DOI: 10.1609/aaai.12025

SPECIAL TOPIC ARTICLE

Feedback-based self-learning in large-scale conversational AI agents

Pragaash Ponnusamy | **Alireza Roshan Ghias** | **Yi Yi** | **Benjamin Yao** | **Chenlei Guo** | **Ruhi Sarikaya**

Amazon Alexa

Correspondence
Chenlei Guo, Amazon Alexa.
Email: chenlei.guo@gmail.com

Abstract

Today, most of the large-scale conversational AI agents such as Alexa, Siri, or Google Assistant are built using manually annotated data to train the different components of the system including automatic speech recognition (ASR), natural language understanding (NLU), and entity resolution (ER). Typically, the accuracy of the machine learning models in these components are improved by manually transcribing and annotating data. As the scope of these systems increase to cover more scenarios and domains, manual annotation to improve the accuracy of these components becomes prohibitively costly and time consuming. In this paper, we propose a system that leverages customer/system interaction feedback signals to automate learning without any manual annotation. Users of these systems tend to modify a previous query in hopes of fixing an error in the previous turn to get the right results. These reformulations, which are often preceded by defective experiences caused by either errors in ASR, NLU, ER, or the application. In some cases, users may not properly formulate their requests (e.g., providing partial title of a song), but gleaning across a wider pool of users and sessions reveals the underlying recurrent patterns. Our proposed self-learning system automatically detects the errors, generates reformulations, and deploys fixes to the runtime system to correct different types of errors occurring in different components of the system. In particular, we propose leveraging an absorbing Markov Chain model as a collaborative filtering mechanism in a novel attempt to mine these patterns, and coupling it with a guardrail rewrite selection mechanism that reactively evaluates these fixes using feedback friction data. We show that our approach is highly scalable, and able to learn reformulations that reduce Alexa-user errors by pooling anonymized data across millions of customers. The proposed self-learning system achieves a win-loss ratio of 11.8 and effectively reduces the defect rate by more than 30 percent on utterance level reformulations in our production A/B tests. To the best of our knowledge, this is the first self-learning large-scale conversational AI system in production.

INTRODUCTION

Large-scale conversational AI agents (2017) such as Alexa, Siri, and Google Assistant are getting more and more prevalent, opening up in new domains and taking up new tasks to help users across the globe. One key consideration in designing such systems is how they can be improved over time at that scale. Users interacting with these agents experience frictions due to various reasons: (1) automatic speech recognition (ASR) errors, such as "*play maj and dragons*" (should be "*play imagine dragons*"), (2) natural language understanding (NLU) errors, such as "*don't play this song again skip*" (Alexa would understand if it is formulated as "*thumbs down this song*"), and even user errors, such as "*play bazzi angel*" (it should have been "*play beautiful by bazzi*"). It goes without saying that fixing these frictions help users to have a more seamless experience and engage more with the AI agents.

One common method to address frictions is to gather these use cases and fix them manually using rules and finite state transducers (FST) as they are often the case in speech recognition systems (2002). This of course is a laborious technique which is: (1) not scalable at Alexa scale, (2) prone to error, and (3) getting stale and even defective over time. Another approach could be to identify these frictions, ask annotators to come up with the correct form of query, and then update ASR and NLU models to solve these problems. This is also: (1) not a scalable solution, since it needs a lot of annotations, and (2) it is expensive and time consuming to update those models. Instead, we have taken a "query rewriting" approach to solve customer frictions, meaning that when necessary, we reformulate a customer's query such that it conveys the same meaning/intent, and is actionable (i.e., interpretable) by Alexa's existing NLU systems.

In motivating our approach, consider the example utterance, "*play maj and dragons*." Now, without reformulation, Alexa would inevitably come up with the response, "*Sorry, I couldn't find maj and dragons*." Some customers give up at this point, while others may try enunciating better for Alexa to understand them: "*play imagine dragons*." Also note that there might be other customers who give up and change the next query to another intent, for example: "*play pop music*." Here, frictions evidently cause dissatisfaction with different customers reacting differently to them. However, quite clearly there are good rephrases by some customers among all these interactions, which beckons the question – how can we identify and extract them to solve customer frictions?

We propose using a Markov-based collaborative filtering approach to identify rewrites that lead to successful customer interactions. We go on to discuss the theory and implementation of the idea, as well as show that this method is highly scalable and effective in significantly reducing customer frictions. We also discuss how this approach was deployed into customer-facing production and what are some of the challenges and benefits of such approach.

RELATED WORK

Collaborative filtering has been used extensively in recommender systems. In a more general sense, collaborative filtering can be viewed as a method of mining patterns from various agents (most commonly, people), in order to help each other out (2001). Markov chains have been used previously in collaborative filtering applications to recommend course enrollment (2016), personalized recommender systems (2012), and web recommendation (2012).

Studies have shown that Markov processes can be used to explain the user web query behavior (2005), and Markov chains have since been used successfully for web query reformulation via absorbing random walk (2015), and modeling query utility (2012). We here present a new method for query reformulation using Markov chain that is both highly scalable and interpretable due to intuitive definitions of transition probabilities. Also, to the best of the authors' knowledge, this is the first work where Markov chain is used for query reformulation in voice-based virtual assistants.

One important difference between the web query reformulation and Alexa's use case is that we need to seamlessly replace the user's utterance in order to remove friction. Asking users for confirmation every time we plan to reformulate is on itself an added friction, which we try to avoid as much as possible. Another difference is how success and failure are defined for an interaction between user and a voice-based virtual assistant system. We use implicit and explicit user feedback when interacting with Alexa to establish the *absorbing states* of success and failure.

SYSTEM OVERVIEW

The Alexa conversational AI system follows a rather well-established architectural design pattern of cloud-based digital voice assistants (2018), that is, comprising, in order, of an ASR system, an NLU system with a built-in dialog manager, and a text-to-speech (TTS) system, as visualized in Figure 1 Conventionally, as a user interacts with their Alexa-enabled device, their voice is first recognized by the ASR engine and decoded into a plain text surface form, which we refer to as an utterance. The utterance is then interpreted by the NLU component to surface the aforementioned user's intent by also accounting for the state of

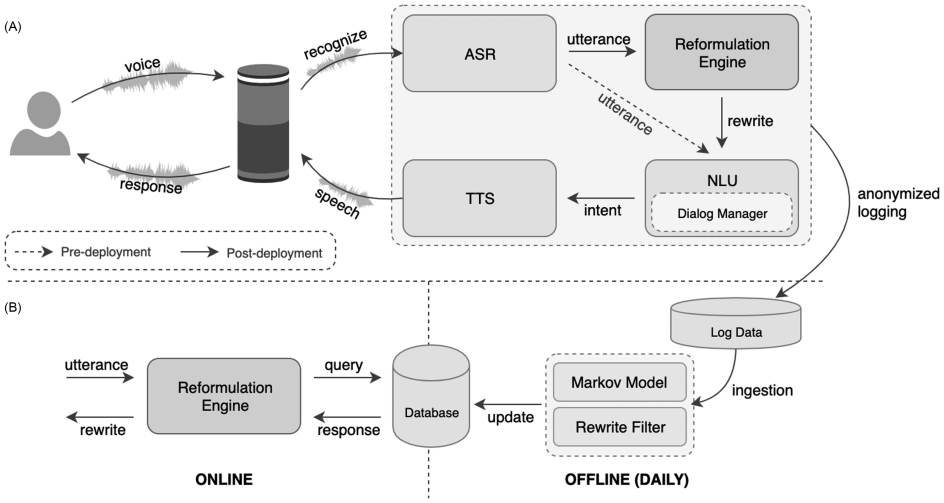

FIGURE 1 A high-level overview of the deployed architecture with our reformulation engine in context of the overall system in (A) and the offline sub-system that updates its online counterpart on a daily cadence

