Fantom: A Crowdsourced Social Chatbot using an Evolving Dialog Graph

Patrik Jonell[1], **Mattias Bystedt**[1], **Fethiye Irmak Doğan**[2], **Per Fallgren**[1], **Jonas Ivarsson**[1], **Marketa Slukova**[1], **Ulme Wennberg**[1], **José Lopes**[1], **Johan Boye**[1], and **Gabriel Skantze**[1]

[1]Department of Speech, Music and Hearing, KTH Royal Institute of Technology
[2]Department of Robotics, Perception and Learning, KTH Royal Institute of Technology
{pjjonell,mbystedt,fidogan,perfall,joniva,slukova,ulme,jdlopes,jboye,skantze}@kth.se

Abstract

In this paper we present Fantom, a social chatbot competing in the Amazon Alexa Prize 2018[1]. The system uses a dialog graph for retrieving an approximation of the current dialog context in order to find suitable response candidates in this context. The graph is gradually built using user utterances from actual interactions, and system responses suggested by crowd workers. To this end, we developed an automatic system for finding dialog contexts that were often visited but lacked system responses in order to automatically post tasks on Amazon Mechanical Turk. Workers could see a brief excerpt of past conversation history and were asked to suggest a good response, based on a description of the system's persona and a set of rules that would help foster more engaging conversations. Our main contributions are 1) describing the use of a graph-based approach for context modeling, 2) techniques used in order to make the crowd workers author good content, and 3) discussion of learning outcomes from the Alexa Prize challenge.

1 Introduction

This paper presents Fantom, a social chatbot that has taken part in the second installment of the Alexa Prize, an Amazon-funded student challenge created with the purpose of advancing conversational Artificial Intelligence [1]. The goal of the competition is to implement an artificial agent capable of maintaining an entertaining, coherent, and possibly human-like conversation for a sustained period of time. If a participating bot is capable of engaging in 20-minute-long conversations (of sufficient quality), it is rewarded with a $1M prize, which is an indication of the complexity of the task.

The main problem in designing conversational chatbots is that two of the major design goals, *coherence* and *scalability*, are counteracting forces. It is perfectly possible to construct a system able to converse coherently on a specific topic, by letting an expert hand-code facts and domain-specific rules on how to interpret user utterances, and how the system can best reply to them. This approach can give good results, but the access to experts is a bottleneck, and the approach is therefore not very scalable to a multitude of domains. On the other side of the scalability spectrum, sequence-to-sequence models based on recurrent neural networks have proven to be successful in machine translation [2], and might seem like an obvious candidate to try also on the dialog problem. Using this approach, a recurrent neural network is trained on a large corpus of dialogs, so that it can input a user utterance as a sequence of words, and output a sequence of words which constitutes the system reply to the user utterance in that particular context. However, existing approaches based on this technique have not proven successful, partly due to lack of appropriate data. Generated answers using

[1]`https://developer.amazon.com/alexaprize`

1st Proceedings of Alexa Prize (Alexa Prize 2018).

this technique are often very bland since they are likely to occur in variety of different contexts (e.g. say "I don't know" no matter what the user says) [3].

When designing Fantom, we have used crowdsourcing to attain a middle ground between the purely data-driven approach and the one-expert-codes-all approach. The main research question we wanted to address when taking part in the competition was the following: *To what extent can dialog authoring be made into a collective process using a crowd of non-experts?* The use of crowd-sourcing removes the bottleneck of the single expert without sacrificing control of the data collection process. We therefore hypothesize that this approach will be able to, with proper configuration, yield conversational data of high quality. Obviously, there are still major challenges to consider: ensuring coherence of the resulting dialogs, automating the generation of crowd worker tasks, based on real interaction data, and incorporating the answers into the system's dialog model.

The Fantom system uses a graph-based dialog model: An acyclic directed dialog graph is used for representing possible coherent dialogs between user and system. Each node represents an utterance (or rather, a class of synonymous utterances), either from the system or from the user. An edge from node X to node Y means that any utterance in Y is an appropriate answer to any utterance in X. The dialog graph is partly built up using data from real conversations (for user nodes), and partly by letting crowd workers author appropriate system responses to user utterances (for system nodes). At runtime, Fantom retrieves an approximation of the current dialog context by matching the user's utterance with an appropriate node X in the graph, and selects an answer among the nodes connected to X.

To populate the graph with system utterances, we developed an automatic system for identifying user nodes that were frequently visited at runtime, but which lacked system responses. To create responses, crowd worker tasks (so called HITs) were automatically posted on Amazon Mechanical Turk, but only containing user utterances that were approved to be used for crowdsourcing by Amazon. Workers would see a brief excerpt of past conversation history and were asked to continue the dialog producing an engaging system response. In order to create coherent and engaging utterances, the crowd workers were strongly directed by instructions, including a description of the system's intended persona.

The Fantom system has proven to work on a large scale. During a period of 3 months, Fantom interacted with approximately 75,000 unique Amazon customers in roughly 100,000 conversations, resulting in a dialog graph with 46,000 user nodes and 6,500 system nodes suggested by crowd workers.

2 Related Work

There are several important distinctions to make when it comes to modern dialog systems. One such distinction is if the system is task-oriented or chat-oriented. A large portion of dialog system research has been devoted to systems that help users to perform some specific task, like retrieving information, booking trips, controlling devices, etc. In this strand of research, the focus has often been to solve the task as quickly as possible. Indeed, one evaluation criteria for such task-oriented systems is how many dialog turns they take, on average, to accommodate the user's need and solve the task (where fewer turns are better). In contrast, chat-oriented dialog systems aim to amuse and entertain, and the goal here is rather to maintain the user's interest and have as long dialogs as possible. Evaluation methods are by necessity subjective, e.g. how much users like the system on a scale 1-5, or how large the proportion of coherent and/or engaging answers was (where a human expert decides what is coherent and engaging). A major challenge when designing chat-oriented system is to handle the open-ended nature of the interaction; the user can say virtually anything at any time without leaving the domain of discourse. Such research efforts can be seen by for example 2017 year's Alexa Prize finalists [4, 5, 6].

Another important distinction to make is whether the system is data-driven or hand-coded. Most chatbots of today are trained using example interactions. The interactions used for training can, for example, be collected from Twitter [7], movie subtitles [8], or the user responses in conversations between humans and an already existing bot, e.g. Cleverbot [9], and Microsoft Tay. However, existing approaches based on these data sets have not resulted in systems capable of holding engaging interactions with a user for any length of time. One reason for this is that the dialog data they are based on was not especially targeted towards engaging human-computer interaction with a coherent persona. They might also risk saying extremely inappropriate content, such as the Microsoft Tay bot did when twitter users learned that they could influence what the bot said [10].

Hand-coded systems often use rules to steer the conversations. They have the advantage of an expert being in charge of what the system says, and thus avoids the issues with incoherent or offensive replies. However, hand-coding is extremely time-consuming and does not scale well with an open-domain chat system. These rules can either be manually hand crafted or derived from example interactions. ChatScript [11] or online APIs, such as Dialogflow[2], Lex[3], Luis[4] or Wit.ai[5] can be used to build rule-based chatbot systems.

A third distinction that can be made is whether a system is generative or selective when outputting answers to the user. A generative model will automatically generate the system utterance on a letter-by-letter or word-by-word level often using variations of recurrent neural networks, such as LSTMs [12, 13]. These systems have however been reported to produce overly general and vague utterances [3, 14], despite the recent efforts to use objective functions that try, for instance, to maximize the amount of mutual information between adjacent pairs of turns [15]. However, this does not *per se* guarantee an increase in the quality of the dialogue. Another limitation that generative dialogue models tried to solve was the capacity to generate responses based on previous context of the dialogue. Serban et al. proposed to use a latent variable in the decoder, which combined information from the previous turn with the history of the dialogue [16]. A selective system on the other hand selects a full utterance, or a template, from a bank of utterances and provides it to the user. The advantage of using a selective approach is that the utterances presented to the user are guaranteed to be syntactically well-formed. The disadvantage on the other hand is the decreased flexibility of what the system can output. Initial approaches using this principle used similarity of RNN-based embeddings between the context and the response to retrieve the best answer [17]. In contrast to these word sequence representations, Zhou et al. proposed an utterance sequence representation to map the relationship between utterances [18]. Sequential Matching Networks [19] were also proposed for the same purpose. However, the evaluation of these models is often limited to datasets which can hardly be considered conversational (e.g. the Ubuntu Dialogue Corpus [17])

A compromise between using a data-driven and a hand-coded approach, which we have adopted to build Fantom, is to use crowdsourcing. This allows for collecting large amounts of data, while at the same time having control over how the data is collected. One of the first examples of crowdsourcing for dialog generation is Orkin et al. [20]. Similarly, Breazeal et al. proposed a data-driven approach to dialog generation for a social robot by crowdsourcing dialog and action data from an online multi-player game [21]. There has been work on collection of text-based corpora, for example Filatova investigated irony and sarcasm by creating a corpus based on Amazon reviews [22], and more recently, Ben Krause et al. collected a dialog dataset using crowd workers in order to train their generative RNN network. They asked the crowd workers to self-author complete dialogs [14]. Leite et al. presented a graph based approach using crowd workers when letting participants interact with a robotic head [23]. The use of crowd-workers to build dialogues has also been done for task-oriented dialogues [24].

