Zensors: Adaptive, Rapidly Deployable, Human-Intelligent Sensor Feeds

Gierad Laput[1] Walter S. Lasecki[2] Jason Wiese[1] Robert Xiao[1] Jeffrey P. Bigham[1] Chris Harrison[1]

[1] HCI Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
{glaput, jwwiese, brx, jbigham, chris.harrison}
@cs.cmu.edu

[2] ROC HCI, University of Rochester
500 Joseph C. Wilson Boulevard
Rochester, NY 14627
wlasecki@cs.rochester.edu

ABSTRACT

The promise of "smart" homes, workplaces, schools, and other environments has long been championed. Unattractive, however, has been the cost to run wires and install sensors. More critically, raw sensor data tends not to align with the types of questions humans wish to ask, e.g., do I need to restock my pantry? Although techniques like computer vision can answer some of these questions, it requires significant effort to build and train appropriate classifiers. Even then, these systems are often brittle, with limited ability to handle new or unexpected situations, including being repositioned and environmental changes (e.g., lighting, furniture, seasons). We propose *Zensors*, a new sensing approach that fuses real-time human intelligence from online crowd workers with automatic approaches to provide robust, adaptive, and readily deployable intelligent sensors. With Zensors, users can go from question to live sensor feed in less than 60 seconds. Through our API, Zensors can enable a variety of rich end-user applications and moves us closer to the vision of responsive, intelligent environments.

Author Keywords: Smart environments; sensing; human computation; computer vision; machine learning; end-user programming;

ACM Keywords: H.5.2. [Information interfaces and presentation]: User interfaces – Input devices and strategies.

INTRODUCTION

For decades, "intelligent" environments have promised to improve our lives by inferring context, activity and events in diverse environments, ranging from public spaces, offices, and labs, to homes and healthcare facilities. To achieve this vision, smart environments require sensors, and lots of them. However, installation continues to be expensive, special purpose, and often invasive (e.g., running power).

An even more challenging problem is that sensor output rarely matches the types of questions humans wish to ask. For example, a door opened/closed sensor may not answer the user's true question: "are my children home from school?" A restaurateur may want to know how many patrons need their beverages refilled, and graduate students want to know, "is there free food in the kitchenette?" Unfortunately, these sophisticated, multidimensional and often contextual questions are not easily answered by the simple sensors we deploy today. Although advances in sensing, computer vision (CV) and machine learning (ML) have brought us closer, systems that generalize across these broad and dynamic contexts do not yet exist.

In this work, we introduce *Zensors*, a new sensor approach that requires minimal and non-permanent sensor installation and provides human-centered and actionable sensor output. To achieve this, we fuse answers from crowd workers and automatic approaches to provide instant, human-intelligent sensors, which end users can set up in under one minute.

To illustrate the utility of Zensors, we return to our restaurant example. John, the proprietor, finds a disused smartphone or tablet and affixes it to the wall of his restaurant. He installs and launches our Zensors app, which uses the front facing camera to provide a live overview of the restaurant. John presses the "new sensor" button and circles the bar countertop, thus specifying a region of interest. John can then enter a plain text question, for example: *"how many drinks are almost empty?"*

By pressing "go", the sensor is activated and starts providing real-time data, in this case numerical. John can now use e.g., a companion app to see a real time visualization of how many drinks need to be refilled, or use an end user programming tool to have the system automatically message a co-worker requesting help if the number exceeds ten. Within a few minutes, John could similarly set up sensors for: *"does table four need to be cleaned?"*, *"are customers wearing their coats inside?"*, *"is a check sitting on the table?"* and other questions relevant to the dining experience.

Unbeknownst to John, his sensors are initially powered by crowd workers interpreting his plain text question, providing immediate human-level accuracy, as well as rich, human-centered abstractions. However, using crowd workers can be costly and difficult to scale, and so ideally it is only used temporarily – answers from the crowd are recorded

CHI 2015, April 18 - 23 2015, Seoul, Republic of Korea

ACM 978-1-4503-3145-6/15/04 $15. http://dx.doi.org/10.1145/2702123.2702416

and used as labels to bootstrap automatic processes. More specifically, our system begins training and testing image-based machine learning classifiers, testing against the ground truth provided by the crowd labels. If the classifiers begin to achieve human-like accuracies, they begin to vote alongside crowd responses. Eventually, if sufficiently robust, the classifiers can take full control. This human-computer handoff is seamless and invisible to end-users; as far as users like John are concerned, they have a sensor with human-level accuracy from minute one and onward. Even when the classifiers do not achieve the needed level of accuracy, we have designed savings measures into our crowd-based method to conserve costs. Through our API, Zensors enables a variety of applications that help realize the potential of smart environments. This abstracts the complexity of crowdsourcing, computer vision and machine learning, enabling developers to treat our "zensors" just as they would traditional electro-mechanical sensors.

We designed Zensors to achieve five key criteria: 1) answer diverse natural language "sensor" questions, 2) with reasonably high-accuracy, 3) while being easy enough to be used by non-experts, 4) requiring zero training, and 5) receive live data within seconds. We are unaware of a single prior computer vision system that achieves these five properties.

We make the following contributions in this work:

- Zensors, a new approach and architecture for hybrid crowd-ML powered sensors, with an API to enable access to sensor data streams.
- A proof-of-concept mobile application for easy, end-user authoring of on-demand intelligent sensors.
- A tool for end-user programming of case-based events that turn sensor output into meaningful actions.
- A study that demonstrates the accuracy and reliability of Zensors in a variety of settings.
- Evidence that our human-powered sensors can be used to train computer vision approaches in situ, leading to an automatic handoff in most cases.

RELATED WORK

Our work touches on several areas including crowd-driven annotation, computer vision, and more generally, smart environments and activity sensing. We now summarize key related work in these respective domains.

Crowd-Driven Annotation

Crowdsourcing allows systems to access human intelligence through online marketplaces such as Mechanical Turk. For example, the ESP Game asked workers to label images with keywords in order to make the web more accessible [24]. VizWiz [3] asks crowd workers to answer visual questions for blind users with a latency of under a minute. However, both VizWiz and the ESP Game have difficulty scaling because they elicit un-typed, natural language responses to non-repeated images, making it difficult for machine learning algorithms to learn key patterns with high accuracy.

