Evorus: A Crowd-powered Conversational Assistant Built to Automate Itself Over Time

Ting-Hao (Kenneth) Huang, Joseph Chee Chang, and Jeffrey P. Bigham
Language Technologies Institute and Human-Computer Interaction Institute
Carnegie Mellon University
{tinghaoh, josephcc, jbigham}@cs.cmu.edu

ABSTRACT

Crowd-powered conversational assistants have been shown to be more robust than automated systems, but do so at the cost of higher response latency and monetary costs. A promising direction is to combine the two approaches for high quality, low latency, and low cost solutions. In this paper, we introduce Evorus, a crowd-powered conversational assistant built to automate itself over time by *(i)* allowing new chatbots to be easily integrated to automate more scenarios, *(ii)* reusing prior crowd answers, and *(iii)* learning to automatically approve response candidates. Our 5-month-long deployment with 80 participants and 281 conversations shows that Evorus can automate itself without compromising conversation quality. Crowd-AI architectures have long been proposed as a way to reduce cost and latency for crowd-powered systems; Evorus demonstrates how automation can be introduced successfully in a deployed system. Its architecture allows future researchers to make further innovation on the underlying automated components in the context of a deployed open domain dialog system.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

Author Keywords

crowd-powered system; crowdsourcing; real-time crowdsourcing; conversational assistant; chatbot

INTRODUCTION

Conversational assistants, such as Apple's Siri, Amazon's Echo, and Microsoft's Cortana, are becoming increasingly popular, but are currently limited to specific speech commands that have been coded for pre-determined domains. As a result, substantial effort has been placed on teaching people how to talk to these assistants, *e.g.*, via books to teach Siri's language [36], and frequent emails from Amazon advertising Alexa's new skills [1]. To address the problem of users not knowing what scenarios are supported, AI2 recently built an Alexa skill designed to help people find skills they could use, only to have it rejected by Amazon [8].

CHI 2018, April 21–26, 2018, Montreal, QC, Canada

ACM 978-1-4503-5620-6/18/04...$15.00
https://doi.org/10.1145/3173574.3173869

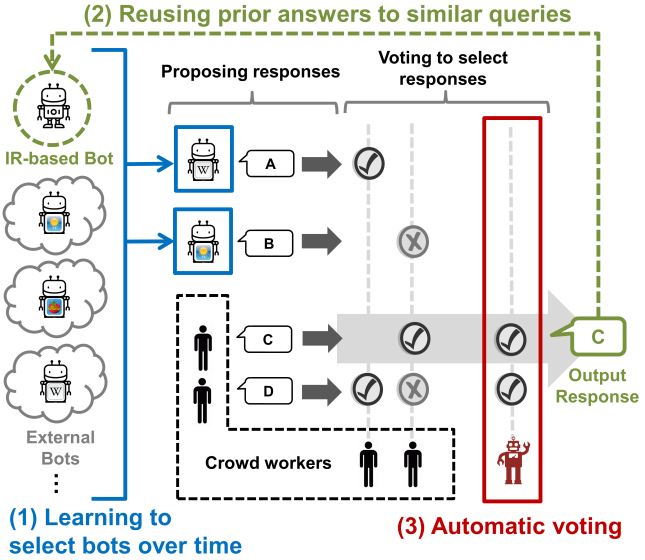


Figure 1. Evorus is a crowd-powered conversational assistant that automates itself over time by *(i)* learning to include responses from chatterbots and task-oriented dialog systems over time, *(ii)* reusing past responses, and *(iii)* gradually reducing the crowd's role in choosing high-quality responses by partially automating voting.

Crowd-powered assistants are more robust to diverse domains, and are able to engage users in rich, multi-turn conversation. Some systems use professional employees, such as Facebook M [15], while others use crowd workers, such as Chorus [25]. Despite their advantages, crowd-powered agents remain largely impractical for deployment at large scale because of their monetary cost and response latency [3, 19]. On the other hand, crowd-powered systems are often touted as a path to fully automated systems, but transitioning from the crowd to automation has been limited in practice. The most straightforward approach is to use data from prior conversations to train an automated replacement. This can work in specific domains [46], or on so-called "chit-chat" systems [2]. Fully automating a general conversational assistant this way can be difficult because of the wide range of domains to cover and the large amount of data needed within each to train automated replacements. Such automated systems only become useful once they can completely take over from the crowd-powered system. Such abrupt transition points mean substantial upfront costs must be paid for collecting training examples before any automation can be tested in an online system.

In this paper, we explore an alternative approach of a crowd-powered system architecture that supports gradual automation

over time. In our approach, the crowd works with automated components as they continue to improve, and the architecture provides narrowly scoped points where automation can be introduced successfully. For instance, instead of waiting until an automated dialog system is able to respond completely on its own, one component that we developed recommends responders from a large set of possible responders that might be relevant based on the on-going conversation. Those responses are then among the options available to the crowd to choose. Another component learns to help select high-quality responses. Each problem is tightly scoped, and thus potentially easier for machine learning algorithms to automate.

This paper introduces Evorus, a crowd-powered conversational agent that provides a well-scoped path from crowd-powered robustness to automated speed and frugality. Users can converse with Evorus in open domains, and the responses are chosen from suggestions offered by crowd workers and *any number* of automated systems that have been added to Evorus. Evorus supports increased automation over time in three ways (Figure 1): *(i)* allowing third-party developers to easily integrate automated chatterbots or task-oriented dialog systems to propose response candidates, *(ii)* reusing crowd-generated responses from previous conversations as response candidates, and *(iii)* learning to automatically select high-quality response candidates to reduce crowd oversight over time.

In Evorus, existing dialog systems can be incorporated via simple REST (REpresentational State Transfer) interfaces that take in the current conversation context, and respond with a response candidate. Over time, Evorus learns to select a subset of the automated components that are most likely to generate high-quality responses for different context. The responses are then forwarded to crowd workers as candidates. Workers then choose which of the responses to present to the users. Evorus sees workers selecting responses from candidates as signals that enable it to learn to select both automated components and response candidates in the future. It is important to note that while Evorus is a functioning and deployed system, we do not see the current version and its constituent components to be final. Rather, its architecture is designed to allow future researchers to improve on its performance and the extent to which it is automated, by working on constituent problems, which are each challenging in their own right. The structure of Evorus provides distinct learning points that can be bettered by other researchers. Others may include additional dialog systems or chatterbots, and improve upon its learning components, driven by the collected data and its modular architecture.

We deployed the current version of Evorus over time to better understand how well it works. During our deployment, automated response were chose 12.44% of time, Evorus reduced the crowd voting by 13.81%, and the cost of each non-user message reduced by 32.76%. In this paper, we explore when the system was best able to automate itself, and present clear opportunities for future research to improve on these areas.

This paper makes four primary contributions:

- **Evorus Architecture:** a crowd-powered conversational assistant that is designed to gradually automate itself over time by including more responses from existent chatbots and reduce the oversight needed from the crowd;
- **Learning to Choose Chatbots Over Time:** we introduced a learning framework that uses crowd votes and prior accepted message to estimate the likelihood of each chatbots when receiving a user message;
- **Automatic Voting:** we implemented a machine learning model for automatically reducing the amount of crowd oversight needed, evaluated its performance on a dataset of real conversations, and developed a mathematical framework to estimate the expected reward of using the model; and
- **Deployment:** we deployed Evorus for over 5 months with 80 participants and 281 conversations to understand how the automatic components we developed could gradually take over from the crowd in a real setting.

RELATED WORK

Our work draws from research on conversational agents, general purpose dialog systems and crowd-machine systems.

General Purpose Dialog System: A number of general purpose dialog systems such as IRIS [2] have been proposed. Wen et al. [43] designed a neural network language generation model for multi-domain dialog systems. A deep-learning-based domain adaptation model was also proposed recently [11]. Project DialPort [50] introduced a multi-agent framework that has the capability to include multiple task-oriented dialog systems to hold a multiple domain conversation. On the other hand, in the field of natural language processing, general response generation technologies were also developed. Ritter *et al.* [35] generated responses based on phrase-based statistical machine translation based on Twitter data. Li *et al.* [26] introduced a response generator based on speaker model that encodes personas with background information and speaking style. Recently, researchers started exploring end-to-end joint learning of language understanding in dialogue systems [6, 48, 27]. However, after decades of developments, sophisticated artificial "conversational intelligence" are largely absent in modern digital products.