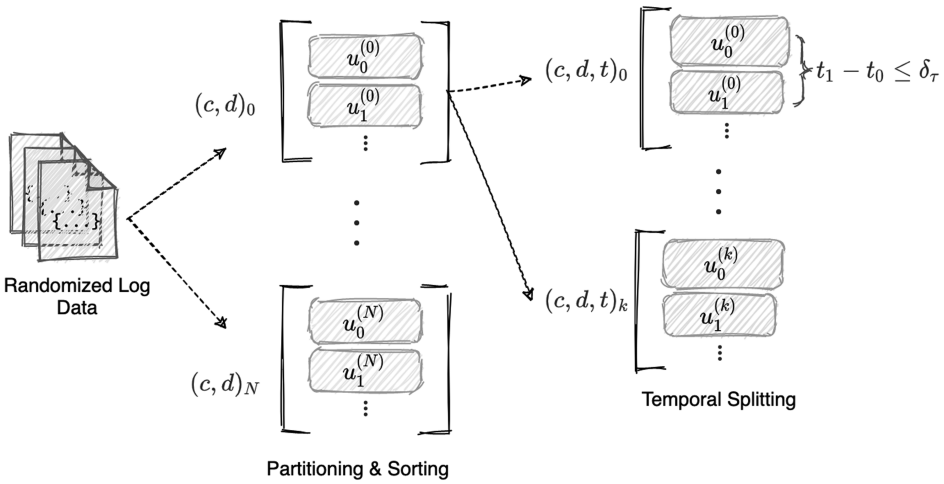

FIGURE 2 Data pipeline for constructing sessions

user's active dialog session. Thereafter, the intent and the corresponding action to execute is passed on to the TTS to generate the appropriate response as speech back to the user via their Alexa-enabled device, thus closing the interaction loop. Also note that the metadata and logs associated with each of the above systems are deidentified and stored asynchronously in an external data storage system, which thereafter is accessible offline.

In deploying our self-learning system online, we augment the original interaction loop by first intercepting the utterance being passed onto the NLU system and rewriting it with our reformulation engine. We then subsequently pass the rewrite in lieu of the original utterance back to NLU for interpretation, and thus restoring the original data flow. This is shown as the post-deployment data flow path in Figure 1. Our reformulation engine here essentially implements a rather lightweight service-oriented architecture that encapsulates the access to a high-performance, low-latency database, which is queried with the original utterance to yield its corresponding rewrite candidate. This along with the fact that the system is fundamentally stateless across users translates to a rather scalable customer-facing system with marginal impact to the overall perceived latency of their Alexa-enabled device.

Now, in order to discover new rewrite candidates and particularly to maintain the recency of the rewrites, the Markov-based model ingests the deidentified Alexa log data on a daily cadence to learn from users' reformulations and subsequently updates the aforementioned online database. This ingestion to update process takes place offline in entirety with the rewrites in the database updated via a low-maintenance feature-toggling (i.e., feature-flag) mechanism. Additionally, in verifying the viability of these rewrites, particularly in the context of improving customer experience, the system is also supplemented by a rewrite selection mechanism. Here, the deidentified log

data is leveraged to evaluate the rewrites from the Markov-based model by independently comparing their friction rate against that of the original utterance both generically via a proportion Z-test and contextually via generalized linear models (GLMs), and subsequently removing them from being uploaded to the database should they perform worse than their no-rewrite counterpart. This rewrite selection process, which allows us to maintain a rather high precision overall system at runtime, also takes place entirely offline and updates the database in similar fashion to that of the Markov-based model, albeit in a much more frequent cadence of every 4 h – defined by striking a balance between immediacy, data availability, and execution time.

In subsequent sections of this paper, we discuss the nature of our dataset, how the Markov-based model learns from the reformulations, and the functional aspects of our rewrite selection mechanism. It is also worth mentioning that the aforementioned friction rate is computed by aggregating across utterances, the result of a pre-trained neural model that leverages a user's utterance, the corresponding Alexa's response, and other contextual signals to detect friction for every user-Alexa interaction exchange. While the extended details of that model is beyond the scope of the paper, in the simplest sense, the model is based on fine-tuning a BERT (2018) model on a rather limited internally annotated dataset.

DATASET

As our objective is to learn the patterns from user interactions with Alexa, we pre-process 3 months of deidentified Alexa log data across millions of customers, which constitutes a highly randomized collection of time-series utterance data, to construct our dataset, $\mathcal{D}$ comprising of a set of sessions, S_i, that is:

$$\mathcal{D} = \{S_0, S_1, ...\} \tag{1}$$

Here, in motivating the definition of a session, it is worth walking through this pre-processing pipeline from a high-level perspective. Given the collection of randomized log, the data is first funneled through a partitioning and sorting stage where each grouping represents a finite ordered set of successive utterances, u for a unique customer-device pair, (c, d). Thereafter, each of these sets are split into smaller sub-sets at points where the time delay between any two consecutive utterances is at most δ_τ. These sub-sets are uniquely denoted by (c, d, t) where t represents the timestamp of the corresponding sub-set's first utterance. We also note that interjecting utterances, J, that is, those leading to StopIntent, CancelIntent, etc., that occur before the end of the aforementioned ordered sets are removed. These processing steps as a collective process is illustrated in Figure 2.

TABLE 1 Top seven domains spanning the dataset with their example utterances

Domain	Example Utterance
Global	What time is it
Music	Play watermelon sugar
HomeAutomation	Turn on the patio light
Notifications	Set a reminder for two p.m.
Knowledge	Who won the n.f. I.
Weather	Is it going to rain this week
Video	Play the new season of shameless

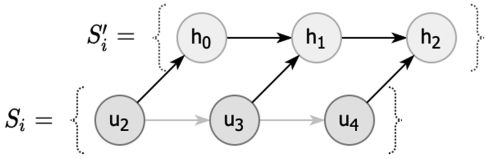

FIGURE 3 Latent session of interpretations

Then, intuitively speaking, any particular session, S_i is effectively a time-delimited snapshot of a user's conversation history with their Alexa-enabled device. We illustrate this in Figure 4A–C where each session is represented as a linear directed chain of successive utterances, for example, $u_2 \rightarrow u_3 \rightarrow u_4$. In this paper, we choose the value of $\delta_\tau = 45$ s as a result from a separate internal analysis.

Additionally, as a matter of principle, we note here that this dataset of sessions still remains true to the underlying natural distributions of the original log data. This is particularly so as the process of constructing the dataset is entirely devoid of any stratification or sampling strategies. To that end, given the anonymity of the sessions, we ensure that every user interaction with their Alexa device is entirely independent and weighted equally across the dataset, where from a linguistic context, it both structurally and semantically spans over a dozen different domains – the top 7 of which along with their examples are summarized in Table 1.