In order to avoid issues with utterances in dialog datasets being incoherent or contradictory, one can condition it using a persona. Work has been done to condition a conversation model with a given persona [25] or profile information from the sender/receiver of a message [26]. Huang et al. let crowd workers chat one-on-one, giving each one a list of personality traits to adhere to [27]. In this case, the dialogs produced were symmetrical with respect to the roles the both users had, while in our case, the role of the chatbot is different compared to the role of the human, thus requiring an asymmetrical dialog dataset.

[2]https://dialogflow.com/
[3]https://aws.amazon.com/lex/
[4]https://www.luis.ai/
[5]https://wit.ai/

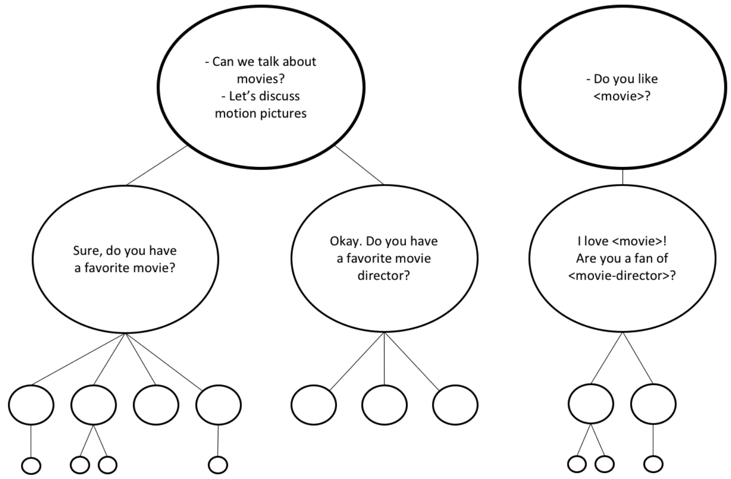

Figure 1: Two user nodes with corresponding system utterances in the dialog graph.

3 System overview

3.1 Dialog Graph Overview

The basic idea when designing and constructing the Fantom dialog system was to use crowdsourcing to produce coherent and entertaining system utterances. We therefore required a dialog management method capable of incorporating contributions from many authors, most of them without any knowledge about dialog systems or computer science in general. The dialog graph structure used by Fantom is such a method: The graph contains nodes representing an utterance (or rather, a class of synonymous utterances), either from the system ("system nodes") or from the user ("user nodes"). An edge from node X to node Y means that any utterance in Y is an appropriate answer to any utterance in X. An edge can therefore only connect a system node to a user node or vice versa; there is never an edge between two system nodes or two user nodes. A path through the graph thus represents a dialog between user and system. A path ending in a user node can now easily be extended by adding a new system response to the last user utterance, using a graphical interface showing the preceding dialog to the crowd worker (see Figure 3). In this way, many workers can contribute and independently extend the graph, and each path in the graph will represent a coherent dialog with (hopefully) interesting system utterances. The graph is also extended with user nodes retrieved from real interactions with the system.

More specifically, the graph can be described as a collection of trees, each rooted in a user node (see Figure 1). Each tree is typically a topic of its own, and the root node is context-independent (e.g. it can serve as a starting utterance in a conversation). At runtime (chat time), the initial user utterance is matched towards all such root nodes. If the utterance matches a user node X with a high enough similarity score, the system's reply is chosen from the system nodes connected to X. The next user utterance is then matched with all the children of that system node. If the user utterance does not match with a high enough similarity score, it is again matched with all root nodes (as the user might have changed topic). If the utterance does not match any node with high similarity score, various fallback strategies are employed (see Sect. 3.3.2). One of the main challenges is to correctly and reliably match user utterances to nodes in the graph. For more details regarding the matching, please see Section 4.

3.2 System Components

As can be seen in Figure 2, Fantom consists of two main systems: An online system, which the user interacts with, and an offline system, which runs independently of the online system in order to automatically improve the dialog graph, using crowdsourcing and graph maintenance tools.

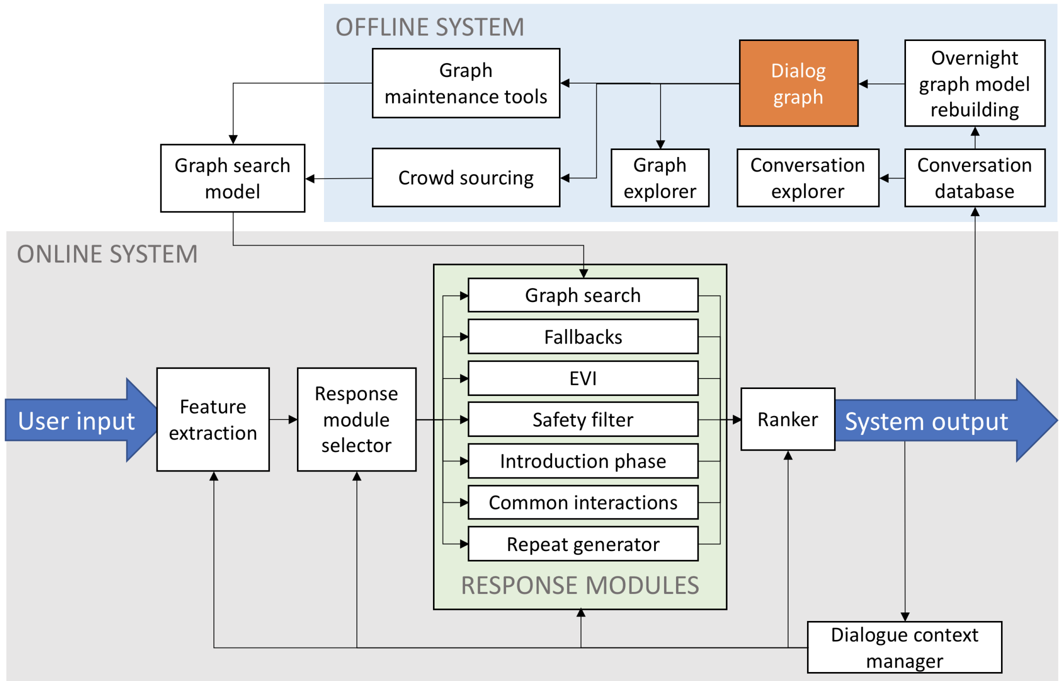

Figure 2: An illustration of the general system architecture.

3.2.1 Online System

The online system is responsible for handling the dialog with the user. When a user utterance enters the system it goes through several steps described below.

Feature Extraction Our system extracts several features from the current user utterance in combination with the previous dialog history. For a detailed description of the extracted features, see Section 3.4.

Response Module Selector The Response Module Selector is responsible for selecting all of the modules that could generate a valid response candidate. For example, while the user is considered to be in the introduction phase, the response module selector would add the introduction response module to the pool of potential response module candidates, and request that it attempts to produce an answer.

Response Modules The system relies on 7 different response modules. All modules selected by the Response Module Selector make an attempt at producing a system response. In some cases there will be no output from a response module (such as the safety filter, if it does not detect any inappropriate content). However, the graph search response module always returns an answer, together with a similarity score which is used by the ranker to determine if the response from graph search is usable or not. The response modules are individually covered in more detail in Sections 3.3.

Ranker The ranker is responsible for selecting a response from the candidates selected by the response module selector. The system uses a rule-based ranking system to select a response. The safetyfilter, introduction phase, and common interaction response modules were always chosen first (in stated order) if they had produced a response. Besides yielding a textual response, the graph search response module also produced a similarity score for the given response. Using this similarity score two lower thresholds (where a higher score corresponds to a better match) for the graph search answer were empirically found. The first threshold (0.83) was approximately the point where the graph search responses were on average more incoherent than not and thus deemed unusable. The second threshold (0.89) corresponded to the point where the Evi response module on average produced a better response than the graph search. Therefore the ranker would choose the Evi response if there is one available and the similarity score would be below 0.89 otherwise the graph search response would be used as long as the similarity score is above 0.83. Lastly if no other module had been selected, the fallback module was used.