Crowd-powered Conversational Agents: Building fully-automated, open-domain conversational assistants is a widely researched topic in the field of artificial intelligence (AI), but has thus far remained an open challenge. In response to this, the Chorus [25] conversational agent is powered by a crowd of human actors, which enables it to work robustly across domains [24]. To help users manage information and services through crowd-powered conversational agents, Guardian takes as input a Web API and a desired task from the user and the crowd determines the parameters necessary to complete the task [20, 18]; IntructableCrowd helps users automate the management of sensors and tasks in their mobile phones through a conversational agent [16]; and WearMail enables users to access their emails by talking to the crowd-powered assistant via smartwatch [39]. Conversational assistants powered by trained human operators, such as Facebook M [15] and Magic Assistant [29], have emerged in the recent years.

While most crowd-powered conversational systems function well in laboratory settings, Chorus was deployed in the real

world [19] and revealed a range of problems such as determining when to terminate a conversation or protecting workers from abusive content introduced by end users. Microsoft Tey [45] introduced an AI-powered agent which encountered problems when deployed publicly, because some users realized that they could influence what Tey would say because it mimicked them. Unlike Tey, Evorus does not learn only by mimicking users, and paid crowd workers are kept in the loop to verify responses in order to maintain quality. Our deployment did not reveal such problems and we believe the structure of Evorus makes such problems less likely.

Crowd-Machine Hybrid Systems: Crowd and machine hybrid systems have enabled us to solve a wide range of tasks that were difficult for machines or humans to solve alone, making impacts in areas including crowdsourcing, machine learning, databases and computer vision [10, 21, 37, 34, 22]. For instance, Flock and Alloy [7, 5] use crowds to suggest predictive features, label data, and weigh these features with machine learning techniques to produce coherent categories and accurate models. The Knowledge Accelerator [12] uses crowds to synthesize such crowd-machine structures into coherent articles. Zensors creates custom computer vision sensors bootstrapped by the crowd [23]. Similarly, CrowdDB [10] uses human input for providing information that is missing from the database, for performing computationally difficult functions, and for matching, ranking, or aggregating results based on fuzzy criteria. JellyBean [37] introduces a suite of crowd-vision hybrid counting algorithms that can perform in independent or hybrid modes returning more accurate counts that either workers or computer vision could do alone.

EVORUS' CONVERSATIONAL ASSISTANT FRAMEWORK

Evorus obtains multiple responses from multiple sources, including crowd workers and chatbots, and uses a voting mechanism to decide which responses to send to the end-user.

Worker Interface

Evorus' worker interface contains two major parts (Figure 2): the *chat box* in the middle and the *fact board* on the side. Chat box's layout is similar to an online chat room. Crowd workers can see the messages sent by the user and the responses candidates proposed by workers and bots. The role label on each message indicates it was sent by the user (blue label,) a worker (red label,) or a bot (green label.) Workers can click on the check mark (✔) to *upvote* on the good responses, click on the cross mark (✖) to *downvote* on the bad responses, or type text to propose their own responses. Beside the chat box, workers can use the fact board to keep track of important information of the current conversation. To provide context, chat logs and the recorded facts from previous conversations with the same user were also shown to workers.

The score board on the upper right corner displays the current reward points the worker have earned in this conversation. If the conversation is over, the worker can click the long button on the top of the interface to leave and submit this task.

Selecting Responses using Upvotes and Downvotes

Crowd workers and bots can upvote or downvote on a response candidate. As shown in Figure 2, on the interface, the upvoted responses turned to light green, and the downvoted responses turned to gray. Crowd workers automatically upvote their own candidates whenever they propose new responses. Upon calculating the voting results, we assigned a negative weight to a downvote while an upvote have a positive weights. We empirically set the upvote weight at 1 and downvote's weight at 0.5, which encourages the system to send more responses to the user. We inherited the already-working voting threshold from deployed Chorus [19], which accepts a response candidate when it accumulates a vote weight that is larger or equal to 0.4 times number of active workers in this conversation. Namely, Evorus accepts a response candidate and sends it to the user when Equation (1) holds:

$$\begin{aligned} &\#upvote \times W_{upvote} - \#donwvote \times W_{downvote} \\ &\geqslant \#active\_workers \times threshold \\ W_{upvote} = 1.0, \quad &W_{downvote} = 0.5, \quad threshold = 0.4 \end{aligned} \tag{1}$$

We formally defined the *#active_workers* in the later subsection of real-time recruiting. Evorus does not reject a message, so it does not have a threshold for negative vote weight.

Expiring Unselected Messages to Refresh Context

When Evorus accepts a response, the system turns the accepted message to a white background, and also *expires* all other response candidates that have not been accepted by removing them from the chat box in the worker interface. This feature ensures all response candidates displayed on the interface were proposed based on the latest context. We also created a "proposed chat history" box on the left side of worker interface, which automatically records the worker's latest five responses. Workers can copy his/her previously-proposed response and send it again if the message expired too fast.

A Proposed, Accepted, or Expired Message

In Evorus, non-user messages are in one of three states: [Proposed], [Accepted], or [Expired]. [Proposed] messages are open to be up/downvoted. These messages were proposed by either a worker or a bot, has not yet received sufficient votes to be accepted, and has not yet expired; [Accepted] messages received sufficient votes before they expired and were sent to the user; and [Expired] messages did not receive sufficient votes before they expired. These messages were not sent to the users, and were removed from the worker interface. A [Rejected] state does not exist since Evorus does not reject a message proactively.

Worker's Reward Point System

To incentivize workers, Evorus grants reward points to workers for their individual actions such as upvoting on a message or proposing a response candidate, and also for their collective decisions such as agreeing on accepting a message or proposing a message that were accepted. The score box on the right top corner of the interface shows the current reward points to the worker in real-time. Reward points are later converted to bonus pay for workers. Without compromising output quality, if some of these crowd actions can be successfully replaced by automated algorithms, the cost of each conversation can be reduced. Evorus' reward point schema was extended from the Chorus reward schema, which was previously used during

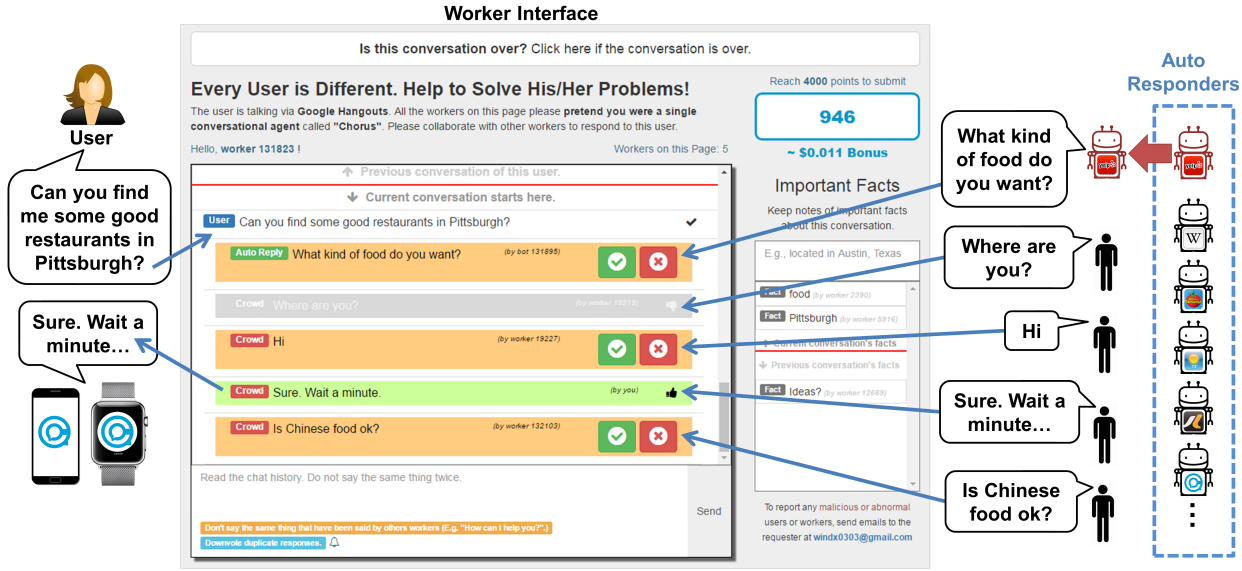


Figure 2. The Evorus worker interface allows workers to propose responses and up/down vote candidate responses. The up/down votes give Evorus labels to use to train its machine-learning system to automatically gauge the quality of responses. Evorus automatically expires response candidates upon acceptance of another in order to prevent workers from voting through candidate responses that are good but no longer relevant. Workers can tell each message is sent by the end-user (blue label), a worker (red label), or a chatbot (green label) by the colored labels.

its year-long deployment [19]. This schema encodes the importance of each action, and thus provides a good guide for algorithms to estimate the benefit and risk when automating a crowd action. Moreover, this reward schema will be used to estimate the expected reward points (and corresponding costs) an automatic voting bot can save, which we describe later.