ABSORBING MARKOV CHAIN

In this section, we show how encoding user interaction history as paths in an absorbing Markov Chain model can be used to mine patterns for reformulating utterances. In particular, we discuss in detail the concept of the interpretation space, H, which functions as the vertex set of the model's transient states. We then elaborate on the construction of the absorbing states, R, the canonical solution

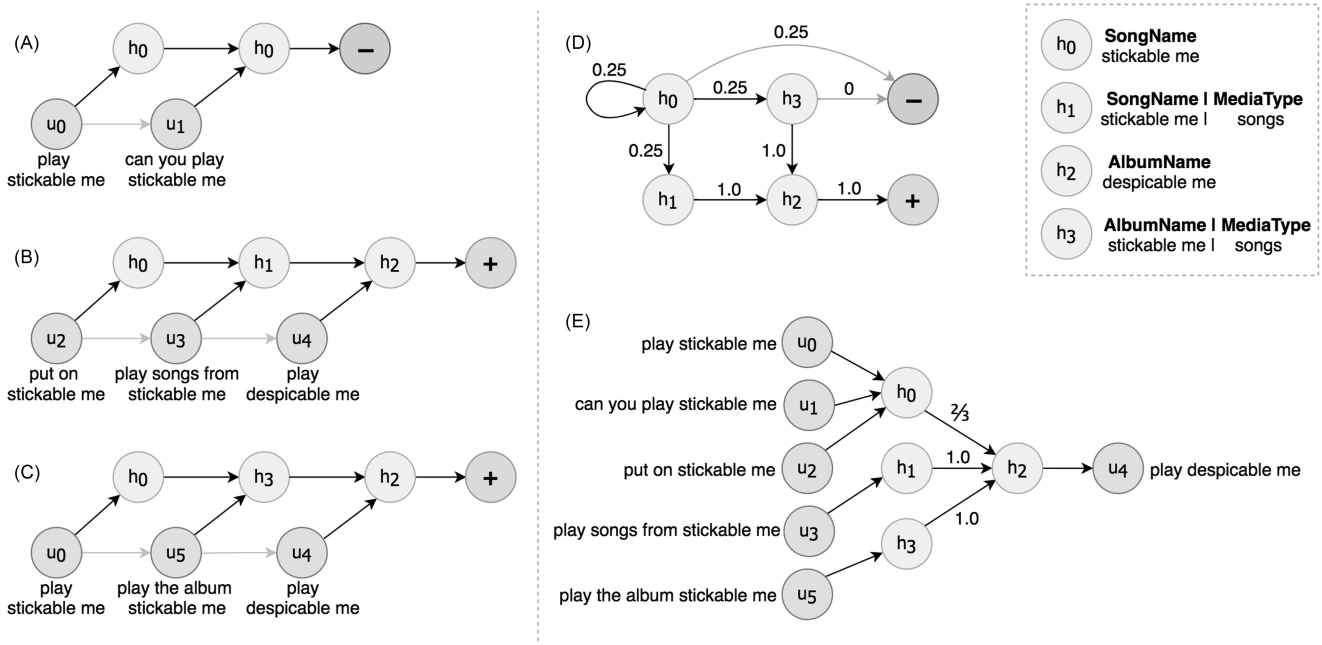

FIGURE 4 A visual representation of the Markov model constructed in the interpretation space, H, over three separate sessions, (a), (b), and (c), of users attempting to play the album "Despicable Me", and how solving for the path with the highest likelihood of success, (+), given by the darkened edged in (d), can allow for the defective utterances to be reformulated into a more successful query, as summarized in (e). Note that here, for demonstration purposes, we only show 3 interactions. However, in practice, we had a higher threshold for the minimum number of customers and interactions to have better estimates for the probabilities

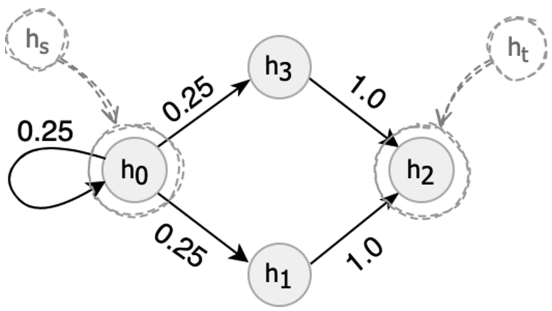

FIGURE 5 Sub-graph of the Markov model in Figure 4D

to the model, and the practical implementation of the model. As the Markov Chain model is inherently a probabilistic graphical model, we can represent it as graph, $G = (V, E)$, where the vertex set, V comprises of both the transient and absorbing states, that is, vertices in H and R, respectively, while the edge set, E comprises of vertex-pairs, (x, y) with x being any vertex in H and y being any vertex in V. We note that from here on out, we use the terms, Graph, and Markov model interchangeably.

Interpretation space

While our definition of a session above naturally extends toward having each ordered linear sequence of utterances as a path in our Markov model, this encoding in the utterance space, U, that is, the space of all utterances u, imposes a limitation on the model by creating heavily sparse connections. This is primarily due to the high degree of semantic and structural variance in U, which would ultimately result in a lower capacity for generalization.

To resolve this, we leverage the domain and intent classifier as well as the named entity recognition (NER) results from Alexa's NLU systems to surface structured representations of utterances, and thus encapsulate a latent distribution over U. Consequently, each utterance in a session is projected into this interpretation space, H which comprises the set of all interpretations h, to define a latent session, S_i':

where placed in context of Figure 4B is illustrated as above in Figure 3. To exemplify this, consider the utterance, "*play despicable me*" (i.e., u_4 in Figure 4), which would be mapped into the H-space as:

Music | PlayMusicIntent | AlbumName:***despicable me***

which is compactly represented as h_2 in Figure 4. As the H-space condenses the semantics of U, this mapping between U and H is inherently a many-to-one relationship. However, given the stochasticity of Alexa's NLU, the original projection itself is not entirely bijective and thus results in

a many-to-one relationship in both the forward and inverse mapping, that is, $U \rightarrow H$ and $H \rightarrow U$, akin to a bipartite mapping. This in turn, yields the conditional probability distributions, $P(H|U)$ and $P(U|H)$, such that for a particular $u \in U$ and $h \in H$, they are defined as follows:

$$P\ (h|u) = \frac{\#(u \rightarrow h)}{\#u} \quad P\ (u|h) = \frac{\#(u \rightarrow h)}{\#h} \tag{2}$$

where $\#(u \rightarrow h)$ represents the total number of times both u and h are mapped onto each other while $\#u$ and $\#h$ are the total occurrences of u and h in the entire dataset, $\mathcal{D}$, respectively.

Transient states

Given our transformed dataset, $\mathcal{D}'$ of latent sessions S', we take each such session and the interpretations within it to represent paths and transient states respectively in our Markov model, such that each successive pair of interpretations would represent an edge in the graph. Then, the corresponding probability that a transition state $h_i \in H$ transitions to $h_j \in H$ in the graph is given by:

$$P\ \left(h_j|h_i\right) = \frac{\#\left(h_i \rightarrow h_j\right)}{\#h_i} \tag{3}$$

where $\#(h_i \rightarrow h_j)$ and $\#h_i$ represent the total number of times h_j is adjacent, that is, directly linked to h_i and the total occurrence of h_i respectively, aggregated across all sessions (i.e., *over 3 months and millions of customers*) in $\mathcal{D}'$.