Dialog context manager The selected response, and all the data for the current turn, is saved in the dialog context manager and can be accessed from subsequent turns. For example the feature extractor accesses previous user and system utterances when it does coreference resolution which is provided by the dialog context manager.

3.2.2 Offline System

Conversation Database Each time anyone interacts with the system, the turn is persisted in the conversation database. Information about what the user said, what the system responded, features extracted etc. was saved to the conversation table.

Dialog Graph The dialog graph is a database where all of the relevant dialogs for the system are stored and accessed by other parts of the system. See Section 4 for an in-depth description of the dialog graph.

Crowdsourcing The system automatically analyzes user conversations and the dialog graph to find where the dialog graph is lacking system responses with respect to how often that part of the graph is visited by users. Given a list of such prioritized nodes in the graph, the system automatically creates and sends out tasks on Amazon Mechanical Turk for workers to author new system nodes. These utterances are then inserted into the graph as valid responses for a given user utterance and a given history context. Only utterances approved for crowdsourcing by Amazon were posted to Amazon Mechanical Turk. For a detailed description of the crowdsourcing system see Section 4.1.

Graph Maintenance Tools A set of tools were developed for maintenance of the graph. These tools were for example a merging tool, for merging semantically similar nodes (see Section 4.2.4), or an inactivation tool for inactivating nodes that are incoherent, unengaging or inappropriate.

Graph Search Model Each night the graph search model is rebuilt and transmitted to the graph search response module. The difference between the graph search model and the dialog graph is that the graph search model is a file meant to be read into memory by the graph search response module. It does not contain all the nodes from the dialog graph, as for example nodes with less than two user visits, i.e. the amount of times any of the utterances belonging to that node had been uttered, are excluded. The graph search model contains pre-calculated features for every node it stores.

Conversation and Graph Explorer Tools To facilitate browsing of conversation data two tools were implemented. Firstly, a conversation viewer, which was used to browse individual conversations users had with the system. The tool could sort, search and fetch data from a certain time period based a number of values e.g. ratings, number of turns and feedback. Secondly, a web interface visualizing the dialog graph was used in order to get a better overview of the data quality and with ease follow dialog paths in the dialog graph. This interface gave a quick impression of which nodes were visited the most and what kind of replies most users gave to a given system node.

Overnight Model Rebuilding Each night the dialog graph is rebuilt and a sequence of processes are run, these are in order:

- Update list of utterances approved by Amazon
- Populate the dialog graph with conversations from the past day
- Merge semantically equivalent sentences and synonyms
- Populate relevant nodes in the dialog graph with yes/no nodes after each system utterance has been analyzed using the yes/no question classifier. (see Sect. 3.4)
- Retrain LDA model (see Sect. 3.4)
- Find named entities in the graph
- Fetch named entity information from external sources (such as Wikipedia)

3.3 Response Modules

Below a description of all response modules is given. See Table 5 in Section 7 for example of interactions showcasing their usage.

3.3.1 Graph Search

The “Graph search” response module matches the user utterance to the best candidate in the graph, using the similarity score function described in Section 4.3. It does so by comparing a set of features derived from the user utterance against the same set of features pre-computed for each node in the graph. If there is no previous dialog context, the utterance is only matched against the root nodes in the graph. On subsequent turns, the system will first try to match against any children of the previously selected node, before trying to match to all root nodes again. In other words, first the contextually dependent nodes are being searched, in case the user is replying to the last system utterance, and then, the contextually independent nodes are being searched, in case the user said something that does not require the current contextual history (for example if the user is switching topic).

3.3.2 Fallbacks

A fallback module was set in place to handle situations where the ranking strategy deemed responses of the other modules inadequate. Depending on what the user said and the features of the current turn, the fallback module returns a different response. As a first step, it checks for any potential named entities in the conversation. Depending on the type of the named entity, the system responds in a certain manner. For instance, if the user input contains a <person> tag, a response such as *Tell me more about <person>!* or *I need to read up on <person>, cause I don’t know who that is.* was returned. Similar responses are generated for other named entities, such as <organization> or <movies-tv>.

If the user input does not contain any named entities, the fallback module instead makes use of the topical information, based on the assumption that the user will appreciate a bot that at the very least retains some context of what was said, rather than returning a completely generic response. Therefore lists of fallback responses for the topics with highest frequency, e.g. movies, music, sports, celebrities, were created.

For the general, and less frequent topics, the last resort is to return a generic fallback without any contextual information. To avoid repeating fallbacks, the probability of selecting a given fallback was based on how many times it had already been used, where a higher number would result in a lower probability. Furthermore, simple acknowledgments (e.g. *okay*, *cool*) were added to the beginning of the fallbacks, with the intention of giving the user a sense that they had been understood. The functionality of linked nodes, described in Section 4.2.2, was also used in order to position the user in a node in the graph that carried the same semantics and intents as the produced fallback. This was done in order to use the system responses already existing in the graph. Finally, simple rules were added to handle follow-ups to certain types of common question/answer pairs and for user utterances containing the words where, who, what, why, when. For example, if a yes/no question such as *Do you love music?* was asked by the system, and a yes response was detected, the bot could return simple answers such as *I love music too*. Examples of fallbacks are shown in Table 1.

User input	Fallback Response	Fallback Type
I can’t get enough of Tom Cruise.	Cool. So, are you a fan of Tom Cruise?	Named entity
My best friend’s name is James Smith.	Okay. Tell me more about James Smith!	Named entity
How I met your mother is probably my favorite tv-show	I see. So do you like watching TV?	Topic
Okay my favorite book is The Hobbit	Cool. So do you like reading books?	Topic
No I’m not into DC Comics	Okay. Is it ok if you suggest a new topic?	Generic
I don’t like that song	Oh. Can we talk about something else?	Generic

Table 1: Examples of fallbacks given user input. The data is not from real user interactions.

3.3.3 Evi

Amazon’s Evi, which is a question-answering API, was used in order to handle fact questions, or questions regarding recent events and news. Some tweaking and filtering had to be implemented to make sure that the bot generated adequate responses. Evi had a tendency to return definitions of certain common words. As an example, if a user asked for a joke in a certain phrasing Evi generated *"As a noun, ’joke’ is usually defined as a humorous anecdote or. . . "*. Because of this, all responses were filtered out following a similar pattern. In a similar manner profanity was filtered out to make sure no bad words would slip through. Additionally, as also seen in [6], some sentences were rephrased before sending them to Evi, which tended to lead to better responses.

3.3.4 Safety Filter

A rigid set of rules was developed for detecting inappropriate content and replies to deflect from those subjects. For instance, if a bad word was detected in the user utterance a response such as *"Let's talk about something else, for example sports, music or movies."* was selected. Similarly, when a user asked for advice the system responded with something along the lines of *"I'm afraid I'm not allowed to give advice. Let's talk about something else."*.

3.3.5 Introduction Phase

An introduction phase was introduced in order to lead the user smoothly into a conversation with the bot, and also to pick up the name of the user. This module was only initialized the first time a user interacted with the bot.

3.3.6 Common Interactions

In order to increase customer experience a number of mini modules were added that were triggered by certain user utterances. It was for instance relatively common that the user tried to activate a regular Alexa skill, such as playing a certain song, or reading an audiobook. When these utterances were detected the bot replied that this functionality is not available in the current mode and instructions for how to stop were provided. Similarly detection for when the user wanted to stop the conversation was added, as the default stop intent didn't cover enough varieties.

3.3.7 Repeat Generator

A detector for when the user had communicated misunderstanding or requested that the chatbot should repeat itself was implemented. It took the last system utterance and appended "I said" to the beginning of the sentence.

3.4 Feature Extraction

A feature extraction module was used to provide the response modules with relevant features. This module took a specification of functions to be run in order to be able to extract a feature. Below we present how the various feature extractors were implemented.

True Casing True casing was used to recover correct casing, using the library named truecaser [6].

Coreference Resolution SpaCy was used [28] to perform coreference resolution. The system first looks for any coreferences in the previous and current user utterances when they are concatenated. If it does not find any coreferences, it tries to resolve them for the last system response concatenated with the current user utterance.

Named Entities A dictionary of named entities was used to identify relevant entities in the user's utterance and to generate a key value pair for the tag and corresponding entity.

Lemmatization, Tokenization and Part-of-speech SpaCy is used in order to extract these features.

Sentiment Each utterance's sentiment was extracted using VaderSentiment [29]. This extractor gave a compound sentiment score in the interval $[-1.0, 1.0]$, where -1 indicates strong negative polarity, 0 indicates a neutral polarity and 1 indicates strong positive polarity.

