FourEyes: Leveraging Tool Diversity as a Means to Improve Aggregate Accuracy in Crowdsourcing

JEAN Y. SONG and RAYMOND FOK, University of Michigan, USA
JUHO KIM, KAIST, Republic of Korea
WALTER S. LASECKI, University of Michigan, USA

Crowdsourcing is a common means of collecting image segmentation training data for use in a variety of computer vision applications. However, designing accurate crowd-powered image segmentation systems is challenging, because defining object boundaries in an image requires significant fine motor skills and hand-eye coordination, which makes these tasks error-prone. Typically, special segmentation tools are created and then answers from multiple workers are aggregated to generate more accurate results. However, individual tool designs can bias how and where people make mistakes, resulting in shared errors that remain even after aggregation. In this article, we introduce a novel crowdsourcing approach that leverages *tool diversity* as a means of improving aggregate crowd performance. Our idea is that given a diverse set of tools, answer aggregation done *across* tools can help improve the collective performance by offsetting systematic biases induced by the individual tools themselves. To demonstrate the effectiveness of the proposed approach, we design four different tools and present FourEyes, a crowd-powered image segmentation system that uses aggregation across different tools. We then conduct a series of studies that evaluate different aggregation conditions and show that using multiple tools can significantly improve aggregate accuracy. Furthermore, we investigate the idea of applying post-processing for multi-tool aggregation in terms of correction mechanism. We introduce a novel region-based method for synthesizing more accurate bounds for image segmentation tasks through averaging surrounding annotations. In addition, we explore the effect of adjusting the threshold parameter of an EM-based aggregation method. Our results suggest that not only the individual tool's design, but also the correction mechanism, can affect the performance of multi-tool aggregation. This article extends a work presented at ACM IUI 2018 [46] by providing a novel region-based error-correction method and additional in-depth evaluation of the proposed approach.

CCS Concepts: • **Information systems → Crowdsourcing**; • **Human-centered computing → Human computer interaction (HCI)**; • **Computing methodologies** → *Computer vision*;

Additional Key Words and Phrases: Crowdsourcing, human computation, multi-tool aggregation, tool diversity, semantic image segmentation, computer vision

The reviewing of this article was managed by special issue associate editors Mark Billinghurst, Margaret Burnett, and Aaron Quigley.

This research was supported in part by the Denso Corporation, Toyota Research Institute, and MCity at the University of Michigan. This work was also supported in part by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. 2017-0-00537, Development of Autonomous IoT Collaboration Framework for Space Intelligence).

Authors' addresses: J. Y. Song, R. Fok, and W. S. Lasecki, University of Michigan, Electrical Engineering and Computer Science, 2260 Hayward Street, Ann Arbor, MI, 49085, USA; emails: {jyskwon, rayfok, wlasecki}@umich.edu; J. Kim, KAIST, School of Computing, Daejeon, Republic of Korea; email: juhokim@kaist.ac.kr.

2160-6455/2019/08-ART3 $15.00
https://doi.org/10.1145/3237188

ACM Reference format:
Jean Y. Song, Raymond Fok, Juho Kim, and Walter S. Lasecki. 2019. FourEyes: Leveraging Tool Diversity as a Means to Improve Aggregate Accuracy in Crowdsourcing. *ACM Trans. Interact. Intell. Syst.* 10, 1, Article 3 (August 2019), 30 pages.
https://doi.org/10.1145/3237188

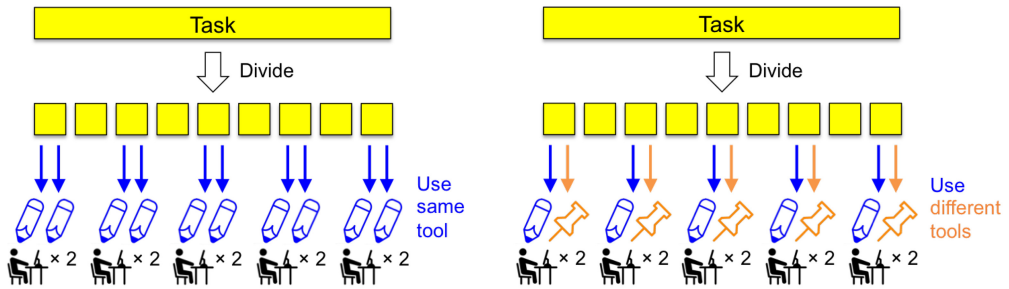
1 INTRODUCTION

The goal of image segmentation is to demarcate objects in a visual scene from the background, allowing computer vision (CV) systems to learn to recognize specific objects. These CV systems can, in turn, enable autonomous cars to identify pedestrians, surveillance drones to focus on potential threats, and in-home robots to help people with motor or mobility impairments live more comfortably and independently.

Perceiving the boundaries of physical objects comes naturally for people, but it remains a challenging open problem for CV systems due to the complexity of understanding the semantics of visual scenes [15, 35]. Crowd-powered object segmentation tools can bridge this gap by leveraging human understanding to produce large, manually demarcated training data sets ([2, 13, 32]) for CV systems. However, designing crowd-powered tools that produce high-accuracy training data and scale efficiently (with respect to human-time cost) remains an open problem, because the task of manually marking object boundaries requires significant hand-eye coordination and fine motor skills, resulting in a high error rate if these tasks are performed too quickly by workers.

Many web-based image segmentation tools ([1, 2, 5, 12, 31, 43]) have been designed to help workers reduce the effort needed to complete a task and to increase the accuracy of their output. However, different tool designs induce different error patterns in worker performance, which can lead to repeated systematic mistakes when only a single tool is used. For example, some tools [2, 43] provide polygon drawing functionality to help trace object boundaries, but Bell et al. [2] reported that workers often skip selecting parts of the object if automatic scrolling during selection is not provided. We consider this to be a *systematic error bias,* because the same error pattern would be unlikely to emerge if the tool were designed differently. In other words, it would be unlikely for worker outputs from Click'n'Cut [5] (which asks workers to use left/right mouse clicks to identify foreground and background regions of an image) to result in the same mistakes as using the polygon drawing tool. However, Click'n'Cut may exhibit its own systematic error pattern induced by limitations in its own design. More generally, we consider error patterns that are found to be common among worker outputs from a single tool to be systematic error biases, as they are likely to be induced by the design of the tool itself. These errors are different from, for example, human perceptual biases that may also systematically affect outcomes [40], in that they are common to the outputs of a tool, not common to the annotations produced by an individual worker.

In this article, we propose the idea of leveraging *tool diversity* as a means of overcoming these systematic error biases to improve aggregate crowd performance. Tool diversity is the extent to which tools designed for the same task differ from one another in the systematic error biases that they induce. Unlike standard aggregation methods in crowdsourcing, which try to design and use the best single tool available with many workers to reach high accuracy, we show that using multiple effective tools can diversify the error patterns in worker responses and help systems achieve higher *combined* accuracy (Figure 1). This insight is motivated by ensemble learning methods in machine learning that use multiple learning algorithms to obtain better prediction than can be obtained from any of the constituent algorithms alone [7]. A strength of leveraging tool diversity is that the approach is orthogonal to, and thus may be combined with, many existing



(a) Conventionally, one tool type is used for the same task. (b) We propose leveraging different tools for the same task.

Fig. 1. This article introduces an approach to leveraging *tool diversity* that uses multiple different tools for the same task (as in (b)) to improve aggregate crowd performance by offsetting systematic error biases that might otherwise result from using any one tool type alone (as in (a)). Our findings on an image segmentation task demonstrate that using a combination of tools can significantly increase aggregate accuracy.

crowdsourcing methods for improving quality over time (e.g., training workers [8, 51] or identifying high-performing contributors [44]).

To demonstrate our proposed workflow, we design four different image segmentation tools and introduce FourEyes, a multi-tool-based crowd-powered system that leverages combinations of tools to generate better aggregate responses. After that, we report results from a series of studies that evaluate different aggregation conditions—such as majority voting versus expectation maximization (EM), and single-tool aggregation versus up to four-tool combination aggregation—with equally sized groups of workers. Our evaluation demonstrates the effectiveness of tool diversity by showing that the output accuracy of heterogeneous tool combinations can be significantly higher than homogeneous sets, providing output at least comparable to the best constituent tool and always yielding significantly better results than the weakest constituent tool.

Moreover, we explore the idea of adding post-processing for multi-tool aggregation with respect to the error correction mechanism. When leveraging tool diversity, once analysis on individual tool performance is conducted and the error pattern of each tool is revealed, a system designer can implement suitable correction mechanisms to further offset error biases. To correct errors in image segmentation tasks, we introduce a new region-based method for synthesizing more accurate bounds through averaging surrounding annotations. We explore the effects of mask size and threshold parameter and show that the proposed method always increases the aggregate accuracy of any tool combination by up to 6%. We also investigate the effect of a threshold parameter in the EM method and show that the threshold parameter value that yields the best performance differs by tool combination types.

Finally, we discuss generalizable guidelines to apply the multi-tool approach in other problem domains. We characterize our problem in a more general form and summarize the properties of crowdsourcing tasks that are amenable to our approach: those that are objective, tractable enough for workers to produce nearly correct responses and increase in correctness as additional answers are provided.

This article presents an extended version of work published at the 2018 ACM International Conference on Intelligent User Interfaces [46], which first introduced the idea of leveraging tool diversity during aggregation as a crowdsourcing technique. In addition to a more in-depth evaluation of tool combinations (aggregation of three- and four-tool combinations), this article introduces a novel region-based error correction method and explores the impact of parameter selection on the region-based and EM methods as a means of post-processing for multi-tool aggregation.

The key contributions of this article are:

- A novel crowdsourcing paradigm that leverages a system's or task's tool diversity to aggregate input across different *types* of tools to improve the combined accuracy of workers' answers by offsetting systematic error biases.
- FourEyes, a crowd-powered image segmentation system that implements our approach, combining the output of four different tool types to improve the collective accuracy of a group of workers using a single segmentation tool.
- Experimental results from 51 objects across 12 indoor scenes segmented by 288 crowd workers using four different tools that validate our system's effectiveness and suggest the benefits of our multi-tool approach.
- An evaluation of the aggregate results of each possible tool combination from FourEyes and an exploration of the ability for correction mechanisms to further improve the accuracy of the combined results by exploiting the error bias patterns of the individual tools.

2 RELATED WORK

Conventional approaches to improving crowd worker output accuracy include microtask decomposition and consensus-based aggregation. These approaches are usually intended to reduce task complexity and correct for the variance in individual worker responses, respectively. However, when it comes to systematic error biases induced by a tool's design, errors can persist even after decomposition or aggregation, since all workers use the same tool and the tool used in the workflow may induce biases into worker responses. Our tool-diversity strategy builds on prior work in crowdsourcing workflows and answer aggregation strategies to reduce these systematic error biases. In this section, we discuss related work in designing crowdsourcing workflows and improving the output quality by answer aggregation.

2.1 Crowdsourcing Workflows

In crowdsourcing, breaking large tasks into smaller microtasks has been a popular strategy to increase the accuracy of crowd workers' answers. Microtasks are small, context-free units of work that are widely used in crowdsourcing workflows. Crowdsourcing platforms, such as Amazon Mechanical Turk, post these small units of work that (typically quasi-anonymous [28]) crowd workers can accept and complete. TurKit [33] introduced the crash-and-rerun programming model to recursively improve output of a challenging task by passing the task from worker to worker. Soylent [3] showed that dividing a larger task into Find-Fix-Verify steps improves the accuracy of crowd workers' answers in document editing tasks. Similarly, ToolScape [19] used a Find-Verify-Expand workflow to enhance the process of extracting different steps in how-to videos. ConceptScape [34] extends multi-stage workflows and divides the concept map generation task into three stages with multiple substeps within each stage. CrowdForge [20] introduced a MapReduce-like workflow to accomplish even complex and interdependent tasks using microtasks. Crowdlines [37] introduced two different workflows for merging information from multiple sources to create an outline. Turkomatic [21] attempted to crowdsource the workflow itself, showing that the planning and execution of a task can be done given some level of requester supervision.

While this prior research has explored how to use crowd workflows to collectively accomplish what no single worker could alone, each task type was done using the same UI, and thus were subject to systematic error biases in each tool. More recently, continuous crowdsourcing has made real-time [24, 26] or even instantaneous [36] crowdsourcing responses from crowds possible. These allow for the creation of interactive systems powered by human contributors. TimeWarp [25] introduced the idea of creating workflows that enable a group of workers to complete tasks in a

matter not possible with a single worker (in that case, provide captions in real time while listening to half-speed audio) to improve individual performance. Plexiglass [42] introduced a workflow that enables a single worker to interleave multiple tasks at a time. The idea is to multiplex "passive" and "active" tasks together in one UI to more efficiently complete work that would otherwise contain time spent idly waiting for a rare event to occur. CrowdMask [18] uses a pyramid workflow to mask private content in images using crowds. Their method segments and distributes the segments of user content so workers can mark potentially private content without viewing enough of it to be harmful. WearMail [47] introduced a privacy-preserving workflow that allowed crowds to train a system on demand to accomplish an email search task without ever revealing the email contents to workers.