Taking this in context of Figure 4, consider the transition probability $P(h_1|h_0)$. From the sessions (a), (b), and (c), we can note that the transition state h_0 is adjacent to the states, $\{h_0, h_1, h_3, (-)\}$ with each of them having a co-occurrence count of 1 with h_0. Here, $(-)$ refers to the failure absorbing state (*defined in the following sub-section*). As such, the probability $P\ (h_1|h_0) = \frac{1}{4} = 0.25$ as shown in Figure 4D.

Absorbing states

In formulating the definition of the absorbing states of the Markov model, we look toward encoding the notion of interpreted defects as perceived by the user. As we have briefly introduced earlier, this concept of defect surfaces in two key forms, that is, via *explicit* and *implicit* feedback.

Here, *explicit feedback* refers to the type of corrective or reinforcing feedback received from direct user engagement. This primarily includes events where users opt to interrupt Alexa by means of an interjecting utterance (*as defined above in Dataset*). This is illustrated in the example below:

User:	*"play a lever"*
Alexa:	*"Here's Lever by The Mavis's, starting now."*
User:	*"stop"*

In contrast, *implicit feedback* is typically observed when users abandon a session following Alexa's failure to handle a request either due to an internal exception or simply unable to find a match for the entities resolved. Case in point:

User:	*"play maj and dragons"*
Alexa:	*"Sorry, I can't find the artist maj and dragons."*

Given this, we define two absorbing states: *failure* (r^-), and *success* (r^+), where *success* is defined as the absence of *failure*. These states are artificially injected to the end of *all* sessions, based on the implicit and explicit feedback we infer from Alexa's response, and user's last utterance.

To clarify this, let us walk through the examples above assuming that they are the last utterances of their corresponding sessions. In the first example, we would drop the "*stop*" turn, and add a *failure* state. In the second example, we simply add the *failure* state to the end of the session. Finally, in the absence of an explicit or implicit feedback, we add a *success* state to the end of the session. Given this, we can then define the probability that a given transient state, h_i is absorbed in much the same way as in Equation 8, for example:

$$P\ \left(r^+|h_i\right) = \frac{\#\left(h_i \rightarrow r^+\right)}{\#h_i} \tag{4}$$

Note that in Figure 4, we refer to the *failure* (r^-), and *success* (r^+) states as $(-)$ and $(+)$, respectively.

Markov model

With the distributions over both the transition and absorbing states defined above, recall that the interpretation space, H is the set of all transient states in the graph. Then, we can summarize the Markov model in its canonical form via the transition matrix, $\mathbf{A}$ as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{Q} & \mathbf{R} \\ 0 & \mathbf{I}_2 \end{bmatrix} \tag{5}$$

where the sub-matrix $\mathbf{Q}$ encompasses the transient probabilities between all states in the interpretation space H, while the sub-matrix $\mathbf{R}$ encapsulates the absorption probabilities of every state in H going to the two absorbing states, r^+ and r^-. Additionally, the sub-matrices 0 and $\mathbf{I}_2$ both

represent the sub-matrix of zeros and the 2×2 identity matrix, respectively.

Now, we generalize the previous notation of probabilities as $P^{(n)}$, that is, the probability at depth-n of the graph, with P implicitly referring to $P^{(1)}$. Note that in the context of transition probabilities, this generalization readily extends to the matrix form where the probability of transitioning from some transient state h_i to some transient state h_j after exactly n steps, that is, $P^{(n)}(h_j|h_i)$ is thus given by the (i, j)-th element of the sub-matrix $\mathbf{Q}^n$ (i.e., $\mathbf{Q}$ *multiplied itselfn times*) and similarly as before with $\mathbf{Q}_{i,j}$ implicitly referring to $P(h_j|h_i)$. Additionally, let h_s and h_t be any given source and target transient states in the graph, respectively. Then, in motivating the optimization objective, consider foremost the Markov model in Figure 4D and take both h_s and h_t to be h_0 and h_2, respectively. The sub-graph in question can be illustrated in Figure 5.

At this point, we can observe the following key facts about the sub-graph above: (1) as there is at least 1 path between h_s and h_t, the latter is considered to be *reachable* by the former; (2) both paths linking h_s and h_t are separated by 1 other transient state with no direct linkage resulting in 2 traversal steps being required, for example, $h_0 \rightarrow h_1 \rightarrow h_2$; and (3) the self-edge on h_0 further extends the original depth-2 path into infinitely many possible path depths by repeatedly circling around h_0, for example, $h_0 \rightarrow \ldots \rightarrow h_0 \rightarrow h_1 \rightarrow h_2$.

Here, while the point (2) above indicates that $\mathbf{Q}^2_{s,t} > 0$, it is far from sufficient from summarizing the total probability of transitioning from h_s to h_t. In fact, the point (3) above cements that h_t can also be reached by h_s with traversals of depths 3, 4, and so on, which inevitably leads to the total probability being the (s, t)-th element of the infinite compound sum of $\mathbf{Q}^2 + \mathbf{Q}^3 + \cdots +$. We can then generalize this across all possible source and transient states by also including $\mathbf{Q}^0$ and $\mathbf{Q}$ to account for all possible depths. Now, given that $\mathbf{Q}$ is a convergent square matrix of probabilities such that $\| \mathbf{Q} \| < 1$, the aforementioned summation then leads to a geometric series of matrices, which as given by Definition 11.3 in (1997), corresponds to the fundamental matrix of the Markov model, denoted by $\mathbf{N}$:

$$\mathbf{N} = \mathbf{Q}^0 + \mathbf{Q} + \mathbf{Q}^2 + \cdots = \left(\mathbf{I}_{|H|} - \mathbf{Q}\right)^{-1} \tag{6}$$

where $\mathbf{I}_{|H|}$ refers to the identity matrix which is equi-dimensional with $\mathbf{Q}$. Given this, we can thus define the probability of *success* after traversing from any given h_s to any given h_t such that h_t is *reachable* by h_s as follows:

$$\boldsymbol{\pi}_\infty(h_s \rightarrow h_t) = P\left(r^+|h_t\right) \cdot \mathbf{N}_{s,t} \tag{7}$$

As such, in the context of reducing defects, we consider h_t to be a possible reformulation candidate for h_s if it is *reachable* by h_s, such that conditioned on h_s, h_t has a higher chance of *success* than h_s on its own, that is:

$$\boldsymbol{\pi}_\infty(h_s \rightarrow h_t) > P\left(r^+|h_s\right) \tag{8}$$

We then frame our objective as identifying the h_t which maximizes the aforementioned probability for the given h_s:

$$h_t^* = \arg\max_{h_t} \boldsymbol{\pi}_\infty(h_s \rightarrow h_t) \tag{9}$$

Intuitively speaking, in the event that $h_t^* \neq h_s$, the model shows that there exists a *reachable* target interpretation that when reformulated from h_s, has a better chance at a *successful* experience than not doing so. In reference to Figure 4E, we can see that reformulating h_0 to $h_t^* = h_2$ increases the likelihood of success as:

$$\boldsymbol{\pi}_\infty(h_0 \rightarrow h_2) = \frac{2}{3} > P\left(r^+|h_0\right) = 0 \tag{10}$$

Suppose that $h_t^* = h_s$. In which case, the source interpretation is already *successful* on its own and hence requires no reformulation. As such, the model is effectively able to automatically partition the vertex space, H into sets of *successful* (H^+) and *unsuccessful* (H^-) interpretations. In extending this reformulation back to the utterance space, U, we leverage the distributions $P(U|H)$ and $P(H|U)$ defined in Equation 5 and re-define our objective as follows for a given source utterance $u_s \in U$:

$$u_t^* = \arg\max_{u_t} \sum_{h_s, h_t} P(u_t|h_t) \cdot \boldsymbol{\pi}_\infty(h_s \rightarrow h_t) \cdot P(h_s|u_s) \tag{11}$$

The intuition described above can similarly be applied here where u_t^* is the more *successful* reformulation of u_s. Note that the self-partitioning feature of the model directly extends to the utterance space, U, allowing it to both surgically and actively target only utterances that are likely to be defective and surface their corresponding rewrite candidates. This is the cardinal aspect of the model that drives the *self-learning* nature of the proposed system without requiring any human in the loop.