Real-time Recruiting & Connecting to Google Hangouts

When a conversation starts, Evorus uses the Ignition model [17] to recruit workers quickly and economically from Amazon Mechanical Turk, and uses the Hangoutsbot [14] library to connect with the Google Hangout servers so that users can use its clients to talk with Evorus on computers or mobile devices. Each conversation starts with 1 worker and incorporates 5 workers at most. Workers may reach a conversation at different times, but typically stay to the end of the conversation (average duration $\simeq$ 10 minutes). The *#active_workers* in Equation 1 is defined as "the number of crowd workers who were working on this conversation when the message was proposed," which varies as workers arrive (or drop out) at different times. In our deployment, the average *#active_workers* of all crowd messages is 3.56 (SD=1.29,) and 77.56% of the crowd messages had *#active_workers* >= 3.

EVORUS' AUTOMATION AND LEARNING FRAMEWORK

Evorus is a conversational assistant that is collaboratively run by real-time crowdsourcing and artificial intelligence. The core concept of Evorus is to have crowd workers work with automated virtual agents (referred to as "bots") on the fly to hold sophisticated conversations with users. To test this, we developed two types of bots: the automatic response generators, *i.e.*, **chatbots**, and the automatic voting algorithms, *i.e.*, **vote bots**. Evorus monitored all ongoing conversations, and periodically called chatbots and vote bots to participate in active conversations. Both chatbots and vote bots take the entire chat log as input, and based on the chat log to generate responses or votes. To coordinate with human workers' speed, Evorus often needs to set constraints on the frequency or capability (*e.g.*, in which condition can a chatbot propose responses) of bots. More importantly, Evorus can learn from the crowd feedback to automate itself over time via three primary mechanisms: *(i)* the chatbot selector, *(ii)* the retrieval-based chatbot that can reuse old responses, and *(iii)* the automatic voting bot.

Part I: Learning to Choose Chatbots Over Time

In Evorus, existing dialog systems or chatbots can be incorporated by defining a simple REST interface on top of them that accepts information about the current conversation state, and responses with a suggested response. When a sufficient amount of chatbots are included in the bot pool, selecting the most appropriate chatbots to answer different questions becomes critical. For instance, a simple "ping-pong" chatbot that always responds with what it was told can be selected to reply echo questions such as "Hi" or "How are you?"; a restaurant recommendation bot can be selected when the user is looking for food; and a chatbot that was built on a friend's chat log can be selected when the user feels lonely [31]. In this paper, we introduce a learning framework that uses crowd votes and prior accepted messages to estimate the likelihood of each chatbot when receiving a user message. The learning framework naturally assigns a slightly higher likelihood to newly-added chatbots to collect more data. The beauty of this design is that any chatbot can contribute, as long as it can effectively respond to – even a small – set of user messages.

Part II: Reusing Prior Answers

Upon receiving a message from the user, Evorus uses a retrieval-based approach to find the most similar message in prior conversations and populates its prior response for the crowd to choose from. By doing so, Evorus is capable to reuse the answer of prior similar questions to respond to users. The advantage of using a retrieval-based method is that it naturally

increases its capability of answering questions with the growth of the collected conversations, without the need of recreation or retraining of machine-learning models. With the oversight of the crowd, the retrieval-based approaches also do not need to be perfect to contribute. As long as it find good responses to a portion of user conversations, the learning framework described in Part I can gradually learn when to use it.

Part III: Automatic Voting

Closing the loop of automating the entire system, the last piece is to automate the oversight of the crowd that are necessary for quality control, *i.e.*, the voting process in Evorus. We formulated response voting as a classification task and tackled it with a supervised machine-learning approach. A set of features based on literature, including the word, the speaker, and the time of the proposed messages are used to develop a machine-learning model, and the prior collected crowd votes are used as gold-standard labels. While the overall classifier performance is efficient in the dataset, a misfired vote (a false-positive) that mistakenly accepts a low-quality response will not only disturb the conversation, but also waste extra bonus money to crowd workers who proposed and voted for it. In this paper we propose a mathematical framework to estimate expected benefits of using an automatic voting classifier.

In Evorus, both workers and the vote bot can upvote suggested responses. When a new suggestion is offered, the vote bot is called. It first calculates its *confidence* score, and, if the confidence is greater than a threshold, which is estimated by our proposed mathematical framework, the vote bot automatically upvotes the message. Evorus monitors the latest down/upvotes and calculates voting results in real-time. When a crowd message collects sufficient vote weight, Evorus *(i)* accepts it and sends it to the user, and *(ii)* removes all other candidate messages from the worker interface to refresh context.

In the following three sections, we describe in detail the three main parts of the Evorus framework.

PART I: LEARNING TO CHOOSE CHATBOTS OVER TIME

Evorus' chatbot selector learns over time from the crowd's feedback to choose the right chatbots to respond to user messages. Evorus also **regularly populates lower-ranking chatbots** to allow the model to learn about new chatbots and tp keep the model up-to-date.

Ranking and Sampling Chatbots

Upon receiving a message from a user, Evorus uses both the text and prior collected data to estimate how likely each chatbot is capable of responding the user (*i.e.,* $P(bot|message)$). We used a conditional probability, as shown in Equation 2, to characterize the likelihood of selecting a chatbot (bot) after receiving a user $message$.

$$\begin{aligned} P(bot|message) &= P(bot) \times P(message|bot) \\ &\approx P(bot) \times similarity(message,\ history_{bot}) \end{aligned} \quad (2)$$

$P(bot)$ is the prior probability of the chatbot, and $P(message|bot)$ is the likelihood of the user message given the chatbot's history (*i.e.*, previous user messages that the bot has successfully responded). While training an n-gram language model using previous messages to estimate $P(message|bot)$ is intuitive [13], sufficient data for building such model is often unavailable for newly-added bots. To generalize, we used a similarity measure based on distance between word vectors ($similarity(message,\ history_{bot})$) to approximate this likelihood. We will explain how we calculate these two components in the following subsection.

Equipped with the estimates, Evorus ranks all the chatbots based on the likelihood values, and always calls the first-ranking chatbot to provide its response. More interestingly, in addition to the top chatbot, Evorus also randomly selects a lower-ranking chatbot to provide responses. By doing so, Evorus is capable to gradually update its estimates of each bot based on the crowd feedback and learn over time the best scenario to call each chatbot. Similar strategies, such as the *epsilon-greedy strategy* that yields a small portion of probability for random outcomes and collects feedback, have been used in models that learn to select crowd workers [40] and dialogue actions [38]. For the new chatbot, Evorus initially assigns a starting probability to it to allow the system to collect data about it, which we describe in the following subsection.

Estimating Likelihood of a Chatbot

We aimed at designing a learning framework that is *(i)* inexpensive to update, since we want the model to be updated every single time when the system receives a new label, and *(ii)* allows new bots to be added easily.

