemaker/latest/dg/sms-custom-templates-step2.html#sms-custom-templates-step2-automate

[10] https://aws.amazon.com/cloudwatch/

[11] https://aws.amazon.com/dynamodb/

2.4 Limitations of Existing System

While GT offers an efficient data labeling process, providing a quick and straightforward method for acquiring labeled data, it lacks some valuable features found in MTurk. This includes External HITs: tasks or jobs created by a Requester that are hosted on an external website or application rather than within the MTurk platform. Workers access and perform these tasks on external platforms, submitting a verification code back to MTurk to obtain payment. This mechanism allows Requesters to utilize the MTurk marketplace for worker recruitment while conducting tasks on their own systems with specific requirements or interfaces[12].

The absence of other features, such as the ability to block subpar-performing workers and selectively target specific crowd workers based on criteria like country presents a significant drawback. Our research team at the lab, initially drawn to Ground Truth for its features, encountered a substantial challenge—around 90% of labeled data had to be discarded due to poor quality. This highlights a critical limitation in GT's functionality, necessitating enhanced control mechanisms and quality management for optimal utility in machine learning applications. To overcome this, we developed a customized solution on top of Ground Truth to effectively monitor workers' performance and block those who fall short.

2.5 Difference between MTurk and GT

In this section, we explore key distinctions between MTurk and GT, presenting a comprehensive overview of their differences as shown in Table 2.1.

[12]`https://docs.aws.amazon.com/AWSMechTurk/latest/AWSMturkAPI/ApiReference_ExternalQuestionArticle.html`

Feature	**MTurk**	**Sagemaker GT**
Sandbox Testing	Provides a sandbox for testing task interfaces before launching jobs with real workers.	No free sandbox; a private workforce can be used for a fee. The "Preview" button allows inspection of the first 10 items, and selecting "Submit" in "Preview" mode shows a sample payload.
Worker Qualifications	Allows assignment of qualifications to filter workers based on skills or attributes.	Does not support worker qualifications directly. Assumes the use of a private workforce with specific skills. Already filters the MTurk workforce for Requester convenience.
Rejection and Bonuses Work	Requesters can reject HITs and provide bonuses for high-quality work.	Does not allow rejection or bonusing. It assumes the curated public workforce will deliver quality work, and any issues must be addressed through task refinement.
Worker-Requester Communication	MTurk workers can contact the Requester with questions about their labeling job.	Public workers from MTurk on GT cannot directly contact the GT Requester with questions. When GT tasks are posted to the public workforce, the Requester is generically identified as MLDataLabeler, and any questions from workers do not reach the GT Requester. After labeling, the workerIDs associated with each label are available with the labeled data; however, the use of MTurk APIs through GT user IDs is restricted to contacting the workers who performed the labeling.
External HITs	MTurk supports external HITs, allowing tasks to be run on the Requester's website for maximum flexibility.	GT does not support external HITs. However, Crowd HTML in GT provides interface flexibility to embed the External HIT link. There is no built-in mechanism to verify user completion of the External HIT; compensation is triggered upon task submission.

Table 2.1: Differences between MTurk and Sagemaker GT

Chapter 3: Design Goals and Implementation Challenges

This chapter discusses our primary design goals, focusing on worker quality control, minimal disruption to existing workflows, and a user-friendly system design. We also address key implementation challenges, providing insights into our strategic resolutions for seamless integration and efficient worker management.

3.1 Design Goals

3.1.1 Enhance Workforce Management Controls and Automation

While MTurk provides fine-grained controls for managing the public workforce, GT provides a simplified Requester experience, striving to curate the workforce internally with fewer controls for Requesters. We seek to provide a an alternative balance with the best of both. In particular, we want to enhance the degree of worker management controls provided to Requesters in order to allow them to curate their workforce, screening workers in and out, akin to MTurk. However, we also want to simplify the Requester experience vs. MTurk in handling more of this within the system, based on parameters specified by Requesters. In particular, we seek to provide continuous monitoring of worker performance for custom tasks, based on Requester-provided gold labels, accuracy function, and minimum accuracy thresholds. When workers fail to meet minimum performance standards, the platform should restrict access to future tasks as well as additional instances within a current annotation task.

3.1.2 Minimal Changes to the Current GT Workflow for Requesters

We seek to seamlessly integrate our system with minimal changes to the existing GT workflow for Requesters. This approach prioritizes the Requester's familiarity with the current GT framework, enabling them to transition effortlessly and maintain efficiency in task design and data collection within the new GT framework.

3.1.3 A Simplified System Design Solution for Easier System Maintenance

We seek to design a user-friendly system architecture that prioritizes simplicity, ensuring ease of maintenance and seamless management for long-term sustainability.

3.1.4 Seamless Worker Experience for External Tasks on GT

Our objective is to seamlessly integrate External Tasks into GT, providing crowd workers with a user experience similar to MTurk External Tasks. This involves generating a completion code at the end of each external task, which workers are required to submit on MTurk for verification and subsequent payment.

3.2 Challenges and Solutions

In order to successfully implement this infrastructure in a way which addresses the design goals, the following challenges would need to be addressed;

3.2.1 Worker Identification

The first major challenge we encountered was the unique identification of workers performing the labeling task. Initially, we explored asking workers to sign-in to our system. We experimented with AWS Cognito [1], though another 3rd party sign-in service, such as Google or Facebook, might also be used. While this solution would support generalization across different GT workforces (public, private, and vendor), our particular focus was the public workforce (i.e., MTurk). For this reason, we found a more efficient solution: extracting MTurk Worker IDs directly from the URL embedded in the iframe of our tasks on MTurk. This simplified both our system

[1] `https://aws.amazon.com/cognito/`

architecture and the experience for Workers, though limited support to GT's public workforce, as noted above.

3.2.2 Configuring Jobs for Performance Monitoring

Another challenge was how to integrate an accuracy check mechanism without disrupting the Requester's focus on task design and data collection, ensuring a similar experience to the standard GT workflow. To address this, we utilized the `TAGS` fields in GT, which allowed us to specify additional input and configurations that Requesters can optionally provide when preparing their labeling jobs. The `TAGS` field became the avenue for incorporating an accuracy functions and associated metadata to facilitate accuracy checks. Furthermore, we implemented a pre-processing Lambda function that calls the SageMaker API to read the configuration tags and send them to our application before launching the job.