Marcus et al. [20] explored how to perform SQL query estimations that computers cannot do alone, e.g., answer how many people in a database of images were of a certain gender. In their system, users could pre-define functions that specified crowd tasks, which were used to find an answer estimation for query filtering. CrowdDB [10] explored a similar crowdsourcing concept, using the crowd to fill in missing data and overcome the closed-world assumption held in traditional DBs. In both of these examples, the language the user must use to pose queries is a slight variant of SQL, and the crowd's input is used to estimate relationships and missing data over large, pre-existing datasets. In contrast, Zensors turn a live stream of images into an easily accessible structured data stream.

Tohme [13] use the crowd to improve accessibility information in maps by asking workers to label curbs in Google StreetView images. Unlike Zensors, Tohme addressed a single, specific question, using a specialized interface to facilitate labeling. Tohme also included a computer vision component that enabled it to learn how to identify curbs in specific areas over time. They demonstrated that this automated approach could be successful. However it, too, relied on the detailed object segmentation and labeling information obtained from workers through the custom interface (and in combination with GPS and other metadata from Google Maps). Zensors, by contrast, targets general-purpose domains in which no additional information is available, and answers a broad range of user-defined questions.

VATIC [25] uses the crowd to annotate video with labels and object bounding boxes. Glance [17] annotates spans of time with user-requested event labels within minutes by leveraging the ability of large sets of crowd workers to concurrently complete an otherwise time-consuming task. Both VATIC and Glance are approaches for analyzing previously recorded, fixed-length video. By contrast, Zensors is focused on real-time analysis of still images, and offers the potential for handing sensor labeling off from the crowd to machine learning algorithms.

Legion:AR [18] recruits the crowd to provide activity labels using data from live video and RFID tags. It uses an active learning approach to request crowd labels for partial streams of live video. While this solution is related to Zensors, the crowd architecture is different. Legion:AR does not use computer vision, only attempts automatic activity recognition via RFID tags, and collects open-ended questions and answers, not *typed* values (i.e., data type). More specifically, Legion:AR collects open-ended plain text answers, in contrast to the structured labels of Zensors. Further, Legion:AR gathers multiple workers into a single session for a long duration, having them synchronously generate multiple incomplete sets of labels that are merged into a final stream—a configuration that costs tens of dollars for a few minutes. Legion:AR monitors an *instrumented* space (RFID scanners, Kinects), and was not designed to be a *generalizable* sensing platform.

Image- and Video-Based Sensors

The field of computer vision has largely developed in pursuit of extracting meaning from images without the need for human assistance. Today, there exist robust and automatic approaches for applications as diverse as face detection [28] and traffic monitoring [15]. Most related to our present work are video-based, end-user sensing systems, such as the techniques proposed in Crayons [8] and the Eyepatch system [21]. Light Widgets [9] allowed users to create virtual interactive elements on everyday surfaces through video-based sensing. Other systems, e.g. Slit-Tear Visualizations [23], aim to help users better recognize environmental events. These systems often rely on users annotating images to provide training data or demarcate regions of interest. Researchers have also explored crowd-based approaches for improving computer vision techniques [7,26], which could be used to enhance systems such as our own.

Smart Environments

A broad body of work in the HCI and UbiComp communities has outfitted home, work and public environments and objects in those environments with sensors to detect various activities in the environment [1]. The Context Toolkit [22] utilized data from numerous sensors to make contextual inferences about a place (e.g. what is going on in a space, or how many people are around). In a project to support aging-in-place, Lee and Dey created an augmented pillbox that detected whether or not an older adult had taken their pills for a given day [19]. In another example, the AwareOffice project [29] outfitted various office artifacts with diverse sensors to detect their state, including: whiteboard pen (writing or not), eraser (wiping whiteboard), chair (in use or free), window & door (open/closed).

While there are clearly many benefits of dedicated hardware sensors in UbiComp smart environments, there are many drawbacks that limit their use. Compared to Zensors, these approaches are fairly heavyweight, requiring one sensor per item that is being sensed. In contrast, by segmenting the view from a single camera to evaluate the current state of many objects, Zensors offers this same functionality with just a single device. Furthermore, Zensors enables almost anyone to answer arbitrary questions about their environment, without requiring the hardware/software technical skills needed to implement traditional sensor approaches.

APPLICATION SPACE

To help us understand the expanse of questions and functionalities enabled by a human-intelligent, camera-driven sensing system, we recruited 13 interaction designers and solicited their input in a structured ideation session.

First, participants were split into four groups, and each group was asked to enumerate scenarios that would greatly benefit from a "super smart environment sensor." After fifteen minutes, groups were asked to share their ideas with one another. Immediately after, four groups were merged into two, and the new teams were asked to discuss potential issues from the ideas mentioned prior. Next, all groups were asked to examine the area of "camera-based sensing." Specifically, we asked them to generate and categorize "questions" that they might want to ask from a "highly intelligent" camera-based sensing system. Finally, using input from the previous round, we asked participants to "program" image-based sensing applications (using a mockup software we provided) inspired from the questions they previously explored. In total, the session lasted 90 minutes. All participants were briefed at the end of the session, and were compensated for their time.

Data gathered from the session were analyzed using affinity-diagramming techniques. From this, several themes emerged from the categories of questions generated by each group, including optimization of domestic tasks (e.g., "what food in my fridge will go bad if I don't cook them the next day?"), health (e.g., "who is experiencing the most severe depression right now?"), public space monitoring (e.g., "how many cars are in this parking lot?"), and item finding (e.g., "where did I last leave my keys?"). These questions were also tied to a strong set of contexts and environments: including the home, urban and public spaces, educational institutions (e.g., "are students interested in the topic?"), health facilities, and supply chains (e.g., "what items should I restock from my inventory?"). When asked about issues concerning the deployment of "super smart environment sensing," participants cited consent and privacy (e.g., data storage), as well as technical feasibility (e.g., sensing capabilities, computation) as chief concerns. Altogether, these results draw out the expansive set of applications that would otherwise be hard or impossible to implement with CV-based technologies alone.

COMPARISION TO COMPUTER VISION

Additionally, we wanted to understand the work requirements and economic implications of building smart environment sensing using existing software engineering approaches. Intuitively, one or more experienced programmers would be able to implement most of our proposed intelligent sensors. But, how long would it take, and at what cost?

