Improving Task-Oriented Dialogue Systems In Production with Conversation Logs

Alon Jacovi*
IBM Research
Bar Ilan University
alonjacovi@gmail.com

Ori Bar El*
IBM Research
mroribarel@gmail.com

Ofer Lavi
IBM Research
oferl@il.ibm.com

David Boaz
IBM Research
davidbo@il.ibm.com

David Amid
IBM Watson
davida@il.ibm.com

Inbal Ronen
IBM Research
inbal@il.ibm.com

Ateret Anaby-Tavor
IBM Research
atereta@il.ibm.com

ABSTRACT

In this work we propose a solution to a significant limitation of task-oriented dialogue systems — their inability to learn and improve over time during deployment. Although current popular task-oriented systems are implemented as rule-based execution graphs, the available solutions for improvement incorporate neural network modules, either fully or partially, despite the poor performance of neural architectures for the task-oriented use-case. We present an algorithm to modify the graph-based system directly, in a manner which improves the system automatically and is simultaneously easy to understand by the system expert. To our knowledge, this is the first method of this type towards automatically improving a dialogue system's coverage in production, without additional explicit labels. Though the system is still evidential, our experiments already show promising results in its ability to usefully modify an existing dialogue system, while improving its coverage.

CCS CONCEPTS

• **Computing methodologies** → **Learning from demonstrations**; **Rule learning**; *Discourse, dialogue and pragmatics*; • **Human-centered computing** → **Natural language interfaces**.

KEYWORDS

dialogue systems, task oriented, closed domain, virtual agent, rule based systems, machine learning

ACM Reference Format:
Alon Jacovi, Ori Bar El, Ofer Lavi, David Boaz, David Amid, Inbal Ronen, and Ateret Anaby-Tavor. 2020. Improving Task-Oriented Dialogue Systems In Production with Conversation Logs. In *Proceedings of KDD Workshop on Conversational Systems Towards Mainstream Adoption (KDD Converse'20).* ACM, New York, NY, USA, 10 pages.

*Both authors contributed equally to this research.

KDD Converse'20, August 2020,

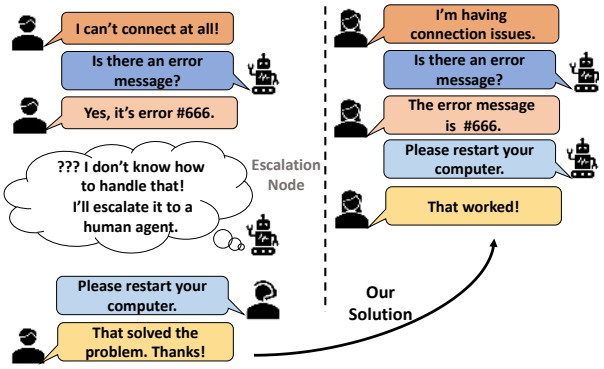


Figure 1: Example of an escalation log and how we adopt it in our solution. The dialogue system fails, causing an escalation to a human who resolves the case; The system then learns from the human's response for similar cases.

1 INTRODUCTION

Dialogue systems, or virtual assistants, are automated systems for interacting with users through a natural language interface. Task-oriented[1] dialogue systems are not only concerned with maintaining coherent interaction with another party (e.g., chit-chat agents, or chatbots), but also leading the interaction towards some goal [8, 11]. These systems have a variety of useful applications, such as customer support [35], restaurant or hotel reservation [24], online shopping [34], and many others.

Recent advances in Natural Language Understanding (NLU), via neural networks, have shown promise to facilitate drastic improvements in such virtual task-oriented dialogue agents [1] — as a major bottleneck in the past has been correct interpretation of the user's natural language utterances. However, the scope of these dialogue systems is still limited by their inability to handle new types of

[1] Also referred to as "goal-oriented" or "closed-domain".

interactions after deployment (e.g., new software product in IT support, or new categories in online shopping) [16].

The dominating task-oriented dialogue systems follow a rule-based architecture where machine learning NLU techniques interpret the user utterances (Figure 2), with an *execution graph* backbone for the dialogue path management [8]. Modelling such a system requires expertise in both the backbone system and the domain the system is planned to operate in (i.e., the concrete use case). This combined knowledge of both the use-case and the system engineering is rare, and requires training. Consequently, as the dialogue management system is rule-based, improving the system's performance based on post-deployment usage requires manual updates by such an expert, as well.

Often, the dialogue management backbone is based on a dialogue graph (Figure 3B). Each node in the graph represents a dialogue state, and each edge a possible transition from one state to another according to the user's utterances and the *condition* derived from it by the NLU system (Figure 2). Changing the dialogue system's behavior involves altering the dialogue system's structure and transition table. But how can we acquire supervision for the changes necessary for these improvements?

Towards this end, we point to a key property of our use-case: Virtual assistants which are the topic of this work are deployed as part of customer support centers. They work in tandem with a fallback to human agents in cases of failure — as a way of maintaining a sufficient service level to customers (users). At any point during the virtual assistant to user interaction, a failure can occur, either when the virtual assistant detects its inability to continue, or when the user directly requests the escalation to a human agent. In these cases, the human agent will assume control of the interaction to properly assist the user. Naturally, a *record* of such interactions is collected during the deployment of the support system, and is used by an expert to manually modify and improve the automatic dialogue system. We refer to these records as *escalation logs*, detailing interactions where the dialogue system assumed initial control, subsequently failed, and control was escalated to a human agent to resolve the case (Figure 1).

In this work, we propose to leverage these escalation logs for completing missing functionality in the dialogue system *automatically*, by introducing new nodes to the dialogue execution graph. A notable attribute of the dialogue systems discussed in this work, based on execution graphs, is their *human-readability*, as they are easy to read and understand by humans (since they are actively designed by humans). Thus, modifying them automatically requires maintaining the system's human-readability by proposing modifications which are also rule-based. This enables the dialogue system developer to thoughtfully handle these updates — adapt them and alter them as necessary. As these systems are designed to be deployed and serve a large sector, this will allow the developer a sufficient degree of confidence in the automatic modifications to allow their usage in production. We are addressing this aspect in the design of our algorithm and assess some readability measures of its results.