Our work contributes to this line of research by introducing a novel crowdsourcing approach that aggregates multiple crowd-powered tools to offer better performance than any of the constituent tools alone. AgentHunt [29] had a similar motivation, using multiple workflows to outperform a single best workflow, but their approach used decision-making models to *choose* among different workflows. To the best of our knowledge, we are the first to study methods for *simultaneously* using and aggregating different tools within a workflow to increase combined accuracy.

2.2 Improving Output Quality by Answer Aggregation and Bias Correction

Crowdsourced data can contain conflicts between workers' answers, thus answer aggregation becomes a necessary step to obtain the final unitary output. In this section, we review answer aggregation strategies and bias correction strategies that help improve crowdsourcing output.

2.2.1 Answer Aggregation Strategies. A common strategy to improve output quality in crowdsourcing systems is to aggregate independent workers' answers on the same task into a single response, typically via a consensus method like voting. Even simple majority voting has been shown to produce accurate results for crowdsourcing tasks, such as linguistic annotation tasks [45] and document editing tasks [3]. In terms of image segmentation tasks, ground truth segmentations of objects have been generated via majority pixel voting with manually collected answers from multiple crowd workers or experts [13, 32]. More sophisticated approaches using unsupervised learning have been used to weight workers' answers by using models of their abilities [4, 30, 49, 50]. Deluge [4] models workers' sensitivity and specificity to detect noisy workers, and LazySusan [30] tracks workers by assigning different weights based on the accuracy of a worker's answers. Researchers have also proposed probabilistic approaches to model not only the workers, but also the properties of the data being labeled [49, 50].

2.2.2 Bias Correction Strategies. Assigning differential weights to workers' answers during aggregation is a preprocessing step that aims to correct individual worker errors before combining the answers [44]. Ipeirotis et al. [16] showed that the EM algorithm can be used to separate biases from unrecoverable errors, providing more reliable scores of the quality of the workers. The EM algorithm [6, 16] predicts unknown (latent) correct answers by estimating weights for each crowd worker's answers. Dawid and Skene [6] showed that the EM algorithm significantly outperforms majority voting when a majority of workers' responses are correct and conditionally independent given the ground truth answer. The EM algorithm is suitable for exploiting tool diversity in image segmentation tasks because: (i) the majority of the pixels selected by any tool are assumed to be correct and (ii) the probability of tools labeling a pixel is independent of any particular chosen pixel. When designing a tool, its exact abilities and error biases are not typically known in advance, because designers are not aware of the input images that the system will see in final use. Because the performance of each tool can vary with images or object types, we can consider a tool's ability

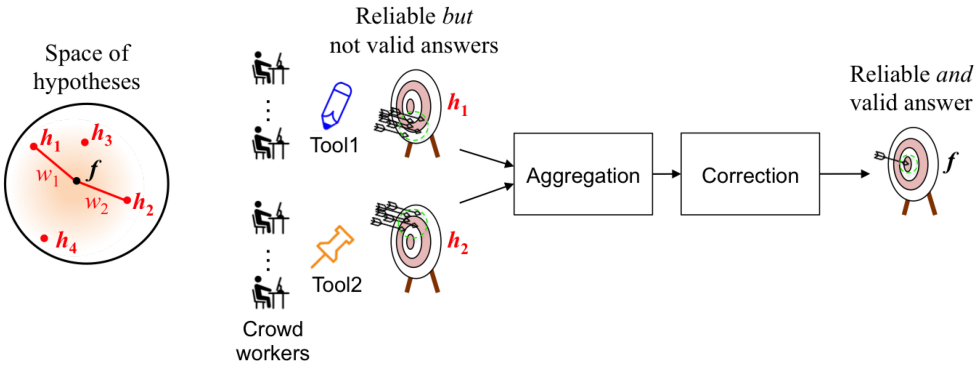


Fig. 2. The left diagram shows the hypotheses space of the possible segmentation tools, including the best performing tool (f) and other possible hypotheses ($h_1 \ldots h_4$). We are motivated by ensemble learning methods that construct a combination of alternative hypotheses (h_1 and h_2) to approximate the best hypothesis f. The right flowchart shows a set of workers using two different tools to perform the same task. An aggregation and correction pipeline can output reliable (consistent) and valid (accurate) aggregate results (f) from two reliable but not valid answers (h_1 and h_2). This diagram represents the end-to-end process of the proposed tool-diversity scheme: preparing different tools, aggregating, and correcting.

as the latent variable to be predicted. Therefore, we apply the EM algorithm across different tools with the goal of maximizing the performance of the aggregated output.

Several approaches have been introduced to combat biases of individual crowd workers; there has been little work on correcting error biases induced by tools or interfaces. For example, References [22] and [17] can be potentially used to correct systematic biases induced by workers but require human mediators to correct biased answers. In this work, we explore the idea of applying error correction when using multiple tools and suggest mechanisms that can offset the trade-offs of different image segmentation tools, which can be done with or without a human mediator.

3 APPROACH

Prior work has used task decomposition—the process of breaking down larger tasks into more manageable, focused pieces of work—to make tasks more approachable for non-expert crowd workers. Once task decomposition has been used to break down a larger unit of work as much as possible within a corresponding workflow, most crowdsourcing systems then recruit multiple workers in parallel to further improve accuracy by aggregating their answers. We propose using multiple different tools across different workers to complete the same (sub)task, instead of having all workers complete the same task with the same interface or tool. Our proposed approach fills in the gap where traditional task decomposition leaves off.

3.1 Motivation from Ensemble Learning

Our work is conceptually motivated by ensemble learning in machine learning. Ensemble learning methods are machine-learning algorithms that construct a set of learning algorithms and predict a new data point by taking a weighted vote of the predictions from each learning algorithm [7, 9]. It has been proven that ensembles often perform better than any single member [7]. Algorithm accuracy (i.e., better than random guessing) and diversity are necessary as well as sufficient conditions for a combination of algorithms to be more accurate than any of its individual constituents [14]. The left diagram in Figure 2 shows how ensemble methods work. In the diagram, a learning algorithm can be viewed as searching a space of hypotheses to identify the best performing hypothesis

f, which can be computationally difficult to find. Ensemble learning constructs a combination of two alternative hypotheses h_1 and h_2 with proper weights (w_1 and w_2), and approximates the best hypothesis f by averaging the two. Our tool-diversity approach is analogous to ensemble learning methods in that multiple image segmentation tools are combined to produce a better final result.

3.2 Aggregation of Reliable but Biased Tools

Even a carefully designed crowdsourcing system may often induce reliable (consistent) but not valid (accurate) answers. For example, a semantic image classification task—of assigning classes that correspond to objects that appear in an image—can have its systematic bias due to the design of the tool. If a tool is designed to type free-form answers, it may bias workers to only use a limited number of words that they can spell or find easier to spell. However, if a tool is designed such that workers can click to select a word from a predefined list, the error pattern would be different. These errors can be defined as systematic error biases, because the same error pattern would be unlikely to arise if the tools were designed differently.

Instead of trying to fix a biased tool, our approach aims to *combine* answers from these multiple biased tools to improve the aggregate result. Analogous to a necessary and sufficient condition in the ensemble learning scheme, a suggested condition for using multiple tools is that the tools are at least reliable, even if they are not valid. This allows for aggregation and correction mechanisms that can offset the expected biases, eventually achieving both reliable and valid results when aggregated. Figure 2 depicts the concept of tool aggregation within a crowdsourcing workflow. A researcher or requester can provide Tool 1 to one set of workers and Tool 2 to a different set of workers. When the tools are reliable but not valid with output hypotheses h_1 and h_2, respectively, the aggregation and correction modules can combine the answers so the final output is approximately f, the best hypothesis. In the next sections, we show how we realized these tools and designed aggregation and the correction mechanisms in the domain of semantic image segmentation.

4 FOUREYES

FourEyes is an image segmentation system that leverages four different crowd-powered tools to produce accurate segmentation results by aggregating answers across different tool types. We describe the individual tool here and then detail the novel aggregation methods in the next sections.

4.1 Choosing the Tools

We introduce four web-based segmentation tools that we designed to instantiate and test the tool-diversity concept. We considered one key question when designing the tools: "How can we diversify the errors produced by different tools?" Because it is hard to predict what errors will be induced by a given tool, we built tools specialized to work well with objects with different characteristics, such as small or transparent objects, objects with fuzzy materials, and reflective surfaces. These objects are current challenges to both automatic segmentation methods and human annotators. We designed these tools to ideally perform differently for different types of objects, resulting in greater error diversity. We categorized these objects into three groups and created tools that are designed to minimize errors in each object category. The spaces we explored and the tools we designed are summarized in Figure 3. We used the Question (Q), Option (O), and Criteria (C) representation [38] of the design space for deciding which tools to build. The Question indicates a key design issue, the Option node suggests possible answers to the Question, and the Criteria item represents the core properties expected from choosing an Option. For one of the Options (O3 in Figure 3), we differed the interface in two ways (Drag-and-Drop and Pin-Placing) so the interaction of users can create different artifacts. We observed that different interactions lead to different

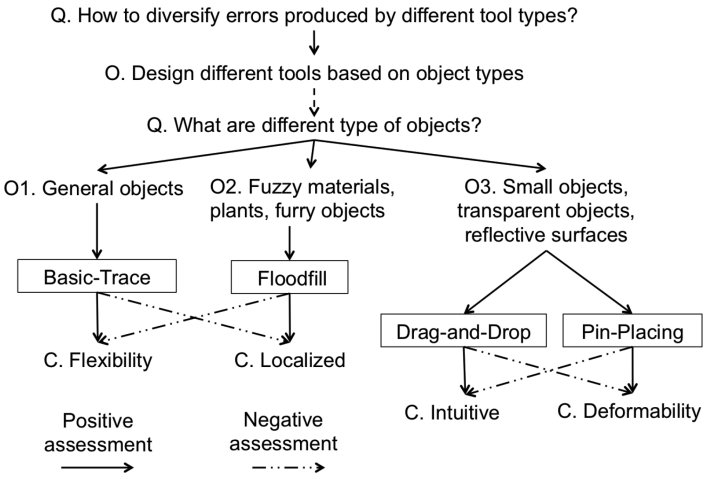


Fig. 3. Design space we considered when choosing the tools for the study. We used the Question (Q), Option (O), and Criteria (C) representation of the design space.

Table 1. A Comparison of the Four Tools Across Two Design Elements

Design Element	Comparison
Degree of freedom & Amount of interaction	Basic Trace > Pin-Placing > Drag-and-Drop > Floodfill
Complexity of interface layout	Pin-Placing > Drag-and-Drop > Floodfill > Basic Trace

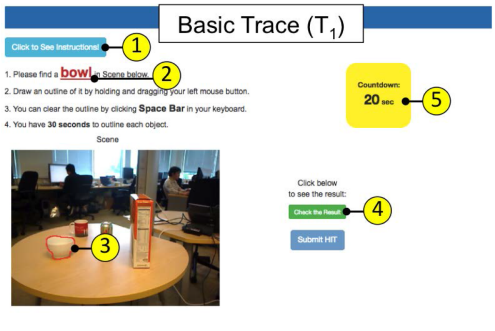
error patterns, so we include both of the tools in the experiment section. In the following section, we provide detailed descriptions of the four tools developed.

4.2 Designing the Tools

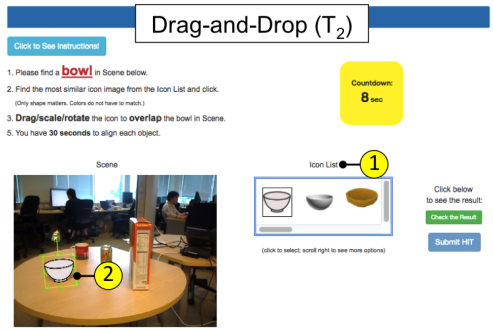
The four tools implemented were Basic Trace, Drag-and-Drop, Pin-Placing, and Floodfill. They vary in the level of degree of freedom, interface layout, and amount of interaction needed from a worker. The differences are summarized in Table 1.

4.2.1 Basic Trace. The first tool is a free-form drawing tool shown in Figure 4(a). With Basic Trace, workers click and drag their mouse to trace the outline of the query object in a scene (Figure 4(a) ③). Once a worker submits the initial trace line, a simple image-processing algorithm connects the gaps and fixes the irregularities in the traced line to form a smooth shape. It then highlights the pixels inside the traced shape and returns the result as the final object segmentation. Of our four tools, the Basic Trace is the most manual and provides the highest degree of control. The strength of this tool is that it is highly flexible and workers can segment any type of object if sufficient time is given. However, the weakness of the tool is that if a worker is idle and not careful enough, the output can easily be very poor, e.g., a worker may draw a rough box around an object instead of carefully following the boundary.