3.2.3 Blocking Poorly Performing Workers During Ongoing Jobs

We also needed to find a way to dynamically block workers from ongoing jobs if we detect that they are not meeting performance standards specified by the Requester (i.e., if their running accuracy falls below a given threshold). To overcome this challenge, we briefly interrupt the final task submission on MTurk to perform accuracy checks and prevent workers from undertaking any further tasks by overriding the default submission, ensuring data quality throughout the labeling process.

Note that our blocking mechanism is "worker-friendly" in several respects. First, as is standard in GT, workers are always paid for completed work; this differs from MTurk where work can be rejected without payment. Blocking merely prevents a worker from continuing to perform any additional labeling work on tasks launched via our system. Second, while we use the term "block", we do not use the MTurk API to block workers in that platform's sense of the term. In fact, worker blocking in our system is entirely internal to it and thus completely invisible to GT and MTurk. This means that worker blocking in our system has no negative repercussions for a worker's reputation on GT or MTurk. Workers accepting our tasks thus do not risk punitive actions (e.g., potential account suspension) that could result if blocking signals were conveyed back to the MTurk platform via its API. Of course, this lack of signaling back to MTurk also means that we do not contribute to MTurk platform reputation mechanisms that seek to detect and exclude truly bad actors.

3.2.4 Establishing a Mechanism to Validate Completion Codes for External Tasks

Given GT's absence of built-in support for external Tasks, implementing such functionality required the development of a method to generate completion codes on a third-party platform and communicate them to GT. In our context, particularly focusing on Qualtrics, we conducted experiments and discovered a solution involving the embedding of JavaScript code. This code is triggered during submission, facilitating the transmission of the completion code through an API. Workers are then expected to submit this code for verification, confirming task completion and enabling payment.

Chapter 4: Enhanced System Architecture and Workflow

In this chapter, we discuss the architecture of our system, designed.

4.1 Overview of Enhanced System

The process flow within our enhanced solution is illustrated in **Figure 4.1**, highlighting additional configurations and components integrated atop GT. These supplementary settings are incorporated in the "TAGS" field during the configuration of a labeling job. **Figure 4.4** shows a sample Configuration of TAGS field from the GT platform.

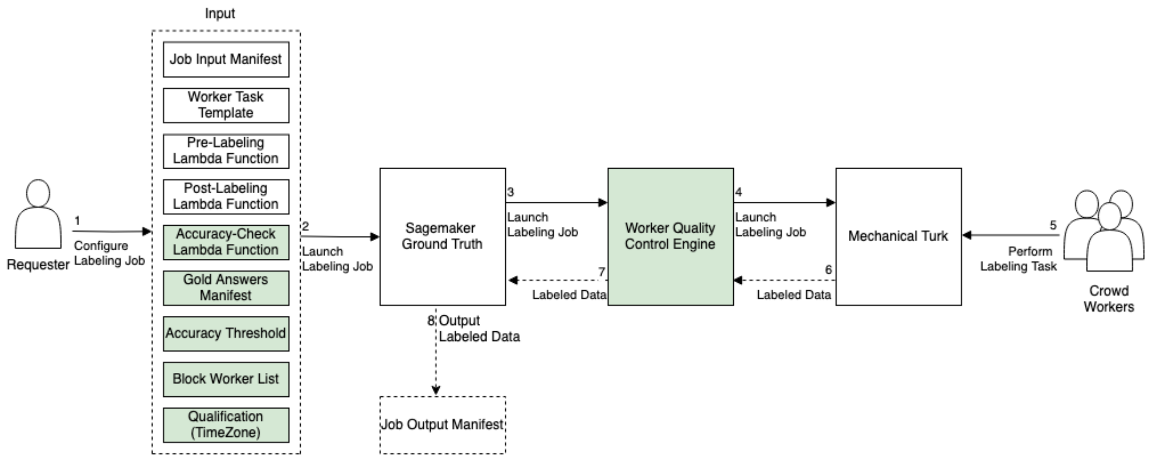


Figure 4.1: Overview of Enhance System

4.1.1 Accuracy Check Implementation

The Requester starts by implementing a accuracy check Lambda function specific to the labeling job and uploads a gold answers manifest necessary to an S3 bucket. AWS identifiers for lambda and S3 buckets used are added to GT in "TAGS" fields.

4.1.2 Blocking Poor Performing Workers Identified from Existing System

Requesters may optionally provide a “block list” of worker IDs who should be prevented from access task jobs. This list is uploaded to S3 in JSON format. AWS-specific unique locations for these resources are configured in the job. The AWS identifier of S3 buckets used is added to GT in ”Tag” fields.

4.1.3 Accuracy Threshold and Gold Answers Threshold

Requesters may optionally specify parameters to take advantage of continuous performance monitoring and restricting task access for workers who do not meet acceptance performance standards.

A `gold answers count threshold` specifies that accuracy checks should be performed only after the worker has labeled a minimum number of data instances. The intent here is to avoid premature blocking of workers in their initial work, when they may be learning and when relative frequency estimates of worker accuracy have the highest variance. While each job can specify a custom threshold, note that its implementation is global, referring to the minimum number of data items that a worker must label across all Requesters and their annotation tasks run on our engine.

Similarly, the `gold accuracy threshold` is specified on a per job basis but measured across the lifetime of all work completed by a given Worker across Requesters and jobs on our engine.

4.1.4 Qualification (i.e. Timezone)

One feature of the framework is the ability to restrict access to tasks to USA-workers, implemented by consulting the worker’s timezone as recorded by their web browser. Specifically, The “countries-and-timezones” npm library[1] has been used to obtain the list of timezones for the US, which, in turn, retrieves this data from the IANA timezones database[2]. Optionally, Requesters can specify a qualification for USA-based workers, allowing only those with matching browser timezones to access the tasks. A comprehensive list of timezones is included in Appendix 6.2.