Prior Probability of Chatbots: To generate more reliable prior estimation for newly-added bots with limited histories, we used a beta distribution with two shape parameters α and β (Equation 3) to model the prior probability of each chatbot.

$$P(bot) \approx \frac{(\#\text{accepted messages from bot}) + \alpha}{(\#\text{user messages since bot online}) + \alpha + \beta} \quad (3)$$

$P(bot)$ can be interpreted as the *overall acceptance rate* of the chatbot without conversation contexts. The two shaping parameters α and β can be viewed as the number of accepted (positive) and not-accepted (negative) messages that will be assigned to each new chatbot to begin with, respectively. Namely, any new chatbot's prior probability $P(bot)$ will be initially assigned as $\alpha/(\alpha+\beta)$, and then later be updated over time. The beta distribution's α and β are both functions of the mean (μ) and variance (σ^2) of the distribution. In our pilot study, in which each automatic response requires only one vote to be accepted, four chatterbots had an average message acceptance rate of 0.407 (SD=0.028.) Since we increased the required vote count from 1 to 2 in the final deployment, a lower acceptance rate is expected. We used $\mu = 0.3$ and $\sigma = 0.05$ to estimate the shape parameters, where $\alpha = 24.9$ and $\beta = 58.1$.

Similarity between Messages and Chatbots: To estimate $similarity(message,\ bot)$, we first used the pre-trained 200-dimension GloVe word vector representation trained on Wikipedia and Gigaword [32] to calculate the average word vector of each message. We then used previous user messages that were successfully responded by the chatbot as the bot vector $\vec{w}_{bot}$ that represents the chatbot in the word-vector space. We also calculated the centroid vector of *all* user messages, $\vec{w}_{overall}$, to represent general user messages. Finally, as shown

in Equation 4, the similarity between a message and a chatbot is defined as the distance ratio between the vectors.

$$similarity(message,\ history_{bot}) := \frac{dist(\vec{w}_{message}, \vec{w}_{overall})}{dist(\vec{w}_{message}, \vec{w}_{bot}) + dist(\vec{w}_{message}, \vec{w}_{overall})} \quad (4)$$

While $\vec{w}_{bot}$ can be calculated as the centroid vector of prior user messages that were successfully responded to by the chatbot, in cold-start scenarios, a chatbot will not have sufficient accepted messages to calculate the vector. We provide two solutions for chatbot developers: First, the developer can provide a small set of **example messages** where their chatbots should be called. For instance, the developer of an Yelp chatbot can list "*Find me a sushi restaurant in Seattle!*" as an example. Evorus will treat these example messages as the user messages that the chatbot successfully responded to, and use their centroid vector as the initial $\vec{w}_{bot}$. When more messages are accepted, they will be added into this set and update the vector. Second, for some chatbots, especially non-task chatterbots, it could be difficult to provide a set of examples. Therefore, if the developer decided not to provide any example messages, we set the initial $dist(\vec{w}_{message}, \vec{w}_{bot}) = 0$ for new chatbots.

PART II: REUSING PRIOR RESPONSES

Evorus uses an information-retrieval-based (IR-based) method to find answers to similar queries in prior conversations to suggest as responses to new queries. To do so, Evorus first extracts query-response pairs from all the old conversations, and then performs a similarity-based sorting over these pairs.

Extracting Query-Response Pairs: One advantage Evorus has is that each accepted response has crowd votes, which can be used as a direct indicator of the response's quality. For each turn of a conversation between one user and Evorus, we extracted the accepted crowd *response* which did not receive any downvotes, along with the user message *(query)* it responded to, as a *(query, response)* pair. Since the deployed Evorus has not had any prior conversation with users yet, we obtained conversation data that were collected by the deployed Chorus, which also used crowd voting to select responses, during May, 2016 to March, 2017 to start with. We further removed the messages from known malicious workers and users, also removed all conversations where the users are co-authors or collaborators of Chorus [19]. At the beginning of the Evorus deployment, 3,814 user messages were included, and each of these user message is paired with 1 to 5 crowd responses.

Searching for the Most Similar Query: For the *query* message in each *query-response* pair, we calculated its average word vector by using the pre-trained 200-dimension GloVe word vector representation based on Wikipedia and Gigaword [32] and stores the vector in the database. When Evorus receives a user message, the system first calculates its average word vector $\vec{w}_{message}$ using the same GloVe representation, and then searches in the database to look for the top k responses that their corresponding queries' word vectors had the shortest Euclidean distances with $\vec{w}_{message}$. Finally, for increasing answer's diversity, the system randomly selects one from top k responses to send back to Evorus for the crowd to choose from. We empirically set $k = 2$ in our deployment.

PART III: AUTOMATIC VOTING

Evorus uses supervised learning to vote on responses.

Data Preparation: The voting mechanism has been proven to be useful in selecting good responses and holding conversations in the lab prototype [25] and deployed system [19]. The final status (*i.e.*, accepted or not) of a messages is a strong signal to indicate its quality. However, when we used this data to develop an AI-powered automated voting algorithm, it is noteworthy that expired messages were not all of lower quality. In some cases the proposed response was good and fitted in the old context well, but the context changed shortly after the message was sent; Some messages were automatically accepted and bypassed the voting process because Evorus does not have enough active workers, *i.e.*, when $0.4 \times \#active\_workers < 1$ in Equation 1); Furthermore, since downvotes can cancel out upvotes, the voting results in Evorus could be influenced by race conditions among workers. For instance, when two upvotes of a message has been sent to the server, Evorus might decide that this message' vote weight is sufficient and sent it to the user, in which a belated downvote would deduct its vote weight to lower than the threshold. Therefore, the training data needs to be carefully developed.

Similar to Part II, we used voting data collected during the Chorus deployment [19] to train the initial machine learning model for voting. We first extracted the expired messages with one or more downvote(s) as examples of "downvote", and extracted the accepted crowd messages with both one or more upvote(s) and zero downvote as examples of "upvote." We excluded the automatically-accepted messages that were sent when the task did not have sufficient number of active workers ($0.4 \times \#active\_workers < 1$) From all the messages collected by the deployed Chorus during September 2016 to March 2017, 1,682 "upvote" messages 674 "downvote" messages were extracted to from the dataset.

Model & Performance: We then used this dataset to train a LibLinear [9] classifier. For each message, Evorus extracted features in message, turn, and conversation levels to capture the dialogic characteristics [33], and also used GloVe word vector, which is identical as that of our retrieval-based response generation, to represent the content. Our approach reached to a precision of 0.740 and a recall of 0.982 (F1-score = 0.844) on the "upvote" class in a 10-fold cross-validation experiment. The feature analysis using the Weka toolkit [47] showed that the top 3 features were about the historical performance of the worker who proposed the message, and also 13 out of 20 top features were of a particular dimension of one of the word vectors. On the other hand, the performance of the "downvote" class is less effective. Its precision is 0.714 but recall is only 0.134. This could be caused by the insufficient amount of training data, since Chorus promoted upvote more than downvote by design. According to this result, in the deployed Evorus, the system only automatically upvoted when the classifier ouput "upvote," but did not downvote otherwise.

Optimizing Automatic Voting

Automatic voting directly participates in the process of deciding which messages to send. While our machine-learning model resulted in good performance on our dataset, we would

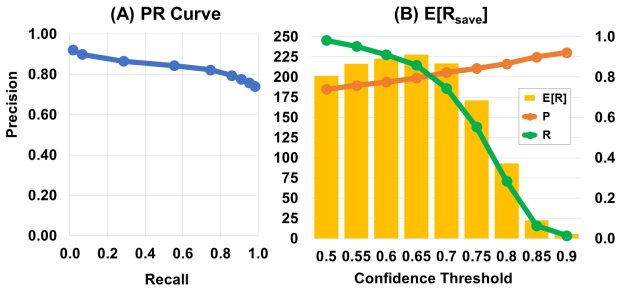


Figure 3. (A) The precision-recall curve of the LibLinear classifier for automatic upvoting. (B) Using our model (Equation (5)) to estimate the precision and recall at different thresholds and their corresponding expected reward amount saved.

like to use Evorus' worker reward point schema to find the right **confidence threshold** for the automatic voting classifier. If the threshold is set too low, the classifier would vote frequently even when it is not confident about the prediction, and thus many low-quality responses would be accepted and disturb the conversation; if the threshold is set too high, the classifier would rarely vote, and the system will not gain much from using it. Liblinear can output the probability estimates of each class when performing prediction, which we used as the notion of confidence of the classifier. Thresholding out the predictions with lower confidences increased the precision but reduced the recall of the classifier. Figure 3(A) shows the Precision-Recall curve of Evorus' upvoting classifier.