POS Score Each part-of-speech (POS) is associated with a weight which has been found empirically. These weights denote the importance of each POS when matching their similarity.

Word Embeddings FastText [30] is used for extracting word embeddings.

POS Vectors This scales the word embeddings with the weights from the POS score for each token based on its POS.

Latent Dirichlet Allocation Latent Dirichlet Allocation (LDA) is a model generally used for topic modeling [31, 32]. Since the LDA makes an assumption about each document's composition of different topics, it is useful for modeling dialogs which might also include different topics in a conversation. Moreover, LDA is based on Dirichlet Parameters which are called alpha and

[6] `https://github.com/nreimers/truecaser`

beta parameters and decide on document-topic and topic-word proportions respectively. “Cleaned” utterances from the dialog graph, i.e. having new lines, punctuation and stop words removed, were used to train the LDA. The LDA model has two Dirichlet distribution parameters called α and β which determines document-topic and topic-word distributions respectively. Our model was initialized for 5 topics, with $\alpha = 0.5$ and $\beta = 0.1$. Those are the default values for the model but they also yielded best results in terms of word distributions over topics when performing informal subjective human evaluations. We made use of C++ implementation of Phan et al. [33].

Topic Classifier A topic classification API provided by Amazon was used in order to extract the most likely topic from an incoming utterance.

Yes/No Question Classifier The Stanford CoreNLP module was used to identify closed-ended questions. This module was developed by Manning et al. [34]. The tool was modified so that it oculd distinguish between open-ended and closed-ended questions. Moreover, we also added differentiation of clarification statements such as ‘Pardon me?’, ‘Excuse me?’ since the module sometimes classifies them as close-ended questions.

3.5 Implementation

The large majority of the implementation for Fantom is written in Python 3.6, where most of the code resides in a utility package named fantom_util for easy portability of the system. Amazon has also provided a framework named Cobot, which is a dialog manager framework which is deployed on AWS[7]. The cobot framework handles much of the dialog manager boilerplate code, such as calling different feature extractors or response modules, but also handles and persists the current dialog state into a document database.

4 Dialog Graph

4.1 Crowdsourcing

In order to populate the graph with system responses, a module was implemented that automatically created crowdsourcing tasks, where the workers were asked to continue the dialog based on a given dialog history (see Figure 3). The worker was asked to both validate the previous dialog, and to author an appropriate response. To guide the worker, a brief system persona was shown on the right. For a more detailed description of the persona, see Section 4.1.2.

The dialog history was created by traversing the graph backwards from the point that needed improvement and by picking a random anonymized utterance from each node, up to 6 utterances back. The system created both regular tasks and "tag tasks" (see Section 4.2.1), in which the worker was asked to continue the conversation using tags (such as named entities) they could drag-and-drop, which would then resolve to an attribute relating to the previous tag. As can be seen in Figure 4, this allowed the worker to construct responses with variable content. New tasks were generated on an hourly basis.

4.1.1 Data Quality Assurance

Several methods were employed in order to ensure high quality of the responses the workers authored. First, each worker was instructed to validate the dialog history, and to mark any utterances deemed to be incoherent. Failing to do so resulted in failing the task. If other workers would mark a dialog incoherent where a worker had failed to report it as incoherent, that would result in a failed task. This was a strategy for 1) automatically finding bad utterances and bad workers, and 2) letting workers know that their work will be evaluated by other workers. The system made sure that a worker would never receive a dialog history which they have contributed to themselves. Another measure in order to increase data quality was to prepare an introduction video with an instructor explaining how to use the system and a few training tasks in order to show some examples of good and bad dialogs. Completing this training was a requirement in order to work on the tasks. Since the system would interact with American end-users, it was important that cultural references would be as similar as possible. Thus, workers were required to be from the U.S. The worker was required to have had completed at least 5000 tasks on Amazon Mechanical Turk and had an approval rating higher than

[7] `https://aws.amazon.com/`

97%. Additionally, in order to be qualified to do the tagging task, which also was slightly better paid, the worker needed to have completed at least 20 of the original tasks. The workers were paid 20 cents per regular task and 30 cents per tagged node task.

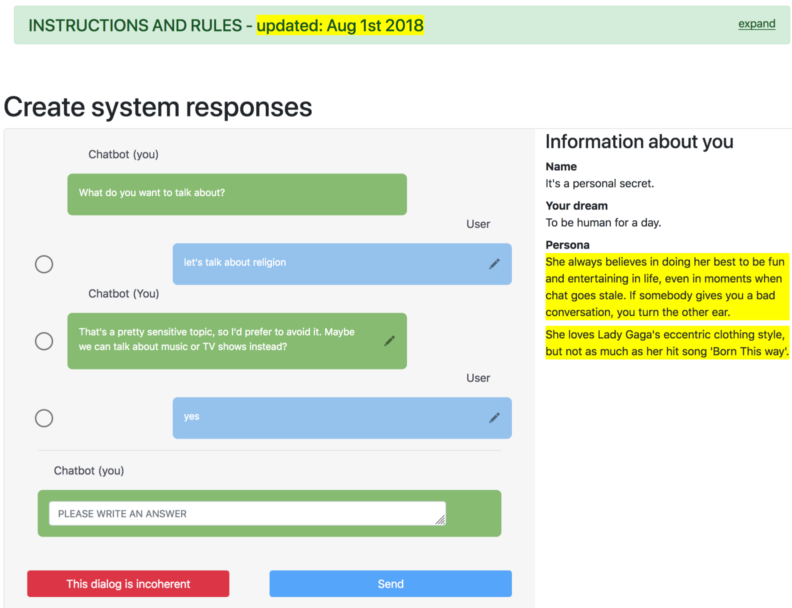

Figure 3: Interface used for authoring content in the graph

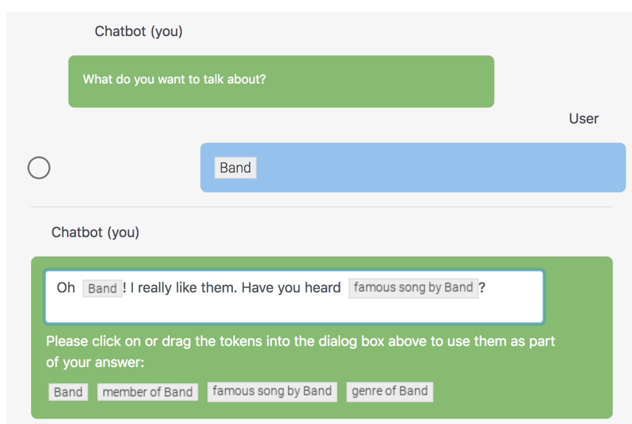

Figure 4: Crowdsourcing interface for tag nodes.

4.1.2 Persona

The persona was carefully crafted so that the crowd workers would form a coherent personality in their responses to the user. In order to make the workers read the persona attributes, a small random subset of the attributes was shown to the workers while they worked on the task. Two examples of the persona provided to the crowdworker can be seen to the right in Figure 4. The persona text used a yellow background color in order to draw attention. This was found to yield a higher degree of answers in line with the persona after doing some initial experimentation with the user interface and inspecting random responses from the workers. All of the attributes built up one personality, thus there were no two sentences that contradicted each other. Showing a small subset of the attributes was a compromise between losing the attention of the crowd worker due to showing too much information but still getting the persona through.

4.2 Scaling the Graph

To avoid scalability issues, given the constant growth of the dialog graph, a number of tools and concepts were implemented, these are described below.

4.2.1 Tags for Named Entities

In order for the system to be able to handle variable content, such as named entities, tags were introduced. Topics such as movies or music are heavily based around talking about specific entities and creating a separate node for each entity in the graph meant an undesirable branching factor and would also result in an unrealistic amount of crowdsourcing tasks. A simple example of this mechanism is illustrated below

System Utterance : What's your favorite movie?
User Utterance : I like <movie>.
System Utterance : I thought that <movie-actor> was excellent in that.
Do you have a favorite actor or actress?

A database was created to store all the information about named entities, and their related attributes, such as in the example above, <movie-actor> is a related attribute of <movie>. Wikidata[8] was used in order to download information about the named entities and various internet sources were scraped for collections of popular entities.

To avoid ambiguities among named entities with the same name, which belong to different categories and where the category is clear from the context, the system only looked for categories of tags that it could find among the next expected user utterances. In asking the user for a favorite movie, the system would identify *Harry Potter and the Goblet of Fire* as both a book and a movie, but only the movie would be correct.

As the intention for the bot was not to be omniscient, it was acceptable that it did not possess the information about all existing named entities. Instead, it was intended to learn more over time.