We conducted informal interviews with nine freelance developers from oDesk.com, a popular online workplace. To recruit capable candidates, we created a targeted job posting (titled "Image-Based Sensing System"), asking experienced software developers to build a computer vision-based automatic "bus detection" system. The job posting included illustrative details and clear task requirements (e.g., system should perform with at least 90+% accuracy), along with a labeled image set (e.g., a view of a bus stop on a city street).

Interested candidates were asked a series of follow-up questions, most notably: "how long do you think the project will take?", "what is your hourly rate?" and "how many hours will you expect to work per week?" Candidates were encouraged to ask follow-up questions. All interviews were conducted online using oDesk's internal messaging system. Afterwards, participants were debriefed (e.g., we clarified the true intent of the job posting), and were paid $25.

Across all users, the average estimated project cost was $3,044 (SD=$2,113), with an average completion time of 4.5 weeks (SD=2.2 weeks), excluding time spent for data collection. While these results are anecdotal – given that none of the developers went ahead and actually built the system – we do believe it illustrates the significant complexity, cost and development time of special purpose computer-vision-driven sensing systems.

ZENSORS

We now describe how Zensors provides end-users with intelligent sensing capabilities, leveraging both crowdsourcing and machine learning. Figure 1 provides an illustrated overview of the architecture.

System Architecture

First, a mobile application serves as the primary end-point for creating and modifying sensors. Users highlight a region of the camera image and an associated question, e.g., "how many parking spots are available?" As images stream from the device's front-facing camera, our system intelligently decides when to push requests to the crowd.

Next, crowd workers process these requests through a web-based interface. To reduce noise and reject malicious workers, several answers are collected per image, then fused together using quality-control algorithms (e.g., voting) to determine the best response for a given instance. Finally, the responses gathered from the crowd are stored into a back-end database. These responses provide immediate, human-intelligent answers to the questions asked by users, and also serve as a corpus for training computer-vision based, machine learning classifiers.

Sensors from Images

Leveraging cameras as multi-purpose sensors. From mobile phones, security cameras, and Kinects in people's living rooms, cameras are everywhere. They continue to become more powerful, while remaining small. More importantly, time-series data from cameras offers rich contextual information about an activity or environment far more than what basic sensors (e.g., proximity) can provide. One can ask several multi-dimensional questions from camera images across a time period, such as "how many people are smiling?", "is it sunny?" or "is the table messy?", all of which provide useful information in learning about the context or activity within an environment. Thus, the cost, availability, and information bandwidth that cameras offer make them an ideal "multi-purpose" commodity sensor.

Repurposing old mobile devices as sensor hosts. Users upgrade their devices on average once every two years [27]. It is not uncommon for people to have a slew of older smart devices stashed in a drawer or closet. Although older, these devices are capable computers, typically featuring one or more cameras, a touchscreen, and wifi connectivity. This is the ideal platform for rapidly deployable, image-based sensing. Users simply download our Zensors app onto the devices, which allows them to create or modify sensors. Users then "stick" the device in a context of their choosing.

WiFi Cameras. Zensors can also utilize stand-alone wifi-enabled cameras, costing as little as $30 today. In this case, a web interface can be used to define sensors (Figure 3).

Privacy Preservation

Image Subregions. Contextual information from cameras creates an inherent tradeoff between information and privacy [4,5,14]. A naïve approach would utilize the entire raw image. However, this can easily violate privacy, especially when personally identifying information is present, or when images depict people in sensitive situations. To partially mitigate this issue, our system asks users to select an arbi-

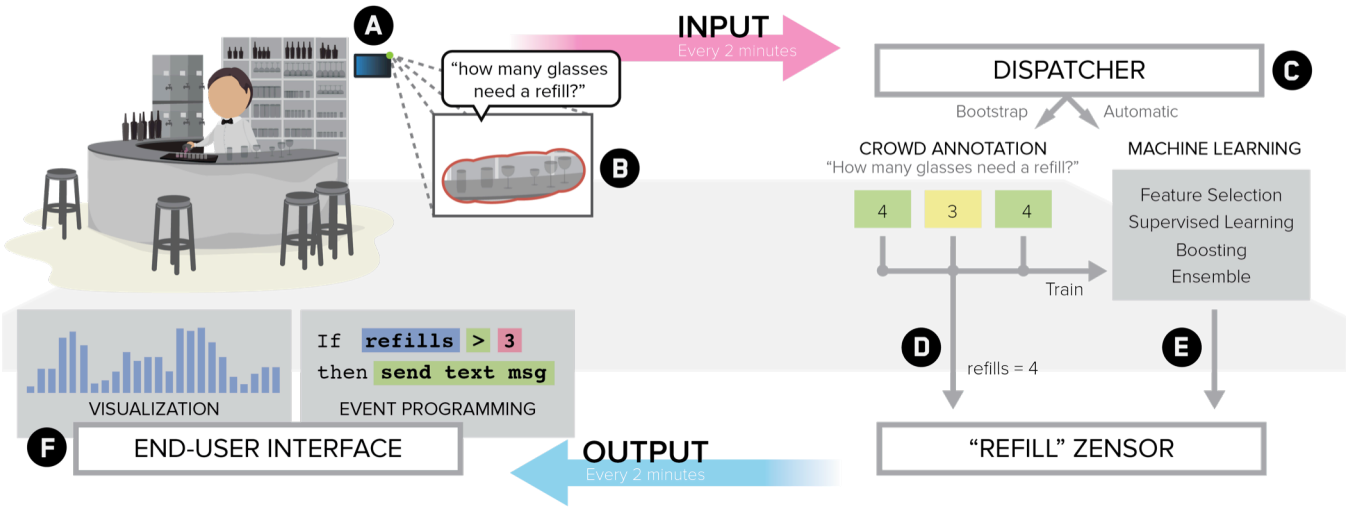

Figure 1. Zensors architecture. A bartender repurposes a tablet as a sensor host, affixing it to the wall behind the bar (A). Using the live view from the front facing camera, he selects a region of the scene and asks, "how many glasses need a refill?" (B). Periodically, the device takes snapshots, and forwards this data to a dispatcher (C). Initially, the dispatcher uses crowd workers to power the sensor, providing immediate human-level accuracy (D). In the background, answers from the crowd train a computer-vision-based, machine learning classifier (E). As it approaches crowd-level accuracy, the system employs a hybrid crowd-ML scheme to power the sensor stream. Sensor output can drive end-user applications, such as a real time visualizer (F, left) or event-based end-user programmable system (F, right).

trarily shaped subregion relating to the question they wish to ask; image data outside the subregion is masked away. This approach helps users to strike a balance between privacy and information content, and as a positive side effect, it reduces file size, removes unnecessary image elements, and simplifies the sensing effort for both human raters and computer vision techniques.