Implementation

With $|H| \sim 10^6$, constructing the matrix $\mathbf{Q}$, let alone inverting it, poses a key challenge towards scaling out the model, particularly in its batched form. As such, we formulate an approximation in computing the probability $\boldsymbol{\pi}_\infty(h_s \rightarrow h_t)$ for all source interpretations, h_s by means of a distributed approach.

We note that from our dataset, D', that in the event that a given source utterance, u_s is defective, users would only attempt at reformulating their query a few times before either arriving at a satisfactory experience or abandoning their session entirely. This translates to most ($\sim$ 97.3 percent) source interpretations, h_s in the Markov model having short path lengths (i.e., typically ≤ 5) prior to them being absorbed by an absorbing state. Consequently, this along with the fact that these reformulations are recurrent across users, most high-confidence reformulations often only involve visiting a relatively much smaller set of target interpretations, h_t when compared to the cardinality of the interpretation space, $|H|$ as a whole.

This leads us to deduce that the matrix $\mathbf{Q}$ is highly sparse and the corresponding graph contains many clustered (i.e., *community*) structures. We then leverage these facts to first collect the paths for every source interpretation, h_s in a series of map-reduce tasks, by means of a distributed breadth-first search traversal up to a fixed depth of 5 using Apache Spark (2016). Thereafter, each task receives the paths corresponding to a single h_s and in turn uses them to construct an approximate transition matrix. As the dimensionality of this matrix is much lower than that of $\mathbf{A}$, we can easily compute approximates for both the fundamental matrix and the probabilities $\boldsymbol{\pi}_{\infty}(h_s \rightarrow h_t)$ for the *reachable* target states within the same task. As a result, we have a distributed solution for parallelizing the computation of the optimization problem in Eqn. [optim] for every $h \in H$.

The breadth-first search traversal, which involves a series of sort-merge joins, does indeed introduce an algorithmic overhead of $\mathcal{O}(d \cdot |E| + |E|\log|E|)$, where d and E refer to the depth of the traversal and the set of all edges in the graph, respectively. We do also note that as this is a distributed join, the incurred network cost due to data shuffles are omitted here for simplicity. That being said, these overheads are offset by the advantage of being able to scale out the model. For purposes of optimization, each successive join is only performed on the set of paths which are non-cyclic and have yet to be absorbed while paths with vanishing probabilities are pruned off.

EXPERIMENTS

Baseline: Pointer-generator sequence-to-sequence model

Sequence-to-sequence (seq2seq) architectures have been the foundation for many neural machine translation and sequence learning tasks (2014). As such, by formulating the task of query rewriting as an extension of sequence learning, we used a long short-term memory-based (LSTM) model as an alternative method to produce rewrites. In short, we first mined 3 months of rephrase data using a rephrase detection ML model such that the first utterance was defective, and the rephrase was successful. We then used this data to train the pointer-generator model, such that given the first utterance, it produces the second utterance. The model is based on well-established encoder-decoder architecture with attention and copy mechanisms (2017). After the model is trained, we then used it to rewrite the same utterances that the graph rewrites.

Offline analysis

In order to evaluate the quality of the rewrites we obtained, we annotated 5679 unique utterance-rewrite pairs generated via the graph, and estimated the accuracy and win–loss ratio to be 93.4 percent and 12.0, respectively. The notion of win–loss ratio here is defined as the ratio of rewrites that result in better customer experience and the rewrites that deteriorate customer experience. We further leveraged the pointer-generator model to generate rewrites for these utterances as a baseline.

Applying the pointer-generator model on this dataset resulted in accuracy of 55.2 percent, that is, significantly lower than the accuracy of graph. This is expected, since the graph (1) aggregates all 3 months of data (and not limited to merely rephrases), (2) takes into account the frequency of transitions whereas the pointer-generator model only has unique rephrase pairs for training, and (3) utilizes the interpretation space to further compact and aggregate the utterances.

In contrast, the pointer-generator model has instead the benefit of a higher recall (*since it can potentially rewrite any utterance*), and it learns the patterns, for example, SongName → play SongName. Another important difference between the graph and the pointer-generator method is that the graph is capable of identifying when an utterance is *successful* via its interpretation, that is, when $h_t^* = h_s$ and thus maximize its precision. This is a signal to *not* rewrite the utterance, since statistically speaking, rewriting could only potentially worsen its likelihood of success. However, the pointer-generator model lacks this capability, and it may rewrite an otherwise successful utterance, which thereafter would cause a friction.

Table 2 shows some examples of good and bad rewrites from the graph. It is clear from the examples that the rewrites are capable of fixing ASR (no. 1–3), NLU (no. 4–7), and even user errors (no. 8). On the other hand, there are cases where the rewrites tend to fail (no. 9–10). One of the recurring cases of failure is when an utterance is rewritten to a generic utterance, like "*play*," or "*shuffle my songs*." This usually happens due to the original utterance not being *successful*, and the users trying many different

TABLE 2 Some example rewrites from the graph

No.	Original utterance	Rewrite	Label
1	Play maj and dragons	Play imagine dragons	Good
2	Play shadow by lady gaga	Play shallow by lady gaga	Good
3	Play rumer	Play rumor by lee brice	Good
4	Play sirius x. m. chill	Play channel fifty three on sirius x. m.	Good
5	Play a. b. c.	Play the alphabet song	Good
6	Don't ever play that song again	Thumbs down this song	Good
7	Turn the volume to half	Volume five	Good
8	Play island ninety point five	Play island ninety eight point five	Good
9	Play swaggy playlist	Shuffle my songs	Bad
10	Play carter five by lil wayne	Play carter four by lil wayne	Bad

paths that eventually lose information, and is consequently aggregated in a generic utterance (due to Eq. [sumProbabilities]). Another common case of failure is when the rewrite changes the intention of the original utterance by changing the song name or artist name. This happens because of various reasons. For example, the data that we use for building the graph may contain a period of time where the original utterance was not usually *successful*, so the users changed their mind by asking to play another similar song (like no. 10).

While the first type of error is easy to correct, by either applying rules or building a learning-based ranker after the graph generation, the second type, however, is rather tricky to detect, since more often than not, changes in the interpretation tend to help. Here, we critically rely on the rewrite selection mechanism to remove these forms of rewrites from the production system.