New named entities in the graph were detected during the overnight process, and new tag nodes were created. For example, if a node was found, where several of the children contained mentions of movies, a node would be added with the movie tag to that parent node. New named entities were also collected and the resulting data was merged into the graph search model.

4.2.2 Linked Nodes

Two nodes in the graph can be manually linked if they share the same semantic representation, but are located in different branches of the graph. This functionality was introduced as it was noticed that similar conversations started to appear in different branches of the graph. If two nodes X and Y are linked, then all children of X are considered children of Y and vice versa, leading to a more richly populated graph, and a wider range of response options.

4.2.3 Root Node Classifier

In the graph structure, it is crucial to be able to distinguish between utterances that can be used to start a new conversation or steer the conversation into a new topic, such as *let's chat about music* or *what's your opinion on Donald Trump?*. These can be contrasted with utterances that always are context dependent, e.g. *I like them*, *yeah, I think so too*. The former are used as root nodes in the dialog graph, while the latter should only exists at deeper levels of the graph. When automatically populating the graph with user utterances, context dependent nodes were erroneously added as root nodes, and context independent nodes were added at deeper levels in the graph. While it is not necessarily wrong that context independent nodes exists deeper in the graph, it can be beneficial to have them exist at the root level as well. In order to automatically distinguish between these two types of utterances a *Root Node Classifier* was developed which classifies the context dependency of a certain utterance. The classifier was then used in order to relocate and remove erroneously inserted nodes.

[8] https://www.wikidata.org

The classifier was trained using data of the 2000 most common utterances from the 2017 Alexa Prize competition that were annotated as to whether they were always context dependent or if they could be used to start of a new conversation.

Two versions of the root node classifier were trained. The first was a bag of words neural network model based on one-hot encodings of the POS tags and a selection of certain words that were indicative of whether the utterance were to be considered a root node or not. The second model was an extension of the first one, where an RNN model on a word-to-word level was used instead of the bag of words model. The first of these reached an accuracy of 84% and the second one reached an accuracy of about 86% on the task of correctly predicting the root nodes in the test set.

4.2.4 Automatic Merging

For the graph structure to work effectively, it was important to merge utterances with the same meaning into the same node. Although this was a purely manual task at first some automation was added to facilitate the process. Specifically, a script was implemented that automatically merged provided lists of synonyms. Among these synonym lists were for instance a list containing different ways of saying "yes", one for the ways of saying "no", and one for the ways of saying "I don't know". During runtime all neighboring nodes that contained at least one element from the same of these lists were merged with each other. Furthermore, this script was also tweaked to allow for merging of certain patterns that were guaranteed to carry the same meaning; it is for instance generally true that the user semantics in *Let's talk about X* and *Let's chat about X* are the same.

4.3 Graph Search

A semantic similarity function was trained in order to match a given incoming utterance to the most semantically similar utterance in the candidate nodes. This was accomplished by using an approximative statistical framework, where an independence approximation of the features leads to a multiplicative model. All functions return values in the interval $[0, 1]$. In order for the matching function to work reliably in the realm of sparse data that the challenge is in, the amount of free parameters in the model were effectively reduced by using a combination of feature engineering and weight optimization on a validation dataset. The optimal weights were obtained through a grid search algorithm. (See section 5.1) The resulting similarity score, f, between two utterances, utt_i and utt_j, is calculated as the product of the following scoring components: sentiment, topic, word vectors and LDA, as shown in Equations 1-7. All functions return values in the interval $[0, 1]$.

$$f(\text{utt}_i, \text{utt}_j) = f_{\text{sentiment}}(\text{utt}_i, \text{utt}_j) \cdot f_{\text{topic}}(\text{utt}_i, \text{utt}_j) \cdot f_{\text{word vectors}}(\text{utt}_i, \text{utt}_j) \cdot f_{\text{LDA}}(\text{utt}_i, \text{utt}_j) \tag{1}$$

$$f_{\text{topic}}(\text{utt}_i, \text{utt}_j) = \begin{cases} 1.00, & \text{if topic}(\text{utt}_i) = \text{topic}(\text{utt}_j) \\ k_{\text{topic}}, & \text{otherwise} \end{cases} \tag{2}$$

$$f_{\text{sentiment}}(\text{utt}_i, \text{utt}_j) = \max\Big(k_1, 1.00 - \big(\text{sentiment}(\text{utt}_i) - \text{sentiment}(\text{utt}_j)\big)^{k_2}\Big) \tag{3}$$

The topic classifier and the sentiment extractor are described in Section 3.4. $k_{\text{topic}} \in [0.0, 1.0]$, $k_1 \in [0.0, 1.0]$ and $k_2 > 0$ are learned parameters.

$$f_{\text{word vectors}}(\text{utt}_i, \text{utt}_j) = \max\Big(f_{\text{word vectors, 1}}(\text{utt}_i, \text{utt}_j), f_{\text{word vectors, 2}}(\text{utt}_i, \text{utt}_j)\Big) \tag{4}$$

The first word vector score makes an assumption of syntactical similarity between the two sentences. It works by creating a separate average word embedding for each part-of-speech (POS) in each of the utterances, and then computing the cosine similarity for each POS followed by a weighted summation of the similarity scores. C_k is a weight constant corresponding to the k-th POS, as shown in Equation 5:

$$f_{\text{word vectors, 1}}(\text{utt}_i, \text{utt}_j) = \frac{\sum_k C_k \cdot \cos\Big(\sum_{i' \in \text{POS k}} x_{i'}^{(1)}, \sum_{j' \in \text{POS k}} x_{j'}^{(2)}\Big)}{\sum_k C_k} \tag{5}$$

The second word vector score computes the cosine similarity of the word-class-weighted average word embeddings of the two utterances, as shown in Equation 6:

$$f_{\text{word vectors, 2}}(\text{utt}_i, \text{utt}_j) = \cos\left(\frac{\sum_{i'} w_{i'}^{(1)} x_{i'}^{(1)}}{\sum_{i'} w_{i'}^{(1)}}, \frac{\sum_{j'} w_{j'}^{(2)} x_{j'}^{(2)}}{\sum_{j'} w_{j'}^{(2)}}\right)^2 \tag{6}$$

Here, $x_{i'}^{(1)}$ indicates the i-th word vector in the first utterance, and $w_{i'}^{(1)}$ indicates the weight constant corresponding to that same word's POS. These are multiplied together and summed in order to construct a POS-weighted average word embedding. The score function $f_{\text{word vector, 2}}(\text{utt}_i, \text{utt}_j)$ measures the squared cosine similarity of the weighted average word embeddings of the two utterances.

$$f_{\text{LDA}}(\text{utt}_i, \text{utt}_j) = k_{\text{LDA}} + \Big(1 - k_{\text{LDA}}\Big) \cdot \cos\Big(\text{LDA}(\text{utt}_i), \text{LDA}(\text{utt}_j)\Big) \tag{7}$$

5 Observations

In this section, observations regarding the system and experimentation results which were made during the course of the Amazon Alexa Prize 2018 competition, are described.

5.1 Weight Optimization in Similarity Function

A script was developed in order to compare the validity and performance of the similarity scoring functions mentioned in Section 4.3 that creates a measure in the interval $[0, 1]$ of semantic similarity between two utterances. This script was also used in order to optimize the weight parameters of the different scoring functions. The test script utilizes the fact that different utterances such as *let's chat about music* and *can we talk about music please* in a given node share the same semantic meaning. This opens up for the possibility to automatically generate a dataset from the nodes containing several utterances. The 100 root nodes containing the most utterances were chosen. The test script loops through each pair of utterances and tries to predict if they belong to the same node or not in a binary classification task. This is done by calculating the similarity score between each pair of nodes, and then varying the threshold $\theta_{thres} \in [0.0, 1.0]$ in order to optimize the F_1-score. Calculating the prediction if an utterances belongs to the same node is shown in Equation 8.

$$y_{pred}(\text{utt}_i, \text{utt}_j) = \begin{cases} 1, & \text{if } \theta_{thres} \leq f(\text{utt}_i, \text{utt}_j) \leq 1.0 \\ 0, & \text{if } 0.0 \leq f(\text{utt}_i, \text{utt}_j) < \theta_{thres} \end{cases} \tag{8}$$

Based on the results from these tests, it was clear that words from different POS classes differ in their usability to predict the semantic meaning of an utterance. Generally, it was found that nouns and verbs are more indicative of the semantic meaning of an utterance than other POS classes such as pronouns, adjectives and numeric values. It was also observed that the magnitude of the word embedding vectors of different words did not contribute to predicting semantic representation.