Image Obfuscation. For users wishing to add an additional level of privacy, we offer several image obfuscation techniques. These are applied to images on the device before they are sent to the cloud. Image obfuscation and privacy has been previously researched, and therefore we integrated the guidelines suggested by Hudson [14] and Boyle [4,5]. Users can choose to leave the image subregion unaltered, or select from four obfuscation methods: light blur, heavy blur, median filter, and edge filter (Figure 2).

Creating New Sensors

Sensor Questions. Every sensor starts with a question. Users create a new sensor by selecting a region of the image, and entering a plain text question. For example, in Figure 1, the bartender highlights the bar area, and asks, "how many glasses need a refill?" Questions define the "capabilities" of a sensor, and thus, the quality of answers depends on several factors, such as the question's context and relevance.

Data Types. To further add context and relevance, our system requires users to define a data type when creating new sensors. Data types curb variance and constrain the range of possible values returned by a sensor (i.e., the answer to the sensor's question), and facilitate simple automated processing of the data stream. To this end, we categorize questions into four example data types:

`YesNo` – This type of question can be answered by either *yes* or *no*. It is analogous to an ON/OFF sensor mechanism. Examples include: "is the door open or closed?", "is there food in the kitchen?", or "can you see a bus in this image?"

`Number` – Number data types are intended for questions that require counting. Numbers are continuous and are bound between a minimum and maximum range. Examples include: "how many cars do you see in the parking lot? (*min*=0, *max*=30)", and "what percentage of the water tank is full? (*min*=0, *max*=100)"

`Scale` – Scale data types are analogous to Likert-scale questions. Answers belong to discrete values specified within an increasing/decreasing scale. For this data type, users are required to supply scale-value pairs. Examples include: "how messy is this table? (1= Clean, 2=Average, 3= Messy)", or "how happy does the person look? (1=Very Sad, 2=Sad, 3=Neutral, 4=Happy, 5=Very Happy)"

`MultipleChoice` – When creating multiple-choice questions, users are required to specify the list of choices. Unlike scale data types, choice order is irrelevant. Examples include: "what type of food do you see? (None, Indian, Thai, Asian, Salad, Bagels, Other)" and "what are people doing? (reading, using computers, eating, other)".

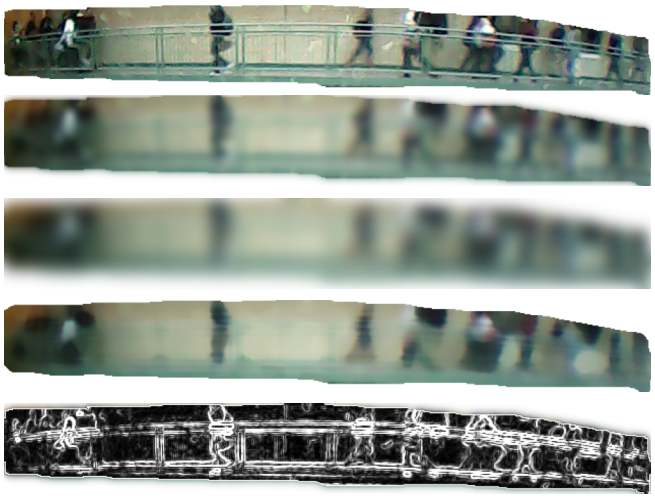

Figure 2. In addition to subregion masking, users can select image obfuscation methods if desired. From top to bottom: raw image, light blurring, heavy blurring, median filter, and edge masking.

Frequency. When creating sensors, users need to specify the frequency at which sensor readings are taken. Depending on the question, frequency readings can range from near real-time (e.g., every one or two seconds for questions like "is the refrigerator door open? [`YesNo`]"), to extended periods (e.g., once per day for: "what product is advertised on the billboard? [`MultipleChoice`])."

Web Interface. Along with the mobile application, we built a companion web interface for sensor management (Figure 3). Users link one or more sensors to a web account, where they can create, modify, and synchronize sensors across all of their devices. The web UI also makes it possible to create new sensors remotely. For example, users can "stick" a sensor device at an elevated vantage point (e.g., for viewing an entire parking lot), and then manage and create sensors without having to physically touch the device.

Similar Image Detection and Rejection

Sensor image streams often have periods of little variation (e.g., buildings after closing hours, outdoor scenes at night). Thus, to avoid soliciting redundant (and thus costly) responses from the crowd on highly similar images, we collapse runs of similar images. We calculate image similarity using a simple technique. First, we count the number of pixels that have changed from the previous frame using their RGB values and a predetermined *pixel difference threshold*. If the number of changed pixels in an image exceeds a certain *image area percentage threshold*, we consider the image to be different. Although this algorithm worked well for our purposes, we note that more sophisticated approaches (see e.g., [16]) could be used.

To determine optimal parameters, we performed a brute force optimization experiment. We compiled a corpus of roughly 6000 time-stamped images taken from multiple pilot sensor streams. We then manually labeled whether or not an image was the same as the previous image, providing

a ground truth set. We then ran our image similarity algorithm, seeded with all combinations of the following thresholds: 2% to 40% *pixel difference threshold*, in 2% increments, and 0.1% to 5.0% *image area percentage threshold*, in 0.1% increments. This produced 130 result sets, which we compare to our ground truth using Jaccard's distance metric. By using a *pixel difference* and *image area* threshold of 10% and 1.0% respectively, a Jaccard distance of .64 is achieved. On average, this process removes roughly 40% of images – a significant saving.

Sensing with the Crowd

Images streamed from sensor devices are stored in a database. Crowd workers process un-answered instances through a web-based interface seen in Figure 4. The interface varies slightly based on the question/response type. Each sensor instance is answered (i.e., labeled) by several different crowd workers; we use voting to determine the best response (at present, we use three workers, but other numbers are possible). The resulting value is then saved in the database and the instance is considered answered and ready for sharing with end users or powering applications.