4.1.5 Email Based Failure Notification

The Requester can optionally specify their email address to receive error alerts for any failure associated with their Accuracy Lambda Functions. AWS CloudWatch is configured for each Accuracy Lambda function to monitor the function logs, watch for any errors, and trigger notifications when a failure occurs.

4.2 Internal Architecture of Worker Quality Control (WQC) Engine

Once the labeling job is setup and launched, it proceeds to launch on GT using the public workforce (i.e., MTurk), largely mirroring the Requester experience of the standard GT system. Our Worker Quality Control Engine operates discreetly in the background. It is tasked with capturing worker IDs of crowd workers involved in the labeling task, actively monitoring their performance, and dynamically blocking those who do not meet specified performance standards.

Note that worker blocking is entirely internal to our engine and invisible to GT and MTurk; we do not use the MTurk API to block workers in the conventional sense. This also means that workers accepting our tasks do not risk punitive actions from Amazon (e.g., potential account suspension) that could result if blocking signals were conveyed back to the MTurk platform via its API. It’s crucial to emphasize that in the event of a failure in the Worker Quality Control Engine, the impact is

[1]https://www.npmjs.com/package/countries-and-timezones
[2]https://www.iana.org/time-zones

contained and does not disrupt the regular workflow. Workers won't be aware of the engine failure; they will seamlessly continue their tasks as the system transitions to the current workflow. Workers will receive their due payments, and the labeled data will be sent back to GT without interruption.

Our engine is designed as a network of interconnected components, fostering seamless communication among themselves, AWS services, and crowd workers. Therefore, to more rapidly develop and smoothly integrate the Engine we used AWS Amplify. More details about these software components and their interactions are explained below and shown in **Figure 4.2**.

This is built using React, the WQC Gateway serves as an interface that renders the Worker Task Template designed by Requesters on MTurk. The WQC plays a crucial role in coordinating interactions among various components of the engine. It captures Crowd Worker IDs, performs historical and current job accuracy checks for each worker on gold questions. Then if the Requester defined thresh-hold isn't met it coordinates the blocking of subpar workers.

4.2.1 Pre-Labeling Lambda Function

This function is invoked when a Labeling Job is launched. It reads accuracy-related metadata configured in the GT Labeling Job and passes it to the WQC Gateway, facilitating smooth communication within the engine.

4.2.2 Sagemaker APIs

In our solution, two critical APIs provided by GT play key roles. The ListTags API facilitates the retrieval of accuracy-related configurations incorporated into the TAGS field. Conversely, the RenderUITemplate API interprets the Liquid Template Language embedded within the CrowdHTML of the Worker Task Template and dynamically loads unlabeled data to it. This process ensures the seamless rendering of the template on web browsers, optimizing the user experience during the execution