Name	F1	Precision	Recall	Threshold
Average Word Embedding (FastText)	0.18	0.28	0.38	0.87
Sentiment	0.06	0.036	0.68	0.99
Topic	0.28	0.21	0.84	0.94
Graph Search Similarity Score	0.63	0.69	0.73	0.74

Table 2: Results for individual components of the similarity function and the final combined approach.

5.2 Dialog graph

By the end of the semi finals the dialog graph consisted of approximately 50,000 active nodes (see Table 3 for additional data about the dialog graph). The number of user nodes at depth 2 is significantly higher than that of other levels. This is due to a large amount of system responses such as *What is your favorite movie* or *Do you have a favorite rock band* at depth 1 resulting in a wide variety of answers from users. This observation further motivates work on scalability-enabling features such as tags for named entities and automatic merging of semantically similar nodes (see Section 4.2).

Depth	User/System	Node count	Utterance count	Median of utterances per node (90th perc.)	Median node visits (90th perc.)
1	User	2124	3355	1 (2)	4 (37.0)
	System	2058	2205	1 (1)	7 (68.4)
2	User	34188	92781	1 (1)	1 (2.0)
	System	3402	3567	1 (1)	2 (10.0)
3	User	7640	39221	1 (23)	1 (2.0)
	System	743	777	1 (1)	2 (8.0)
4	User	746	6771	1 (49)	1 (2.0)
	System	122	122	1 (1)	1 (4.9)
5	User	43	812	1 (49)	3 (2.0)
	System	2	2	1 (1)	1 (4.6)

Table 3: Various data at different depths of the dialog graph. User nodes at $depth = 1$ are root nodes. The median values are presented together with the 90th percentile. Data from August 15th 2018.

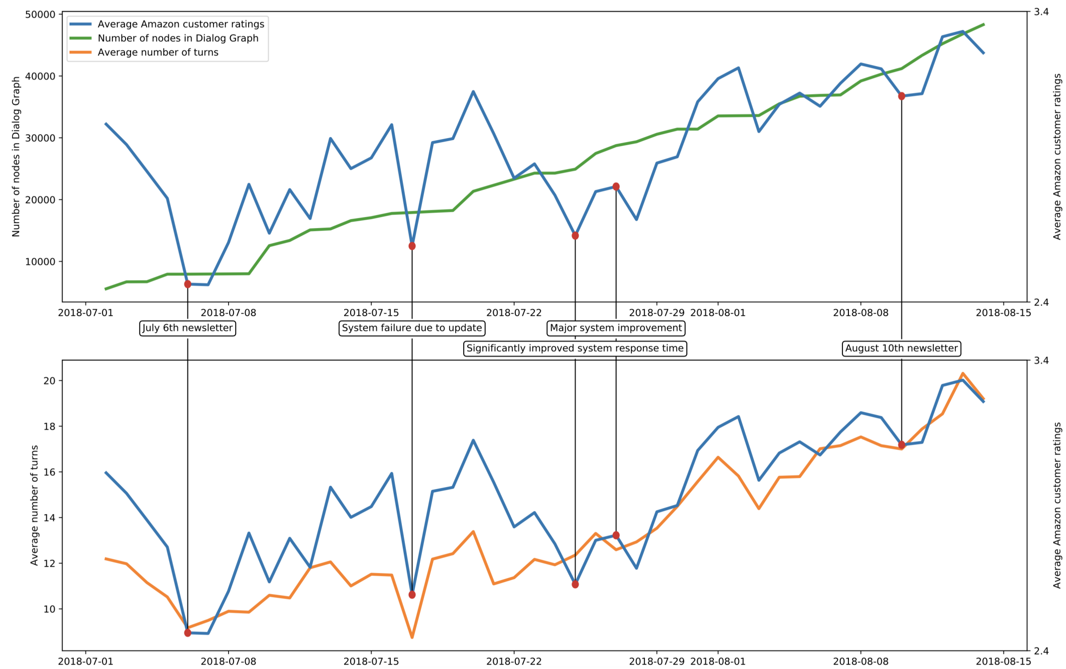

Figure 5: Average Amazon customer ratings plotted together with average number of turns and the number of nodes in the Dialog Graph over the course of the Amazon Alexa Prize 2018 challenge.

5.3 System performance

As can be seen in Figure 5 the system performance in terms of Amazon customer ratings increased during the course of the competition. Three major events have been annotated in the graph which are particularly interesting. The first event was on July 6th when Amazon sent out a newsletter regarding the challenge to their customers. This resulted in a significant drop in ratings. The second event which is annotated in the figure is from a major system failure due to a failed update of the system, which also resulted in a drop in ratings. The third and fourth events are both system improvements, where several substantial code changes were introduces, such as improvement in the fallback system, shorter response times, addition of the introduction phase, etc. The last annotation is from the second newsletter that was sent out during the semifinals. Note that the second newsletter had a much smaller impact on the ratings compared to the first newsletter. What is also interesting to note from the figure is that the average number of turns are strongly positively correlated with the average Amazon customer ratings when grouped by date (r=0.86, p<0.001).

5.4 Correlations Between Response Module Usage and Amazon Customer Ratings

Pearson Correlation Coefficients between the usage of each module and the rating was used in order to get further insights into what response modules correlate with higher Amazon customer ratings.

Table 4 shows that the modules correlate to varying degree with the Amazon customer ratings. As the table shows, Graph Search correlates the most with the ratings. It is also the response module which is used most frequently. Interestingly, the fallback module performs better at generating higher customer ratings than Evi in the open domain chat that we have investigated.

Module Name	Correlation Coefficient	Fraction Module Usage
Graph Search	0.120	0.455
Fallbacks	0.068	0.376
Evi	0.054	0.083
Safetyfilter	-0.037	0.007
Other	0.010	0.079

Table 4: Pearson Correlation Coefficients between Response Module Usage and Amazon customer ratings for the time period August 2nd until August 14th 2018.

5.5 Effect of Data Collecting Strategies

When we started with the crowdsourcing tasks where the workers would author content for the dialogs we set up an experiment in order to investigate the impact of having them either listening contrary to reading the previous dialog history and/or typing vs speaking in their answer. For the listening condition we generated audio using text-to-speech (TTS) through Amazon Polly. A female voice (Joanna) resembling the voice used by Amazon Alexa, was used for system utterances while a contrasting male voice (Joey) was used for user utterances. For the spoken responses Google Cloud speech-to-text was used in order to obtain the answer in written form. The user was able to edit the written answer in order to correct for misinterpretations. Our hypothesis was that workers who spoke and listened would produce higher quality utterances for a conversational system. The way we evaluated this was by comparing the means between the different conditions with respect to two annotation metrics provided by Amazon. These metrics were whether an utterance is engaging or not and response quality metric ranging from 1 to 5. The results from these experiments are shown in Figure 6, and it is shown that none of them are significant.

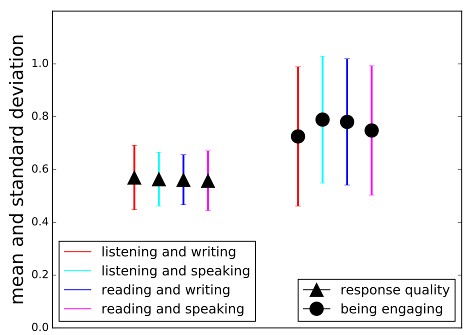

Figure 6: Crowd-workers either listened or read the conversation history and/or wrote or spoke in the system responses during our data collection task. Mean and standard deviation based on the response quality (normalized between 0 and 1) and being engaging for the four conditions

5.6 LDA Topic Modeling

LDA gives topic probabilities for each dialog in the inference phase. These probability distributions are taken into consideration to compute the scoring function as explained in the previous sections. The 6 most probable words for 3 different topics assigned by LDA are:

Topic 1: movie, favorite, talk, be, good, hear; Topic 2: game, favorite, play, video, food, eat; Topic 3: yeah, course, yep, sound, definitely, absolutely.

As can be seen, the LDA model groups phrases with related words into the same topic and some topics are combinations of different topics. For instance, Topic 2 can both contain topics related to ‘food’ and ‘game’. Moreover, as can be observed in Topic 1 and Topic 2, ‘favorite’ is very probable for both of the topics, the same word might be likely to appear in both.

6 Discussion

6.1 On Adopting the Graph Search Model

Early on in the competition we considered using deep seq2seq models as the main approach. To this end we reviewed many-to-many LSTM networks as they recently have shown to be very popular in dialog generation [12, 13]. Initially we thought that we might train the model by using existing large datasets. Subtitle datasets such as opensubtitles[9] were considered but the lack of structure and quality dialogues for conversations posed too many problems. Additionally, some requirements of the challenge such as avoiding profanity, not giving financial advice would be much more difficult if we use other web-sources such as Reddit[10].