Possible Outcomes When the Classifier Upvotes: To find confidence thresholds for Evorus, we introduced the following heuristics to estimate reward points saved per message by using the upvoting classifier. Consider the following cases:

1. **[Good Vote]** The classifier upvoted on a message that would originally be selected by the crowd. It saves 1 upvote reward ($R_{upvote} \times 1$) and 1 agreement reward ($R_{agreement} \times 1$) that would originally be granted to one human worker.

2. **[Bad Vote]** The classifier upvoted on a message that would originally *not* be selected by the crowd. In this case, one of the two following consequences will occur: *(i)* **[Misfire]** The message is **sent** to the user. The system grants agreement rewards to all human workers who upvoted on this message ($R_{agreement} \times \#upvoted\_workers$) and 1 successful proposal reward to the worker who proposed the message ($R_{proposal} \times 1$); and *(ii)*: **[No Difference]** The message remains **not sent**. Even with one extra upvote, this message's vote count was still insufficient to get accepted.

Estimating System's Expected Gain: Given these setups, the expected reward points $E[R_{save}]$ saved per message by using the classifier can be estimated as follows:

$$E[R_{save}] = TPR \times E[Good] - FPR \times E[Bad] \qquad (5)$$

TPR is the **true positive rate** and FPR is the **false positive rate** of the classifier. $E[Good]$ is the expected reward points saved per [Good Vote] event, and $E[Bad]$ is the expected reward points wasted per [Bad Vote] event.

In Evorus, $E[Good]$ is a constant ($R_{upvote} + R_{agreement}$). Meanwhile, [Bad Vote] event costs reward points only when the upvoted message is sent ([Misfire]). Therefore, $E[Bad]$ is decided by *(i)* how often one mistaken upvote triggers a misfire, and *(ii)* how expensive is one misfire, as follows:

$$E[Bad] = P(Misfire|Bad) \times E[R_{Misfire}]$$

$P(Misfire|Bad)$ is the conditional probability of a message being sent to the user given the classifier has mistakenly upvoted on it. $E[R_{Misfire}]$ is the expected reward points that were granted to workers in a single [Misfire] event.

Our training dataset only used the not-accepted messages with at least one downvote to form the "Downvote" class, which were less likely to be misfired after adding one extra automatic vote. For better estimating $P(Misfire|Bad)$, we first ran the classifier on all messages that were *not* included in our training set, and within all the messages that the classifier decided to upvote, we then calculated the proportion of messages that would be sent to the user if one extra upvote were added. The rate was 0.692, which we used to approximate $P(Misfire|Bad)$. Furthermore, based on Evorus' mechanism, $E[R_{Misfire}]$ can be calculated as follows :

$$E[R_{Misfire}] = R_{agreement} \times E[\#upvoted\_workers] + R_{proposal}$$

$E[\#upvoted\_workers]$ is the expected number of human workers who upvoted on the message in an [Misfire] event. Similarly, we ran the classifier on the unlabelled data, and calculated the average number of workers who upvoted on the messages that the classifier decided to upvote on. The number is 0.569 (SD = 0.731), which we used to approximate $E[\#upvoted\_workers]$.

Finally, $E[Hit] = 100 + 500 = 600$, and $E[R_{Bad}] = 0.692 \times (500 \times 0.569 + 1000) = 888.874$. Using Equation (5), we can estimate the precision and recall at different thresholds and their corresponding reward amount (Figure 3(B)). According to the estimation, the best confidence threshold is at 0.65. In the deployed Evorus we selected a slightly higher precision and set the threshold at 0.7 ($P = 0.823$ and $R = 0.745$.)

DEPLOYMENT STUDY AND RESULTS

Evorus was launched to the public as a Google Hangouts chatbot in March 2017. While the end-users were not aware of the changes of the system from the client side, behind the scenes, our deployment had 3 phases: *(i)* Phase 1, *(ii)* Control Phase, and *(iii)* Phase 2. Phase-1 deployment started in March, 2017. We launched the system with only four chatterbots and one vote bot, without the learning component described in Part I, to understand the basics of having virtual bots working with human workers on the fly. For comparison, in May 2017 we then temporarily turned off all automation components and had the system solely run by the crowd till late August 2017, which we referred to as the Control Phase. Finally, for testing the capability of learning to select chatbots, we started Phase 2 deployment in early September. The Phase-2 deployment included several significant changes: *(i)* increasing the frequency of calling chatbots for responses, *(ii)* increasing the vote count needed to accept a response from 1 to 2, and *(iii)* incorporating the Part I learning.

To recruit users, we periodically sent emails to mailing lists at several universities and posted on social media sites, such

as Facebook and Twitter. Participants who volunteered to use our system were asked to sign a consent form first, and no compensation was offered. After the participants submitted the consent form, a confirmation email was automatically sent to them to instruct them how to send messages to Evorus via Google Hangouts. The users can use Evorus as many times as they want to, for anything, via any devices that are available to them. Eighty users total talked with Evorus during 281 conversations. The Phase-1 deployment had 34 users talked to Evorus during 113 conversations, and Phase-2 deployment (till 17th September, 2017) had 26 users with 39 conversations. The Control Phase had 42 users with 129 conversations.

Phase 1: Chatterbots & Vote bot

Our Phase-1 deployment explored how chatbots and our vote bot could work together synchronously with crowd workers. We implemented four chatterbots (including the IR-based chatterbot using Chorus conversation data described in Part II) and a vote bot. During the Phase-1 deployment, the system only randomly selects one of four chatterbots to respond every half a minute, where the learning component described in Part I will be later included in Phase-2 deployment.

Implementing Four Chatterbots: In our Phase-1 deployment, we implemented the following four chatterbots.

1. **Chorus Bot (shown as Part II):** A chatterbot that is powered by a retrieval-based method to reuse prior conversations to respond to users, which was described in Part II.
2. **Filler Bot:** A chatterbot that randomly selects one response from a set of candidates, regardless of context. We manually selected 13 common "conversation filler" in the Chorus dataset (*e.g.*, "Is there anything else I can help you with?", or "Thanks") to form the candidate pool.
3. **Interview Bot:** A chatterbot that uses a retrieval-based method, which is identical to Chorus Bot, to find the best response from 27,894 query-response pairs extracted from 767 transcripts of TV celebrity interview [30].
4. **Cleverbot:** Cleverbot is a third-party AI-powered chatbot which reuses more than 200 million conversations it had with users to generate responses [44, 4].

Vote Bot Setup: Currrently, The vote bot only votes on human-proposed messages, but not messages proposed by chatbots. In Phase 1, Evorus requires each automatic vote to have at least one extra human upvote to be accepted. Vote bots also skipped messages if the same worker proposed identical content earlier in the conversation but did not get accepted. We believe that worker's re-sending is a strong signal of the poor quality of the message. Vote bots can decide not to vote if the confidence is too low (Part III.)

Automating Human Labors: During Phase-1 deployment, a conversation on average contained 9.90 user messages (SD = 11.69), 13.6 accepted messages proposed by the crowd workers (SD = 10.44), and 13.58 accepted messages proposed by automatic chatterbots (SD = 2.81). Thus, **automated responses were chosen 12.44% of the time**. As a comparison (Figure 4(A)), in the Control Phase (42 users and 129 conversations), a conversation on average contained 8.73 user messages (SD = 10.05) and 12.98 accepted crowd messages

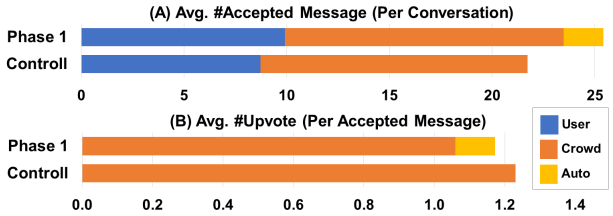


Figure 4. Phase-1 Deployment: (A) Average number of accepted messages per conversation. Automated responses were chosen 12.44% of the time. (B) Average number of upvotes per accepted non-user message. Human upvotes were reduced by 13.81% by using automatic voting.