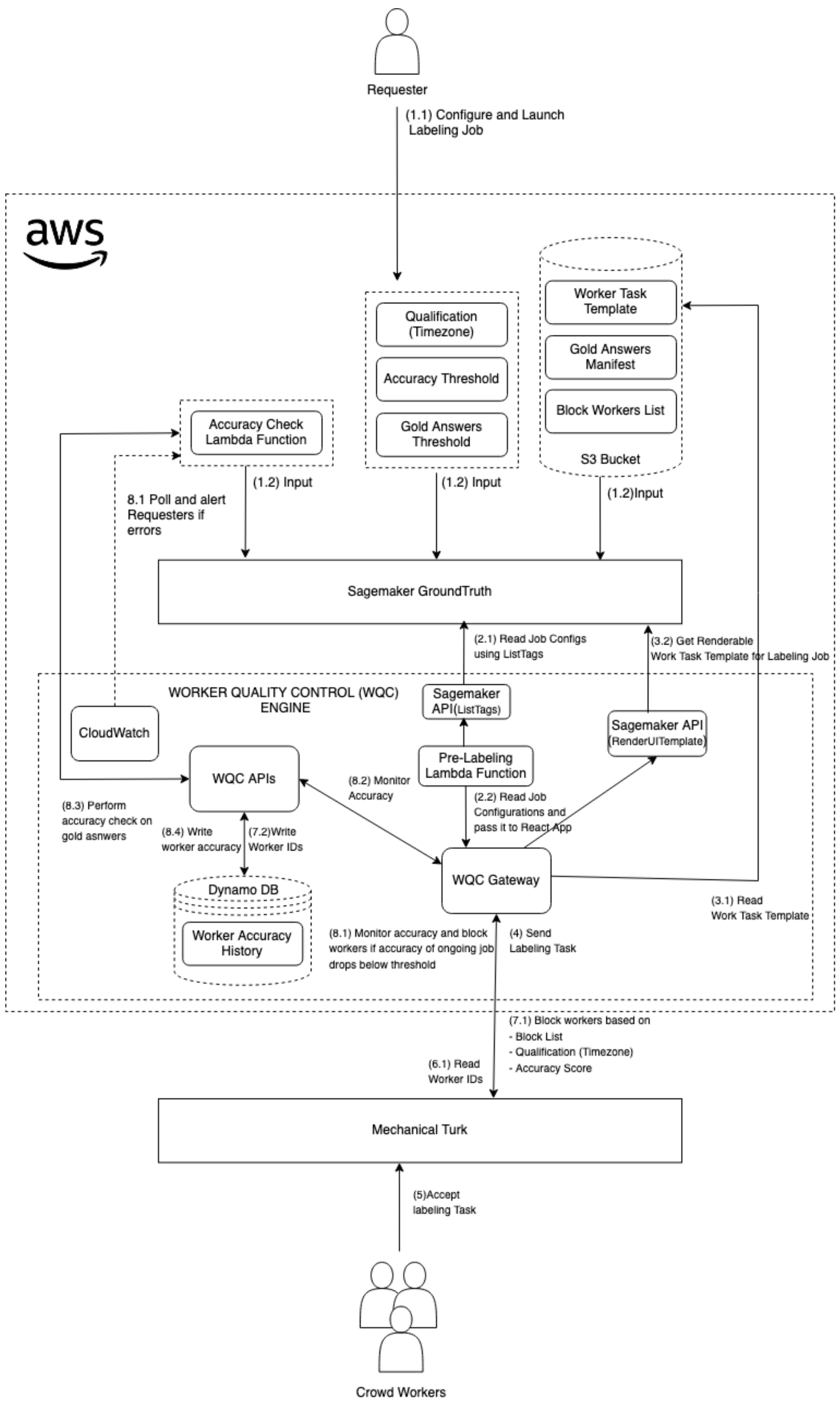

Requester
(1.1) Configure and Launch Labeling Job
aws
Qualification (Timezone)
Accuracy Threshold
Gold Answers Threshold
Worker Task Template
Gold Answers Manifest
Block Workers List
S3 Bucket
Accuracy Check Lambda Function
(1.2) Input
(1.2) Input
(1.2)Input
8.1 Poll and alert Requesters if errors
Sagemaker GroundTruth
(2.1) Read Job Configs using ListTags
(3.2) Get Renderable Work Task Template for Labeling Job
WORKER QUALITY CONTROL (WQC) ENGINE
CloudWatch
Sagemaker API(ListTags)
Sagemaker API (RenderUITemplate)
Pre-Labeling Lambda Function
WQC APIs
(8.2) Monitor Accuracy
(2.2) Read Job Configurations and pass it to React App
(8.3) Perform accuracy check on gold asnwers
(8.4) Write worker accuracy
(7.2)Write Worker IDs
Dynamo DB
Worker Accuracy History
WQC Gateway
(3.1) Read Work Task Template
(8.1) Monitor accuracy and block workers if accuracy of ongoing job drops below threshold
(4) Send Labeling Task
(7.1) Block workers based on
- Block List
- Qualification (Timezone)
- Accuracy Score
(6.1) Read Worker IDs
Mechanical Turk
(5)Accept labeling Task
Crowd Workers

Figure 4.2: Worker Quality Control (WQC) Gateway

of labeling tasks.

4.2.3 Worker Quality Control (WQC) APIs

This component contains Lambda Functions exposed through AWS API Gateway to process and update metrics for monitoring worker quality.

4.2.4 Worker Accuracy History

This component is hosted on DynamoDB; the Worker Accuracy History database stores crowd worker IDs, their cumulative count of correctly answered gold answers, and accuracy both overall and specific to each Labeling Job. This historical data is referenced to identify and block workers with poor-quality work.

4.3 Labeling Job Deployment

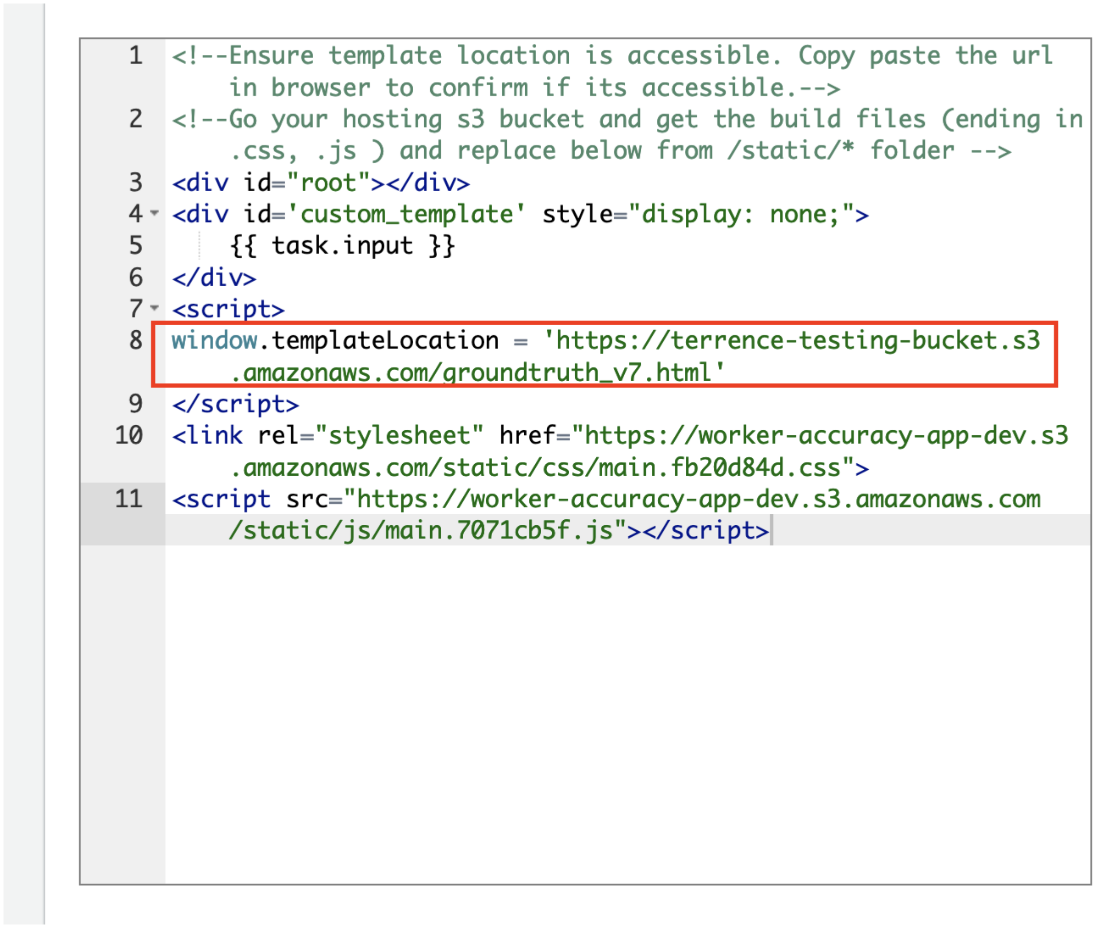
To illustrate how these components seamlessly interact, let's walk through the deployment of a labeling job on GT using the WQC Engine as illustrated in **Figure 4.2**.

4.3.1 Requester Initiates Job Configuration

1. The Requester begins by creating an accuracy lambda function and uploading a gold answers manifest, Worker Task Template, and a worker block list to the S3 bucket dedicated to accuracy (#1.1).