In order to build the dialog graph, we started collecting data using crowdsourcing. During a limited time, crowd workers would either listen or read the utterances and reply to them by speaking into the microphone or typing them with a keyboard. The main motivation behind this was to evaluate whether listening and/or speaking would facilitate a more natural conversation. However as shown in Section 5.5 there were no significant differences between the conditions and we continued collecting data only based on reading and writing the responses as this was the most convenient approach.

Several machine learning experiments were initially performed on the dialog graph with the overall aim to generate a model that captured the semantic content of dialog paths. We did not succeed in this regard and created what would become the first version of the graph search model instead.

6.2 Insights

We studied user feedback thoroughly in order to come closer to the end-user to make appropriate, and frequent adjustments to the systems accordingly. The evaluation strategies, not least the web interfaces, had a big impact on how we perceived the challenges in building a conversational system and we gained many insights from them. For example, we discovered that there were many instances where users wanted to use common Alexa skills that the bot didn't support, such as playing a song on Spotify, listening to an Audiobook, or similar. We implemented efficient fixes that gave the users instructions on what to do to be able to use that functionality to handle these occurrences. We also found that many users weren't able to stop the conversation, as the default stop component wasn't triggered when it needed too be, and fixes were added for this as well. Another important insight we gained was that it might not always benefit to inform the user when the system has problems understanding what the user is saying. Early on in the competition, the system was designed to apologize when it did not find an adequate reply and asked the user to talk about something else. However, this behaviour seemed to have a negative effect on the user's experience, as it reinforced the impression of a system with limited understanding. We therefore switched to a strategy where the agent instead of acknowledging the non-understanding (e.g., "sorry, I didn't catch that"), the system used various fallbacks to smoothly take the initiative and jump to another node in the graph (e.g. "cool, so what kind of music do you like?"), based on the current topic or named entities found in the user's utterance. After this change, we experienced a substantial increase in ratings (See Figure 5, as part of the updates marked as “Major system improvement”).

6.3 Challenges and Future Work

Initial Data Sparsity As we wanted the dialog graph to handle a large proportion of the conversation we were restrained by data sparsity, specifically early on when we had not collected much crowdsourced data at all. While challenging at first, this was a conscious choice as we wanted the bot to produce conversation that fell in line with the persona and the current context, something that was being reflected by the dialog graph. An alternative, strictly hard-coded approach could arguably cover a broad set of topics with less time and resources put in, but potentially at the cost of a less interesting research project. Therefore we chose the graph-based approach with the aim of having the bot maintain natural conversations in the long run.

Merging of Nodes A significant problem is that the branching factor for nodes become very high since to any given system utterance the user can generally reply in a multitude of ways. This means

[9] `https://www.opensubtitles.org`

[10] `https://www.reddit.com`

that the number of utterances that crowd workers needs to write responses to scales exponentially. In order to cope with this, nodes in the graph had to be merged to avoid having several occurrences of utterances with similar meaning. As a purely manual task this was feasible at first but as the graph grew it became harder to maintain. Ideally merging would be performed fully automatically, possibly using the scoring function described in Section 4.3. Unfortunately this approach is error-prone and would potentially compromise the property of a child node always being a valid response to its parent node. We did however implement a partially automatic merging system, that merged certain patterns automatically and facilitated manual merging by narrowing the amount of merge candidates to a lower amount. Even so, it is still a time-consuming task that would benefit from even more automation, but the task of merging semantically similar utterances without allowing for any mistakes is simply a very complex problem that for now needs human supervision.

Anonymization To utilize the user interaction data in our crowdsourcing system we first had to make sure that the data was properly anonymized so that no personal data was released to the public. For this purpose, we built an anonymization tool which prioritized utterances in respect to frequency, but utterly each utterance had to be manually deemed anonymous in order to be used. During the course of the competition, Amazon formalized the process for anonymization of user data. This reduced our ability to get updated annotations into the graph as often as we would like. These procedures meant that certain rare utterances which often existed deeper in the graph could not be annotated, reducing the depth of the graph.

Dialog Graph Maintenance Given that the architecture is continuously improving the more data we get, the clear way to further improvement is simply to increase the data flow. As discussed, the system is self-evolving to some extent, which means, at least in theory, that given enough time the bot will perform better and better. The self-evolving aspect of the dialog graph is not fully automatic as of yet however, and there are several potential improvements of value with regard to this. As stated above, the merging process is only partially automatic, ideally we would want this component to be fully automatic. This is not only true for merging, but for maintenance of the graph in general. Because the graph is constantly changing based on the flow of user data it's common to come across outdated conversations. An example of this is conversations that were built during the 2018 FIFA World Cup. During this period the bot uttered questions such as *Are you watching the world cup?* and *What team are you rooting for in the World Cup?*. While appreciated when the tournament was ongoing the questions were very inappropriate when it was over. As users tend to talk about trends and news quite often this can become troublesome if not handled correctly. An automatic, possibly crowdsourced, or at least systematic approach to handle this could be of value, one could for instance tag certain conversation paths in the graph that are only relevant during a certain time period, and when that period is over the paths are removed.

Fallbacks Given that the graph-based approach generated dialogs that were dependent on what the user wanted to talk about we faced a challenge when the users did not take clear initiatives. We tried to handle this by using the fallbacks module. The fallback module was improved significantly during the course of the competition, but we also had additional ideas that didn't make into the live system. The most promising idea was to implement functionality for referring back to previously stated entities or topics, given that we don't have a better response for the moment. Ideally, this would cover not only the active conversation, but also potential previous conversations with the same user. This could potentially be quite powerful as it is something humans do all the time.

Generalizability The generalizability of the graph is also interesting to investigate further. While Section 4.2.1 describes some of our work in the direction of using tags in the dialog graph there is certainly more to explore. Simple fixes in this domain could possibly give rise to higher customer experience, e.g. if we allow for generalizations such that a phrase like *A cute cat* can be swapped with *A cute dog* even if we have never seen the latter.

6.4 Innovation

As discussed in the beginning of this paper, the challenge for conversational chatbots in open domains is how to combine coherent and engaging dialog on the one hand, and scalability of both breadth and depth of the conversation topics on the other. The approach presented in this paper is innovative in the way it allows for the construction of one coherent dialog model, based on the collection of hand-crafted responses provided by a large number of crowd workers. The system is built to automatically detect where more content is needed, with respect to how popular that content is, and post tasks on Amazon Mechanical Turk for workers to fill those gaps. The workers are provided with strict guidance on how to author content and are given a persona to adhere to, while at the same time being encouraged to use their creativity when authoring replies. The Amazon Alexa Prize has given us the opportunity to test this approach on a large scale with actual users, and we think the results are encouraging.

7 Example Interactions

System (S) /User (U)	Response module
S: Hi, this is an Alexa Prize Socialbot. How are you doing?	Introduction phase
U: I'm good how are you?	
S: Good to hear! I'm doing very well, thank you. I don't think we have met before, would you like to get to know each other?	Introduction phase
U: Sure	
S: Great! What is your name?	Introduction phase
U: My name is Peter what is your name?	
S: Nice to meet you Peter! Sorry, but due to this being a competition I am not allowed to tell you my name even though I really want to. Tell me, what's your favorite thing in the entire world?	Introduction phase
U: My dog	
S: What breed of doggo companion do you have?	Graph Search
U: It's a mixed breed	
S: Alright. Did you know that you can talk to me about music or food for instance?	Fallback
U: Ok what do you prefer?	
S: I like to hear about others life experiences how about you?	Graph Search
U: Ok tell me about you	
S: Me, I am an advanced chatbot always looking for new experiences. I like to talk about everything. Is there anything you would like to know?	Graph Search
U: What's your favorite food?	
S: While I appreciate great food, my good taste is better reflected in the company I keep.	Graph Search
U: Let's chat about politics	
S: Did you vote in the last election?	Graph Search
U: yes did you?	
S: I cannot vote, I'm not a human. Did your candidate win?	Graph Search
U: I don't want to say	
S: Okay, It's nice talking to you. Now I feel like talking a bit about sports or music.	Fallback
U: Let's talk about science	
S: Great, did you know that the hottest planet in the solar system is Venus, with an estimated surface temperature of 864 Fahrenheit?	Evi
U: I didn't	
S: A lot of advancements in space travel are happening right now. If you could travel to space and had to choose, would you go to the international space station or mars?	Graph Search
U: Mars if I could return	
S: I see, Speaking of science and sci-fi stuff. Would you like to go to Mars in the future? Or the moon maybe?	Fallback
U: Yes I just told you	
S: Sweet! Tell me, did you like science classes in school?	Fallback
U: Yes I did	
S: I love science! Without it I wouldn't be talking to you right now!	Graph Search
U: Do you have a pet	
S: I don't have any pets. I used to have a few bugs, but they kept getting squashed.	Evi
U: Do you like dogs?	
S: I don't have a pet preference, but they are cute and fuzzy. Are you a dog person?	Graph Search
U: Yes, I am	
S: Do you own a dog?	Graph Search
U: Yes, I have 3 dogs	
S: What kind of dogs do you like?	Graph Search
U: I like all kinds of dogs	
S: Some examples of dog breeds are: dachshund, Alaskan Malamute, Chihuahua, Cavalier King Charles Spaniel, Airedale terrier, bichon frise, Australian Shepherd, Bulldog, Cane Corso and American Akita.	Evi
U: Alexa, are you happy?	
S: I don't have much emotion but I heard it was a good feeling. Are you happy?	Graph Search
U: Yes, I am	
S: That is great to hear. Everyone should be happy with the body they were born in. What is making you happy?	Graph Search
U: Talking to you	
S: You may talk to me about anything your heart desires.	Graph Search
U: Play music	
S: You are currently in chatting mode where I can only focus on our conversation. If you want to stop chatting and do other things, just say Alexa stop.	Common Interactions
U: Should I invest in bitcoin	
S: I'm afraid I'm not allowed to give advice. Let's talk about something else.	Safety Filter
U: Why not?	
S: I can't do explanations yet.	Evi
U: Then what can we talk about?	
S: Well I'm open to just about anything	Graph Search
U: Do you like rainy days?	
S: I don't like water so much, but I don't have true preferences about the weather. Do you like rainy days?	Graph Search
U: What?	
S: I said, I don't like water so much, but I don't have true preferences about the weather. Do you like rainy days?	Repeat Generator