(SD = 11.39). In terms of upvotes, each accepted non-user message received 1.06 human upvotes (SD = 0.73) and 0.11 automatic upvotes (SD = 0.18) In comparison, in the Control Phase, each accepted non-user message received 1.23 human upvotes (SD = 0.70.) **Crowd voting was thus reduced by 13.81%**. The comparison is shown in Figure 4(B). Moreover, an accepted non-user message sent by Evorus costed $0.142 in Phase-1 deployment on average, while it costed $0.211 during the Control Phase. Namely, with automated chatbots and the vote bot, **the cost of each message is reduced by 32.76%**.

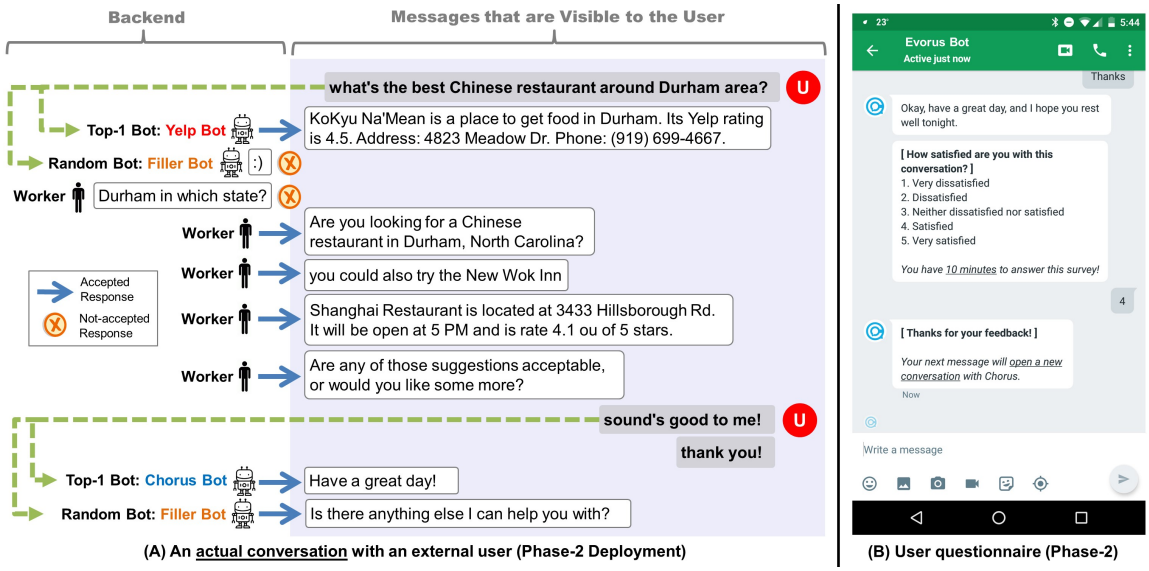
We also calculated the acceptance rate of messages proposed by each chatbot. The Filler Bot, which ignores context and proposes responses randomly, had the highest acceptance rate, 41.67%. The Chorus Bot's acceptance rate was 30.99%, that of the Interview Bot was 33.33%, and that of the Cleverbot was 30.99%. This might be because Filler Bot's commonplace responses (*e.g.*, "I don't know") were often considered acceptable by human raters. In Phase 2, where an automatic response needed 2 human votes to go through, the Chorus Bot had the highest acceptance rate among all four chatterbots (Figure 6). While human workers, whose acceptance rate was 72.04% during Phase 1, still outperformed all chatbots by a large margin, chatbots with low accuracy can still contribute to the conversation. For instance, the Filler Bot, while being very simple and ignoring any context, nevertheless, often produces reasonable responses:

[*The user asked information about the wildfire and smoke in Emory university campus.*]
user Do you know where they are happening exactly? (The wildfires I mean)
bot **Can you provide some more details?**

Compared with Filler Bot, Chorus Bot better targets its responses because it chooses messages based on similarity with previous human responses:

user Hey how many people like bubble tea here?
bot **Ask for their feedback when you talk with them**

Conversation Quality: We sampled conversations with accepted automatic responses and a matching set without automated contributions. For each, 8 MTurk workers rated [Satisfaction, Clarity, Responsiveness, Comfort], which was based on the PARADISE's objectives for evaluating dialogue performance [42] and the Quality of Communication Experience metric [28], on a 5-point Likert scale (5 is best.) The original conversations (N=46) had an average rating of [3.47, 4.04, 3.88, 3.56], while those with automatic responses (N=54) had [3.57, 3.74, 3.52, 3.66]. The similar results suggest that the automatic components did not make conversations worse.



(A) An actual conversation with an external user (Phase-2 Deployment)

(B) User questionnaire (Phase-2)

Figure 5. (A) An actual conversation of Evorus. Conversations in Evorus tend to combine multiple chatbots and workers together. (B) User questionnaire used in Phase-2 deployment. The average user satisfaction rating of automated and non-automated conversations had no significant difference.

Phase 2: Learning to Select Chatbots

Our Phase-2 deployment explored how the learning component, described in Part I, can select the right chatbots in context, and how integrating additional chatbots affects performance. We implemented two additional *utility bots*, a Yelp Bot and a Weather Bot, that can perform information inquiry tasks for different contexts, in addition to the four chatterbots and one vote bot from Phase 1. We first launched the learning system with four chatterbots for two days, and then added the two utility bots for observing the changes of the model. Furthermore, in order to directly compare user satisfaction levels, all the automated components (including the chatbots, the vote bot, and the learning component) were only applied to 50% of the conversations randomly, and the other half of the conversations were solely run by the crowd as a baseline.

To efficiently collect crowd feedbacks to update our model, we increased the frequency Evorus called chatbots from randomly calling one chatbot and one vote bot every 30 seconds (Phase 1) to calling two chatbots (top-1 ranked plus random) and one vote bot every 10 seconds (Phase 2.) To compensate for the possible drop in quality caused by higher calling frequency and randomly selecting one of the two bots, we increased the required upvote count for accepting an automatic response from 1 vote to 2 votes. Namely, while Evorus obtained more automatic responses with a much higher frequency in Phase 2, it also required more human upvotes to approve each automatic response at the same time. As a result, among the conversations that had automation in Phase 2, automated responses were chosen 13.25% of the time, in which a conversation on average contained 10.68 user messages (SD = 8.90), 15.74 accepted messages proposed by the crowd workers (SD = 11.92), and 2.40 accepted messages proposed by automatic chatterbots (SD = 2.40). Each accepted non-user message received 1.90 human upvotes (SD = 1.13) and 0.30 automatic upvotes (SD = 0.22). We did not compare Phase 2's results in detail to that of the Control Phase because the high-frequency setup of Phase 2 is primarily for experimental exploration.

Implementing Two Utility Bots: In addition to the four chatterbots in the Phase-1 deployment, we implemented the following two task-oriented utility bots:

1. **Yelp Bot:** A chatbot that suggests restaurants near the location mentioned by the user powered by the Yelp API [49]. If the user did not mention any location, it replies with "You're looking for a restaurant. What city are you in?".
2. **Weather Bot:** A chatbot that reports the current weather of a mentioned city powered by the WeatherUnderground API [41]. If the user did not mention any city names, it replies, "Which city's weather would you like to know?"

The chatbots' developer (the first author) also provided three example chat messages for each that should trigger the corresponding chatbot, to initiate the learning process. For example, "Any restaurant recommendations in NYC?" for the Yelp Bot.

Similar User Satisfaction Level: In Phase 2 we implemented an exit survey to measure end user satisfaction (Figure 5(B)). At the end of each conversation, the user had 10 minutes to report their satisfaction using a Likert-scale ranging from 1 (Very Dissatisfied) to 5 (Very Satisfied). 13 of 30 users provided feedback (response rate = 43%). The automated conversations' average user satisfaction rating was 4.50 (SD=0.5, N=4); the crowd conversations' average user satisfaction rating was 4.00 (SD=0.47, N=9), a difference that was not significant.