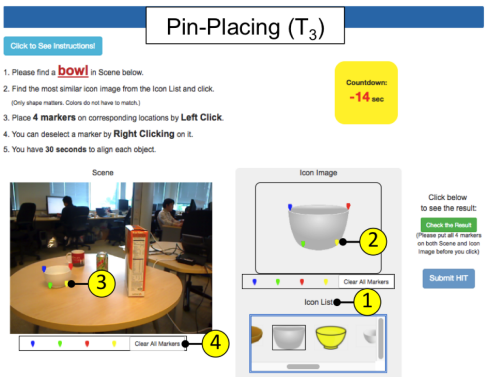
4.2.2 Drag-and-Drop. The second tool lets workers select an object template from a list (Figure 4(b) ①), which is generated by searching images of a target object from an image search engine such as Google or Bing. These images are then filtered for transparency and size, and the top N (in this article, we use $N = 12$) are downloaded to construct a template list for each query object. Workers are asked to select the template that most accurately matches that object



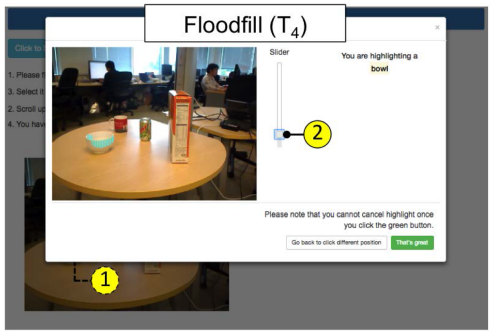
(a) Overview of the Basic Trace image segmentation tool. (1) shows the full instruction when clicked. (2) describes the query object which is shown to the workers in random order. (3) shows an example of a trace line a worker provided. (4) shows the segmentation result when clicked. (5) is the task timer, which serves as an encouragement to workers to consider time in their work.



(b) Overview of the Drag-and-Drop image segmentation tool. (1) shows the template images list. (2) shows an example of the chosen template image being aligned to the query object.



(c) Overview of the Pin-Placing image segmentation tool. (1) shows template images list. (2) and (3) show examples of chosen points by a worker. (4) lets workers reset the points to start over.



(d) Overview of the Floodfill image segmentation tool. (1) shows that a worker can click on a query object in the given scene to initiate segmentation. (2) is a slider that allows workers to adjust a parameter to control the propagation of the selection area.

Fig. 4. Worker interface of the four segmentation tools used in our experiments.

in the scene based on its shape, proportion of dimensions, and perspective. In Drag-and-Drop, workers overlay their selected template onto the object identified in the scene (Figure 4(b) ②). Workers are able to scale, rotate, and drag the template to adjust the angle its dimensions in an attempt to closely match the shape of the actual object. Based on a worker's transformation of the template, the system determines the final object segmentation by identifying and annotating the overlapping pixels between the template and the scene. The strength of this tool is that it is very intuitive to use. However, the weakness of the tool is that it is hard to map deformable objects, or even rigid objects if being viewed from different perspectives.

4.2.3 Pin-Placing. The third tool is also a template-based tool called Pin-Placing. A template list is generated in the same manner as in Drag-and-Drop (Figure 4(c) ①). With this tool, workers select four arbitrary points on their selected object template (Figure 4(c) ②) and pair them with

four corresponding points on the object in the scene (Figure 4(c) ③). Workers can modify individual points or clear all points at once (Figure 4(c) ④). After the four pairs of points are submitted, an automatic transformation algorithm is run to transform the template image to produce the final object segmentation. Note that four pairs of control points are the minimum necessary to perform a non-linear deformation between two images, given the perspective limitation inherent in a fixed-angle view in two dimension. Pin-Placing's working mechanism is similar to sophisticated techniques (e.g., which professional radiologists use for diagnosing lesions), but it is not intuitive to novice workers.

One drawback of template-based approaches is that if an object in a scene has an atypical shape, none of the template images in the list may have a shape similar to the object. In this case, a possible solution would be allowing workers to switch to a different non-template-based tool. We note that the two template-based tools, Drag-and-Drop and Pin-Placing, force workers to select occluded parts of a target object when it overlaps with other objects. This is useful in domains like robotics, where ground truth object geometry includes hidden parts. However, in this study, we only consider the visible parts of a target object as the region of interest, because it is a more general way of indicating objects in two-dimensional image segmentation. As a consequence, these two tools necessarily select more false-positive regions than the other tools.

4.2.4 Floodfill. Floodfill (AKA Bucket-fill) is a mostly autonomous tool, combining a simple region-growing method [48] with minimal human input to initialize the seed point and tune a threshold parameter. Workers click on the object they want to segment (Figure 4(d) ①) and adjust a slider to tune one of the algorithm's threshold parameters (Figure 4(d) ②). This triggers the RGB Floodfill algorithm that highlights all neighboring pixels sharing an RGB value similar to the seed point that was clicked. If the segmentation is unsatisfactory, either failing to select the entire object or exceeding the object boundary, workers can adjust the slider to modify the highlighted area. The tool is effective if the shape of an object is complex with many curves, but only when the object is mostly monochromatic. If a query object is polychromatic or contains shaded regions, the selection area can be smaller than the actual object boundaries, because the algorithm cannot propagate across these regions.

4.3 System Interfaces

FourEyes begins by receiving a scene image and the user's request in the form of a natural language query, e.g., "mark the bowl." The query is parsed to find nouns that are then displayed to workers (Figure 4(a) ②) as objects that need to be segmented from the scene. For each tool, a short series of instructions (including the target object in bold) is displayed to workers while they perform the task. Workers can also check the segmentation result before they submit their work ("Check the Result" button in Figure 4(a) ④). To discourage workers from idling, a task timer is embedded (Figure 4(a) ⑤) that counts down from t seconds and turns negative when time runs out. In this article, we used $t = 30$ in all experimental conditions. The timer serves as encouragement to complete the task in a timely manner and does not otherwise affect the workers.

5 MEASURING THE PERFORMANCE OF INDIVIDUAL TOOLS

To understand the effect of tool diversity on improving aggregate crowd performance, we recruited 288 crowd workers from Mechanical Turk using LegionTools [11]. Workers were given one of the four tools to perform a task of image segmentation. Note that we gave different tools to different workers, because we consider the smallest unit of microtask as one worker segmenting one object using a single tool. To avoid learning effects and worker-induced bias in the annotation results, workers were randomly assigned to an annotation task (segmenting one scene using one tool),

and they could not choose which tool they were given. We recruited six unique workers for each tool-scene pair, resulting in a total of 1,224 object segmentations.

5.1 Dataset

We chose a dataset that included various indoor objects. The dataset included 12 different visual scenes, each containing 3 to 7 objects, for a total of 51 objects. The scenes were gathered from publicly available datasets[1,2] and represent typical indoor scenarios with commonplace objects. They ranged from a living room to a tabletop and contained everyday objects (e.g., a plant, laptop, soda can, cereal box, flashlight, etc.). In the experiment, each worker was shown one scene and a series of object names to segment depending on the number of objects in the scene. For each task, the order of the objects in each list was randomized to avoid any ordering bias. Each worker was given one scene with one tool to perform a segmentation task.

5.2 Instructions and Payment

Before crowd workers could begin the task, they were shown a short instructional video demonstrating the goal of the task and how to use the tool they would be provided with. The lengths of the instructional videos were 36s, 54s, 78s, and 33s, respectively, for each tool: Basic Trace, Drag-and-Drop, Pin-Placing, and Floodfill. Two of the tools (Drag-and-Drop and Pin-Placing) had longer videos, because they explained how to choose the most similar template image. This additional step delayed workers' task completion time in the actual experiment as well. Workers were also shown pictures exemplifying desired and undesired segmentations (the same example images were used for all tools) so they understood the aim of the task to create a detailed boundary of a target object in a scene. If the worker decided to proceed after watching the instruction video, they were directed to the FourEyes's worker UI and their subsequent interactions with the UI were recorded. Task instructions were also accessible at any time if necessary (Figure 4(a) ①). Each worker was paid between $0.35 and $0.60 per task, proportional to the number of objects they had to segment and the expected completion time using a given tool (a pay rate of ~$10/hr). The expected time of each tool was determined by its average latency time from a dozen of preliminary experiments.

5.3 Segmentation Quality Evaluation

To assess success on the image segmentation task, we measured the accuracy of each by comparing the output similarity to the ground truth segmentation that was generated manually by the authors prior to the experiments. One author carefully completed the task and another author verified the quality of the resulting ground truth.

We used precision, recall, and F_1 score (the harmonic mean of precision and recall) to compute the pixel-level similarity (Equation (1)). To do this, the number of true positive, false positive, and false negative pixels were counted for each crowdsourced segmentation.

$$\begin{aligned} \text{Precision} &= \frac{\text{true positive}}{(\text{true positive} + \text{false positive})} \\ \text{Recall} &= \frac{\text{true positive}}{(\text{true positive} + \text{false negative})} \\ \text{F}_1\ \text{Score} &= \frac{2 \times \text{Precision} \times \text{Recall}}{(\text{Precision} + \text{Recall})} \end{aligned} \tag{1}$$

[1] https://rgbd-dataset.cs.washington.edu/dataset.html/.
[2] https://www.doc.ic.ac.uk/~ahanda/VaFRIC/iclnuim.html/.

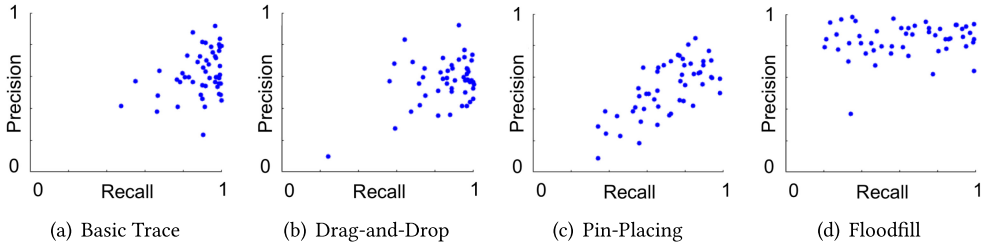


Fig. 5. Precision-recall scatter plot of our four different tools. The different tools have different error patterns (trade-offs) in terms of precision-recall metrics. (a) Basic Trace and (b) Drag-and-Drop show high recall but low precision tendency, implying that the tools are reliable but not valid. (c) Pin-Placing shows the most scattered pattern, implying that the tool's performance highly depends on the query object, which makes the tool neither reliable nor valid. (d) Floodfill shows high precision but low recall tendency, implying that the tool is reliable but not valid.

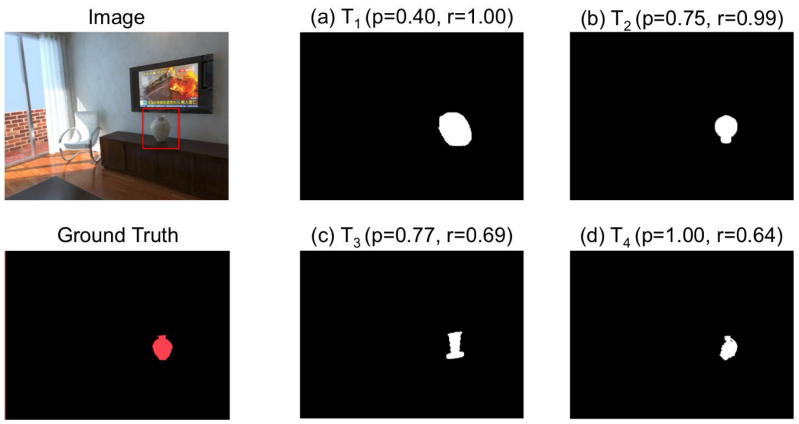


Fig. 6. Original image (top left), ground truth image (bottom left), and exemplar segmentations using the four tools with their precision and recall values reported on top. (a) Basic Trace, (b) Drag-and-Drop, (c) Pin-Placing, and (d) Floodfill. The exemplar images represent a typical output of each tool.

5.4 Results

The different tools had different error patterns (trade-offs) in terms of precision and recall. Figure 5 shows scatter plots of the overall segmentation result with each dot representing an average precision-recall of one object being segmented using one of the tools in FourEyes. That is, each dot is an average of six workers' segmentation results. As shown in Figure 5(a) and (b), Basic Trace and Drag-and-Drop tended to show high recall but low precision. We observed that with these two tools, workers tended to select objects by putting large margins around the objects, resulting in high recall but low precision. Examples of segmentation using these two tools are shown in Figure 6(a) and (b), respectively. Meanwhile, Pin-Placing resulted in the most scattered performance, as shown in Figure 5(c). This implies that the performance of the tool varies a lot depending on object types. We presume that the underlying mechanism of computing non-linear transformation of Pin-Placing is unfamiliar to novice workers, which led to scattered and low overall performance. An example of using Pin-Placing is shown in Figure 6(c). In the example, a worker selected a template image that is very different from the query object, resulting in both low precision and low recall. Last, Figure 5(d) shows that Floodfill tended to give high precision but low

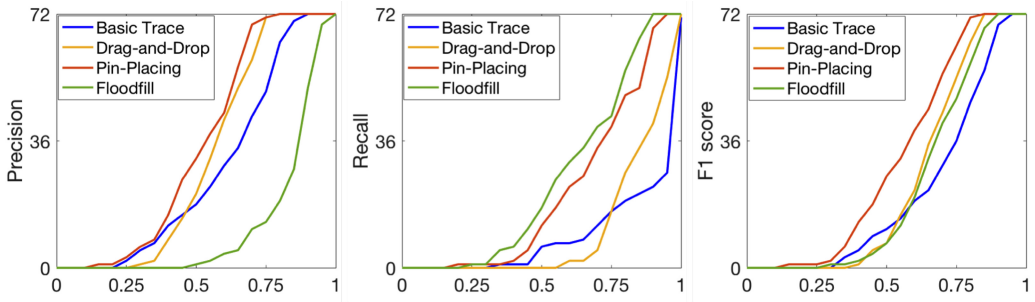


Fig. 7. Precision (left), recall (center), and F_1 score (right) plots of the cumulative distribution functions of performances of a single worker per tool. In terms of precision, Floodfill has the highest number of workers with high performance (>0.8). In terms of recall, Basic Trace has the highest number of workers with high performance. The F_1 score performance per worker is similar between tools compared to precision or recall, because the two offset each other when combined.