REWRITE SELECTION

Despite the graph's ability to surgically target only utterances which are potentially defective, it certainly is not devoid of false positives and may very well over-trigger by surfacing rewrites that worsen user experiences as shown in Table 2. In fact, extending from the previous section, while high path entropy leading to overly generalized utterances is a cause of concern, the greater requisite for a rewrite selection mechanism stems from the latter issue of intent changes in the proposed rewrite, which is largely prompted by the unavailability of more recent content requested by the original utterance. Here, the rewrite offers but only a temporary solution to address user demand. However, when the newer content becomes available, a system more reactive than the graph is imperative in suppressing this. Additionally, the collaborative filtering nature of the graph translates to surfacing rewrite paths that while may improve the experience for most users, may very well worsen for those who fall short of being in the majority preference pool. These factors suffice in giving rise to our rewrite selection system centered on the premise of comparatively evaluating the friction rates between the rewrites and their original counterparts, and thus compensates for the graph's limitations by (1) reactively supplanting graph's local Markov property in assessing the viability of a rewrite for an utterance post-launch, (2) leveraging more granular contextual and feedback signals as afforded by the pre-trained neural defect predictor than the coarser definitions used in the absorbing states, and (3) potentially feeding back the actions of rewrite not being selected as implicit failure nodes allowing for a better re-routing of paths in the graph.

Generalized selection

Given the original utterance, the selection system can decide to rewrite it to one or more candidate utterances, or not to rewrite at all if none of the candidates reduces frictions.

In its rather simplest form, the generalized rewrite selection both posits and attempts to answer the question of whether a rewrite for a given utterance would significantly worsen user experience when compared to that of the original utterance itself, solely by virtue of their respective friction rates. This lends itself to the notion of evaluating on the basis of hypothesis testing, and more specifically a proportion Z-test between the friction distributions of the rewrite and no-rewrite cases. To drive this home, consider the case where we rewrite, "*play walk hard by dewey cox*" → "*play walk hard*" by excising the artist name for the requested song. Note that the rewrite is actively altering a portion of original interpretation as follows:

$$\text{ArtistName}:\textbf{dewey cox}|\text{SongName}:\text{walk hard}$$
$$\downarrow$$
$$\text{SongName}:\text{walk hard}$$

Now, let μ_0 and μ_1 be the friction rates of the original and rewrite, respectively. Then, we subsequently construct the one-sided proportion Z-test to question if the rewrite is indeed worsening the experience by defining both the null and alternative hypotheses such that:

$$\begin{aligned} H_0 &: \mu_0 = \mu_1 \\ H_1 &: \mu_0 < \mu_1 \end{aligned} \tag{12}$$

Leveraging actual post-launch data, we observed that the original resulted in about six friction experiences out of eight, that is, an effective friction rate of 75 percent while the rewrite had about eight friction experiences out of 24, that is, an effective friction rate of 33 percent. These in combination yields a P-value of 0.98 and with a significance level of 0.01 suggest that the inverse alternative hypothesis, that is, where $\mu_0 > \mu_1$ is statistically significant and is thereafter sufficient to reject the null hypothesis – thus indicating that the rewrite actually improves overall user experience. Now, extending this back to the example no. 10 in Table 2, where we rewrite "*play carter five by lil wayne*" → "*play carter four by lil wayne*," we note that while initially the content unavailability of the original utterance would push the Z-test in favor of the rewrite, this is only temporary as the Z-test would eventually favor preserving the original as soon as the more recent content becomes available and thus driving down the friction rates of the original. This reactive behavior is critical to ensure customer satisfaction at optimal rates.

Contextualized selection

Despite the simplicity of the generalized selection mechanism, its reactivity is very much limited by the volume of observations for the given hypothesis tests and this is especially so for rewrites with rather low accrual rates where a far longer period would be requisite toward achieving similar statistical significance for rewrite selection. This inevitably necessitates a supplementary system to bridge this reactivity gap. Now, consider the two proposed rewrites for a similar original utterance, that is, "play a. b. c." → "play the song a. b. c." and "play a. b. c." → "play the a. b. c. song." Here, while the original utterance and the former rewrite has an effective friction rates of 79 percent (given approximately 100 observations) and 32.3 percent (given approximately 3000 observations), respectively, the latter rewrite appeared to have only occurred once and without friction. Evidently, a single friction-free observation of the latter rewrite would be insufficient to perform any meaningful hypothesis testing as in the generalized setting. However, given the fact that both the rewrites here share a similar resolved interpretation as follows:

$$MediaType{:}song|SongName{:}the\ alphabet\ song$$

It does indeed beckon the question of whether the friction rate and by extension the overall rewrite viability of the latter rewrite could possibly be inferred from those of the former. In fact, our contextualized rewrite selection (CRS) proposes to answer this question by means of featurizing the rewrites so as to leverage and transfer their content information across rewrites, thus scaling out the rewrite selection impact regardless of the reactivity of the generalized setting.

Friction rate prediction

Motivated by the requisite for bridging the reactivity gap and generalizing friction distributions across utterances in traffic, the CRS aims to infer the friction rates for rewrites in this contextualized setting by ultimately establishing two conditional distributions, $P(Y|u)$ and $P(Y|u_s, u_t)$ where Y corresponds to a binary response variable denoting the friction indicator while u, u_s, and u_t each correspond to any valid utterance, the source utterance, and the target utterance, that is, the rewrite, respectively. Now, in learning these conditional distributions, the CRS abstracts the input space leveraging a feature extractor, $\phi : u \to X_u$ to contextualize the inputs, that is, utterances and thereafter employs logistic regression classifiers in constructing the following predictive models:

$$\begin{aligned} P(Y|u) \quad &\sim P(Y|X_u) \\ P(Y|u_s, u_t) \quad &\sim P(Y|X_s, X_t) \end{aligned} \tag{13}$$

Here, the feature space for each utterance includes (but is not limited to) the domain, intent, NER, and entity resolution values. Extending this formulation back to the original example, we observe that both the rewrites, "*play the song a. b. c.*" and "*play the a. b. c. song*" for the original utterance, "*play a. b. c.*" would fundamentally have the same feature spaces, X_s and X_t purely on the basis of having similar resolved interpretations. As such, the predicted friction rates would also inevitably be the same and thus allowing the model to generalize its learning across utterances.

As a matter of performance, these two predictive models achieve an ROC-AUC of 66 percent and 62 percent, respectively within the confines of our testing dataset comprising of millions of historical utterances built upon the aforementioned Alexa log data. In addition to the predicted friction binaries, the standard errors based on the asymptotics of generalized linear methods (GLMs) (1985) and the delta

TABLE 3 *Z*-test on predicted friction rates

	Friction rate (SE)	*p*-value	Decision
Original utterance: play a. b. c.	78.5% (1.26%)		
Rewrite candidate: play the a. b. c. song	32.8% (0.73%)	1.0	Better
Rewrite candidate: play the alphabet song	34.1% (6.27%)	0.999	Better
Original utterance: play big shrimp	15.3% (1.49%)		
Rewrite candidate: play big shrimp by flatbush zombies	17.1% (6.87%)	0.597	Tie
Original utterance: play happier by d. j. marshmello	30.9% (9.43%)		
Rewrite candidate: play happier	60.5% (8.14%)	0.009	Worse

method (1935) are computed across the dataset to capture relevant uncertainties.

In the interest of better evaluating the functional performance of the models in selecting rewrites, we annotated 152 random samples of rewrites that were decidedly chosen to be removed by the models, and we observed that 113 of them are veritable, leading toward an effective precision of 74.3 percent. Additionally, we estimated the overall percentage of bad distinct rewrites in our traffic to be approximately 6.4 percent, based on a separate annotation of 921 random rewrites. Now, given that approximately 12.9 percent of them are reactively removed, we conclude that this effectively translates to the recall of our rewrite selection system. It is worth mentioning however that these number would be further inflated if they were otherwise re-weighted by traffic.