Our goal is to ensure workers are presented with a simple task that they can answer quickly. As such, we present one image to each worker and collect a single response. If workers are unable to answer an image-based question, they can mark it as an exception ("I can't tell" button, Figure 4), which informs the system that there is something amiss with the sensor itself (e.g., occlusion, insufficient lighting, poorly defined question). In addition, workers are prompted to provide textual descriptions when exceptions occur. This approach provides actionable user feedback to help remedy the problem.

To recruit users fast enough to receive answers in real time, we use LegionTools [17], a toolkit for quickly recruiting workers from Mechanical Turk using a web-based interface.

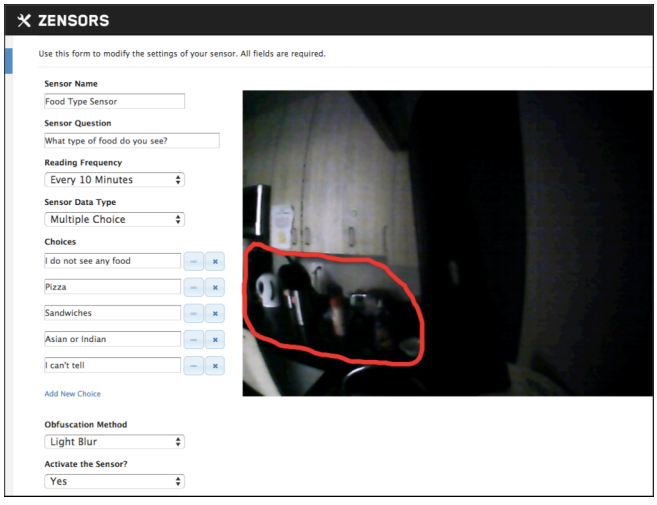

Figure 3. Users create and synchronize Zensors across all of their devices using a web API, allowing them to manage sensors without physical device interaction.

It leverages a retainer model [2], which pre-recruits workers so they are ready to respond to a task within as little as two seconds. When sensors are crowd-powered, this sets the lower-bound on our system latency.

Figure 4. Our Mechanical Turk interface, which lets workers answer the question or raise an exception.

Training Automated Sensors

Solutions using crowd-power alone can be costly and difficult to scale (i.e., more sensors requires more people). Zensors reduces its reliance on crowd workers over time by using crowd-provided answers to train machine learning classifiers, which are fast and inexpensive. However, even after machine learning has taken over processing sensors feeds, crowd workers are still needed to provide a periodic human-accuracy baseline to ensure high accuracy. Here, we describe the machine learning framework for our prototype deployment.

The classifier is trained on all the input data to date, except for the most recent full day of data, which is set aside for evaluation. For sensors that produce data infrequently, the test set can be extended to one week or more to ensure there are sufficient test instances.

Histogram equalization is applied to each image to reduce the effect of lighting variances. Then, each input image is processed into a large number of global features. Additionally, each sensor image (which is itself a subregion of the larger original image) is broken into a grid of sub-images. In addition to a 1x1 grid (simply the image unchanged), we also use 4x4, 7x7 and 10x10. Each of these variously-sized sub-images is then converted to a luminance image, and the mean, standard deviation and mean-squared error across the window are used as numerical features. This produces a total of 332 image features. Feature selection is used as a post-process to extract exemplar features for a given sensor feed. Of course, much more advanced computer vision and scene understanding approaches exist that are beyond the scope of this work.

We use correlation-based feature selection [12] to select features from the training set, coupled with a backtracking best-first attribute searcher. Both algorithms are implemented in the Weka [11] machine learning toolkit. This selection process typically chooses between 10 and 30 features. Of note, the feature sets for different sensors rarely overlap.

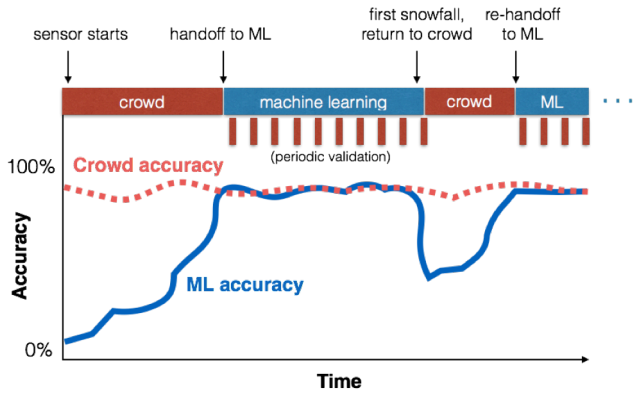

Figure 5. Sensors can toggle between the crowd and machine learning to adapt to environment changes. Note that end users and applications only ever see the max of the crowd and ML accuracies.

We then train classifiers depending on the type of sensor. A "pre-classifier" is first trained to distinguish between exceptions and non-exceptions, to ensure that the main classifier is not polluted by incorrect data. For continuous (numeric or scale) sensors, we train a support vector machine regression classifier using the SMOReg algorithm. For discrete sensors (yes/no, multiple choice), we use a one-versus-one multiclass quadratic-kernel SVM trained with the SMO algorithm, and for simple binary sensors we train a single SVM. The SVM was chosen as the basic classifier because of its ease of training and predictable behavior, though other classification approaches are certainly possible and valid. A more robust version of this system would maintain a library of feature extractors and classification algorithms, selecting those exhibiting the best performance for a given sensor.

Machine Learning Handoff

As the training corpus grows from crowd labeled instances, the accuracy of the machine learning classifiers typically improves. Once the accuracy of the machine learning exceeds a predefined threshold (e.g., 95%) for several days, the sensor hands off classification tasks to the machine learning algorithm. It is also possible to do a soft handoff, where the relative weighting between crowd and machine learning labels shifts over time.

Periodic Ground Truth Validation and Adaptation

To ensure continued sensor accuracy after the handoff to machine learning, the system periodically submits small batches of fresh sensor data to the crowd for labeling. This is used to benchmark the classifier accuracy. If accuracy is sufficiently robust, the machine learning can continue to power the sensor. However, if the accuracy has fallen below a threshold, the system can revert to crowd-power. This serves two immediate purposes: 1) the sensor immediately regains human-intelligence level accuracy, and 2) the system can incorporate new labels in its training for a hopeful future handoff.