Table 2. Average Performance (and Standard Deviation) of the Four Individual Tools

	Precision	Recall	F_1 score
Basic Trace	0.62 (0.14)	**0.89** (0.12)	**0.71** (0.13)
Drag-and-Drop	0.57 (0.14)	0.86 (0.15)	0.66 (0.13)
Pin-Placing	0.53 (0.17)	0.71 (0.17)	0.58 (0.17)
Floodfill	**0.84** (0.11)	0.63 (0.25)	0.67 (0.20)

recall performance. We observed that the selection area with Floodfill tended to be smaller than the actual object boundaries due to boundaries that were shaded or colored differently. An example segmentation of using Floodfill is shown in Figure 6(d). Because one side of the vase was much brighter, a worker could not select the entire image with the seeded region-growing algorithm. Figure 6 shows typical example worker segmentations from each tool, alongside the ground truth.

In terms of reliability and validity (as discussed in Section 3.2), Basic Trace, Drag-and-Drop, and Floodfill can be considered reliable, since their output pattern is expectable (either high recall or high precision). However, they are not valid, because their output is biased (either low precision or low recall). However, Pin-Placing is neither reliable nor valid, because the output pattern is not predictable, being highly dependent on the query object.

For each tool, we recruited 72 workers. Each worker performed segmentation for one scene, where each scene contained three to seven objects. The cumulative distribution functions of performances (precision, recall, and F_1 score) of a single worker are summarized in Figure 7. From the precision plot (left), we can see that the Floodfill tool has more workers with high scores (>0.8) compared to the other tools. From the recall plot (center), we can see that Basic Trace and Drag-and-Drop tools have more workers with high scores compared to the other tools. The F_1 score plot suggests that the tool's harmonic performance between tools are less diverse compared to precision or recall, due to the offset between the two. Average accuracy metrics for each tool are summarized in Table 2. We use F_1 score as our performance measurement. In general, Floodfill gave the best performance in terms of precision, and Basic Trace gave the best recall and F_1 score.

The average performances of each object are summarized in Figure 8. The hollow dots represent performance for individual objects (average performance of 24 workers who segmented that object), and the filled dots are average performance over all objects in a single scene. The

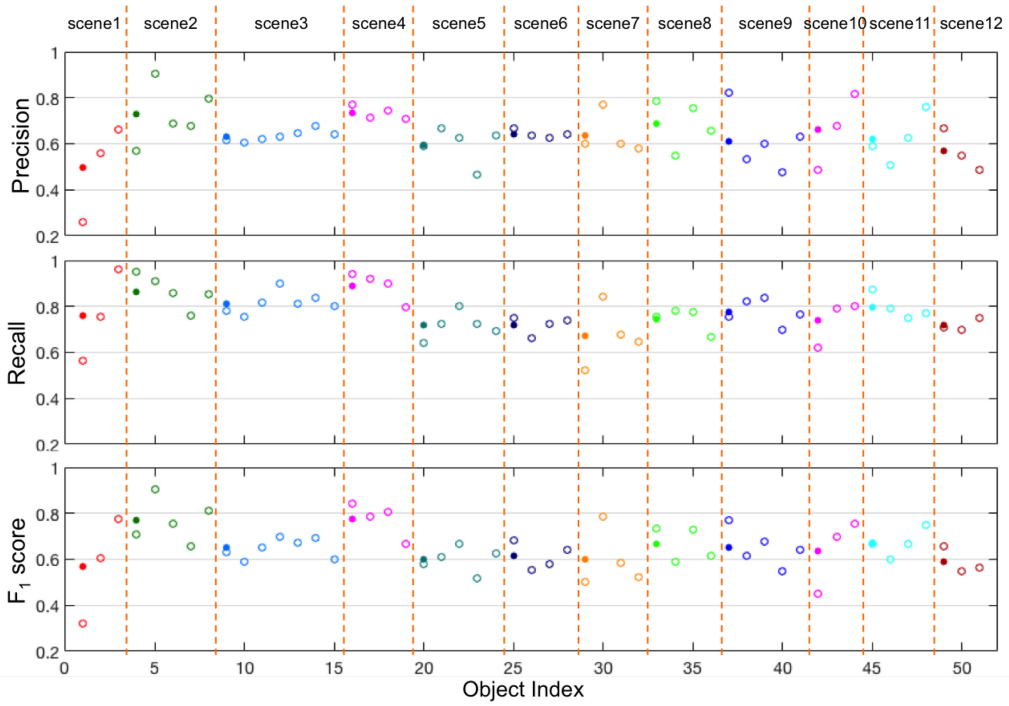


Fig. 8. Precision (top), recall (middle), and F_1 score (bottom) of average segmentation result of each object and scene. The hollow dots represent performance for individual objects (average performance of 24 workers who segmented that object), and the filled dots are average performance over all objects in a single scene. Different scenes are separated with dotted vertical lines. The average performance of objects varied across different scenes but lied in between 0.5 to 0.8 in terms of the F_1 score.

performance between scenes varied due to the different characteristics of each scene. For example, one scene was shot in front of a window, which added a lot of lighting to the scene, and another scene had many rigid objects that were relatively easy to demarcate from the background. Regardless of the characteristics, the average F_1 score of scenes lay in between 0.5 to 0.8.

To calculate latency, we measured overall task time starting from when the worker began interacting with the task to when the worker clicked "submit" at the end of the task. After dropping outliers more than two standard deviations (2σ) from the mean latency, Basic Trace's average latency was 14.37s (σ = 8.08), Drag-and-Drop's was 24.89s (σ = 11.25), Pin-Placing's was 20.77s (σ = 7.90), and Floodfill's was 12.62s (σ = 10.41). The template-based tools had a higher segmentation latency than the other two tools. This was expected, because the template-based tools are more involved and perhaps less intuitive to general-purpose crowd workers. Using Floodfill, some workers managed to produce a satisfactory segmentation within 3s, but others spent extra time trying to perfect their segmentation, with diminishing returns in accuracy.

From these primary results, we observed different error patterns across the four tools that we designed. The result matches our design intent to diversify the errors produced by different tools. Now, we can think of each tool as alternate hypotheses h_1, h_2, h_3, and h_4 of the optimal hypothesis f, with different error biases b_1, b_2, b_3, and b_4, respectively. As in ensemble learning, we expect that aggregating the different tool pairs will improve the output accuracy by reducing accumulated systematic error biases, especially when the combined tools are reliable but not valid with complemented biases (as portrayed in Figure 2).

Table 3. Average Performance (and Standard Deviation) of Majority Voting on Single-tool Aggregation

	Precision	Recall	F_1 score
Basic Trace	0.61 (0.18)	**0.99** (0.06)	**0.73** (0.15)
Drag-and-Drop	0.57 (0.15)	0.95 (0.11)	0.70 (0.13)
Pin-Placing	0.57 (0.16)	0.83 (0.16)	0.66 (0.16)
Floodfill	**0.85** (0.12)	0.70 (0.23)	**0.73** (0.18)

6 EVALUATION OF MULTI-TOOL AGGREGATION SCHEME

To evaluate the effectiveness of our tool-diversity approach, we conducted a series of studies to examine the performance improvement achieved from an ensemble of different tools. In the studies, we compared the performance of every possible tool aggregation: from single-tool to four-tool aggregations. As a baseline condition, we first investigate the segmentation quality of single-tool aggregation based on majority voting of four different workers. We implement a pixel-level majority voting method, with each answer weighted equally. In the second study, we do the same majority voting, except on two, three, and four tool combinations aggregating four workers' answers from different tools. In the last study, we apply the EM method on multi-tool aggregation to optimize the tools' weights adaptively per pixel. Our studies show that the tool-diversity approach is a workflow design strategy that can achieve aggregate performance at least as good as the superior constituent tool, and always significantly better than the inferior constituent tool.

6.1 Method 1. Single-Tool Aggregation with Majority Voting (Baseline)

Single-tool aggregation combines answers from the four workers who used the same tool to segment target objects. We randomly picked 15 worker combinations from the collected data. This was performed to avoid any bias from accidentally choosing a good or bad combination of workers. For each query object in the scene, pixel-level majority voting was performed to annotate each pixel as either background or object. If more than two workers labeled a pixel as belonging to the query object, then the pixel was included. The accuracy of the final segmentation was computed as in the Segmentation Quality Evaluation section (Section 5.3). The results of 15 randomly drawn combinations were averaged for all query objects. We summarized the average results in Table 3.

The change in precision was not significant with single-tool aggregation compared to the average precision without aggregation (see Table 2). However, recall and F_1 scores improved. For example, recall of Basic Trace increased by 10% ($p < .01$) compared to its average performance without aggregation. The increase in recall is a natural consequence of answer aggregation with low agreement thresholds. If the agreement threshold is higher, recall would decrease, because more consensus is needed to annotate a pixel as an "object." In the next sections, we observe if and how further improvements can be achieved with multi-tool aggregation.

6.2 Method 2. Multi-Tool Aggregation with Majority Voting

Adding multiple tools for the same task can improve the aggregate accuracy when the tools compensate for systematic error biases of each other. In this section, we look at the results of all possible tool combinations aggregated using pixel-level majority voting. We start by focusing on two-tool aggregate performance and then investigate three- and four-tool performance.

6.2.1 Two-Tool Aggregation. There are six possible two-tool aggregations for FourEyes, $\binom{4}{2} = 6$ (4 choose 2). For each tool pair, we randomly picked 15 pairs of workers from each tool, for a total of

Table 4. Average Performance (and Standard Deviation) of Majority Voting on Two-tool Aggregation

	Precision	Recall	F_1 score
Basic Trace × Drag-and-Drop	0.61 (0.13)	**0.98** (0.03)	0.74 (0.10)
Basic Trace × Pin-Placing	0.62 (0.13)	0.95 (0.08)	0.73 (0.11)
Basic Trace × Floodfill	**0.74** (0.12)	0.94 (0.11)	**0.81** (0.11)
Drag-and-Drop × Pin-Placing	0.57 (0.14)	0.92 (0.11)	0.69 (0.13)
Drag-and-Drop × Floodfill	0.71 (0.11)	0.93 (0.09)	0.79 (0.09)
Pin-Placing × Floodfill	0.69 (0.13)	0.86 (0.14)	0.75 (0.12)

Table 5. Average Performance (and Standard Deviation) of Majority Voting on Three-tool Aggregation

	Precision	Recall	F_1 score
Basic Trace × Drag-and-Drop × Pin-Placing	0.60 (0.13)	0.96 (0.05)	0.72 (0.11)
Basic Trace × Drag-and-Drop × Floodfill	**0.70** (0.12)	**0.97** (0.04)	**0.80** (0.09)
Basic Trace × Pin-Placing × Floodfill	0.69 (0.12)	0.94 (0.09)	0.78 (0.10)
Drag-and-Drop × Pin-Placing × Floodfill	0.65 (0.13)	0.92 (0.09)	0.75 (0.11)

four workers. As in Method 1, we computed pixel-level majority voting. The average performance of all possible tool pairs is summarized in Table 4.

Two-tool aggregation improves F_1 scores compared to single-tool aggregation. Every tool pair except Drag-and-Drop × Pin-Placing (0.69) showed increased F_1 scores compared to the single constituent tools. The pair gave better F_1 scores than aggregating Pin-Placing alone (0.66), but gave a 0.9% lower F_1 score than aggregating Drag-and-Drop alone (0.70). However, there was no statistically significant difference between single-tool aggregation of Drag-and-Drop versus multi-tool aggregation of Drag-and-Drop × Pin-Placing pair. We believe this pair did not increase performance, because Pin-Placing is a tool that is neither reliable nor valid, with the lowest and scattered performance distribution in terms of precision and recall metrics. One notable finding about the result is that the highest F_1 score achievable from single-tool aggregation is 0.73 (aggregating Floodfill alone), whereas that from multi-tool aggregation is 0.81 (aggregating Basic Trace × Floodfill pair), which is a 9.8% ($p < .005$) performance improvement with mixing tools. To emphasize the performance improvement in terms of F_1 score, we compared the F_1 scores of two-tool aggregations (blue bars) with their constituent tools (red and green bars) in Figure 10(a).