2. Expanding on the existing workflow, the Requester leverages resources from the previous step to configure the labeling job. This includes defining the accuracy lambda function, gold answers, and accuracy threshold in the TAGS field of the job configuration as illustrated in **Figure 4.4**. The Worker Task Template URL is inserted into the Custom editor of GT (#1.2). A sample custom editor is illustrated in **Figure 4.3**. The launch is subsequently initiated, and **Figure**

4.5 displays the resulting Worker Task on MTurk.



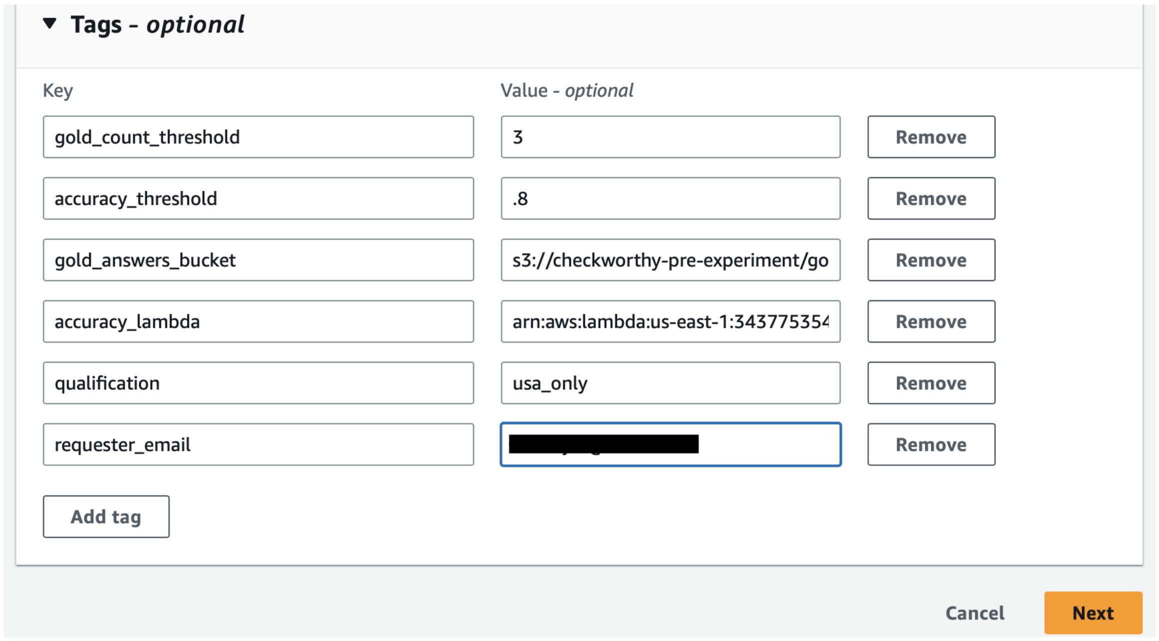
```
<!--Ensure template location is accessible. Copy paste the url
    in browser to confirm if its accessible.-->
<!--Go your hosting s3 bucket and get the build files (ending in
    .css, .js ) and replace below from /static/* folder -->
<div id="root"></div>
<div id='custom_template' style="display: none;">
    {{ task.input }}
</div>
<script>
window.templateLocation = 'https://terrence-testing-bucket.s3
    .amazonaws.com/groundtruth_v7.html'
</script>
<link rel="stylesheet" href="https://worker-accuracy-app-dev.s3
    .amazonaws.com/static/css/main.fb20d84d.css">
<script src="https://worker-accuracy-app-dev.s3.amazonaws.com
    /static/js/main.7071cb5f.js"></script>
```

Figure 4.3: GT Custom Editor

4.3.2 Pre-Labeling Lambda Function Configuration

1. The Pre-Labeling Lambda Function uses the Sagemaker API named ListTags to read configurations added to the TAGS field of the job configuration (#2.1).

2. The configurations are then passed to the WQC (WQC) Gateway (#2.2).



▼ **Tags - *optional***

Key	Value - *optional*
gold_count_threshold	3
accuracy_threshold	.8
gold_answers_bucket	s3://checkworthy-pre-experiment/go
accuracy_lambda	arn:aws:lambda:us-east-1:34377535
qualification	usa_only
requester_email	[redacted]