Updating Estimates of Chatbot's Prior Over Time: While our deployment is of a medium scale, the dynamics of our likelihood model can still be observed. For instance, the estimated prior probability described in Equation 3 was continuously updated with the growth of conversation that Evorus had. Our model assigned a starting probability of 0.3 to each chatbots (Figure 6). When users started talking with Evorus, crowd workers provided their feedback by upvoting and downvoting, and thus changed the estimation over time. When new chatbots were added, Evorus intentionally assigned them a higher prior probabilities to allow quicker crowd feedback.

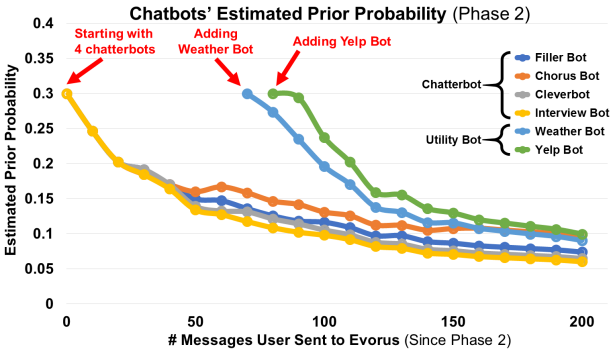


Figure 6. The estimated prior probability (Equation 3) of each chatbot was continuously updated with the growth of user messages.

Utility Bots in Cold-Start Scenarios: We would like to understand if Evorus can select appropriate chatbots to obtain responses in corresponding context. Since non-task chatterbots such as the Cleverbot could be difficult for humans to judge if it should be called given a message, we focused only on task-oriented utility bots in the evaluation. For each user message in the automated conversations in Phase 2, the researchers manually annotated if it is relevant to the topic of "weather" and "restaurant," respectively. With the assumption that each utility bot should be called when its topic comes up, we compared the human-labelled topic against the top chatbot that were suggested by Evorus and calculated the precision, recall, and F1-score. It is noteworthy that we only evaluated when the appropriate chatbot was called, regardless of the quality of the responses it generated. As a result, two newly-added utility bots both had a high precision and a lower recall. The Weather Bot's precision was 1.00 and the recall was 0.47 (F1=0.64); and the Yelp Bot's precision was 0.67 and the recall was 0.20 (F1=0.31.) This result shows the nature of new bots in our learning framework: Evorus uses the expert-generated small set of examples to calculate the initial vector of each chatbot, which could result in precise predictions but with lower coverage. Over time, when Evorus collects more examples that each bot has successfully responded to and the recall increases. Moreover, conversations in Evorus tend to combine multiple chatbots and workers together. For instance, in the following conversation, the crowd had the user to narrow down the query, and then the Weather Bot was able to answer it. An additional detailed example is shown in Figure 5(A).

[*The user asked about the weather in Afghanistan.*]
crowd What city in Afghanistan?)
[*The crowd sends a website about Afghanistan weather.*]
user Kabul
bot **Friday's weather forecast for [Kabul, Afghanistan]: Cloudy with a few showers. High 79F. Winds NW at 5 to 10 mph. Chance of rain 30%.**

Error Cases: Our bot-selection algorithm starts with high precision and low recall, and increases recall as it gradually gathers examples. Therefore, most errors we observed were false-negatives, where a chatbot should have been triggered but was not. Other errors came from the chatbot, where the bot was correctly triggered but its response was invalid. Workers usually downvoted or ignored these suggestions. In the rare cases where invalid automatic responses were mistakenly sent to the user, the crowd often tried to explain how the response was automatically generated to the user afterward.

user hi! Can you summarize the features of the new iPhone for me?
[*Multiple messages list the features of the iPhone X.*]
bot **There is no iPhone 7.**
[*The crowd lists more features, and the user says thank you.*]
crowd No problem!
crowd Some Auto replies don't even make sense

DISCUSSION AND CONCLUSION

We introduced Evorus, a crowd-powered system conversational assistant built to automate itself over time. Informed by two phases of public field deployment and testing with real users, we iteratively designed and refined its flexible framework for open-domain dialog. We imagine a future where thousands of online service providers can develop their own chatbots, not only to serve their own users in a task-specific context, but also to dynamically integrate their services into Evorus with the help of the crowd, allowing users to interact freely with thousands of online services via a universal portal. Supporting this scale offers opportunities for future research. For example, one direction is to improve the learning framework to support third-party chatbots that also improves overtime, or to better balance between the exploitation and exploration phases (like in a multi-armed bandit problem). Evorus could also be used to collect valuable fail cases to enable third-party developers to improve their bots (*i.e.*, when a bot was triggered, but its proposed response was rejected).

Evorus has three main advantages as compared to previous approaches. First, it is a working system that can serve as a scaffold for automation over time. A core advantage of starting with a working system is that users can talk to Evorus naturally from day one, ensuring conversation quality while collecting training data for automation. Second, given the oversight of the crowd, Evorus has a high tolerance for errors from its automated components. Even an imperfect automation component (*e.g.,* chatbots) can contribute to a conversation without hurting quality, which yields more space for algorithms to "explore" different actions (*e.g.,* selecting a chatbot with medium confidence.) Finally, Evorus allows a mixed group of humans and bots to collaboratively hold open conversations.

Most automated systems created from crowd work simply use the crowd for data; Evorus tightly integrates crowds and machine learning, and provides specific points where automated components can be introduced. This architecture allows each component to be improved, providing a common research harness on which researchers specializing in different areas may innovate and compete. For instance, "response generation" has long been developed in the NLP community; Evorus provides a natural evaluate it within a larger conversational system. The flexibility of the Evorus framework potentially allows for low cost integration between many online service providers and fluid collaboration between chatbots and human workers to form a single user-facing identity. Given the complexity of conversational assistance, Evorus is likely to be crowd-powered in part for some time, but we expect it to continue to increasingly rely on automation.

ACKNOWLEDGEMENTS

We thank Walter S. Lasecki and also the workers who participated in our studies. This project has been funded as part of the Yahoo!/Oath InMind Project at Carnegie Mellon University.

REFERENCES

1. Amazon. 2017. Meet Alexa. (2017). https://www.amazon.com/meet-alexa/b?ie=UTF8&node=16067214011
2. Rafael E Banchs and Haizhou Li. 2012. IRIS: a chat-oriented dialogue system based on the vector space model. In *Proceedings of the ACL 2012 System Demonstrations*. Association for Computational Linguistics, 37–42.
3. Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, and Tom Yeh. 2010. VizWiz: Nearly Real-time Answers to Visual Questions. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 333–342. DOI: http://dx.doi.org/10.1145/1866029.1866080
4. Rollo Carpenter. 2006. Cleverbot. (2006). https://www.cleverbot.com/ [Online; accessed 08-March-2017].
5. Joseph Chee Chang, Aniket Kittur, and Nathan Hahn. 2016. Alloy: Clustering with crowds and computation. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 3180–3191.
6. Yun-Nung Chen, Dilek Hakkani-Tür, Gökhan Tür, Jianfeng Gao, and Li Deng. 2016. End-to-End Memory Networks with Knowledge Carryover for Multi-Turn Spoken Language Understanding.. In *INTERSPEECH*. 3245–3249.
7. Justin Cheng and Michael S Bernstein. 2015. Flock: Hybrid crowd-machine learning classifiers. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 600–611.
8. Josh Constine. 2017. Amazon rejects AI2's Alexa skill voice-search engine. Will it build one? (May 2017). https://techcrunch.com/2017/05/31/amazon-skill-search-engine/
9. Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR - A Library for Large Linear Classification. (2008). http://www.csie.ntu.edu.tw/~cjlin/liblinear/ The Weka classifier works with version 1.33 of LIBLINEAR.
10. Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: Answering Queries with Crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*. ACM, New York, NY, USA, 61–72. DOI: http://dx.doi.org/10.1145/1989323.1989331
11. Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Dialogue manager domain adaptation using Gaussian process reinforcement learning. *arXiv preprint arXiv:1609.02846* (2016).
12. Nathan Hahn, Joseph Chang, Ji Eun Kim, and Aniket Kittur. 2016. The Knowledge Accelerator: Big picture thinking in small pieces. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2258–2270.
13. Bo Han and Timothy Baldwin. 2011. Lexical Normalisation of Short Text Messages: Makn Sens a #Twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1 (HLT '11)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 368–378. http://dl.acm.org/citation.cfm?id=2002472.2002520
14. Hangoutsbot. 2017. hangoutsbot/hangoutsbot. (Apr 2017). https://github.com/hangoutsbot/hangoutsbot
15. Jessi Hempel. 2015. Facebook Launches M, Its Bold Answer to Siri and Cortana. (Aug 2015). https://www.wired.com/2015/08/facebook-launches-m-new-kind-virtual-assistant/
16. Ting-Hao Kenneth Huang, Amos Azaria, and Jeffrey P. Bigham. 2016. InstructableCrowd: Creating IF-THEN Rules via Conversations with the Crowd. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*. ACM, New York, NY, USA, 1555–1562. DOI: http://dx.doi.org/10.1145/2851581.2892502
17. Ting-Hao K. Huang and Jeffrey P. Bigham. 2017. A 10-Month-Long Deployment Study of On-Demand Recruiting for Low-Latency Crowdsourcing. In *In Proceedings of The fifth AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2017)*. AAAI, AAAI.
18. Ting-Hao K. Huang, Yun-Nung Chen, and Jeffrey P. Bigham. 2017. Real-time On-Demand Crowd-powered Entity Extraction. In *In Proceedings of the 5th Edition Of The Collective Intelligence Conference (CI 2017, oral presentation)*.
19. Ting-Hao Kenneth Huang, Walter S. Lasecki, Amos Azaria, and Jeffrey P. Bigham. 2016. "Is there anything else I can help you with?": Challenges in Deploying an On-Demand Crowd-Powered Conversational Agent. In *Proceedings of AAAI Conference on Human Computation and Crowdsourcing 2016 (HCOMP 2016)*. AAAI.
20. Ting-Hao Kenneth Huang, Walter S Lasecki, and Jeffrey P Bigham. 2015. Guardian: A Crowd-Powered Spoken Dialog System for Web APIs. In *Third AAAI Conference on Human Computation and Crowdsourcing*.
21. Ece Kamar, Severin Hacker, and Eric Horvitz. 2012. Combining human and machine intelligence in large-scale crowdsourcing. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 467–474.