6.2.2 Three-Tool Aggregation. There are four possible three-tool aggregations for FourEyes, $\binom{4}{3} = 4$ (4 choose 3). For each tool aggregation, we randomly picked 15 combinations of workers: two from the first tool and one from each of the second and third tools, for a total of four workers. We maintained the same group size with two-tool aggregation to avoid interference from the effect of group size during comparison. The same pixel-level majority voting was conducted. The average performance of all possible tool combinations is summarized in Table 5.

Three-tool aggregation also improves F_1 scores compared to single-tool aggregation. Every tool aggregation except Basic Trace × Drag-and-Drop × Pin-Placing (0.72) showed increased F_1 scores compared to the single constituent tools. The aggregation gave a better F_1 score than aggregating Drag-and-Drop or Pin-Placing alone ($p < .005$), but there was no significant difference compared to Basic Trace. From the result, we observed that the aggregations that include both Basic

Table 6. Average Performance (and Standard Deviation) of Majority Voting on Four-tool Aggregation

	Precision	Recall	F_1 score
Basic Trace × Drag-and-Drop × Pin-Placing × Floodfill	0.65 (0.12)	0.95 (0.07)	0.76 (0.09)

Trace and Floodfill showed a significant performance improvement even compared to the superior constituent tools ($p < .05$). We hypothesize that including Floodfill in the tool set significantly improves accuracy, because it has the most different error bias compared to the others. That is, the diversity of systematic error biases affects the multi-tool aggregation performance. However, compared to two-tool aggregation, adding a third tool did not improve the performance compared to only combining Basic Trace with Floodfill. This could be because we lost the benefits of within-group aggregation, since only one worker contributed to each of the second and third tool types. We compared the F_1 scores of three-tool aggregations (blue bars) with their constituent tools (red and green bars) in Figure 10(b).

6.2.3 Four-Tool Aggregation. For aggregation of four tools, we randomly picked 15 combinations of workers, one from each tool. The average performance is summarized in Table 6. The comparison of F_1 scores of four-tool aggregation with that of the constituent tools is summarized in Figure 10(c).

Four-tool aggregation improves the F_1 score compared to any of the constituent tools. The four-tool aggregation results give us insight that increasing the number of tools to be combined does not linearly increase the aggregate performance. We hypothesize that the small group size hinders performance more than the benefits from adding more tool types, since having only one worker from one tool type results in lack of error correction from within-tool aggregation. That is, small groups with more tools do not necessarily improve performance, and to fully benefit from adding more tools, the group size should increase as well.

6.3 Method 3. Multi-Tool Aggregation with EM Method

In this section, we model the multi-tool aggregation problem as an optimization problem and use expectation maximization (EM) to estimate consensus-based semantic image segmentations. For certain tool aggregations, EM-based multi-tool aggregation significantly improved output accuracy over majority voting.

We model our problem as follows: Assume M crowd workers segment an object in an image A having N total pixels. Each pixel is labeled as either 1 (object) or 0 (background) by workers. The label a worker m assigns to each pixel is denoted as $z_{mn} \in \{0, 1\}$. We denote all labels from worker m as a vector $\boldsymbol{Z_m}$. The true label y_n, where $n = 1, \ldots, N$, of each pixel is unknown. The true labels of A to be estimated are denoted as a vector $\boldsymbol{Y}$. In the Dawid-Skene algorithm, it is assumed that the probability of worker m labeling a pixel is independent of choosing a pixel, i.e., it is a constant over n. That is, we assume i.i.d. (independent and and identical distributed) pixels. This assumption is acceptable, because we do not have *a priori* knowledge about the relationship between different pixels, making all pixels have the same chance of being included in a selection. In addition, we denote by $\boldsymbol{\theta}$ the confusion matrices set to be estimated. We can estimate the true labels Y by maximizing the marginal log-likelihood of the observed worker labels.

$$l(\boldsymbol{\theta}) := \log\left(\sum_{Y \in \{0,1\}^n} L(\boldsymbol{\theta}; \boldsymbol{Y}, \boldsymbol{Z})\right). \tag{2}$$

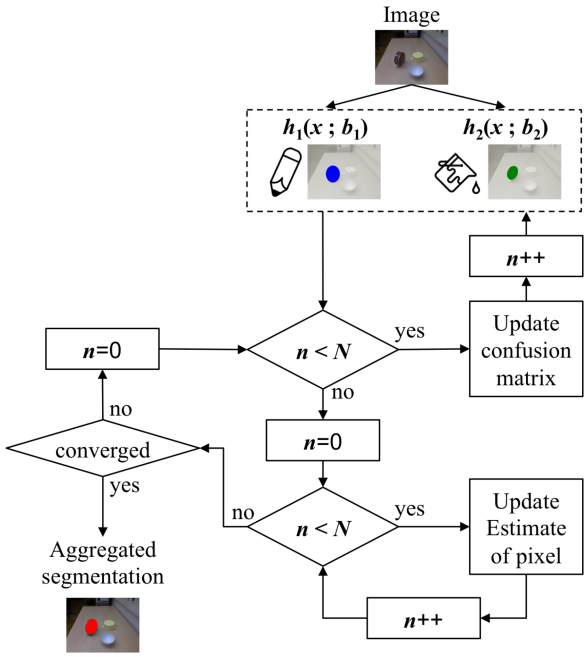


Fig. 9. The flowchart shows the EM algorithm we adopted for the optimization. Two different image segmentation tools, h_1 and h_2, each with different biases, b_1 and b_2 (respectively), pass segmented images to the system. We estimate the weights, w_1 and w_2, to approximate the performance of f.

The EM algorithm applies an expectation (E) step and a maximization (M) step iteratively:

E Step: Calculate the expected value of the log-likelihood function, with respect to the conditional distribution of $\boldsymbol{Y}$ given $\boldsymbol{Z}$ under the current estimate of $\boldsymbol{\theta}$.

M Step: Find the estimate $\boldsymbol{\theta}$ that maximizes the expectation of marginal log-likelihood.

The E and M steps are repeated until the estimations converge. The diagram in Figure 9 shows how the process is applied to our problem. The scene image is given as an input to two tools, h_1 and h_2, that have different error models, b_1 and b_2, respectively. Crowd workers use the tools to segment a query object, and the responses are transferred to the EM algorithm. Initial latent variables are set as the majority voting result, and the confusion matrix for each response is updated based on the initial assumption of the latent variables. Confusion matrices are updated by counting the number of false positive, false negative, true positive, and true negative pixels. Once the confusion matrices are updated for every pixel, the new estimations of latent variables are updated until convergence.

6.3.1 Two-Tool Aggregation. For a fair performance comparison, we used the same 15 worker groupings from Method 1 (Single-Tool Aggregation with Majority Voting) and 2 (Multi-Tool Aggregation with Majority Voting). We consider the ground truth labels as latent variables and estimate them jointly with the unknown parameters—the weight per tool on each pixel—of our model.

The accuracy of two-tool aggregation with the EM method is summarized in Table 7. The comparison of F_1 scores of the tool pairs with that of the constituent tools is summarized in Figure 10(a). The numbers for majority voting on single-tool aggregation are obtained from Method 1, and the numbers for majority voting on multi-tool aggregation are obtained from Method 2. The p-values were computed using two tailed t-tests with Bonferroni correction applied after each t-test. The results show that the EM-based multi-tool aggregation always performed significantly better than

Table 7. Average Performance (and Standard Deviation) of the EM Method on Two-tool Aggregation

	Precision	Recall	F_1 score
Basic Trace × Drag-and-Drop	0.63 (0.14)	**0.98** (0.02)	0.75 (0.11)
Basic Trace × Pin-Placing	0.63 (0.14)	0.93 (0.09)	0.74 (0.12)
Basic Trace × Floodfill	**0.75** (0.13)	0.93 (0.12)	**0.81** (0.12)
Drag-and-Drop × Pin-Placing	0.59 (0.15)	0.90 (0.12)	0.70 (0.13)
Drag-and-Drop × Floodfill	0.71 (0.13)	0.90 (0.11)	0.78 (0.10)
Pin-Placing × Floodfill	0.72 (0.14)	0.81 (0.14)	0.75 (0.14)

Table 8. Average Performance (and Standard Deviation) of the EM Method on Three-tool Aggregation

	Precision	Recall	F_1 score
Basic Trace × Drag-and-Drop × Pin-Placing	0.61 (0.13)	**0.99** (0.02)	0.74 (0.10)
Basic Trace × Drag-and-Drop × Floodfill	0.60 (0.13)	0.95 (0.07)	0.72 (0.11)
Basic Trace × Pin-Placing × Floodfill	**0.74** (0.13)	0.93 (0.12)	**0.81** (0.12)
Drag-and-Drop × Pin-Placing × Floodfill	0.57 (0.15)	0.92 (0.11)	0.69 (0.13)

the inferior constituent tool and performed at least as well as the superior constituent tool. The summarized result shows that the EM method significantly improves the performance of the tool pairs compared to uniform majority voting, except for two tool pairs (Drag-and-Drop × Floodfill pair and Pin-Placing × Floodfill pair). We observed that the highest aggregate-performance tool pairs were combinations of a high-precision (but low-recall) tool and a high-recall (but low-precision) tool, as shown in the third and fifth bar groups in Figure 10(a). While there was a significant performance improvement, the gain using EM was small (under 3%).

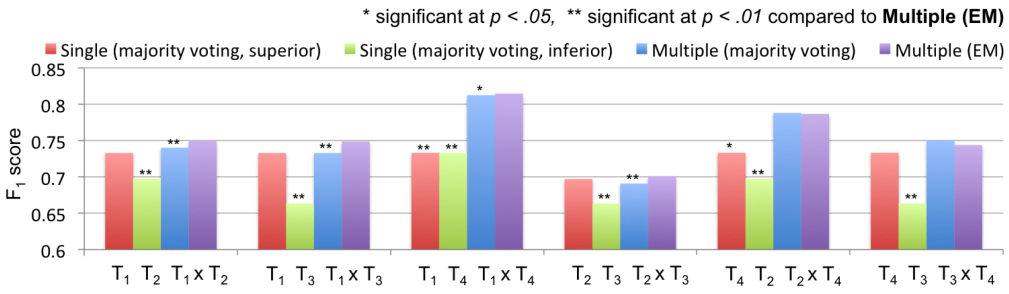
6.3.2 Three-Tool Aggregation. For a fair performance comparison, we used the same 15 worker groupings from Methods 1 and 2. We applied EM-based weight assignment for each pixel by setting the majority voting result as the initial weights. The accuracy of three-tool aggregation with EM is summarized in Table 8.

The comparison of F_1 scores of the three-tool aggregations with that of the constituent tools is summarized in Figure 10(b). Similar to two-tool aggregation result, the EM method improved the F_1 score significantly, but the gain was small (below 2%). The EM method significantly improved the performance of three-tool aggregations compared to majority voting, except for the aggregation of Drag-and-Drop × Pin-Placing × Floodfill. It is worth noting that while the EM method significantly improved accuracy for the tool pair Drag-and-Drop × Pin-Placing, adding Floodfill and forming a three-tool aggregation limited the benefits of the EM method. This implies that a more adaptive distribution of tool weights might be necessary when increasing tool aggregation complexity. This is discussed further in Section 7, where a correction mechanism is proposed to overcome the limitation of typical consensus-based pixel-level aggregation.

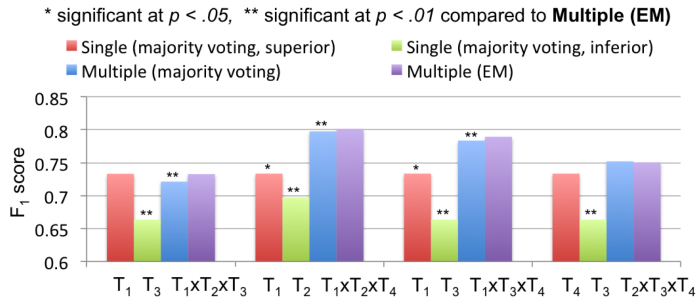
6.3.3 Four-tool Aggregation. We apply EM-based pixel-level weight assignment to the four-tool aggregation condition to evaluate if the method could further improve aggregate accuracy. We used the same 15 worker groupings from Methods 1 and 2. The result shows that the EM method significantly ($p < .01$) improves aggregate performance of four-tool aggregations. However, as in other tool aggregations, the gain was small (below 1%). The accuracy is summarized in Table 9. The comparison of F_1 scores of the tool pairs with those of the constituent tools is summarized in Figure 10(c).

Table 9. Average Performance (and Standard Deviation) of the EM Method on Four-tool Aggregation

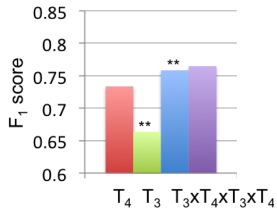
	Precision	Recall	F_1 score
Basic Trace × Drag-and-Drop × Pin-Placing × Floodfill	0.61 (0.13)	0.99 (0.02)	0.74 (0.10)



(a) Two-tool aggregation



(b) Three-tool aggregation



(c) Four-tool aggregation

Fig. 10. Accuracy comparison of different aggregation methods based on four tools: Basic Trace (T_1), Drag-and-Drop (T_2), Pin-Placing (T_3), and Floodfill (T_4). The blue bars are multi-tool aggregation with majority voting, and the purple bars are multi-tool aggregation with the EM method. The red bars are single-tool aggregation of the best-performing tool, and the green bars are single-tool aggregation of the worst-performing tool among all constituent tools. * significant at $p < .05$; ** significant at $p < .01$, both compared to EM-based multi-tool aggregation (two-tailed t-test). Leveraging tool diversity always performed significantly better than the inferior constituent tool, and performed at least as well as the superior tool.