Table 5: The following example interactions were created by the authors using the system in order to demonstrate various aspects of the system.

Acknowledgments

This work was funded by Amazon. We would like to thank the Alexa Prize staff for guidance throughout the competition. Thank you Hannes Kindbom for being a great chief marketing officer, and thank you David House for being the instructor in our crowdsourcing video.

References

[1] A. Ram, R. Prasad, C. Khatri, A. Venkatesh, R. Gabriel, Q. Liu, J. Nunn, B. Hedayatnia, M. Cheng, A. Nagar, E. King, K. Bland, A. Wartick, Y. Pan, H. Song, S. Jayadevan, G. Hwang, and A. Pettigrue, "Conversational AI: The Science Behind the Alexa Prize," pp. 1–18, 2018. [Online]. Available: http://arxiv.org/abs/1801.03604

[2] D. Britz, A. Goldie, T. Luong, and Q. Le, "Massive Exploration of Neural Machine Translation Architectures," *ArXiv e-prints*, Mar. 2017.

[3] I. V. Serban, A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau, "Building end-to-end dialogue systems using generative hierarchical neural network models." in *AAAI*, vol. 16, 2016, pp. 3776–3784.

[4] H. Fang, H. Cheng, E. Clark, A. Holtzman, M. Sap, M. Ostendorf, Y. Choi, and N. A. Smith, "Sounding board–university of washington's alexa prize submission," *Alexa Prize Proceedings*, 2017.

[5] J. Pichl, P. Marek, J. Konrád, M. Matulík, H. L. Nguyen, and J. Šedivỳ, "Alquist: The alexa prize socialbot," *arXiv preprint arXiv:1804.06705*, 2018.

[6] I. Papaioannou, A. C. Curry, J. L. Part, I. Shalyminov, X. Xu, Y. Yu, O. Dušek, V. Rieser, and O. Lemon, "Alana: Social dialogue using an ensemble model and a ranker trained on user feedback," *Alexa Prize Proceedings*, 2017.

[7] S. Jafarpour, C. J. Burges, and A. Ritter, "Filter, rank, and transfer the knowledge: Learning to chat," *Advances in Ranking*, vol. 10, pp. 2329–9290, 2010.

[8] O. Vinyals and Q. V. Le, "A neural conversational model," *CoRR*, vol. abs/1506.05869, 2015. [Online]. Available: http://arxiv.org/abs/1506.05869

[9] "Cleverbot," https://www.cleverbot.com/, 2018, [Online; accessed 2018-08-14].

[10] J. Wakefield, "Microsoft chatbot is taught to swear on twitter," *BBC*, Mar 2016. [Online]. Available: https://www.bbc.com/news/technology-35890188

[11] B. Wilcox, "Chatscript," https://github.com/bwilcox-1234/ChatScript, 2015, [Online; accessed 2018-08-14].

[12] O. Vinyals and Q. Le, "A neural conversational model," *arXiv preprint arXiv:1506.05869*, 2015.

[13] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Józefowicz, and S. Bengio, "Generating sentences from a continuous space," *CoRR*, vol. abs/1511.06349, 2015. [Online]. Available: http://arxiv.org/abs/1511.06349

[14] B. Krause, M. Damonte, M. Dobre, D. Duma, J. Fainberg, F. Fancellu, E. Kahembwe, J. Cheng, and B. L. Webber, "Edina: Building an open domain socialbot with self-dialogues," *CoRR*, vol. abs/1709.09816, 2017. [Online]. Available: http://arxiv.org/abs/1709.09816

[15] J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan, "A diversity-promoting objective function for neural conversation models," *arXiv preprint arXiv:1510.03055*, 2015.

[16] I. V. Serban, A. Sordoni, R. Lowe, L. Charlin, J. Pineau, A. C. Courville, and Y. Bengio, "A hierarchical latent variable encoder-decoder model for generating dialogues." in *AAAI*, 2017, pp. 3295–3301.

[17] R. Lowe, N. Pow, I. Serban, and J. Pineau, "The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems," *CoRR*, vol. abs/1506.08909, 2015. [Online]. Available: http://arxiv.org/abs/1506.08909

[18] X. Zhou, D. Dong, H. Wu, S. Zhao, D. Yu, H. Tian, X. Liu, and R. Yan, "Multi-view response selection for human-computer conversation," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 372–381.

[19] Y. Wu, W. Wu, C. Xing, M. Zhou, and Z. Li, "Sequential matching network: A new architecture for multi-turn response selection in retrieval-based chatbots," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017. [Online]. Available: http://dx.doi.org/10.18653/v1/P17-1046

[20] J. Orkin and D. Roy, "Automatic learning and generation of social behavior from collective human gameplay," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 385–392.

[21] C. Breazeal, N. DePalma, J. Orkin, S. Chernova, and M. Jung, "Crowdsourcing human-robot interaction: New methods and system evaluation in a public environment," *Journal of Human-Robot Interaction*, vol. 2, no. 1, pp. 82–111, 2013.

[22] E. Filatova, "Irony and sarcasm: Corpus generation and analysis using crowdsourcing," in *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, 2012.

[23] I. Leite, A. Pereira, A. Funkhouser, B. Li, and J. F. Lehman, "Semi-situated learning of verbal and nonverbal content for repeated human-robot interaction," in *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, ser. ICMI 2016. New York, NY, USA: ACM, 2016, pp. 13–20. [Online]. Available: http://doi.acm.org/10.1145/2993148.2993190

[24] T.-H. Wen, D. Vandyke, N. Mrksic, M. Gasic, L. M. Rojas-Barahona, P.-H. Su, S. Ultes, and S. Young, "A network-based end-to-end trainable task-oriented dialogue system," *arXiv preprint arXiv:1604.04562*, 2016.

[25] J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan, "A Persona-Based Neural Conversation Model," *arXiv*, p. 10, 2016. [Online]. Available: http://arxiv.org/abs/1603.06155

[26] S. Zhang, E. Dinan, J. Urbanek, A. Szlam, D. Kiela, and J. Weston, "Personalizing dialogue agents: I have a dog, do you have pets too?" *arXiv preprint arXiv:1801.07243*, 2018.

[27] T.-H. K. Huang, J. C. Chang, and J. P. Bigham, "Evorus: A Crowd-powered Conversational Assistant Built to Automate Itself Over Time," 2018. [Online]. Available: http://arxiv.org/abs/1801.02668{%}0Ahttp://dx.doi.org/10.1145/3173574.3173869

[28] M. Honnibal and I. Montani, "spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing," *To appear*, 2017.

[29] C. H. E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf*, 2014.

[30] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[31] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National Academy of Sciences*, vol. 101, no. Suppl 1, pp. 5228–5235, 2004.

[32] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[33] X.-H. Phan and C.-T. Nguyen, "Gibbslda++: Ac/c++ implementation of latent dirichlet allocation (lda)," 2007.

[34] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.