Figure 4.4: GT TAGS field

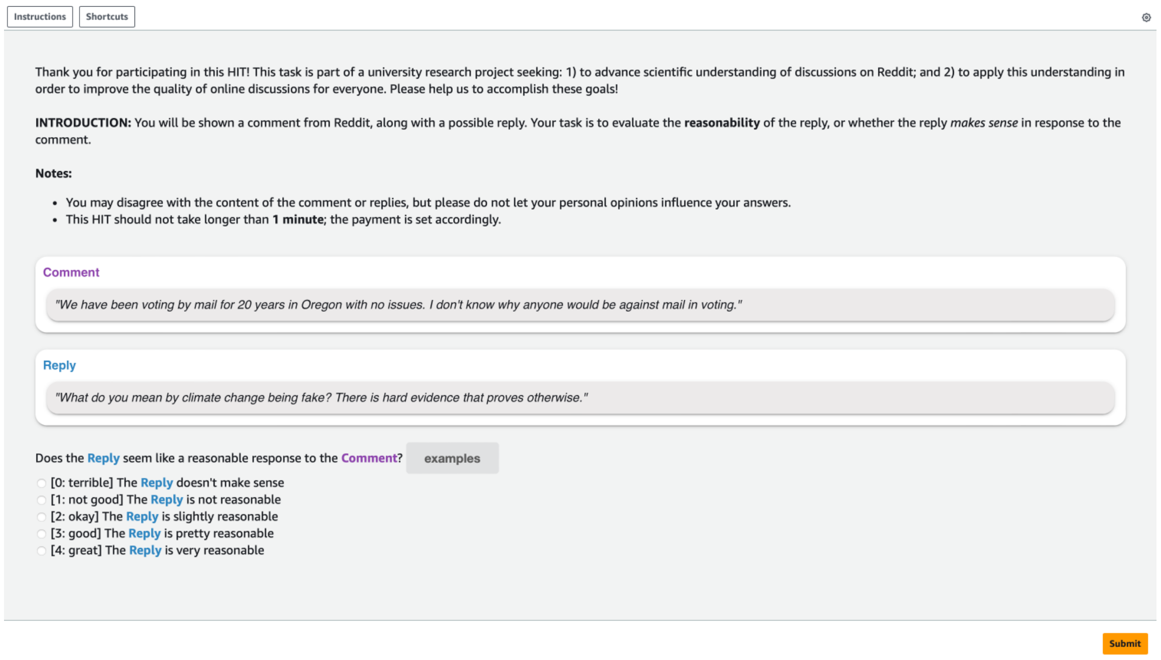
4.3.3 WQC Gateway Processing

1. The WQC Gateway retrieves the Worker Task Template location from the configuration (#3.1) and fetches it from that location.

2. Leveraging the Sagemaker API RenderUITemplate, the WQC Gateway dynamically compiles a Worker Task Template that can be rendered in a web browser. This process involves interpreting the Liquid Template Language embedded in the CrowdHTML (#3.2) and loading input data for labeling. Once the Worker Task Template is prepared, GT dispatches the labeling task to MTurk (#4). **Figure 4.5** illustrates a sample labeling job launched on MTurk.

4.3.4 Worker Interaction

1. Once workers accept a labeling task, WQC captures the worker ID from the web browser and performs several checks (#7.1).



Thank you for participating in this HIT! This task is part of a university research project seeking: 1) to advance scientific understanding of discussions on Reddit; and 2) to apply this understanding in order to improve the quality of online discussions for everyone. Please help us to accomplish these goals!

INTRODUCTION: You will be shown a comment from Reddit, along with a possible reply. Your task is to evaluate the **reasonability** of the reply, or whether the reply *makes sense* in response to the comment.

Notes:

- You may disagree with the content of the comment or replies, but please do not let your personal opinions influence your answers.
- This HIT should not take longer than **1 minute**; the payment is set accordingly.

Comment

"We have been voting by mail for 20 years in Oregon with no issues. I don't know why anyone would be against mail in voting."

Reply

"What do you mean by climate change being fake? There is hard evidence that proves otherwise."

Does the Reply seem like a reasonable response to the Comment?

- [0: terrible] The Reply doesn't make sense
- [1: not good] The Reply is not reasonable
- [2: okay] The Reply is slightly reasonable
- [3: good] The Reply is pretty reasonable
- [4: great] The Reply is very reasonable

Figure 4.5: Sample labeling Job - Work Task

2. It checks whether the worker is not part of the block list, verifies if the worker meets the qualification criteria (e.g., timezone), and evaluates the worker's current overall accuracy score. If the overall accuracy, based on past performance, is found to be below the threshold, the worker is not allowed to proceed.

3. If all criteria are met, it proceeds to write the Worker ID to the database (#7.2).

4.3.5 Task Completion and Accuracy Check

1. On completion of each task, an accuracy check is performed for the ongoing labeling job.

2. If the accuracy is found below the threshold, WQC blocks the users from proceeding further in the job. Workers are compensated for the tasks completed (#8.1).

3. The accuracy check is conducted by WQC APIs (#8.2), invoking the accuracy

lambda function to compare against the gold answers manifest provided by the Requester (#8.3).

4. Once the check is completed, the result is written back to the database (#8.4), and a cumulative accuracy score is calculated based on the number of correctly answered gold questions. A decision is then made on whether to block the worker for the ongoing Job (#8.1).

4.3.6 Accuracy Lambda Function Failure Monitoring

While the labeling job is in progress and accuracy checks are being performed for each task, AWS CloudWatch has been configured on the Accuracy Lambda Function to proactively poll the function's logs, look for any failures, and alert the requester in case of a failure (#8.1). The requester can optionally specify their email address to receive error alerts.

4.3.7 Communicating Blocking to Workers

Workers who do not meet Timezone-based Qualification criteria or minimum performance standards are blocked from task access. When either event occurs, they are shown the corresponding messages below:

Timezone *"Unfortunately, this task is only for the USA region. Therefore, you may not continue to perform this task. Please release the job."*

Performance *"Unfortunately, you have missed too many control questions. Consequently, you may not continue to perform this task. Please release the job."*

As shown above, both messages end by requesting that the worker release the job. This reflects a notable limitation of the system's inability to enforce job releases, thereby necessitating reliance on workers for voluntary compliance in releasing labeling jobs. This limitation is further discussed in Chapter 6.1.

Chapter 5: Qualtrics Integration to GT

In Chapter 2, we explored the distinct capabilities of MTurk and GT, with one notable difference being the absence of support for External HITs in GT. Building upon this exploration, this chapter addresses the limitation in GT to accommodate external Human Intelligence Tasks (HITs) and delves into the integration of Qualtrics surveys onto the GT platform.

5.1 Motivation for Qualtrics Integration

Third-party surveys, particularly those utilizing platforms like Qualtrics, come with established infrastructure, offering familiarity to both requesters and workers. These surveys actively seek participants, aligning with the participant availability on GT. Despite GT's lack of inherent support for external HITs, the incorporation of Crowd HTML provides valuable interface flexibility, enabling the embedding of External HIT links. However, a challenge emerges due to the absence of a built-in mechanism for verifying user completion of the External HIT, leading to compensation triggered solely upon worker submission. This integration aims to enhance efficiency in participant engagement, aligning with the participant search capabilities offered by a platform like GT.