In summary, combining answers from multiple tools increased the final segmentation accuracy compared to using the best single constituent tool alone. This performance improvement could be achieved by simple majority voting of segmentation results from different tools, and EM-based weight assignment to different tools could further improve the performance gains from majority voting. We analyzed multi-tool aggregation by varying tool combinations and learned that (1) the diversity of systematic error biases across tools can lead to further improvement in the aggregation performance, (2) increasing the number of tools does not necessarily improve the multi-tool aggregation accuracy, and (3) offsetting the trade-off between precision and recall is critical to improving aggregate performance in the image segmentation domain. We believe that maintaining a sufficient amount of within-group aggregation for each tool by increasing the group size of total workers is necessary to improve the multi-tool accuracy when increasing the number of tool types. In the next section, we will investigate how to further improve aggregate performance by exploring the error-correction mechanism for multi-tool aggregation.

7 ERROR CORRECTION METHODOLOGIES FOR MULTI-TOOL AGGREGATION

To further improve the accuracy of our tool-diversity scheme, we explore the idea of using error-correction mechanisms—post hoc processes to further improve the aggregate performance by correcting errors that remain even after aggregation. The exploration extends the use of our tool-diversity approach by providing options to improve aggregate accuracy even further when combining workers' answers across different tools. We first propose a morphological masking technique that can automatically balance errors between two biased tools. In image processing, morphological operations are defined as non-linear operations that transform images according to the shape or feature in an image. Our proposed method uses a non-linear operation to alter the label of each pixel by referring to the spatial feature of an aggregated result. Next, we explore the effect of varying the threshold parameter of the EM algorithm and suggest a simple methodology to adjust the threshold to find the best optimization parameter for each object to be segmented.

7.1 Morphological Masking Technique to Offset Biases between Different Tools

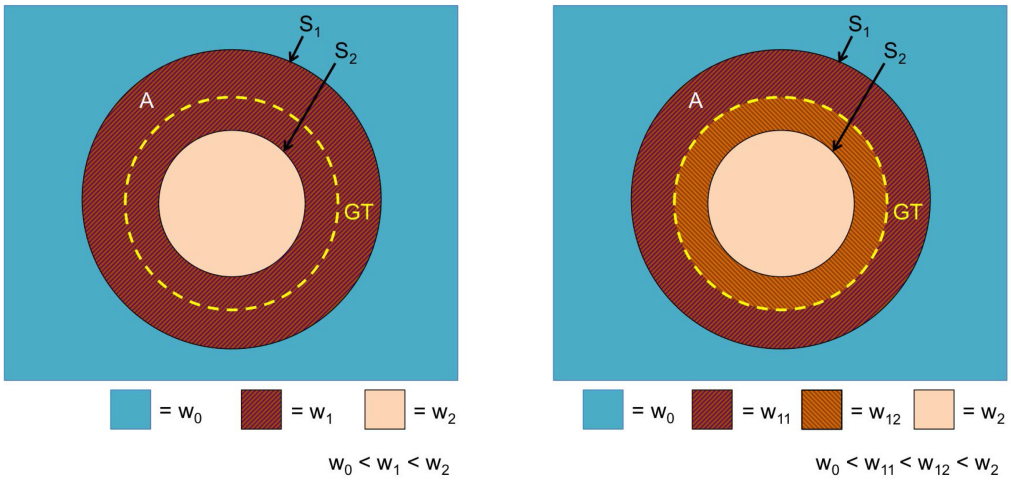
In Section 6, we observed that the aggregate accuracy of multiple tools can be improved when the tools each have different systematic error patterns. In this section, we introduce a region-based morphological masking technique that synthesizes more accurate segmentations by propagating the level of agreement of neighboring annotations. The morphological masking technique further compensates for the precision-recall trade-off by assigning an updated level of agreement to each pixel and segmenting the image based on a new threshold parameter.

The proposed correction mechanism can help improve aggregate accuracy by addressing several limitations faced by many consensus-based pixel-level aggregation methods, in particular those due to the fact that they do not fully make use of the rich spatial correlations between pixels in an image. The limitations and our corresponding solutions can be summarized as follows:

(1) Conventionally, the label of each pixel is estimated independently without referring to its neighboring pixels' labels, despite the fact that they can have strong spatial correlations. We propose utilizing the spatial correlation by considering the average level of agreement of neighboring pixels when deciding the final level of agreement of a single pixel.
(2) In most consensus-based methods, pixels with the same level of agreement for the same label are treated as equivalent, making it impossible to divide them into subgroups to increase the precision in labeling. We combat this problem by updating the level of agreement of pixels based on the average agreement of the neighboring pixels. For example, as in Figure 11(a), let us assume that S_1 and S_2 are two different segmentation results, while GT is the ground truth segmentation to be estimated. In terms of precision and recall, S_1 results in low precision and S_2 results in low recall. With general consensus-based methods, we cannot approximate the ground truth boundary, because the area $A(S_1 \cap S_2^c)$ with agreement level w_1 cannot be labeled with two different labels, e.g., foreground and background. However, by assigning updated level of agreement to pixels as in Figure 11(b), we can approximate GT by setting a threshold value between w_{11} and w_{12}.

Therefore, we propose a technique that applies morphological masking on each pixel to refer to its neighboring pixels, so, as in Figure 11(b), pixels can have better updated level of agreements.

7.1.1 Method. The morphological masking that we introduce is a region-based operation that can modify the aggregated segmentation result by synthesizing more accurate bounds through referring to the average of the surrounding annotations. In the method, the label of a pixel is



(a) Level of agreement by consensus-based aggregation (b) Ideal level of agreement to approximate ground truth

Fig. 11. The motivational concept of the morphological masking scheme. (a) S_1 indicates one segmentation, and S_2 indicates another. The yellow GT line indicates ground truth segmentation. Using general consensus-based aggregation (majority voting or EM), all the pixels within the area between S_1 and S_2 have the same level of agreement, w_1. However, to approximate GT, ideally, the area A ($S_1 \cap {S_2}^c$) needs different level of agreement as in (b), with w_11 and w_12. Our correction mechanism can approximate GT by giving updated level of agreement to pixels by referring to the agreement level of neighboring pixels.

updated by referring to the sum of the neighboring pixels' level of agreement:

$$\bar{w}_p = \sum_{i=0}^{M-1} \frac{w_i}{M}, \tag{3}$$

where $\bar{w}_p$ is the updated level of agreement of pixel p, M is the number of pixels inside the mask, and w_i is the original level of agreement of pixel i (the neighboring pixels). We also set a threshold parameter that decides the label of each newly updated pixel.

$$l_p = \begin{cases} 1, & \bar{w}_p > t \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where l_p is the label of pixel p, and t is the threshold parameter. The threshold parameter can be arbitrarily chosen within $0 < t < 1$ by the system designer.

The morphological masking updates the pixel agreement level around the transition area, for example, pixels close to line S_1 and S_2 in Figure 11(a), where the level is changing from w_0 to w_1 and from w_1 to w_2, respectively. Note that the pixels far from the transition area do not get influenced by the masking. If t is set small, the aggregated recall increases, because the pixels with small agreement level can be labeled as an object. If t is set large, the precision increases, because only pixels with large level of agreement can be labeled as an object. This feature makes it possible to better offset the precision-recall trade-off in multi-tool aggregation.

7.1.2 Evaluation and Results. We applied five different masking sizes and two different threshold parameters to explore the effect of our morphological masking technique. We randomly picked 10 samplings of four workers for each tool combination types as in Section 6 to avoid any bias from repeatedly choosing a good or bad combination of workers. The mask sizes chosen are $N = [5, 15, 25, 35, 45]$ for $N \times N$ masks, and the threshold parameters chosen are $t = 0.2$ and

$t = 0.5$. The experiment was implemented in Matlab 9.4 on a 3.5GHz Intel Core i7. We computed all five mask-size conditions at once, and it took about a minute per object to compute majority voting of four workers, and about 1.4mins per object to compute the EM-based weighting of four workers per object.

Figure 12 shows the morphological masking results of two different threshold parameters: 0.2 and 0.5. As expected, a low threshold ($t = 0.2$) increases recall but decreases precision, and a high threshold ($t = 0.5$) increases precision but decreases recall. The masking size also affected the performance. With $t = 0.2$, as the mask size increased, F_1 score decreased because of the steep decrease in precision but the low increase in recall. With $t = 0.5$, as the mask size increases, F_1 score increased except for Floodfill (annotated as T_4 in the figure), because precision largely increased while recall degraded no smaller than 0.55. The Floodfill (T_4) results in both Figures 12(b) and (d) show a nonlinear spike at mask size 25×25. This is because there were less valid data points for larger thresholds and larger mask sizes. There were many zero precision data points using Floodfill, which we didn't include in computing the average.

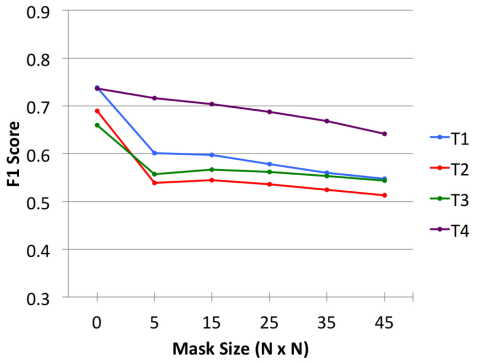
To further explore the effect of our morphological masking, we investigate the effect of mask size with $t = 0.5$ for all possible tool combinations of FourEyes. The result is shown in Figure 13. The left column shows the effect of masking on majority voting and the right column shows that on EM-based weighted aggregation. It is observed that masking *always* improves the aggregated result further up to maximum 5.8% for majority voting and 2.4% for EM. The mask size that induced the largest performance improvement varied by tool combination types. The mask size with 5, 15, and 35 induced maximum accuracy at least for one combination type.

In summary, applying our proposed morphological masking technique could further correct aggregation errors remaining in multi-tool aggregation. The effect of the masking technique was observed in every tool combination possible by FourEyes. This implies that not only the tools' design, but also the settings of the correction mechanism can affect the aggregate accuracy of multi-tool combinations.

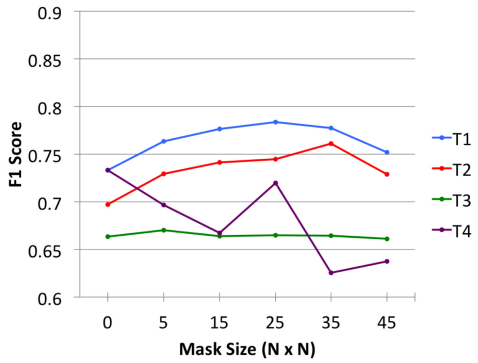
7.2 The Effect of the EM Threshold

In Section 6.3, we saw that EM-based weighted aggregation can improve the aggregate accuracy of the tool-diversity approach. However, we did not fully investigate the effect of the threshold parameter of EM on performance. Threshold parameter adjustment can be a simple and easy technique to apply in the post-processing stage if it gives a better result. Thus, we investigate the effect of different threshold parameters for EM-based weighted aggregation. We used the same 10 random samplings of four workers in Section 6 to avoid any sampling bias. Figure 14 shows F_1 scores of each tool combination with different EM thresholds. In Figure 14(a), we see that while the best performing EM threshold was 0.5, for Basic Trace × Drag-and-Drop, the highest performance was achieved when the threshold was 0.7. This is likely, because Basic Trace and Drag-and-Drop are the two tools with high recall but low precision characteristics. The large threshold parameter compensates the precision and recall trade-off by inducing higher precision when aggregated. In Figure 14(b), the highest accuracy was achieved when the threshold is 0.5, except for Basic Trace × Drag-and-Drop × Pin-Placing, which achieved the highest accuracy when the threshold is 0.7. This is expected, because without the Floodfill tool, which has the opposite characteristics from Basic Trace and Drag-and-Drop, trying to induce higher precision can help gain better compensation between the precision and recall trade-off. Thus, we suggest that future system designers optimize the threshold parameter when using our multi-tool aggregation approach by using methods like a parameter sweep to find the best threshold to leverage the characteristics of each tool.