The contributions of this paper are three-fold: First, we formulate the node-completion problem for the dialogue execution graph based on escalation logs; Next, we propose a method for automatically deriving node transition rules based on user-to-human escalation logs; Finally, we present an automatic evaluation setup in order to assess the quality of the suggested updates to the solution, which can also serve other future dialogue system methods in this area.

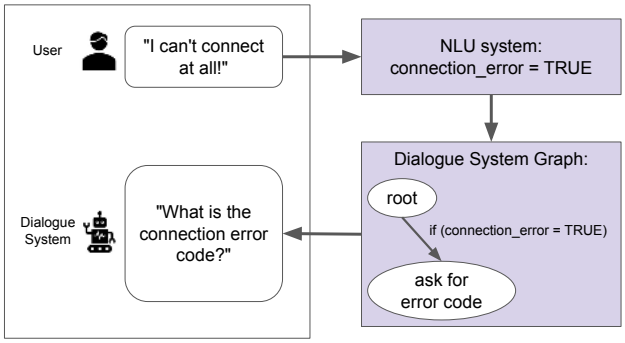


Figure 2: A schema for one step (response to a user utterance) in the dialogue system [8]. Following the user's utterance, the NLU system interprets it to derive various values and flags. This serves the dialogue system to decide on the response.

The rest of this paper is structured as follows: In Section 2 we provide background on different types of dialogue systems and scope the discussion to the more prevalent type we deal with in this paper. Then, in Section 2.2 we establish the importance of improving such dialogue systems based on post-deployment execution logs. In Section 3 we introduce our solution for automatically improving these systems by means of learning from logs, a solution which we provide implementation details for in Section 4. We evaluate our solution in Section 5 and sum up with a short discussion and conclusions in Section 6.

2 BACKGROUND: IMPROVING DIALOGUE SYSTEMS IN PRODUCTION

We give a brief overview on learning-based methodologies for improving and updating dialogue systems without manual annotation by an expert.

2.1 Terminology and Notation

Execution Graph Dialogue System (Figure 3B). We focus on the prevalent dialogue systems where the system is a directed *"execution graph"*, in which each node edge represents a binary *decision function* (or *condition*) and an *action*. The decision function, based on the current state of the environment (conversation), results in a decision on whether to perform the action. If so, a change in the environment is observed as a result of the action, and the execution flow proceeds to the children of the node, in a pre-defined order. If the condition is not satisfied, the action is not performed, and the execution flow proceeds to the next sibling of the current node. The action to perform may be a communication with the user, or a concrete action to perform to help the user, and the observable result will be the user's response to the action.

Escalation Logs (Figure 1). The core supervision to drive learning in production is collected in escalation logs — logs of interactions where the deployed system assumed initial control of handling the case, and subsequently it failed to complete the goal of the interaction. This resulted in escalation of the case to a human agent, who properly handled the case to its conclusion. In this work, we propose a method to utilize the human agent's handling of the dialogue system's failure in order to improve the dialogue system.

2.2 Motivation

In this section we elaborate on the core motivation behind this work — namely, the answer to the question: *Why is it valuable to develop a method of updating dialogue systems after their deployment?* We give two central answers, detailed below.

2.2.1 Distribution Shift Over Time. The main motivation is simple indeed, and uncontroversial: Even in the event where the initially manually designed dialogue system is perfect for its use case, as time goes by and new capabilities are required, we would like the system to be able to manifest them automatically. This motivation also shares common themes with the areas of *lifelong machine learning* [27] and *never-ending learning* [7].

As an example, consider the case of a technical customer support virtual agent — which attempts to help incoming users with technical issues and requests regarding a specific software product. The virtual agent, although properly designed at deployment time, must be continuously augmented with additional information to reflect updates in the software product, as these updates introduce new capabilities and issues.

2.2.2 Reference Logs Are Naturally-Occurring. Another key motivation relates to the ease of obtaining these reference escalation logs. Evidently, the system has been expertly designed to be used in some practice, and thus, it will be deployed. As a result, instances of escalated conversations where the bot has failed will be gathered. These reference conversations can be considered "free": they will exist during production phase *by default*, and if they can be utilized, no additional effort is necessary to gather supervision for the improvement of the deployed system.

Unfortunately, as explained in Section 2, there is currently no method available for making use of this supervision to improve a non-neural dialogue system (the prevailing type of virtual agents in task-oriented settings). In other words, there exists a gap between the relative ease of obtaining reference supervision for the improvement of the currently deployed solution and the lack of available techniques to make use of it.

2.3 Related Work

2.3.1 Execution Graph Solutions. An execution graph [18] is one of the most popular methods for modeling task-oriented dialogue systems. The vast majority of solutions of this type are created manually by an expert [8], and to our knowledge, after being deployed, they are either static, or manually updated by an expert. One notable exception is by Volkova et al. [31], which attempts to create an initial graph-based model by using explicit natural-language instructions on how the execution graph should act. This method can be used to update the graph by redoing the process with additional instructions. Additionally, [23] have proposed a system designed for multi-domain sets of slot values in order to remain scalable to new domains of conversations (we elaborate on slots later).

2.3.2 Neural End-to-End Solutions. Recent advances in deep learning has caused a surge in proposed neural solutions for dialogue systems in the open-domain chit-chat setting [15, 16, 21]. Unfortunately, although these end-to-end models can be improved relatively easily using reference conversation logs, current solutions are ill-equipped to deal with the challenging setting of task-oriented conversation — where the automatic solution must achieve some purpose at the end of the interaction, via a natural language interface and performing actions — and the insufficient quantity of data which can be gathered[2]. Typically these neural solutions involve a component of generating responses [20, 32] or ranking and retrieving them from data [2, 4, 29, 33].

2.3.3 Hybrid Solutions. As previously mentioned, neural models under-perform in task-oriented settings. However, the standout quality of these models is their ability to learn by their design from reference conversation logs. As such, hybrid models have been proposed to combine the strengths of an execution graph backbone with a neural fall-back which can learn to adapt and improve after deployment. For example, Tammewar et al. [28] propose a hybrid model in which every decision of the execution graph has a neural fall-back in case of no appropriate response.