Decision modeling

With the predictive models being able to generalize friction distributions across utterances, we turn to leveraging them in modeling the decision boundaries for rewrite selection. In particular, consider the examples in Table 3 where the rewrites:

"play a. b. c." → "play the a. b. c. song"

"play a. b. c." → "play the alphabet song"

both significantly reduce the friction rate from 78.5 percent (with a standard error, SE of 1.26 percent) to 32.8 percent (0.73 percent) and 34.1 percent (6.27 percent), respectively, and thus evidently improving the overall user experience. In contrast, it is far from arguable that the rewrite "*play happier by d. j. marshmello*" → "*play happier*" has significantly worsened the friction rate, that is, from 30.9 percent (9.43 percent) to 60.5 percent (8.14 percent). However, not all rewrites have such marked decision boundaries for strict selection. In the rewrite, "*play big shrimp*" → "*play big shrimp by flatbush zombies*," with 95 percent confidence intervals of their predictions heavily overlapping with each other, that is, 15.3 percent $\pm$ 1.96$*$1.49 percent and 17.1 percent $\pm$ 1.96$*$6.87 percent, the decision was restricted to a mere tie. Extending from this, we look toward two separate decision making strategies, namely a conservative setting and a more explorative approach.

Conservative: The goal here is to be more risk-averse so as to ensure safety of the system, whereby explorations are disfavored and decisions are solely based on significant insurmountable evidence. While still leveraging the notion of *Z*-tests, we directly utilize the predicted friction rates and their corresponding standard errors in lieu of inferring the mean and standard error from empirical observed friction rates. Applying a significance level of $\alpha = 0.05$, in these four examples, we will only remove the rewrite "*play happier by d. j. marshmello*" → "*play happier*" and preserve the viability of the other rewrites in production traffic.

Explorative: The goal here instead would be to optimize the long-term performance by aggressively exploring rewrites until such a time significant confidence has been accrued to reach a decision. Suppose that the initial predicted friction rate is not too high, but the SE is rather sufficiently high as is the case with the rewrite, "*play big shrimp*" → "*play big shrimp by flatbush zombies*" where the friction rate is 17.1 percent, but with a rather high SE 6.87 percent. This means its true friction rate could fluctuate between lower (down to ~10 percent if we use 1 SE as the fluctuation) and higher values. Now, should an increase in observation volume lower its standard error, we would then inevitably conclude that it as a good rewrite if the new predicted friction rate is closer to the tune of 13 percent (1 percent), and a bad rewrite if instead it is closer to 18 percent (1 percent). Even if the rewrite turns out to be bad after our active explorations, the data is still not wasted, as the investment is only short-term and the data for a particular rewrite is also shared in model learning by other rewrites with similar features.

This kind of reward-driven exploration ideas has been well researched in multi-arm bandit algorithms. As a matter of reference, LinUCB (2010) and derived applications using penalized logistic regression as mentioned in (2011). Here, UCB stands for upper confidence bound. Our example of using 17.1 percent − 6.87 percent for the possible true friction rate of "*play big shrimp*" → "*play big shrimp by flatbush zombies*" is similar to UCB ideas. Strictly applying

UCB ideas in our example, we make the decision based on the lower bound of predicted friction rate, that is, rewrite 17.1 percent − a ∗6.87 percent, as opposite to original utterance 15.3 percent − a ∗ 1.49 percent. Here a is a technical parameter related to sample size and confidence. Additionally, from a Thompson Sampling (1933) and related applications for example (2010), and (2017) perspective, one can consider having three arms for the example "*play a. b. c.*" in Table 3 – no-rewrite, rewrite "*play the a. b. c. song*," or rewrite "*play the alphabet song*." The idea would be along this line: every time we see an utterance traffic with "*play a. b. c.*," we draw one random sample from each arm's predicted defect distribution approximated by Gaussian distribution with estimated mean and SE, N(78.5 percent,1.26 percent), N(32.8 percent, 0.73 percent) and N(34.1 percent, 6.27 percent). Then we select the arm with the lowest sampled friction rate as our decision for that particular traffic.

APPLICATION DEPLOYMENT

Offline rewrite mining

Since there are thousands of new utterances per day, and there are constant changes to the upstream and downstream systems in Alexa on a daily basis, it is important to update our rewrites on a regular basis to remove stale and ineffective rewrites. We run daily jobs to mine the most recent rewrites in an offline fashion. This allows us to find the most recent rewrites and serve them to users. It is noteworthy that in case of conflicts between the rewrites, we pick the most recent rewrite, since it has the latest data. We have online alarms and metrics to monitor daily jobs, since sometimes changes to the upstream and downstream Alexa components can impact our rewrite mining algorithm. In case of large changes in our metrics, we do a dive deep into the data to find the root cause.

Online service

Since the graph is static during the period it is used, and there are many repetitive utterances per day, we opted to mine the rewrites as key-value pairs, where the original utterance is the key, and the rewrite is the value. For example, we store "*play babe shark*" → "*play baby shark*" as one entry. We then serve these pairs in a high-performance database to meet the low latency requirement. This allows us to decouple the offline mining process and the online serving process for high availability and low latency requirements.

Rewrite selection system

Due to the quick change of contents in Alexa traffic as well as its evolving ASR & NLU systems, graph alone is not quick enough to adapt to all changes. The guardrail rewrite selection system runs its algorithms every 4 h, refresh and store selection actions for all rewrite pairs in another high-performance database that is coupled with graph online service.

Online performance

Following the offline analysis and traffic simulations, we launched the graph rewrites in production in an A/B testing setup. We monitored the performance of our rewrites against no-rewrites for over 2 weeks, and we observed more than 30 percent average reduction in defect rate (p − value < 0.001), helping millions of users. Here, the notion of defect is based on an ML model which scores user dissatisfaction at every turn. In a separate 9 week randomized control trial, we also noted as defect decreased, a new dialog interaction was created for every two corrections made by the system (p − value < 0.01), which in turn translates to greater user engagement. We further measured the win–loss ratio 3 months after the system's release by tallying the number of unique rewrites where rewriting significantly improved i.e. wins – or worsened i.e. losses over the no-rewrite option (we used Z-test to test the significance, and set p-value threshold of 0.01). The post-launch win–loss ratio closely matched our offline estimate (11.8 online vs. 12.0 offline).

We have been running this application for over 15 months in production, and it has been serving millions of users, since improving their experience on a daily basis without getting in their way. We know this for a fact since we have been monitoring customer satisfaction metrics on a weekly basis. We monitor the total number of rewrites, and the average friction rate for the rewrites, along with average friction for no-rewrites, where for the latter two, the aforementioned 30 percent margin still prevails. On top of tracking online metrics, we continue doing offline evaluations on a weekly basis, where we sample our traffic, and send it for annotation. Combining the online and offline metrics in a longitudinal fashion allows us to closely follow the changes in the customer experience, which is the ultimate metric for our system.