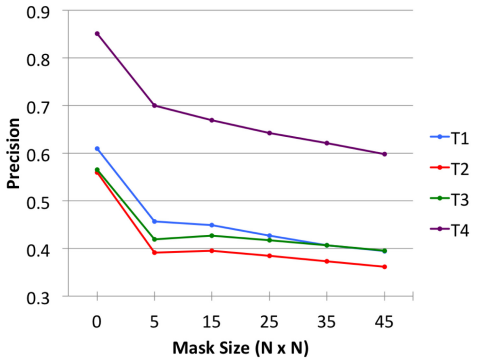
In summary, the threshold parameter of EM affected the aggregate performance. Different tool combinations had different threshold parameters that maximize their performance. This implies



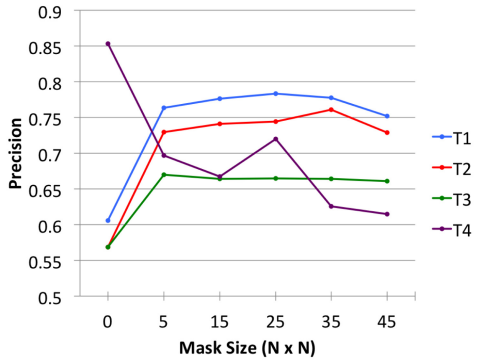
(a) F1 score $t = 0.2$



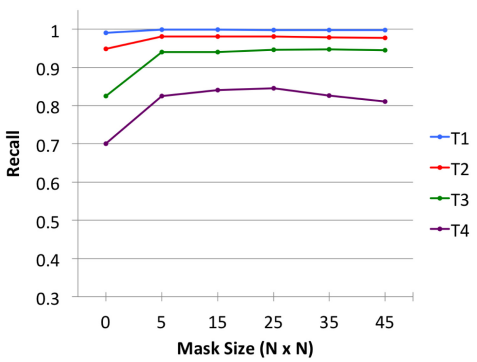
(b) F1 score $t = 0.5$



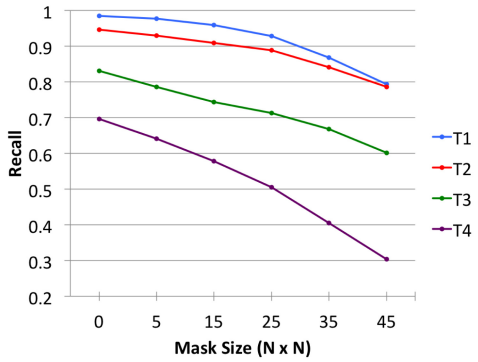
(c) Precision $t = 0.2$



(d) Precision $t = 0.5$

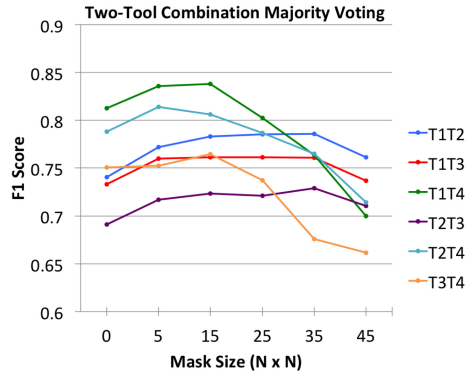


(e) Recall $t = 0.2$

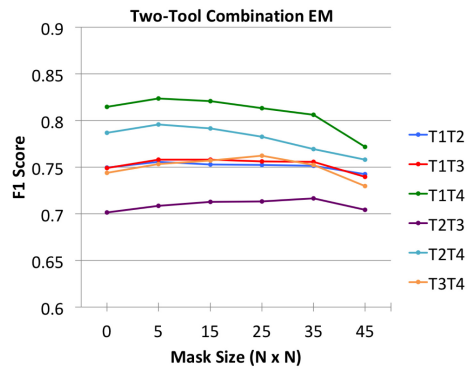


(f) Recall $t = 0.5$

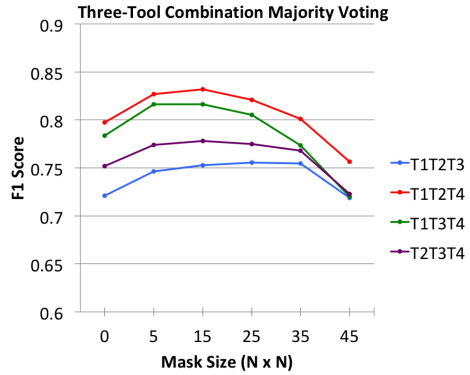
Fig. 12. Results of single-tool aggregation with different threshold parameters for the morphological masking ($T1$ = Basic Trace, $T2$ = Drag-and-Drop, $T3$ = Pin-Placing, and $T4$ = Floodfill). The left column shows F_1 score, precision, and recall for $t = 0.2$ and the right column shows F_1 score, precision, and recall for $t = 0.5$. With $t = 0.2$, the F_1 score degraded by applying the mask. This is because of the large decrease in the precision with only a small increase in recall. With $t = 0.5$, the F_1 score improved by applying the mask up to 6% (except for Floodfill). This is because the precision largely increased while recall degraded no larger than 0.23.



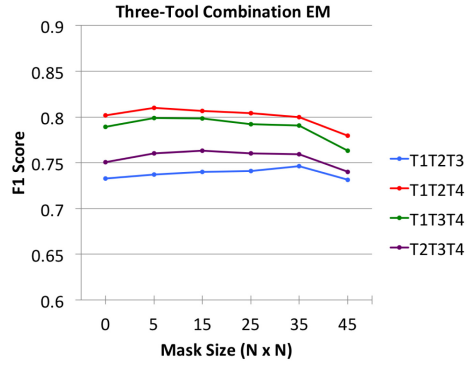
(a) Two tools combination (Majority Voting)



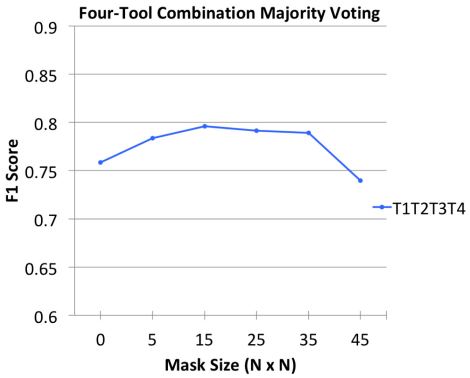
(b) Two tools combination (EM)



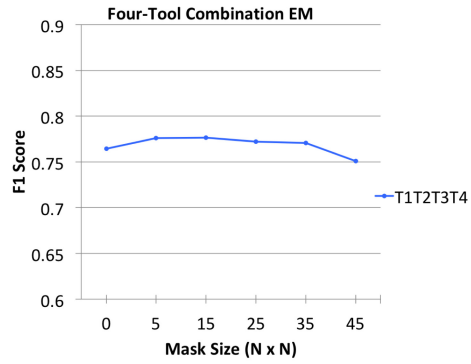
(c) Three tools combination (Majority Voting)



(d) Three tools combination (EM)

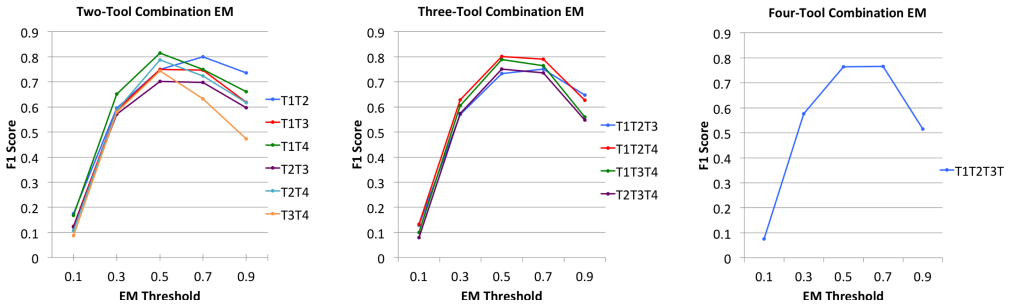


(e) Four tools combination (Majority Voting)



(f) Four tools combination (EM)

Fig. 13. F_1 scores of multi-tool aggregation with different masking sizes ($T1$ = Basic Trace, $T2$ = Drag-and-Drop, $T3$ = Pin-Placing, and $T4$ = Floodfill). The left column is F_1 score of majority voting and the right column is F_1 score of EM-based weighted aggregation. First row is two tools pairs, second row is three tools combinations, and third row is four tools combination results. Every multi-tool combination condition improved accuracy up to 6% by applying our masking technique. The mask size that induced the largest performance improvement varied by tool combination types.



(a) Two tools combination (b) Three tools combination (c) Four tools combination

Fig. 14. F_1 scores of every tool combination with five different EM thresholds (uniform intervals from 0.1 to 0.9). The result shows that the maximum performance that can be achieved varies by the threshold value, implying that correctly setting the EM threshold parameter can further improve the aggregate accuracy.

that a post hoc process of choosing the best threshold parameter can further improve the aggregate accuracy of crowdsourced answers. In the next section, we discuss some guidelines for system designers who aim to use tool diversity to improve performance of their crowd-powered system.

8 DISCUSSION

FourEyes's approach of leveraging tool diversity in designing a crowd-powered system goes beyond the paradigm of conventional crowdsourcing strategies, which divide a task into smaller microtasks and aggregate answers from workers using a single tool. FourEyes divides *tools*—and uses multiple different tools with different systematic error biases—to improve the accuracy of aggregate crowd answers. A practical concern in applying the tool-diversity approach is the cost and effort in building multiple tools with different (and even complementing) characteristics. In domains where multiple standardized tools already exist, e.g., image labeling or handwriting transcription, system designers can simply import and aggregate the existing tools without having to develop multiple customized tools. In this setting, we suggest that it is worth using all tools available and applying the various techniques introduced in this article, not just testing to find the best *one* tool.

However, we found that simply adding more tools does not linearly increase accuracy. This might be due to the small aggregate group size that we purposely constrained for fair comparisons with single-tool aggregation. That is, the insufficient number of *within-tool* workers contributing might have led to limited improvement when leveraging multiple tools. This implies that additional judgments such as the within-tool-group size could have effects on the overall accuracy of the multiple tools configuration. From our observation, we expect the multi-tool approach would improve accuracy with a larger aggregate group size, and it would be the same as we would expect for single-tool aggregation—adding more answers can improve the result, but sub-linearly. Also, the unique characteristics of a tool need to be considered, as they might affect the crowd's answer: e.g., the amount of interaction, the complexity of interface layout, and do on. To maximize the benefit of plugging in multiple tools, the tools combination should be carefully chosen by the system designer to maximize benefits from leveraging tool diversity. The following section discusses guidelines on using the tool-diversity approach in the system design.

8.1 Compensation of Biases in Leveraging Tool Diversity

To benefit from leveraging tool diversity in building crowdsourcing systems, a system or a task should have clear and distinct trade-offs in its accuracy, and it should be possible to build tools that can target one aspect at a time. Once different tools are built with distinctive properties in

terms of their systematic biases, an aggregated method that can offset the biases should be applied to merge the answers. Errors remaining after aggregation should be corrected using an error-correction mechanism such as our morphological masking. Precision and recall often have an inverse relationship, where one can be increased at the cost of reducing the other. In the crowdsourcing literature, research has investigated different payment schemes to observe the trade-off between precision and recall on object annotation tasks [39]. Our work suggests that different tools can be built to target either high precision or high recall so the harmonic means of both can be maximized by aggregating results from different methods. More generally, our results indicate that leveraging tool diversity in crowdsourcing tasks can improve aggregate crowd performance by compensating for various types of inherent individual systematic error biases.

8.2 Generalizability

While we demonstrate this new crowdsourcing paradigm using an image segmentation task, it could benefit any task where different approaches to solving the same problem can be devised. Specifically, tasks that have the following properties would be especially amenable to our approach:

- Expected correctness grows non-negatively with added worker input. In other words, on average, quality improves (collective answers converge to correct) as more worker responses are collected. Problems where majority voting works would belong to this class.
- The task is tractable enough to yield approximately correct responses from workers, but responses can be expected to have imperfections. Tasks such as real-time captioning [24] or handwriting recognition [41] are examples of such tasks.
- The task has an objectively correct answer, but also tolerates imperfections from workers' responses. For example, creative-writing tasks would *not* be a good fit, because there is no single correct answer, and they do not tolerate imperfections well (e.g., incomplete sentences).
- The expected human error is distributed differently when using different tools. This way, a diverse tool set can complement a broad range of error types. If this were not the case (i.e., if the errors were all biased in the same direction), then we would not expect multiple tools to be significantly more effective than a single one alone.

Many common crowdsourcing problems (e.g., in computer vision, natural language processing, or robotic/UI manipulation) have these properties, suggesting that a range of domains beyond the one explored in this article may also benefit from our approach.

8.3 Envisioned Scenario

In this section, we illustrate how our proposed multi-tool approach and the post-processing methods, aggregation and correction, can be strategically used in a probable scenario.

Crystal is a developer at a computer-vision startup company. Her team recently built several crowd-powered image segmentation tools that work pretty well in the lab, but she is not sure which one will work the best when deployed in the wild. Instead of trying to find the best performing tool among them, she decides to use all the tools that the team built by leveraging tool diversity (Section 3). Therefore, for a single segmentation task, the crowd answers from every tool are collected. She knows that the results will get better than any single tool used alone if the EM method described in this article (Section 6.3) is applied. This improvement can be gained without having to know the characteristics of each tool *a priori*. To further improve the final accuracy, she performs a parameter sweep to find the best EM threshold for the tool sets based on a small manually annotated ground truth sample from her dataset (Section 7.2). Additionally, she applies

a correction method that updates the level of agreement on each pixel being labeled as foreground or background. The correction method leverages the characteristics of the tools to more precisely correct for each tools' biases (Section 7.1). With this process, the team built a system that gives more accurate segmentation results than any single tool they built.