In this way, Zensors can automatically handle infrequent changes (such as the first snow fall in a parking lot; Figure 5) that would prove challenging for most computer-vision-driven systems (which are first trained and then deployed). This ability to seamlessly toggle between crowd and automatic approaches, without sensor interruption, makes our approach highly adaptive and robust.

End-User Programming

We built a basic end-user programming tool that lets users design event-based notifications using data from one or more sensors (e.g., "send an email when the stove is ON *and* ZERO people are in the house"). These directives can be chained together as sets of conjunctions ("and" clauses), and "or" clauses for alternative responses. Multiple chains can be defined to represent different configurations. These disjunctions of conjunctions comprise a fully expressive set of logical terms.

This interface works similar to the popular *If This Than That* tool (ifttt.com) – users can select a sensor, select a value and comparison operator, and then select an action type from the set of supported APIs. Our current implementation allows users to select from an email, text message, audio, or chat alert. For each alert type, users are able to define a custom message, which can also display the current value of the sensor by using a specially marked variable.

Implementation and API

As a proof of concept, our Zensors mobile application was developed on Android, and deployed on a fleet of DOPO 9" Internet tablet M975, each costing less than $50. Our web UI, backend server, and end-user programming interfaces were implemented using PHP, MySQL, jQuery, d3.js, node.js, and socket.io. For our machine-learning component, we used the Weka toolkit [11]. We also provide a web API that allows services and end-user applications to leverage sensor data. As a proof of concept, we built two applications using this facility: our end user programming tool, described in the previous section, and a basic data visualization tool – both can be seen in our Video Figure.

PROTOTYPE DEPLOYMENT

The goal of our prototype deployment was to illustrate that even with a basic approach, the Zensors architecture can achieve high accuracy, at low cost, quickly, and author-able by end users using plain text queries..

We deployed 16 sensors across four diverse environments: a home kitchen, office kitchenette, building food court, and parking lot (see Figure 6 for four examples). Sensor questions ranged from "is this café door open?" to "what type of food is on the counter?" A range of reporting frequencies (from once per minute to twice per hour) and deployment durations (10 days to 3 weeks) were represented by our sample sensor set. We also manually labeled images from seven sensors to create a ground-truth dataset for later experiments. These "expert" labels have the advantage of superior context understanding as well as being able to view the entire dataset, not just a small snapshot.

Accuracy of the Crowd

To analyze how well our sensors can quickly provide accurate sensing data, we measured the precision, recall, and latency of our aggregated crowd responses. Figure 7 shows the accuracy of crowd workers' ratings, using expert labels as the ground truth. Cohen's kappa [6] is calculated to mitigate the effects of skewed class distribution (e.g., the leftover food sensor returned "no" over 75% of the time). Crowd accuracy reaches as high as 96.8% (kappa score 0.859), with mean accuracy 77.4% (median 76.0%). The crowd performed very well on three sensors (accessible parking spots occupied, number of cars in parking lot, and dishwasher door), moderately well on one sensor (leftover food), and poorly on three sensors (food type, line length sensor, countertop messy).

The food type sensor required users to distinguish between seven types of cuisine ("I do not see any food", "Pizza", "Sandwiches", "Cake or pastries", "Asian or Indian", "Salad", "Bagels or doughnuts", "Other cuisine or I can't tell") based on a very low-resolution image, while the line length sensor and countertop sensors both involved subjective judgments (e.g. "is the line orderly", "how messy is the countertop"). By contrast, quantitative questions ("is there food here", "is the door closed", "how many cars are there") generally had superior performance.

In designing questions to be posed to the crowd, operators may make assumptions that are not obvious to crowd workers, leading to incorrect results. In one example, workers were asked to identify the presence of food on a kitchen countertop. The countertop has a permanent candy jar, which the experimenters assumed would not be classified as food, yet several crowd workers marked the otherwise-empty countertop as having food. Based on the observed results, the question was amended to explicitly exclude candy, after which the expected results were obtained.

Estimating Live Accuracy

This experiment sought to estimate the accuracy of a sensor, over the course of its deployment, as viewed "live" by its operator. For each sensor, we defined ten time periods each covering one-tenth of the data, numbered t=0.1 through t=1.0. To estimate live accuracy at time t, we trained on all data up to time t, and then tested on all data from time t to time $t+0.3$ (i.e. we tested on a sliding window of 30% of the data). The results for three representative sensors are shown in Figure 7, compared against the crowd accuracies. In many cases, a relatively small portion of the data is needed to reach crowd-level accuracies.

Assessing Future Accuracy Post ML Handoff

Alternatively, it is equally important to assess what the accuracy of a sensor would be going forward, assuming a ML handoff occurs at time t. To assess this, we simulate a complete ML handoff at each time increment. All data up to that point is used for training, while all future data is used for testing. We stop this analysis when less than 30% of the data is available for testing, to avoid noisy results from insufficient test data. These results are summarized in Figure 8, compared against the overall crowd accuracies. The accuracies follow similar curves to the curves of Figure 7, suggesting that live classification accuracy may be able to predict future post-handoff classification accuracy.

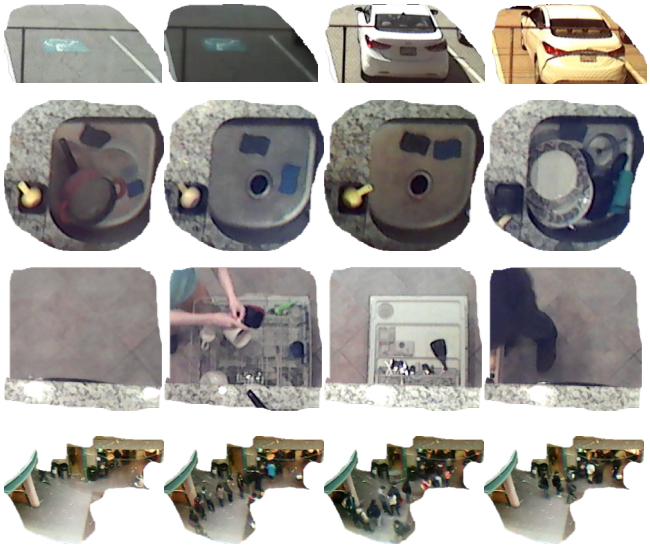

Figure 6. Sensor image time series. Associated questions top to bottom "do you see a parked car?", "how many dishes in the sink?", "do you see a dishwasher door?", and "how orderly is the line?"