Although these models are indeed able to learn from escalation logs after deployment, in truth the only component which is able to learn is the neural model. As mentioned before, these models are as of yet unconvincing in their ability to uphold the task-oriented use-case — due to their inability to rigorously conform to completing the goal of the conversation, and requiring a significant amount of data to learn on any level.

Another alternative to the neural fall-back is a hybrid model that offers redirection of the misunderstood utterance to a search engine and returning its result, relying on an up-to-date search index such as the *search skill* described in [26]. However, a search user experience is substantially different from a conversation one.

2.4 Conclusion

We have discussed three possible solutions for task-oriented systems, and their ability to learn automatically from reference logs after deployment. Specifically, while execution graph-based models are the most robust solutions, they are also rigid and require updates by a manual expert to be continuously improved. Neural models go to the other extreme, and are able to learn freely at any point by optimizing their performance against reference logs. However their overall performance at the task-oriented use-case is severely lacking in comparison to the execution graph based models.

In order to bridge the gap, hybrid models have been proposed to embody the best of both worlds, such that they employ an execution graph backbone and a neural fallback in case of failure. However, the only component which is able to learn and improve in these models is the neural component — which is anyway of negligible

[2]While out of the scope of this work, neural models indeed dominate the open-domain chit-chat settings which don't suffer from these constraints [1].

value in the overall usefulness of the model — and so they suffer the same issues as all of the previous solutions.

3 OUR SOLUTION

We elaborate on our proposed solution in order to concretely improve an existing execution graph dialogue system, by using reference escalation logs, obtained after deployment of the existing virtual assistant.

The procedure is conceptually divided into five steps. At the end of the procedure, the algorithm recommends new edges and nodes (composed of decisions and actions) to be integrated into the execution graph currently in production. These new nodes can be integrated as-is into the execution graph, to be evaluated in a test environment, or they can be verified by an expert before being integrated in order to guarantee their relevance before deployment.

3.1 Step 1: Gathering Failure Points

As mentioned in Section 2.2, to update the existing execution graph, we utilize escalation logs obtained following its deployment.

(1) The before-escalation section of the log describes the dialogue between the user and the dialogue system and ends at a *failure point*. A failure point in a conversation is the point where the control is escalated to a human agent. This conversation corresponds to a single path in the dialogue execution graph, terminating at some node we refer to as the *escalation node* — a graph node from which some failure points escalated to a human agent. Figure 3A illustrates a single escalated conversation. The dialogue system understood that the user wishes to transfer money and escalated to a human agent in the next node.
(2) The after-escalation section of the log describes the interaction from the failure point on, occurring between the user and the human agent. Since this part of the conversation is external to the dialogue system, there is no path corresponding to it in the execution graph (Figure 3B).

Our goal is to derive new nodes to attach to the execution graph at the escalation node, so that failure points corresponding to that node, occurring in multiple conversations, will be handled, or at minimum delayed by an addition step in the execution graph. For a single conversation we look at the execution path up to the escalation node, and at the first response of the human agent after the failure point. In order to generalize we gather multiple conversations that were escalated at that specific escalation node. We thus obtain a set of conversations along with their matching path up to the escalation node, and the appropriate response for this conversation as given by the human agent. We refer to these responses as *gold responses*.

3.2 Step 2: Clustering Gold Responses Into Response Types

Given the collection of human agent responses we obtained in the previous step, it is necessary to divide this collection into categories: Although all of these conversations passed through the same escalation node in the execution graph, they have each possibly originated from different paths, and thus each of the human agents' responses may be different based on the context of the interaction. For this reason, we cluster the human agent responses into *response types* based on semantic similarity. Figure 4 illustrates clustering of multiple conversations based on the agent's responses into 3 response types.

In the case of textual responses, we utilize a neural model to encode the text in a continuous embedding space [14] for clustering. The clustering algorithm attempts to divide the human agents' responses into different response types.

3.3 Step 3: Affixing Actions to Response Types

Each response type will be attributed by a concrete action — such as a text message, value retrieval from a database, and/or miscellaneous actions. This representative action can be derived in one of multiple possible methods:

(1) The action can be chosen by some metric (such as quantity of similar occurrences in the cluster) from among the actions in the response type.
(2) The action can be chosen as the closest response to the centroid of the cluster (Figure 5).
(3) In the case of a text message, the response can be generated via some text generation component by utilizing the collection of text responses in the cluster for the generation process.[3]

3.4 Step 4: Deriving Boolean Conditions

Our next goal is to derive *boolean conditions* that will correctly map a conversation to its response type, and trigger the chosen action. In dialogue systems that use an execution graph as their dialogue management backbone this is equivalent to adding one node per response type with a decision function that takes the dialogue state and context as its input.

In Figure 6 we illustrate eight conversations clustered by the agent's response into three response types. Each cluster is marked by a different type of line (solid, dashed, dotted). Within each cluster, every conversation holds its own different dialogue state captured when the conversation passed through the escalation node. The table illustrates the state of each conversation represented as a set of features, together with the assigned cluster for each conversation. A decision function is then learned, taking the dialogue state as input to discriminate between the three clusters. In the illustration we can see three boolean conditions taking into account the payment amount and customer VIP flag to differentiate between the three response types based on the dialogue state. Note that the boolean conditions ignore the account number feature.

3.5 Step 5: Recommending New Nodes

At the final step of the procedure, various nodes are derived to model the responses of human agents at various failure points. This step attempts to rank these nodes so that only a confident subset of the suggested nodes will be recommended for integration in the deployed dialogue system. This is done for two reasons:

[3] Within the scope of this work, we do not consider the text generation case.