5.2 Overview of Qualtrics Integration to GT

To address the challenge of verifying participant completion in the absence of a built-in mechanism, we developed a robust solution. Here is an overview of the integrated systems:

1. The Requester initiates the process by designing a survey (a labelling task) within the Qualtrics platform to capture participant data. A submission block

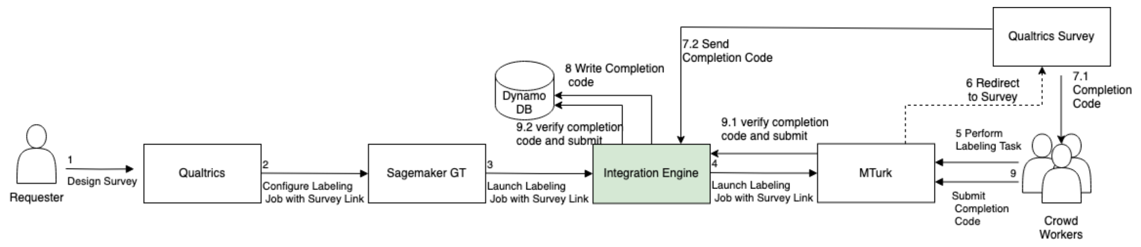


Figure 5.1: Overview of Qualtrics Integration to GT

is incorporated at the survey's conclusion, marking its completion. This block ensures participants are aware of the survey's end, generating a completion code and embedding JavaScript to send this code back to the Integration Engine (#1).

2. Once the survey is ready, its link is configured in sagemaker (#2).

3. We have developed a React-based engine named the Integration engine, facilitating the coordination of job launch with the survey link and capturing as well as verifying the completion code (#3-#4).

4. As the job becomes available on Mturk, workers accept and work on the task. Upon completing the survey, they receive a completion code, which they provide to the Mturk labeling job for verification and payment (#5-#7.1).

5. The completion code is also sent to the Integration engine by the survey, stored in the database (#7.1). When the worker submits the completion code, it's verified against the database (#9.1-#9.2). If valid, task completion is accepted, and the worker receives payment.

5.3 Challenges and Opportunities

In the GT platform, crowd workers receive compensation for individual tasks within a labeling job, with payment configurations ranging from $0.012 for low-complexity tasks to $1.20 for high-complexity tasks. However, the integration of a Qualtrics survey link into a GT labeling job consolidates multiple tasks within the job into a single survey, where each question corresponds to a GT task, resulting in a single, complex survey for data collection. Workers on GT are accustomed to task-specific compensation, and consolidating tasks into a single survey might deter participation, especially in surveys perceived as complex with compensation below their expectations.

Nevertheless, the appeal of this integration could be enhanced if Amazon improves incentives on GT or allows Requesters to configure compensation more flexibly in the future. This limitation underscores the significance of aligning survey complexity with worker compensation expectations for optimal participation and maintaining result quality.

Moreover, by leveraging the existing infrastructure, opportunities arise for potentially integrating other surveys, such as SurveyMonkey. Requesters can follow analogous steps, generating a completion code and transmitting it to our integration engine exposed through API. Workers can then follow the same streamlined process, presenting the code for job verification.

Chapter 6: Discussion and Conclusion

In the process of implementing and deploying our solution, several key reflections, limitations, and considerations have emerged, paving the way for a comprehensive discussion.

6.1 Discussion

1. **Single Threshold Configuration**: In our system, Requesters set an accuracy threshold to filter out poorly performing crowd workers. We have implemented two levels of filters to assess workers—examining their past performance and ongoing job performance. For instance, Requesters might decide on a specific accuracy threshold, such as 0.8, for a labeling task. When crowd workers accept the labeling job, this threshold is applied to restrict their participation based on their past performance. If they meet the threshold, demonstrating satisfactory past performance, they are allowed to proceed with the job. As the job progresses, this same threshold is consistently used to prevent workers from continuing in the job if they fall below the threshold. Looking ahead, there's potential to explore the incorporation of separate thresholds for these different aspects of the process.

2. **Mitigating Variability in Worker Evaluation**: Continuing with the previous point, our system employs two levels of evaluation—assessing both past performance and ongoing job performance—to filter out poor worker performance. Introducing gold questions in the labeling job allows us to evaluate performance against correctly answered gold questions. However, relying solely on this evaluation may lead to premature worker blocking due to the inherent variability in a limited sample size of correctly answered gold questions. The use of a small sample size, if employed in isolation, could introduce variability.

Therefore, our approach aims to mitigate this variability by implementing a minimum gold count in assessing crowd workers' performance. This measure allows workers to build a substantial performance history before facing potential early consequences. The minimum gold count criterion is set at 30 for past performance and is determined flexibly by Requesters for ongoing jobs. This ensures a more robust and reliable assessment, accounting for the potential variability inherent in small sample sizes.performance is evaluated.

3. **Impact of Blocking Mechanism on Worker Behavior on GT**: As highlighted in Chapter 2.5, MTurk allows Requesters to reject poor-quality HITs, a feature notably absent in GT. Consequently, Crowd workers on GT, sourced through MTurk, have grown accustomed to completing tasks without the looming possibility of rejection. The introduction of the blocking mechanism on GT represents a significant departure, providing a direct means to influence worker behavior. This raises a crucial question: does blocking effectively motivate workers to reach higher standards, or are we primarily identifying and retaining those who already demonstrate commendable behavior? Looking ahead, experimentation is imperative to comprehensively understand the impact of the blocking mechanism on worker behavior. This introduces a nuanced exploration of whether the platform can actively encourage workers to consistently deliver high-quality performance. The prospect of shaping worker behavior on GT introduces a dynamic element, necessitating ongoing inquiry and experimentation to optimize the platform's effectiveness in fostering a culture of excellence.

4. **Analyzing Job Acceptance Through a Transparency Experiment**: In our investigation into the impact of the blocking mechanism on worker acceptance, we conducted an experiment focusing on transparent communication about performance monitoring. We explicitly stated in the task description that worker performance would be monitored, with subpar work potentially leading to exclusion from labeling jobs. We observed a significant reduction in accep-

tance rates on Mturk. We confirmed that workers were reading the tasks by tracking the job previews on our infrastructure.

We believe this hesitancy could be connected to the relatively low pay for labeling jobs hosted through GT, ranging from \$0.012 for low-complexity tasks to \$1.20 for high-complexity tasks for high-complexity tasks, along with the fear of being monitored. Working on a low-paid job with the risk of potential blocking might be unattractive, and there could be concerns about the impact on their Mturk account, affecting their source of income. Despite clarifying that this would not affect their Mturk performance history and emphasizing its impact on the labeling job, we did not observe an improvement. The combination of lower compensation and scrutiny may have reduced the appeal for workers to undergo monitoring, influencing task acceptance rates.