Similar Image Rejection in Practice

As described previously, our system sends labeling requests to the crowd only when there is a sufficient visual change between consecutive images, otherwise the last sensor value is simply copied forward. In our deployment, we found that our image similarly mechanism rejected an average of 61.2% of images (SD=17.2%, minimum=40.5%, maximum=93.7%).

Sensors that Fail

It is important to acknowledge that some of our sensors failed to produce reliable output. We initially hypothesized that failure would primarily be due to shortcomings in our computer vision implementation. However, we found that our classifiers work well in practice, with six of our seven sensors (for which we had expert labels, and thus a ground truth) getting to within 90% of crowd accuracy when we trained on half of the crowd-labeled data (mean 98.1%, SD=14.4%; tested on the second half). Instead, we found that the classification bottleneck for several of the sensors was caused by the poor accuracy of the crowd answers (as compared against our ground truth). For these underperforming sensors, we found a common theme: the sensor questions were subjective or required additional context.

For example, one sensor asked, "how orderly is the line?" (Figure 6) with three possible answers: "no people visible in image", "people present, but no obvious organization", and "people standing in a line". Because this is subjective (e.g., relative to local cultural norms) we found that crowd workers provided widely varying answers. Another sensor was

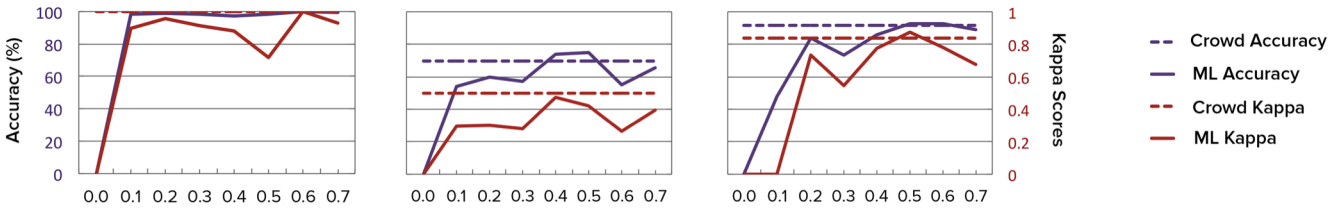

Figure 7. "Live" accuracy evaluation of three sensors. This is the accuracy of a sensor over the course of its deployment, as viewed "live" by its operator. X-axes represent training set cutoff time *t*. Left: "do you see a dishwasher door." Middle: "how messy is the counter." Right: "which parking spots are occupied."

tasked with sensing whether a dishwasher was opened or closed (see Figure 6 for some example images). In piloting, the question was defined as "is the dishwasher door open?" However, this appeared to confuse crowd workers, reducing sensor accuracy. We hypothesize that this problem was caused by the fact that most of the time, no dishwasher was visible in the closed state. When presented with the question of "is the dishwasher door open?", the crowd presumably wondered "what dishwasher?". We found that rephrasing the question to be more context-free – "Do you see a dishwasher door?" – significantly boosted accuracy.

There are a number of ways to alleviate "bad" sensor questions. One approach is to suggest example questions or provide structured question templates (e.g., "do you see a _____ [in/on] the _____?"), helping end-users formulate questions with less ambiguity. Additionally, the "I can't tell" button in the crowd interface (see "sensing with the crowd" section) could allow the system to flag sensors causing confusion and suggest the question or image subregion be modified. Another approach is for the crowd labeling interface to provide exemplar images, one for each possible answer (e.g., show exemplars of both dishwasher door states). Finally, we can also show crowd workers a random set of previously collected sensor images that hopefully better capture the full range of possible states (e.g., orderly line to chaotic), so they can make better relative judgments.

Zensors Economics

The HITs we use for Zensors paid 2 cents each, a pay rate chosen to be above the U.S. minimum wage even for slower workers. This means, e.g., that a sensor that takes images every 10 minutes would cost roughly $100 per month (assuming an average similar image rejection rate) to be fully human-powered (30 days * 24 hours * 6 images per hour * 40% different images * $0.02 * 3 workers). To offer a concrete example, our median sensor in terms of images captured was the dishwasher door sensor, which triggered every minute. It captured 528 non-similar images over a 7-day deployment, which translates to $135/month in costs assuming only human-power.

For many of our sensors, we found that our automatic classification pipeline could reasonably approximate the crowd's answers using the first week as training data. Once there is sufficient agreement between crowd and machine learning, we can decrease the number of human workers from three to two, and eventually to one (and recruit more workers when there is significant disagreement). This means we can reduce the price by 67% even before the machine learning is fully trained, without reducing accuracy. To get to a point where machine learning can shoulder the entire load, we found that our test sensors took between 90 and 687 data points (depending on polling rate and setting). This means that we can train an automated sensor for as little as $5.40 (and our worst sensor for $41).

One of the strengths of Zensors is its ability to use human-intelligence to handle previously unseen scenarios. As such, even if a handoff to ML is possible, there is a continued cost to validate that the automated system is still working By periodically having the crowd spot-check the output of sensors, we can detect e.g., errors and scene changes, switching back to human-power (and thus training) if needed. This periodic validation can run at different intensities. Validating e.g., 1 in every 50 sensor instances would cost roughly $1/50^{th}$ the typical human-powered cost, which is perhaps a few dollars per month.

When ML Handoff is Not Possible

Finally, there may be cases where the system cannot attain a full ML handoff (e.g., poor image resolution, noisy training data, or simply a hard question incompatible with CV ap-

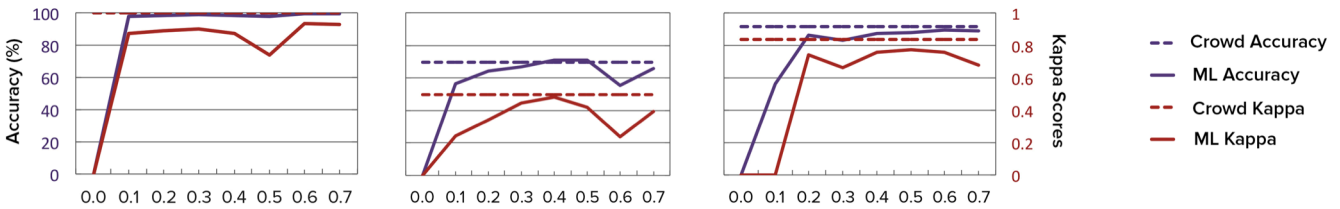

Figure 8. "Future" accuracy evaluation of three sensors. This is the accuracy of the sensor assuming ML handoff at time *t*. X-axes represent ML handoff time *t*. Left: "do you see a dishwasher door." Middle: "how messy is the counter." Right: "which parking spots are occupied."