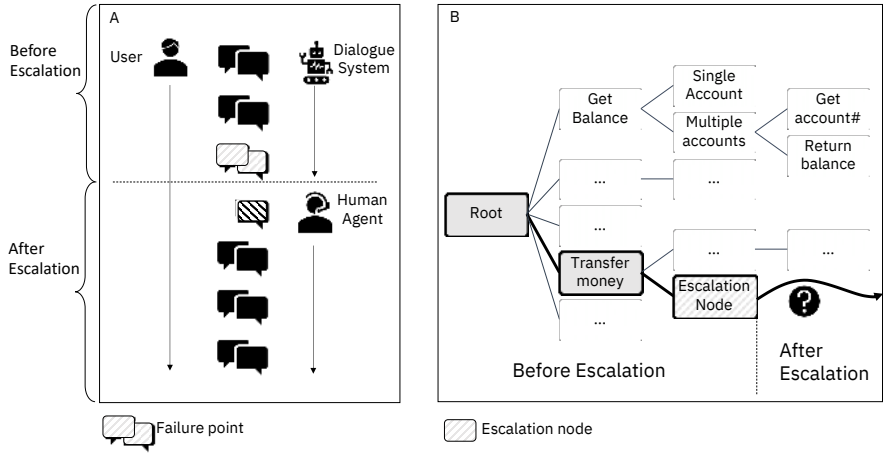


Figure 3: Step 1 of our solution (see Section 3.1).

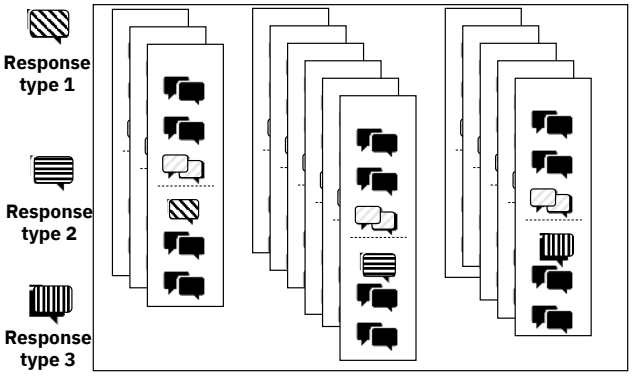


Figure 4: Step 2 of our solution (see Section 3.2).

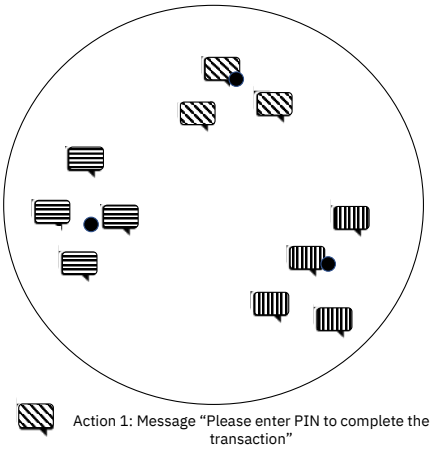


Figure 5: Step 3 of our solution (see Section 3.3).

(1) By choosing a specific amount k of nodes as the top-k nodes in the recommendation ranking, the balance between *precision* and *recall* can be controlled: It is up to the expert to prioritize quality of responses at the failure points versus the potential coverage of failures.
(2) In the event that the expert will be interested in verifying the suggested nodes before they are integrated in the deployed dialogue system, to guarantee their validity, the procedure must filter the nodes by confidence to alleviate the workload of the expert.

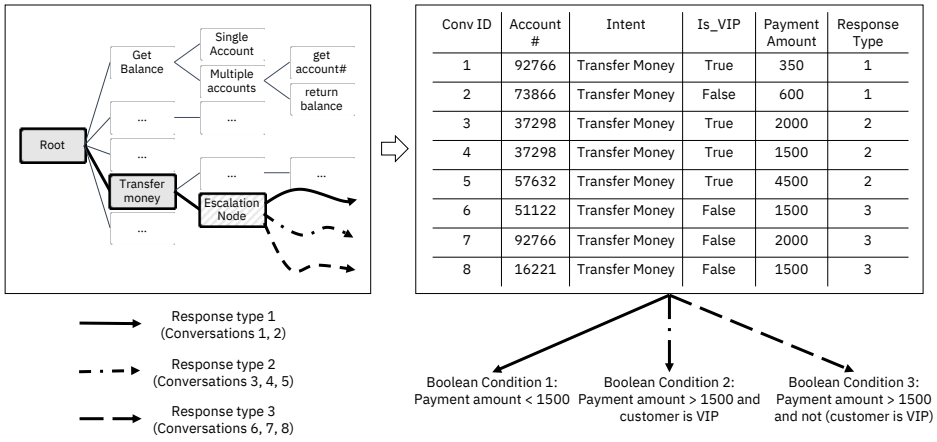
We consider the quality of the suggested nodes (and specifically their conditions) via several heuristics that conform to notions of human-readability[4] for two main purposes: (i) Decision functions that are easier to understand will be preferred, as the expert may still attempt to understand them and verify their functionality to gain confidence in their integration in the deployed product; (ii) The human-readability of the boolean conditions can be viewed as regularization to mitigate overfitting.

3.6 Solution Summary

We propose a five-step procedure for improving an execution graph's ability to handle failure points by using escalation logs as the source of supervision. To our knowledge, this is the first method of this type towards automatically improving a dialogue system's coverage after deployment, without labels that require external feedback — outside of the already available escalation logs — and without manual annotation by an expert. As mentioned, the procedure requires a collection of escalation logs and results in a set of new nodes to be integrated in the current dialogue system's execution graph. These nodes are ranked by some metric, and can be further verified by an expert with minimal overhead to guarantee their behavior for a deployed model.

At the end of the integration of the new nodes, the execution graph will be able to progress an additional step beyond what were considered its failure points previously, thus increasing its coverage. Once the new execution graph is deployed, more escalation logs can be gathered to iteratively improve the system by repeating the procedure.

[4]Such heuristics may include the length of the decision function, the amount of *nesting* (such as "*A or (B and C)*"), the number of negation elements in the function, and so on.



Conv ID	Account #	Intent	Is_VIP	Payment Amount	Response Type
1	92766	Transfer Money	True	350	1
2	73866	Transfer Money	False	600	1
3	37298	Transfer Money	True	2000	2
4	37298	Transfer Money	True	1500	2
5	57632	Transfer Money	True	4500	2
6	51122	Transfer Money	False	1500	3
7	92766	Transfer Money	False	2000	3
8	16221	Transfer Money	False	1500	3

Figure 6: Step 4 of our solution (see Section 3.4). "Decision function" refers to boolean condition.