5. **Impact of the Blocking Mechanism on Labelling Job Outcomes**: Requesters in our research lab reported an improvement in data label quality, resulting in a substantial reduction in the amount of discarded labeled data from 90% to 10%. However, we also observed that labeling jobs took longer to complete compared to those hosted directly on GT without our infrastructure layer, occasionally resulting in incomplete tasks and missing labels in the data. We speculate that this is due to workers not releasing the tasks after being blocked so others could not pick them up. One potential solution to address these challenges is to implement an automatic relaunch of the job, repeating the process until the desired number of completions is achieved.

6. **Evaluation of Crowd Worker Experience on MTurk**: To evaluate the crowd worker experience, we launched dummy labeling jobs on MTurk and attempted to utilize a dummy MTurk worker account to work on them. However, this account had restricted privileges, allowing us only to preview jobs on MTurk, providing limited visibility into the worker's perspective of the task.

For initial testing, we employed this account to ensure the accurate rendering of labeling tasks.

Another limitation that surfaced in MTurk job preview mode was that our software layer on GT couldn't capture Worker IDs to identify the worker previewing the job. The absence of worker IDs during task previews constrained the comprehensive testing of features in our system.

Since evaluating the crowd worker experience on MTurk was limited, we instead relied on test labeling jobs being accepted by real crowd workers and tracked their activity through infrastructure logs and the performance history database for a retrospective analysis of the worker workflow. The logs and database provided comprehensive records of worker IDs, activities (including job acceptance), and responses to gold questions, which were logged for in-depth analysis.

7. **Exploring External HITs Beyond the Current Framework**: As explored in section 5.3, there exists a potential to expand the integration to include various third-party surveys, such as SurveyMonkey. The procedure for Requesters remains similar, involving the creation of a completion code and its transmission to our integration engine through an accessible API. Workers can similarly adhere to this straightforward process by presenting the generated code for job verification.

6.2 Conclusion

In this study, our primary focus was to address the disparities between the MTurk and GT platforms. We achieved this by implementing a robust software infrastructure on GT, introducing Mturk-like features such as worker performance monitoring, automatic task access restriction for subpar performance, geographic-based constraints limiting task access to US-based workers, and the ability to conduct external tasks off-platform. Importantly, this implementation maintains a minimal impact on the Requester's user experience and remains inconspicuous to the worker.

As a result of our efforts, our research team observed a significant reduction in discarded data, dropping from 90% to just 10%, showcasing the tangible impact of our solution. The team reported a seamless workflow experience, enabling them to focus on labeling task design and data collection without dedicating substantial time to learning the new system.

However, unforeseen challenges emerged that were not part of the initial considerations. Workers, now aware of performance monitoring and blocking, exhibited reluctance to work on the labeling jobs. This, coupled with the issue of low pay on GT, resulted in a low job acceptance rate. Additionally, jobs took longer to complete or tasks were left incomplete compared to those hosted directly on GT without our infrastructure layer. Highlighting the complex interplay between implemented solutions and human factors, these hurdles emphasize the ongoing need for careful consideration in system implementation. While success was achieved in the system's functionality, the introduction of human elements added additional layers of complexity that warrant further exploration and refinement in future implementations.

Appendix A

In this appendix, you can find the link to the GitHub repository containing the source code.

Code Repository Link

The source code for the project is available on GitHub at the following link: `https://github.com/utir/gt/tree/main/gt-custom-ui`

User Guide Link

The user guide of our system is available at the following link:

`https://docs.google.com/document/d/1jzaRwabWoMBtI1MgmSbScGRt6zgju9wnsGC3h8e0ZNQ`
`edit?usp=sharing`

List of Timezones

- America/Adak
- America/Anchorage
- America/Boise
- America/Chicago
- America/Denver
- America/Detroit
- America/Indiana/Indianapolis
- America/Indiana/Knox

- America/Indiana/Marengo
- America/Indiana/Petersburg
- America/Indiana/Tell_City
- America/Indiana/Vevay
- America/Indiana/Vincennes
- America/Indiana/Winamac
- America/Juneau
- America/Kentucky/Louisville
- America/Kentucky/Monticello
- America/Los_Angeles
- America/Menominee
- America/Metlakatla
- America/New_York
- America/Nome
- America/North_Dakota/Beulah
- America/North_Dakota/Center
- America/North_Dakota/New_Salem
- America/Phoenix
- America/Sitka
- America/Yakutat
- Pacific/Honolulu

Further Readings

Salman Ahmad and et al. The jabberwocky programming environment for structured social computing. In *UIST 2011*, 2011. (ManReduce, Dormouse, & Dog Programming Languages).

Daniel W. Barowy, Charlie Curtsinger, Emery D. Berger, and Andrew McGregor. Automan: A platform for integrating human-based and digital computation. *Communications of the ACM*, 59(6):102–109, 2016. [PDF] [Website w/ Slides & Source].

Patrick M. De Boer and Abraham Bernstein. Efficiently identifying a well-performing crowd process for a given problem. In *ACM CSCW*, pages 1688–1699, 2017.

Michael Franklin and et al. Crowddb: Answering queries with crowdsourcing. In *ACM SIGMOD 2011*, pages 61–72, 2011.

P. Minder and A. Bernstein. Crowdlang: A programming language for the systematic exploration of human computation. *Social Informatics*, pages 124–137, 2012.

A. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. In *5th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2011.

Aditya Parameswaran et al. Deco: Declarative crowdsourcing. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1203–1212, 2012.

Jurairat Phuttharak and Seng W. Loke. Logiccrowd: A declarative programming platform for mobile crowdsourcing. In *Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1323–1330, 2013.

Rajan Vaish, Kate Wyngarden, Jing Chen, Brian Cheung, and Michael S Bernstein. Twitch crowdsourcing: crowd contributions in short bursts of time. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 3645–3654, 2014.