9 CONCLUSION AND FUTURE WORK

In this article, we have introduced a generalizable crowdsourcing approach of leveraging tool diversity to increase the output accuracy. When building a system, different tool designs can induce different worker performance, leading to different systematic error biases. Prior work has used task decomposition (into microtasks) to increase the reliability of a single task. However, systematic error biases can persist even after a task is divided as much as possible, if only a single tool is used for the task. We claim that these systematic error biases can be reduced by using multiple tools for the same task resulting in improved aggregate crowd performance. We demonstrated the effectiveness of the tool-diversity strategy in the domain of the semantic image segmentation problem. In our experiments, we used FourEyes, a crowd-powered image segmentation system that consists of four different image segmentation tools to segment diverse objects in different visual scenes. A series of studies showed that using multiple tools can significantly improve the aggregate accuracy of a single task, especially when the trade-off between the aggregated tools is high and the aggregation and correction method offsets the trade-off in the right direction. Overall, our findings present new opportunities and directions for gaining a deeper understanding of how tool designs influence the aggregate performance on crowdsourcing tasks, and introduces a new way of thinking about decomposing tasks: based on tools instead of subtasks.

Future work may investigate methodologies for leveraging tool diversity in other domains, such as video coding [23], annotation of fine-grained categories [10], or activity recognition [27]. For instance, using multiple tools for the same task may benefit any NLP task with multiple channels. A system designer can devise a tool that focuses on processing a text channel while sacrificing the audio channel, and aggregate the result with a tool that focuses on the audio channel while sacrificing processing of the text channel. Furthermore, this approach may open new ways of optimizing the effort from both humans and computers—considering them as different resources with different systematic error biases—to leverage the best of both worlds.

ACKNOWLEDGMENTS

The authors would like to thank Stephanie O'Keefe, Alan Lundgard, Fan Yang, Kyle Wang, and Markos V. Koutras for their input on this work.

REFERENCES

[1] Amy Bearman, Olga Russakovsky, Vittorio Ferrari, and Li Fei-Fei. 2016. What's the point: Semantic segmentation with point supervision. In *Proceedings of the European Conference on Computer Vision*. Springer, 549–565.

[2] Sean Bell, Paul Upchurch, Noah Snavely, and Kavita Bala. 2013. OpenSurfaces: A richly annotated catalog of surface appearance. *ACM Trans. Graph.* 32, 4 (2013), 111.

[3] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. 2010. Soylent: A word processor with a crowd inside. In *Proceedings of the 23rd ACM Symposium on User Interface Software and Technology*. ACM, 313–322.

[4] Jonathan Bragg, Mausam, and Daniel S. Weld. 2013. Crowdsourcing multi-label classification for taxonomy creation. In *Proceedings of the 1st AAAI Conference on Human Computation and Crowdsourcing*.

[5] Axel Carlier, Vincent Charvillat, Amaia Salvador, Xavier Giro-i Nieto, and Oge Marques. 2014. Click'n'Cut: Crowdsourced interactive segmentation with object candidates. In *Proceedings of the International ACM Workshop on Crowdsourcing for Multimedia*. ACM, 53–56.

[6] Alexander Philip Dawid and Allan M. Skene. 1979. Maximum likelihood estimation of observer error-rates using the EM algorithm. *Appl. Stat.* 28, 1 (1979), 20–28.

[7] Thomas G. Dietterich et al. 2000. Ensemble methods in machine learning. *Mult. Class. Syst.* 1857 (2000), 1–15.
[8] Steven Dow, Anand Kulkarni, Scott Klemmer, and Björn Hartmann. 2012. Shepherding the crowd yields better work. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. ACM, 1013–1022.
[9] Yoav Freund and Robert E. Schapire. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the European Conference on Computational Learning Theory*. Springer, 23–37.
[10] Timnit Gebru, Jonathan Krause, Jia Deng, and Li Fei-Fei. 2017. Scalable annotation of fine-grained categories without experts. In *Proceedings of the International Conference on Human Factors in Computing Systems*. ACM, 1877–1881.
[11] Mitchell Gordon, Jeffrey P. Bigham, and Walter S. Lasecki. 2015. LegionTools: A toolkit+ UI for recruiting and routing crowds to synchronous real-time tasks. In *Adjunct Proceedings of the 28th ACM Symposium on User Interface Software & Technology*. ACM, 81–82.
[12] Sai Gouravajhala, Jean Y. Song, Jinyeong Yim, Raymond Fok, Yanda Huang, Fan Yang, Kyle Wang, Yilei An, and Walter S. Lasecki. 2017. Towards hybrid intelligence for robotics. In *Proceedings of the Collective Intelligence Conference (CI'17)*.
[13] Danna Gurari, Mehrnoosh Sameki, and Margrit Betke. 2016. Investigating the influence of data familiarity to improve the design of a crowdsourcing image annotation system. In *Proceedings of the AAAI Conference on Human Computation & Crowdsourcing (HCOMP'16)*.
[14] Lars Kai Hansen and Peter Salamon. 1990. Neural network ensembles. *IEEE Trans. Pattern Anal. Machine Intell.* 12, 10 (1990), 993–1001.
[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. 2017. Mask R-CNN. *R*etrieved from: *CoRR* abs/1703.06870.
[16] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. 2010. Quality management on Amazon Mechanical Turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*. ACM, 64–67.
[17] Alexandre Kaspar, Genevieve Patterson, Changil Kim, Yagiz Aksoy, Wojciech Matusik, and Mohamed Elgharib. 2018. Crowd-guided ensembles: How can we choreograph crowd workers for video segmentation? In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'18)*. ACM, New York, NY, Article 111, 111:1–111:12 pages.
[18] Harmanpreet Kaur, Mitchell Gordon, Yiwei Yang, Jeffrey P. Bigham, Jaime Teevan, Ece Kamar, and Walter S. Lasecki. 2017. Crowdmask: Using crowds to preserve privacy in crowd-powered systems via progressive filtering. In *Proceedings of the AAAI Conference on Human Computation (HCOMP'17)*, Vol. 17.
[19] Juho Kim, Phu Tran Nguyen, Sarah Weir, Philip J. Guo, Robert C. Miller, and Krzysztof Z. Gajos. 2014. Crowdsourcing step-by-step information extraction to enhance existing how-to videos. In *Proceedings of the 32nd ACM Conference on Human Factors in Computing Systems (CHI'14)*. ACM, New York, NY, 4017–4026.
[20] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. 2011. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology*. ACM, 43–52.
[21] Anand Kulkarni, Matthew Can, and Björn Hartmann. 2012. Collaboratively crowdsourcing workflows with Turkomatic. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. ACM, 1003–1012.
[22] Walter Lasecki and Jeffrey Bigham. 2012. Self-correcting crowds. In *CHI'12 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2555–2560.
[23] Walter S. Lasecki, Mitchell Gordon, Danai Koutra, Malte F. Jung, Steven P. Dow, and Jeffrey P. Bigham. 2014. Glance: Rapidly coding behavioral video with the crowd. In *Proceedings of the 27th ACM Symposium on User Interface Software and Technology*. ACM, 551–562.
[24] Walter S. Lasecki, Christopher Miller, Adam Sadilek, Andrew Abumoussa, Donato Borrello, Raja Kushalnagar, and Jeffrey Bigham. 2012. Real-time captioning by groups of non-experts. In *Proceedings of the 25th ACM Symposium on User Interface Software and Technology*. ACM, 23–34.
[25] Walter S. Lasecki, Christopher D. Miller, and Jeffrey P. Bigham. 2013. Warping time for more effective real-time crowdsourcing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'13)*. ACM, New York, NY, 2033–2036.
[26] Walter S. Lasecki, Kyle I. Murray, Samuel White, Robert C. Miller, and Jeffrey P. Bigham. 2011. Real-time crowd control of existing interfaces. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology*. ACM, 23–32.
[27] Walter S. Lasecki, Young Chol Song, Henry Kautz, and Jeffrey P. Bigham. 2013. Real-time crowd labeling for deployable activity recognition. In *Proceedings of the Conference on Computer Supported Cooperative Work*. ACM, 1203–1212.
[28] Matthew Lease, Jessica Hullman, Jeffrey P. Bigham, Michael S. Bernstein, Juho Kim, Walter S. Lasecki, Saeideh Bakhshi, Tanushree Mitra, and Robert C. Miller. 2013. Mechanical Turk is not anonymous. *Soc. Sci. Res. Netw.* (2013). DOI: http://dx.doi.org/10.2139/ssrn.2228728
[29] Christopher Lin, Mausam Mausam, and Daniel Weld. 2012. Dynamically switching between synergistic workflows for crowdsourcing. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
[30] Christopher H. Lin, Mausam Daniel, and S. Weld. 2012. Crowdsourcing control: Moving beyond multiple choice. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI'12)*.

[31] Di Lin, Jifeng Dai, Jiaya Jia, Kaiming He, and Jian Sun. 2016. Scribblesup: Scribble-supervised convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3159–3167.

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision*. Springer, 740–755.

[33] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. 2010. Turkit: Human computation algorithms on Mechanical Turk. In *Proceedings of the 23nd ACM Symposium on User Interface Software and Technology*. ACM, 57–66.

[34] Ching Liu, Juho Kim, and Hao-Chuan Wang. 2018. ConceptScape: Collaborative concept mapping for video learning. In *Proceedings of the Conference on Human Factors in Computing Systems*. ACM, 387.

[35] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*.

[36] Alan Lundgard, Yiwei Yang, Maya L. Foster, and Walter S. Lasecki. 2018. Bolt: Instantaneous crowdsourcing via just-in-time training. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'18)*. ACM, New York, NY.

[37] Kurt Luther, Nathan Hahn, Steven P. Dow, and Aniket Kittur. 2015. Crowdlines: Supporting synthesis of diverse information sources through crowdsourced outlines. In *Proceedings of the 3rd AAAI Conference on Human Computation and Crowdsourcing*.

[38] Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. 1991. Questions, options, and criteria: Elements of design space analysis. *Human–Comput. Interact.* 6, 3–4 (1991), 201–250.

[39] Andrew Mao, Ece Kamar, Yiling Chen, Eric Horvitz, Megan E. Schwamb, Chris J. Lintott, and Arfon M. Smith. 2013. Volunteering versus work for pay: Incentives and tradeoffs in crowdsourcing. In *Proceedings of the 1st AAAI Conference on Human Computation and Crowdsourcing*.

[40] Christian A. Meissner and John C. Brigham. 2001. Thirty years of investigating the own-race bias in memory for faces: A meta-analytic review. *Psych., Pub. Polic. Law* 7, 1 (2001), 3.

[41] Tom Ouyang and Yang Li. 2012. Bootstrapping personal gesture shortcuts with the wisdom of the crowd and handwriting recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'12)*. ACM, New York, NY, 2895–2904.

[42] Akshay Rao, Harmanpreet Kaur, and Walter S. Lasecki. 2018. Plexiglass: Multiplexing passive and active tasks for more efficient crowdsourcing. In *Proceedings of the AAAI Conference on Human Computation*. ACM.

[43] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. 2008. LabelMe: A database and web-based tool for image annotation. *Int. J. Comput. Vis.* 77, 1 (2008), 157–173.

[44] Jeffrey M. Rzeszotarski and Aniket Kittur. 2011. Instrumenting the crowd: Using implicit behavioral measures to predict task performance. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology*. ACM, 13–22.

[45] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. 2008. Cheap and fast—but is it good?: Evaluating non-expert annotations for natural language tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 254–263.

[46] Jean Y. Song, Raymond Fok, Alan Lundgard, Fan Yang, Juho Kim, and Walter S. Lasecki. 2018. Two tools are better than one: Tool diversity as a means of improving aggregate crowd performance. In *Proceedings of the 23rd International Conference on Intelligent User Interfaces (IUI'18)*. ACM, New York, NY, 559–570.

[47] Saiganesh Swaminathan, Raymond Fok, Fanglin Chen, Ting-Hao Kenneth Huang, Irene Lin, Rohan Jadvani, Walter S. Lasecki, and Jeffrey P. Bigham. 2017. WearMail: On-the-go access to information in your email with a privacy-preserving human computation workflow. In *Proceedings of the 30th ACM Symposium on User Interface Software and Technology*. ACM, 807–815.

[48] Shane Torbert. 2016. *Applied Computer Science*. Springer. 158 pages.

[49] Peter Welinder, Steve Branson, Pietro Perona, and Serge J. Belongie. 2010. The multidimensional wisdom of crowds. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2424–2432.

[50] Jacob Whitehill, Ting Fan Wu, Jacob Bergsma, Javier R. Movellan, and Paul L. Ruvolo. 2009. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2035–2043.

[51] Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z. Gajos, Walter S. Lasecki, and Neil Heffernan. 2016. Axis: Generating explanations at scale with learnersourcing and machine learning. In *Proceedings of the 3rd ACM Conference on Learning@ Scale*. ACM, 379–388.

Received May 2018; revised July 2018; accepted July 2018