4 IMPLEMENTATION

To verify our suggested approach we implemented each of the 5 steps in our solution on top of IBM Watson Assistant (WA) [13]. However, we stress that while we exemplify our approach on top of IBM Watson Assistant, it can be comfortably generalized to other popular competing execution graph systems, such as Google Dialogflow [25] and Microsoft Bot Framework [5].

WA uses an execution graph as its dialogue management backbone. The graph is designed by the system's author such that at each visited node, the system interprets a user's utterance in the context of the current conversation using natural language understanding and chooses the appropriate transition to the next node based on the execution graph design and the current dialogue state.

The dialogue state is encoded with a set of *contextual variables* characterizing the user's intents (e.g., opening a new account), identity (e.g., account number or country of origin) and relevant details from the user utterances (dates, times, names, etc.). Some of these contextual variables are extracted by WA automatically from the user's utterances and others are "injected" from outside the system (e.g. the account number of the logged in user). Additional variables can be calculated based on the values of exitsing ones during the conversation.

Each node in WA's execution graphs contains a boolean *condition* over the set of contextual variables and an *action* (e.g., a system response). When the system arrives at a specific node during a conversation, the next action in the conversation is chosen to be the action attached to the first child node whose condition is satisfied.

Below we describe our implementation in accordance with the 5 steps of Section 3):

(1) *Step 1: Gathering Failure Points.* Escalation nodes in WA's execution graph are nodes from which dialogues were escalated to a human agent. These nodes are in fact sink nodes for all points in conversations that did not satisfy any condition of the children of the current node. For each escalation node we gather all conversations that were escalated in that node.

(2) *Step 2: Clustering Gold Responses Into Response Types.* We first embed the agent's response following the escalation in a continuous space. For this purpose we use BERT [12] based embedding. Specifically, we use the `[CLS]` token which is the output of employing the BERT model over the responses. We then cluster the resulting vectors using the Mean Shift [10] clustering algorithm[5].

(3) *Step 3: Affixing Actions to Response Types.* Each node in the execution graph is the combination of both an entry condition and an action to follow. Each cluster from the previous phase is associated with a centroid. For the recommended nodes' actions we use the human response of the nearest neighbor to the centroid inside the cluster.

(4) *Step 4: Deriving Boolean Conditions.* Every point in the conversation is associated with a dialogue state constituting a feature vector defined by the values of its contextual variables. For each cluster obtained in step 2, we train a binary decision tree classifier over the dialogue state at the escalation node. The label of each conversation is 1 (positive) if the decision tree associated it with the cluster, and 0 (negative) otherwise. Specifically, we used the implementation offered by `scikit-learn` [19]. This decision tree is then converted into a boolean expression by collapsing sibling sub-trees as *or* and collapsing parent-children sub-trees as *and*. Optionally, the decision tree or boolean expression can be pruned or simplified to increase generalization and readability [3, 17]. We implemented pruning using the `min-leaf-size` parameter of `scikit-learn`. Notably, the decision trees are trained to classify between a given cluster and *all* other clusters, mitigating any issue with order-dependent movement along the execution graph.

(5) *Step 5: Recommending New Nodes.* Clustering high-dimensional vectors is likely to result in a long tail of very small clusters pertaining to outlier responses. To mitigate this, we bound the minimum size of a cluster (as a percentage of the number of responses) to be considered for new node recommendation. Our recommendations constitute k nodes resulting from the k largest clusters.

[5] Although any other clustering algorithm is applicable, we chose Mean Shift since it does not require the number of clusters to be predefined.

5 EVALUATION

Qualitative evaluation of dialogue systems, and particularly task-oriented systems, is a very challenging open problem [11]. Deriu et al. [11] emphasize the need for automated evaluation methods as collecting human judgement for the quality of a dialogue system is laborious and costly. To this end, we devised an automated evaluation method for our solution which does not involve measurement of the dialogue system performance during deployment, but rather utilizes current dialogue systems' logs without the escalation to human agents for evaluating the method itself.

Instead of adding a new node and evaluating its quality, we take an existing dialogue system as reference and destructively modify it by choosing a node (which we refer as *simulated escalation node*) and removing all its outgoing nodes (descendants) from the execution graph. We then use our method to predict the removed outgoing nodes, and compare the behavior of the system prior to the removal with its behavior after adding the predicted nodes. We then measure the quality of our recommendations in an automated manner. A "high quality" node should capture a previously unhandled case, properly act upon it, and be human-readable. Our automatic evaluation is based on the following observation: in the original (unmodified) graph, the removed nodes induce a partition of the conversations that went through the simulated escalation node. We call this partition the *reference partition*.

Similarly, the predicted nodes induce a partition on the same set of conversations. The nodes' conditions and execution order may not necessarily resemble the original ones, but the functionality of the system should be preserved. This preservation can be measured by the level of similarity of the two partitions, the one induced by the removed nodes, and the one induced by the predicted nodes. Our simulated escalation node can be viewed as an escalation node in the human agent escalation case. Once we remove the outgoing nodes, we consider only the conversation log before escalation, ignoring the dialogue state and the continuation of the paths in the execution graph.

We also use the original node conditions to assess the quality of our recommendations for example by comparing the length of the recommended conditions to the original ones in terms of the number of variables in the condition.

Our experiments include two evaluation methods: (1) Automatic evaluation of our solution to assess the quality of the partition and the readability of the conditions. We experiment with different hyperparmeters and implementations of the components in our solution. This evaluation is performed on an internal dataset, using our method for simulating escalation nodes. (2) Human evaluation of the recommended conditions and clustering. This evaluation is performed on a public dataset.

The evaluation method proposed in this paper is standalone, and is neither contingent upon the dialogue system nor the embedding, clustering, and condition inference techniques.

5.1 Datasets

For our evaluation we use different datasets for each evaluation method. For the automatic evaluation we use an internal real-life (non-public) dataset from the banking domain. The dataset includes 7605 real-world conversations of users with a WA dialogue system without escalations to human agents during a period of 10 days of operation. Each conversation includes an average of 6.05 turns between a user and the dialogue system. The execution graph includes 135 intents with 62 entities. It has 1528 nodes and an average depth of 2.59. The dataset handles several customer service issues, such as opening a new account and transferring money. We use this dataset by simulating escalation nodes as explained above. We consider only escalation nodes with at least 50 conversations passing through them. This results in a total of 39 escalation nodes with an average of 535.98 conversations passing through each of them (stdev: 760.12, min: 55, max: 3386) and an average of 2.46 child nodes each. The feature vector used for training the decision tree in step 4 of our solution includes 1070 features.