CONCLUSION

As conversational agents become more popular and grow into new scopes, it is critical for these systems to have

self-learning mechanisms to fix the recurring issues continuously with minimal human intervention. In this paper, we presented a self-learning system that is able to efficiently target and rectify both systemic and customer errors at runtime by means of query reformulation. In particular, we proposed a highly scalable collaborative-filtering mechanism based on an absorbing Markov chain to surface *successful* utterance reformulations in conversational AI agents. Our system achieves a high precision performance thanks to aggregating large amounts of cross-user data in an offline fashion, without adversely impacting users' perceived latency by serving the rewrites in a look-up manner online. This coupled with our rewrite selection mechanism which reactively evaluates the viability of rewrites at a greater cadence has helped maintain the aforementioned high precision at runtime. We have tested and deployed our system into production across millions of users, reducing customer frictions by more than 30 percent and achieving a win–loss ratio of 11.8. Our solution has been customer-facing for over 36 months now, and it has helped millions of users to have a more seamless experience with Alexa.

ACKNOWLEDGEMENTS

The authors would very much like to thank Steven Wasik and his team for their analysis into customer engagement and Jin Hock Ong and his team, particularly Karen Stabile and Vincent Ly for their continued engineering support in maintaining and monitoring the system as a whole.

REFERENCES

Chapelle, O., and L. Li. 2011. "An empirical evaluation of thompson sampling." In Advances in Neural Information Processing Systems, NIPS'11, 2249–57. Granada, Spain: Curran Associates, Inc.

Devlin, J., M. Chang, K. Lee, and K. Toutanova. 2018. "BERT: Pre-training of deep bidirectional transformers for language understanding." CoRR abs/1810.04805.

Doob, J. L. 1935. "The limiting distributions of certain statistics." *The Annals of Mathematical Statistics* 6(3): 160–9.

Fahrmeir, L., and H. Kaufmann. 1985. "Consistency and asymptotic normality of the maximum likelihood estimator in gener- alized linear models." *The Annals of Statistics* 13: 342–68.

Fouss, F., S. Faulkner, M. Kolp, A. Pirotte, and M. Saerens 2005. "Web recommendation system based on a markov-chain model." In Proceedings of the Seventh International Conference on Enterprise Information Systems, 56–63. Miami, USA: SciTePress.

Gao, J., M. Galley, and L. Li. 2018. "Neural approaches to conversational AI." *CoRR* abs/1809.08267.

Graepel, T., J. Q. Candela, T. Borchert, and R. Herbrich. 2010. "Web-scale bayesian click-through rate prediction for spon- sored search advertising in microsoft's bing search engine."' In Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10, 13–20. Madison, WI, USA: Omnipress.

Grinstead, C. M., and J. L. Snell. 1997. *Introduction to Probability*. American Mathematical Society.

Hill, D. N., H. Nassif, Y. Liu, A. Iyer, and S. Vishwanathan. 2017. "An efficient bandit algorithm for realtime multivariate optimization." In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1813–21. Halifax, NS, Canada: Association for Computing Machinery.

Jansen, B. J., D. L. Booth, and A. Spink. 2005. "Patterns of query reformulation during web searching." *Journal of the American Society for Information Science and Technology* 60(7): 1358–71.

Khorasani, E. S., Z. Zhenge, and J. Champaign. 2016. "A markov chain collaborative filtering model for course enrollment recommendations." In IEEE International Conference on Big Data, 3484-3490. Washington, DC, USA: IEEE.

Li, L., W. Chu, J. Langford, and R. E. Schapire. 2010. "A contextual-bandit approach to personalized news article recommen- dation." In Proceedings of the 19th International Conference on World Wide Web, 661–70. Raleigh, North Carolina, USA: Association for Computing Machinery.

Mohri, M., F. Pereira, and M. Riley. 2002. "Weighted finite-state transducers in speech recognition." *Computer Speech & Language* 16(1): 69–88.

Sahoo, N., P. V. Singh, and T. Mukhopadhyay. 2012. "A hidden Markov model for collaborative filtering." *MIS Quarterly* 36: 1329–56.

Sarikaya, R. 2017. "The technology behind personal digital assistants: An overview of the system architecture and key compo- nents." *IEEE Signal Processing Magazine* 34(1): 67–81.

See, A., P. J. Liu, and C. D. Manning. 2017. "Get to the point: Summarization with pointer-generator networks." *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* 1: 1073–83.

Sutskever, I., O. Vinyals, and Q. V. Le. 2014. "Sequence to sequence learning with neural networks." In Proceedings of the 27th International Conference on Neural Information Processing Systems, 3104–3112. Montreal, Canada: MIT Press.

Terveen, L., and W. Hill. 2001. "Beyond recommender systems: Helping people help each other." In *The New Millennium, Jack Carroll*. New York, N.Y., USA.

Thompson, W. R. 1933. "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples." *Biometrika* 25(3/4): 285–94.

Wang, J., J. Z. Huang, and D. Wu. 2015. "Recommending high utility queries via query-reformulating graph." *Mathematical Problems in Engineering*, 2015: 1–14.

Zaharia, M., R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, et al. 2016. "Apache spark: A unified engine for big data processing." *Communications of the ACM* 59(11): 56–65.

Zhu, X., J. Guo, X. Cheng, and Y. Lan. 2012. "More than relevance: High utility query recommendation by mining users' search behaviors." In CIKM'12, October 29–November 2, Maui, HI, USA.

AUTHOR BIOGRAPHIES

Pragaash Ponnusamy is an applied scientist at Amazon Alexa AI. He received his B.S. degree in electrical engineering and computer science from the University of California, Berkeley in 2016.

Alireza Roshan Ghias is an applied science manager at Amazon. He received his B.S degree from the University of Tehran in 2004 and his M.S. degree from Sharif University of Technology in 2006, both in Mechanical Engineering. In 2012, he received his Ph.D. degree from Ecole Polytechnique Feťdeťrale de Lausanne in Biomedical Engineering.

Yi Yi is a senior applied scientist at Amazon. He received his B.S degree in Mathematics and Applied Mathematics from Shanghai University in 2010; and his Ph.D. degree in statistics from the University of California, Los Angeles in 2016.

Benjamin Yao is a senior applied science manager at Alexa AI at Amazon. He received his B.S degree in Electrical Engineering from the University of Science and Technology of China in 2003 and his M.S. degree in Image Analysis in 2006 from Chinese Academy of Sciences. In 2011, he receieved his Ph.D. degree in Statistics from the University of California, Los Angeles.

Chenlei Guo is the director of applied science at Amazon. He received his B.S. degree in 2005 and his M.S. degree in 2008 from Fudan Univeristy, both in Electronics Engineering. In 2009, he received his M.S. degree in Computer Engineering from Carnegie Mellon University. He was a principal engineering manager at Microsoft from 2009 to 2017, where he led the team to launch multiple products such as the entity ranking for Bing search, the office insight and people/location/time search in Office 365 product.

Ruhi Sarikaya is the director of applied science at Amazon. He received his B.S. degree from Bilkent University, Ankara, Turkey, in 1995; his M.S. degree from Clemson University, South Carolina, in 1997; and his Ph.D. degree from Duke University Durham, North Carolina, in 2001, all in Electrical and Computer Engineering. He was a principal science manager at Microsoft from 2011 to 2016, where he founded and managed the team that built language understanding and dialog management capabilities of Cortana and Xbox One. Before Microsoft, he was with IBM Research for 10 years. Prior to joining IBM in 2001, he was a researcher at the University of Colorado at Boulder for 2 years.

How to cite this article: Ponnusamy, P., A. R. Ghias, Y. Yi, B. Yao, C. Guo, and R. Sarikaya. 2021. "Feedback-based self-learning in large-scale conversational AI agents." *AI Magazine* 42: 43–56. https://doi.org/10.1609/aaai.12025