Sensor Name / Question	Freq.	CP Cost per Mo.	Exp. Cost per Mo.
Do you see a dishwasher door?	1 min	$135	$35
How messy is the counter?	10 Min	$82	$22
Do you see a parked car?	30 Min	$30	$8
How many dishes in the sink?	10 Min	$28	$7
How many cars do you see?	30 Min	$43	$12
What type of food do you see?	10 Min	$87	$23

Table 1. Estimated monthly costs if sensors were fully crowd-powered (CP cost), as well as expected cost (Exp. Cost) assuming ML handoff after week one and continued periodic validation.

proaches). As a result, the system will need to rely on a fully crowd-powered approach for an indefinite period, which can be relatively expensive; Table 1 offers some example costs from our deployment. However, even if ML handoff never occurs, it is important to note that end users and applications only ever see human-accuracy level answers.

CONCLUSION

Zensors enables easy end-user creation of arbitrary sensors for any visually observable property. Zensors uses crowd-powered answers to produce near-instant sensor readings with high accuracy while requiring no explicit training. Once enough data labels have been collected, Zensors can seamlessly hand-off image classification to machine learning utilizing computer-vision-derived features. Our formative design exercises highlight the potential that Zensors holds for enabling a broad variety of applications beyond those available using contemporary electro-mechanical sensors. We conclude with results from our prototype deployment, which suggest our approach is feasible, accurate and can be cost effective.

ACKNOWLEDGEMENTS

This work was supported by Yahoo! InMind, National Science Foundation award #IIS-1149709, NSERC of Canada, an Alfred P. Sloan Foundation Fellowship, and fellowships from Microsoft Research, Disney Research, and Qualcomm.

REFERENCES

1. Abowd, G. D. and Mynatt, E. D. Designing for the human experience in smart environments. *Smart environments: technologies, protocols, and applications* (2004).
2. Bernstein, M.S., Brandt, J., Miller, R.C., Karger, D.R. Crowds in two seconds: enabling realtime crowd-powered interfaces. In *Proc. UIST '11*.
3. Bigham, J. P., et al. VizWiz: nearly real-time answers to visual questions. In *Proc. UIST '10*.
4. Boyle, M. and Greenberg, S. The language of privacy: Learning from video media space analysis and design. *ACM ToCHI*, 2005.
5. Boyle, M., Edwards, C., Greenberg, S. The effects of filtered video on awareness and privacy. In *Proc. CSCW '00*.
6. Cohen, J. A coefficient of agreement for nominal scales. Educational and Psychological Measurement 20(1), 1960.
7. Deng, J., Krause, J., Fei-Fei, L. Fine-grained crowdsourcing for fine-grained recognition. In *Proc. CVPR '13*.
8. Fails, J.A. and Olsen, D. A design tool for camera-based interaction. In *Proc. CHI '03*.
9. Fails, J.A. and Olsen, D. Light widgets: interacting in every-day spaces. In *Proc. IUI '02*.
10. Franklin, M. J., Kossmann, D., Kraska, T., Ramesh, S. and Xin, R. CrowdDB: answering queries with crowdsourcing. In *Proc. SIGMOD '11*.
11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I. H. The WEKA data mining software: an update. ACM SIGKDD explorations, 11(1) '09.
12. Hall, M.A., Correlation-based Feature Subset Selection for Machine Learning. 1998.
13. Hara, K., Sun, J., Moore, R., Jacobs, D., Froehlich, J.E. Tohme: Detecting Curb Ramps in Google Street View Using Crowdsourcing, Computer Vision, and Machine Learning. In *Proc. UIST '14*.
14. Hudson, S.E. and Smith, I. Techniques for addressing fundamental privacy and disruption tradeoffs in awareness support systems. In *Proc. CSCW '96*.
15. Kastrinaki, V., Zervakis, M., Kalaitzakis, K. A survey of video processing techniques for traffic applications. *Image and Vision Computing* 21(4), 2003.
16. Khan, M., Acharya, B. and Verm, S. Comparison between different illumination independent change detection techniques. In *Proc. ICCCS '11*.
17. Lasecki, W., Gordon, M., Koutra, D., Jung, M., Dow, S., Bigham, J. Glance: Rapidly Coding Behavioral Video with the Crowd. In *Proc. UIST '14*.
18. Lasecki, W.S., Song, Y., Kautz, H., and Bigham, J.P. Real-time crowd labeling for deployable activity recognition. In *Proc. CSCW '13*.
19. Lee, M. L., Dey, A. K. Sensor-based observations of daily living for aging in place. *Pers. Ubiquit. Comp.*, 2014, 1-17.
20. Marcus, A., Karger, D., Madden, S., Miller, R., Oh, S. Counting with the crowd. In *Proc. VLDB '12*.
21. Maynes-Aminzade, D., Winograd, T., Igarashi, T. Eyepatch: prototyping camera-based interaction through examples. In *Proc. UIST '07*.
22. Salber, D., Dey, A. K., Abowd, G. D. The context toolkit: aiding the development of context-enabled applications. In *Proc. CHI '99*.
23. Tang, A., Greenberg, S., and Fels, S. Exploring video streams using slit-tear visualizations. *Proc. AVI '08*.
24. von Ahn, L. and Dabbish, L.. Labeling images with a computer game. In *Proc. CHI '04*.
25. Vondrick, C., Patterson, D., and Ramanan, D. Efficiently Scaling Up Crowdsourced Video Annotation. *International Journal of Computer Vision*.
26. Wah, C. Crowdsourcing and its applications in computer vision. University of California, San Diego. 2006.
27. Walton, A. Life Expectancy of a Smartphone. Retrieved from chron.com, September 21, 2014.
28. Zhao, W., Chellappa, R., Phillips, P. and Rosenfeld, R. Face recognition: A literature survey. *ACM Comput. Surv.* 35(4), 2003.
29. Zimmer, T., & Beigl, M. AwareOffice: Integrating Modular Context-Aware Applications. In *Proc. ICDCSW'06.*