In order to experiment with a different type of data, which reflects a prevailing use case of task oriented dialogue systems, we use the MultiWOZ dataset. [6]. The dataset contains 10,000 conversations of humans in multiple domains (including hotels, taxi and restaurant booking). Each conversation in MultiWOZ is labeled using contextual variables similar to those of WA. Moreover, each agent response is labeled with the actual agent's action. For example, many agent responses ask the user, in different ways, to specify a certain area. All these responses are labeled as an "area" action in the dataset. Despite the fact that MultiWOZ does not include a built-in backbone execution graph, we simulated the state of conversations by querying the agent actions' labels.

5.2 Experimental Setup

As we noted earlier, our solution is to the best of our knowledge the first to tackle the problem of improving dialogue systems' coverage in production, without explicit external feedback. We thus have no baselines to compare our solution to.

Our solution contains (in step 4) a decision tree (DT) classifier. We compare it to reference solutions employing other classification models — Random Forest (RF) and the state-of-the-art XGBoost (XGB) [9]. Note that both of these models do not fit our complete solution, as they do not offer an interpretable mechanism from which node conditions can be derived. Nevertheless we use these references as an unrealistic upper-bound for the classification part.

To evaluate various aspects of our solution we experimented with different values of the the hyper-parameter τ in the decision tree and random forest models defining the ratio of minimum number of samples required to be at a leaf node.

5.3 Automatic Evaluation

In this section we detail an experimental setup for automatically evaluating our solution. These automatic methods allow a straightforward verification of the effectiveness of our solution.

We use the following evaluation metrics:

(1) Adjusted Rand Index (ARI). [22] To evaluate the partition induced by our model's recommended conditions, we use the ARI between the recommended partition and the gold reference partition, which measures the level of similarity between the two clusterings. **(2) Clustering Coverage.** Ratio of failure points that were eventually mapped to one of the response type clusters. Note that our solution does not require that every failure point is mapped.

(3) #Child Nodes. Compares the number of recommended nodes to the original number of nodes in the execution graph. **(4) COND-Length.** Evaluates the level of readability of the conditions in our solution (this is relevant only for the decision tree model). We compare the length of the recommended conditions of the nodes to the original conditions of the nodes in the execution graph. The length is calculated by the number of variables in the condition.

5.3.1 Results. We evaluated different versions of our model and different reference classifiers as mentioned in Section 5.2 over the banking dataset as shown Table 1.

In spite of the decision tree being the weakest classification model in our comparison, it outperformed all other models in terms of ARI. Moreover, in contrast to the random forest model, the decision tree got consistently high ARI scores independently of τ. The clustering coverage of all variants was above 0.9, with the decision tree model only slightly worse than the other models. Our decision tree solution also outperformed the other models in terms of the number of child nodes, being closest to the expected average number of nodes in the dataset, 2.46. Regarding the condition length measure, only a high value of τ achieved conditions with length close to the original length of the conditions. However, our experiments showed that this metric tended to have a high variance due to extreme outliers. These outliers were conditions corresponding to "outlier clusters" of all conversations that did not map to any of the other clusters. When discarding in step 5 all nodes with conditions of length ≥ 10 with $\tau = 0.01$, our coverage of conversations decreased to 95% of the original clustering coverage. In this case the average condition length was only 1.88. As expected, the lower τ, the more aggressive our pruning becomes, which results in less number of child nodes, shorter conditions, but also lower clustering coverage.

Figure 7 shows the distribution of the Adjusted Rand Index (ARI) for the decision tree for $\tau = 0.01$. Our solution achieved high ARI scores for most of the escalation nodes. Note that in our scenario the number of child nodes is quite small (as can also be seen in Table 1 in comparison to the number of conversations that are clustered (at least 50). This fact sometimes results in low ARI scores (and even a score of 0) and is a known drawback of ARI [30]. Nevertheless, we use the ARI measure as it is the *de facto* standard to estimate the level of similarity between two clusterings. Note that our findings are consistent for all decision tree configurations.

5.4 Experimenting with human-to-human logs

Our proposed solution for suggesting conditions assumes a dialog graph as a backbone model. We are aware that this is not the only dialog system backbone representation possible and that human-to-human conversation logs may not reflect any backbone at all. Yet, we wanted to both evaluate our solution on human-to-human logs, and extend the method so we can learn such a backbone from human-to-human logs. We started with the modest task of recovering conditions for single nodes.

To this end, we used the MultiWOZ dataset which contains both the dialog utterances, and context variables extracted throughout the conversation by human annotators and is aligned with each turn in the conversation. We simulated a single node by collecting all agent utterances asking for a specific detail based on annotations

Table 1: Automatic evaluation results for the Banking Dataset (averaged over all escalation nodes). The values in parentheses refer to the original graph

Model	τ	ARI	Clustering Coverage	#Child Nodes (2.46)	COND-Length (1.40)
DT	0.001	0.68	0.95	2.79	8.05
	0.01	**0.71**	0.95	**2.58**	6.32
	0.05	0.64	0.92	2.02	**1.64**
RF	0.001	0.68	**0.97**	4.97	-
	0.01	0.47	0.95	4.12	-
	0.05	0.27	0.95	2.33	-
XGB	-	0.66	**0.97**	2.33	-

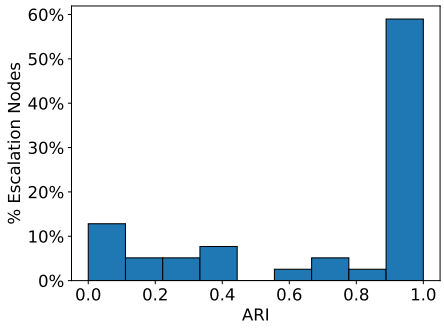


Figure 7: Distribution of ARI over escalation nodes in Banking Dataset for the decision tree with $\tau = 0.01$.

supplied with the dataset. In particular in the MultiWOZ hotel booking scenario, we use the action annotation "area" to collect all utterances where the agent asks about the booking area. We declare all turns in this collection as if they are assigned to the same simulated node, e.g. "ask area node". Our task then is to create additional nodes corresponding to actions taken in the consecutive turn following that node in different conversations, and to recover the conditions to be used for directing a dialogue system towards the correct action.