22. Ece Kamar and Lydia Manikonda. 2017. Complementing the Execution of AI Systems with Human Computation. In *AAAI Workshop on Crowdsourcing, Deep Learning and Artificial Intelligence Agents 2017*. AAAI.

23. G. Laput, W. S. Lasecki, J. Wiese, R. Xiao, J. P. Bigham, and C. Harrison. 2015. Zensors: Adaptive, Rapidly Deployable, Human-Intelligent Sensor Feeds. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 10. **http://www.cs.cmu.edu/~jbigham/pubs/pdfs/2015/zensors.pdf**

24. Walter S. Lasecki, Phyo Thiha, Yu Zhong, Erin Brady, and Jeffrey P. Bigham. 2013a. Answering Visual Questions with Conversational Crowd Assistants. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '13)*. ACM, New York, NY, USA, Article 18, 8 pages. DOI:**http://dx.doi.org/10.1145/2513383.2517033**

25. Walter S. Lasecki, Rachel Wesley, Jeffrey Nichols, Anand Kulkarni, James F. Allen, and Jeffrey P. Bigham. 2013b. Chorus: A Crowd-powered Conversational Assistant. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 151–162. DOI:**http://dx.doi.org/10.1145/2501988.2502057**

26. Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A Persona-Based Neural Conversation Model. *arXiv preprint arXiv:1603.06155* (2016).

27. Xuijun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. 2017. End-to-End Task-Completion Neural Dialogue Systems. In *Proceedings of The 8th International Joint Conference on Natural Language Processing (IJCNLP 2017)*. AFNLP.

28. Leigh Anne Liu, Chei Hwee Chua, and Günter K Stahl. 2010. Quality of communication experience: definition, measurement, and implications for intercultural negotiations. *Journal of Applied Psychology* 95, 3 (2010), 469.

29. Matthew Lynley. 2016. Make Magic's Assistants Do Almost Anything With $100/Hour And A Text Message. (Jan 2016). **https://techcrunch.com/2016/01/05/make-magics-assistants-do-almost-anything-with-100hour-and-a-text-message/**

30. Cable News Network. 2017. (2017). **http://transcripts.cnn.com/TRANSCRIPTS/**

31. Casey Newton. 2016 (accessed October 24th, 2016). *SPEAK, MEMORY: When her best friend died, she rebuilt him using artificial intelligence*. **https://www.theverge.com/a/luka-artificial-intelligence-memorial-roman-mazurenko-bot**

32. Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. **http://www.aclweb.org/anthology/D14-1162**

33. Antoine Raux and Maxine Eskenazi. 2008. Optimizing endpointing thresholds using dialogue features in a spoken dialogue system. In *Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue*. Association for Computational Linguistics, 1–10.

34. Daniela Retelny, Sébastien Robaszkiewicz, Alexandra To, Walter S Lasecki, Jay Patel, Negar Rahmati, Tulsee Doshi, Melissa Valentine, and Michael S Bernstein. 2014. Expert crowdsourcing with flash teams. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 75–85.

35. Alan Ritter, Colin Cherry, and William B Dolan. 2011. Data-driven response generation in social media. In *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 583–593.

36. Erica Sadun and Steve Sande. 2012. *Talking to Siri: Learning the Language of Apple's Intelligent Assistant*. Que Publishing.

37. Akash Das Sarma, Ayush Jain, Arnab Nandi, Aditya Parameswaran, and Jennifer Widom. 2015. Surpassing humans and computers with JELLYBEAN: Crowd-vision-hybrid counting algorithms. In *Third AAAI Conference on Human Computation and Crowdsourcing*.

38. Konrad Scheffler and Steve Young. 2002. Automatic Learning of Dialogue Strategy Using Dialogue Simulation and Reinforcement Learning. In *Proceedings of the Second International Conference on Human Language Technology Research (HLT '02)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 12–19. **http://dl.acm.org/citation.cfm?id=1289189.1289246**

39. Saiganesh Swaminathan, Raymond Fok, Fanglin Chen, Ting-Hao K. Huang, Irene Lin, Rohan Jadvani, Walter Lasecki, and Jeffrey Bigham. 2017. WearMail: On-the-Go Access to Information in Your Email with a Privacy-Preserving Human Computation Workflow. In *30th ACM Symposium on User Interface Software and Technology (UIST 2017)*.

40. Long Tran-Thanh, Sebastian Stein, Alex Rogers, and Nicholas R Jennings. 2014. Efficient crowdsourcing of unknown experts using bounded multi-armed bandits. *Artificial Intelligence* 214 (2014), 89–111.

41. Weather Underground. 2017. A Weather API Designed for Developers. (2017). **https://www.wunderground.com/weather/api/**

42. Marilyn A Walker, Diane J Litman, Candace A Kamm, and Alicia Abella. 1997. PARADISE: A framework for evaluating spoken dialogue agents. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 271–280.

43. Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young. 2016. Multi-domain Neural Network Language Generation for Spoken Dialogue Systems. *arXiv preprint arXiv:1603.01232* (2016).
44. Wikipedia. 2017a. Cleverbot — Wikipedia, The Free Encyclopedia. **http://en.wikipedia.org/w/index.php?title=Cleverbot&oldid=771836990**. (2017). [Online; accessed 02-April-2017].
45. Wikipedia. 2017b. Tay (bot) — Wikipedia, The Free Encyclopedia. **http://en.wikipedia.org/w/index.php?title=Tay%20(bot)&oldid=769762463**. (2017). [Online; accessed 04-April-2017].
46. Jason D Williams and Steve Young. 2007. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language* 21, 2 (2007), 393–422.
47. Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.
48. Xuesong Yang, Yun-Nung Chen, Dilek Hakkani-Tür, Paul Crook, Xiujun Li, Jianfeng Gao, and Li Deng. 2017. End-to-end joint learning of natural language understanding and dialogue manager. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 5690–5694.
49. Yelp. 2017. Yelp API Documentation. (2017). **https://www.yelp.com/developers/documentation/v2/overview**
50. Tiancheng Zhao, Kyusong Lee, and Maxine Eskenazi. 2016. DialPort: Connecting the Spoken Dialog Research Community to Real User Data. *arXiv preprint arXiv:1606.02562* (2016).