The actions taken consider the user's answer and the context of the conversation so far. For example, one action could be to ask for more constraints from the user such as hotel grade, another could be to suggest a small set of specific hotels matching the user's constraints supplied so far, and a third option could be to ask the user to relieve a constraint because no hotels matching the constraints were found. Applying our solution, it clusters the agent responses to their types, and then discovers the condition, based on the context of the conversation and the user's response, that would lead to each of these types.

We created such simulated nodes and found that clustering the agent responses resulted with a small number of clusters, and the corresponding conditions turned out to be long and hard to interpret. Inspecting them we saw that they consist of conjunctions of

clauses connected with an "or" operator. This reflects multiple, and sometimes disjoint paths reaching our simulated node, with very different contexts leading to the same node. We suspect this is due to the slot-filling nature of the MultiWOZ dataset, where different combinations of filled slots lead to the same question asking for a specific slot not filled yet. This result led us to add a calculated context feature, counting the number of hotels that satisfy all constraints set by the filled slots so far. Adding this feature yielded clear conditions that separate the conversations into distinct actions based on this feature.

While we saw that following our solution in this hotel booking use case data set resulted in hard to interpret conditions, the process taught us how to analyze conversations with respect to the context variables, and come up with an extra variable that may lead to an interpretable condition, albeit some additional manual analysis. A complete fully automated solution would probably employ means to automatically detect these missing variables for example by analyzing agents' actions which could be queries to an external back-end system.

6 CONCLUSION AND FUTURE WORK

We presented a method to automatically improve goal-oriented dialogues after deployment. The method offers a way for ongoing learning, utilizing the data that is collected in the customer care center. We challenge a fatal limitation of deployed task-oriented dialogue systems: These systems, while initially useful, cannot improve during production without manual updates by an expert. Previous methods have attempted to incorporate learning into the systems via neural network fall-backs, which has shown to be an ineffective band-aid solution, as neural models have little guarantee to the correctness of their behavior, and are seldom deployed in practice.

We propose a five-step procedure, which can be employed on a deployed system and uses conversation logs collected during run time. These logs named "escalation logs" include interactions where the dialogue system assumed initial control, subsequently failed, and control was escalated to a human agent to resolve the case. Our procedure yields an improved version of the system, where the modifications fulfill additional behaviors in cases where the system failed to provide a satisfactory response.

Future Work. This research is aimed to help in real customer care environments in which human agents and virtual assistants work in tandem. We propose a first step towards relieving the need of manual expert annotations for the improvement of the system. Future work on this topic will naturally involve a thorough evaluation in a production setting, where the system is deployed, improved, and evaluated for its quality in comparison to the previous version. This procedure can be repeated multiple times to iteratively improve the system.

The MultiWOZ dataset poses a real-life scenario of slot filling, in which a user needs to provide several slots of information before the system can respond. The system will then consider the entire context, e.g. all slots filled so far, the new value from the current user utterance, and evaluate the current state to decide on an action. This dependency between the system response and the anticipated result of its action (based on slot value filled and the system state), makes the prevalent slot filling case a challenging scenario for our clustering step, which needs to take into account not only the agent response but also the context of the conversation, the current user utterance and the state of the system. On top of the calculated feature we suggest in the paper, we plan an in-depth analysis of such cases in future work.

REFERENCES

[1] Daniel Adiwardana, Minh-Thang Luong, David R. So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, and Quoc V. Le. 2020. Towards a Human-like Open-Domain Chatbot. *CoRR* abs/2001.09977 (2020). arXiv:2001.09977 https://arxiv.org/abs/2001.09977

[2] Rami Al-Rfou, Marc Pickett, Javier Snaider, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2016. Conversational Contextual Cues: The Case of Personalization and History for Response Ranking. *CoRR* abs/1606.00372 (2016). arXiv:1606.00372 http://arxiv.org/abs/1606.00372

[3] Hussein Almuallim. 1996. An efficient algorithm for optimal pruning of decision trees. *Artificial Intelligence* 83, 2 (1996), 347–362.

[4] Alexander Bartl and Gerasimos Spanakis. 2017. A retrieval-based dialogue system utilizing utterance and context embeddings. *CoRR* abs/1710.05780 (2017). arXiv:1710.05780 http://arxiv.org/abs/1710.05780

[5] Manisha Biswas. 2018. Microsoft Bot Framework. In *Beginning AI Bot Frameworks*. Springer, 25–66.

[6] Pawel Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. 2018. MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling. *CoRR* abs/1810.00278 (2018). arXiv:1810.00278 http://arxiv.org/abs/1810.00278

[7] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. 2010. Toward an Architecture for Never-Ending Language Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, Maria Fox and David Poole (Eds.). AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1879

[8] Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. 2017. A Survey on Dialogue Systems: Recent Advances and New Frontiers. *SIGKDD Explorations* 19, 2 (2017), 25–35. https://doi.org/10.1145/3166054.3166058

[9] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 785–794. https://doi.org/10.1145/2939672.2939785

[10] Yizong Cheng. 1995. Mean Shift, Mode Seeking, and Clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* 17, 8 (1995), 790–799. https://doi.org/10.1109/34.400568

[11] Jan Deriu, Álvaro Rodrigo, Arantxa Otegi, Guillermo Echegoyen, Sophie Rosset, Eneko Agirre, and Mark Cieliebak. 2019. Survey on Evaluation Methods for Dialogue Systems. *CoRR* abs/1905.04071 (2019). arXiv:1905.04071 http://arxiv.org/abs/1905.04071

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[13] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. 2010. Building Watson: An Overview of the DeepQA Project. *AI Magazine* 31, 3 (Jul. 2010), 59–79. https://doi.org/10.1609/aimag.v31i3.2303

[14] Quoc V. Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014 (JMLR Workshop and Conference Proceedings)*, Vol. 32. JMLR.org, 1188–1196. http://proceedings.mlr.press/v32/le14.html

[15] Bing Liu and Ian Lane. 2017. Iterative policy learning in end-to-end trainable task-oriented neural dialog models. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2017, Okinawa, Japan, December 16-20, 2017*. IEEE, 482–489. https://doi.org/10.1109/ASRU.2017.8268975

[16] Bing Liu, Gökhan Tür, Dilek Hakkani-Tür, Pararth Shah, and Larry P. Heck. 2018. Dialogue Learning with Human Teaching and Feedback in End-to-End Trainable Task-Oriented Dialogue Systems. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, Marilyn A. Walker, Heng Ji, and Amanda Stent (Eds.). Association for Computational Linguistics, 2060–2069. https://doi.org/10.18653/v1/n18-1187

[17] Maher Nabulsi, Ahmad Alkatib, and Fatima Quiam. 2017. A New Method for Boolean Function Simplification. *International Journal of Control and Automation*

10 (12 2017), 139–146. https://doi.org/10.14257/ijca.2017.10.12.13

[18] Mohammad Nuruzzaman and Omar Khadeer Hussain. 2018. A Survey on Chatbot Implementation in Customer Service Industry through Deep Neural Networks. In *15th IEEE International Conference on e-Business Engineering, ICEBE 2018, Xi'an, China, October 12-14, 2018*. IEEE Computer Society, 54–61. https://doi.org/10.1109/ICEBE.2018.00019

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[20] Jiahuan Pei, Pengjie Ren, Christof Monz, and Maarten de Rijke. 2019. Retrospective and Prospective Mixture-of-Generators for Task-oriented Dialogue Response Generation. *ArXiv* abs/1911.08151 (2019).

[21] Janarthanan Rajendran, Jatin Ganhotra, and Lazaros C. Polymenakos. 2019. Learning End-to-End Goal-Oriented Dialog with Maximal User Task Success and Minimal Human Agent Use. *TACL* 7 (2019), 375–386. https://transacl.org/ojs/index.php/tacl/article/view/1622

[22] William M. Rand. 1971. Objective Criteria for the Evaluation of Clustering Methods. *J. Amer. Statist. Assoc.* 66, 336 (1971), 846–850. http://www.jstor.org/stable/2284239

[23] Abhinav Rastogi, Dilek Hakkani-Tür, and Larry P. Heck. 2017. Scalable Multi-Domain Dialogue State Tracking. *CoRR* abs/1712.10224 (2017). arXiv:1712.10224 http://arxiv.org/abs/1712.10224

[24] Alexander I Rudnicky, Eric Thayer, Paul Constantinides, Chris Tchou, R Shern, Kevin Lenzo, Wei Xu, and Alice Oh. 1999. Creating natural dialogs in the Carnegie Mellon Communicator system. In *Sixth European Conference on Speech Communication and Technology*.

[25] Navin Sabharwal and Amit Agrawal. 2020. Introduction to Google Dialogflow. In *Cognitive Virtual Assistants Using Google Dialogflow*. Springer, 13–54.

[26] Navin Sabharwal, Sudipta Barua, Neha Anand, and Pallavi Aggarwal. 2020. Integrating with Advance Services. In *Developing Cognitive Bots Using the IBM Watson Engine*. Springer, 197–239.

[27] Daniel L. Silver, Qiang Yang, and Lianghao Li. 2013. Lifelong Machine Learning Systems: Beyond Learning Algorithms. In *Lifelong Machine Learning, Papers from the 2013 AAAI Spring Symposium, Palo Alto, California, USA, March 25-27, 2013 (AAAI Technical Report)*, Vol. SS-13-05. AAAI. http://www.aaai.org/ocs/index.php/SSS/SSS13/paper/view/5802

[28] Aniruddha Tammewar, Monik Pamecha, Chirag Jain, Apurva Nagvenkar, and Krupal Modi. 2018. Production Ready Chatbots: Generate if Not Retrieve. In *The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018 (AAAI Workshops)*, Vol. WS-18. AAAI Press, 739–745. https://aaai.org/ocs/index.php/WS/AAAIW18/paper/view/17357

[29] Chongyang Tao, Wei Wu, Can Xu, Wenpeng Hu, Dongyan Zhao, and Rui Yan. 2019. Multi-Representation Fusion Network for Multi-Turn Response Selection in Retrieval-Based Chatbots. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (Melbourne VIC, Australia) *(WSDM '19)*. Association for Computing Machinery, New York, NY, USA, 267–275. https://doi.org/10.1145/3289600.3290985

[30] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *J. Mach. Learn. Res.* 11 (Dec. 2010), 2837–2854.

[31] Svitlana Volkova, Pallavi Choudhury, Chris Quirk, Bill Dolan, and Luke S. Zettlemoyer. 2013. Lightly Supervised Learning of Procedural Dialog Systems. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*. The Association for Computer Linguistics, 1669–1679. https://www.aclweb.org/anthology/P13-1164/

[32] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve J. Young. 2016. A Network-based End-to-End Trainable Task-oriented Dialogue System. *CoRR* abs/1604.04562 (2016). arXiv:1604.04562 http://arxiv.org/abs/1604.04562

[33] Rui Yan, Yiping Song, and Hua Wu. 2016. Learning to Respond with Deep Neural Networks for Retrieval-Based Human-Computer Conversation System. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Pisa, Italy) *(SIGIR '16)*. Association for Computing Machinery, New York, NY, USA, 55–64. https://doi.org/10.1145/2911451.2911542

[34] Zhao Yan, Nan Duan, Peng Chen, Ming Zhou, Jianshe Zhou, and Zhoujun Li. 2017. Building Task-Oriented Dialogue Systems for Online Shopping. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, Satinder P. Singh and Shaul Markovitch (Eds.). AAAI Press, 4618–4626. http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14261

[35] Guoguang Zhao, Jianyu Zhao, Yang Li, Christoph Alt, Robert Schwarzenberg, Leonhard Hennig, Stefan Schaffer, Sven Schmeier, Changjian Hu, and Feiyu Xu. 2019. MOLI: Smart Conversation Agent for Mobile Customer Service. *Information* 10 (02 2019), 63. https://doi.org/10.3390/info10020063