OCC: A Smart Reply System for Efficient In-App Communications

Yue Weng*
Uber AI
San Francisco, California
yweng@uber.com

Huaixiu Zheng*
Uber AI
San Francisco, California
huaixiu.zheng@uber.com

Franziska Bell
Uber AI
San Francisco, California
fran@uber.com

Gokhan Tur
Uber AI
San Francisco, California
gokhan@uber.com

ABSTRACT

Smart reply systems have been developed for various messaging platforms. In this paper, we introduce Uber's smart reply system: one-click-chat (OCC), which is a key enhanced feature on top of the Uber in-app chat system. It enables driver-partners to quickly respond to rider messages using smart replies. The smart replies are dynamically selected according to conversation content using machine learning algorithms. Our system consists of two major components: intent detection and reply retrieval, which are very different from standard smart reply systems where the task is to directly predict a reply. It is designed specifically for mobile applications with short and non-canonical messages. Reply retrieval utilizes pairings between intent and reply based on their popularity in chat messages as derived from historical data. For intent detection, a set of embedding and classification techniques are experimented with, and we choose to deploy a solution using unsupervised distributed embedding and nearest-neighbor classifier. It has the advantage of only requiring a small amount of labeled training data, simplicity in developing and deploying to production, and fast inference during serving and hence highly scalable. At the same time, it performs comparably with deep learning architectures such as word-level convolutional neural network. Overall, the system achieves a high accuracy of 76% on intent detection. Currently, the system is deployed in production for English-speaking countries and 71% of in-app communications between riders and driver-partners adopted the smart replies to speedup the communication process.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Classification and regression trees**; **Neural networks**;

KEYWORDS

smart reply; machine learning; natural language processing; intent detection; unsupervised learning; distributed embedding; neural networks

ACM Reference format:
Yue Weng, Huaixiu Zheng, Franziska Bell, and Gokhan Tur. 2019. OCC: A Smart Reply System for Efficient In-App Communications. In *Proceedings of The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Anchorage, AK, USA, August 4–8, 2019 (KDD '19),* 8 pages.
https://doi.org/10.1145/3292500.3330694

1 INTRODUCTION

Uber's ride-sharing business connects driver partners to riders who need to transport around cities. The app provides navigation and many other innovative technology and features that assist driver-partners with finding riders at the pick-up locations. Uber's in-app chat [10], a real-time in-app messaging platform launched in early 2017, is one of them. Before having the functionality to chat within the app, communication between customers occurred outside of the mobile app experience using third-party technologies. This resulted in safety concerns, higher operational costs, fewer completed trips, and most importantly, limits the company's ability to understand and resolve challenges both riders and driver partners were having while using the app. Although the newly added chat feature has solved many of these problems by bringing the chat experience into the app, it still requires driver-partners to type messages while driving, which is a huge safety concern. According to a public study, compared to regular driving, accident risk is about 2.2 times higher when talking on a hand-held cell phone and 6.1 times higher when texting [13]. Therefore, to provide a safe and smooth in-app chat experience for driver-partners, we developed One-Click Chat (OCC), a smart reply system that allows driver-partners to respond to messages using smart replies selected dynamically according to the conversation context, as shown in Figure 1.

There has been a surge of interest in developing and using smart reply and chatbot systems on commercial platforms [2, 6, 15]. However, building an intelligent system to automatically generate suggested replies is not a standard machine learning problem. Anjuli et al. [6] proposed to divide the task into two components: predicting responses and identifying a target response. Specifically, to predict responses, they leveraged a sequence-to-sequence (Seq2seq) [12]

*Equal Contribution

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330694

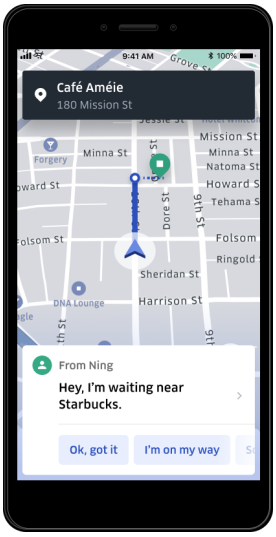

Figure 1: With one-click chat, driver-partners can more easily respond to rider messages.

framework with long short-term memory (LSTM) [4] trained on large-scale email conversations; to obtain the final response, they first proposed a semi-supervised approach to generate a response pool and then select from it based on the LSTM predictions to control the actual replies as free text generation is still not mature enough for commercial use [6]. Similarly, LinkedIn used a statistical model for predicting responses for incoming messages [5].

In contrast, instead of predicting responses directly, our work experiments with techniques that perform language understanding (i.e., intent detection) and reply retrieval separately. Our approach requires a much smaller scale labeled dataset for training. Compared to the generic smart replies, OCC is designed for Uber's domain-specific use case to streamline communications between driver-partners and riders during the pick-up stage. In addition, in-app messages on the Uber platform are typically very short (averaging 4-5 words) and non-canonical (with typos, abbreviations etc.) compared to other platforms such as email. This poses unique challenges to designing and developing such a smart reply system. In this paper, we share our experiences building and integrating Uber's smart reply system. The main contributions of this work are as follows:

- Introducing an end-to-end smart reply system architecture, a mobile-friendly solution, in Section 2.
- Presenting a novel approach to break the task of smart reply down into two steps - intent detection and reply retrieval, tailored specifically for short messages on the mobile platform.
- Proposing a mixture of unsupervised embedding and nearest-neighbor supervised learning approaches which do not require a large amount of labeled data. This combined approach achieves comparable performance to deep learning architectures, but is much easier to implement and deploy. The step-by-step algorithmic approach is discussed in Section 3.

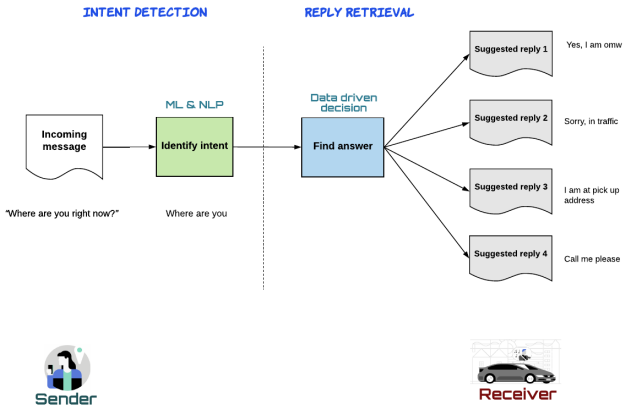

Figure 2: The machine learning algorithm empowers the flow of the OCC experience. Two key steps are involved: 1) intent detection and 2) reply retrieval.

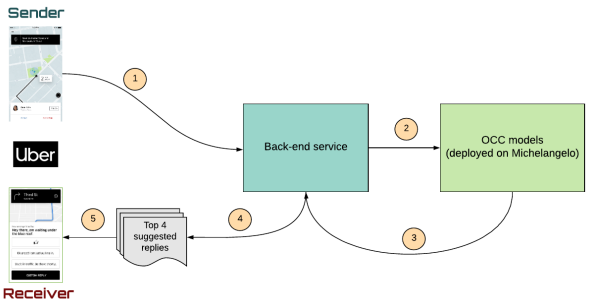

Figure 3: The architecture for Uber's smart reply system, OCC, consists of a five-step workflow.

- Conducting comprehensive experiments to compare different models and uncover their underlying mechanisms and shortcomings. Experiments are discussed in Section 4.

2 ONE-CLICK CHAT SYSTEM

As one of the world's largest and most recognized rider-sharing providers, there are hundreds and thousands of messages exchanged on the platform every day. OCC, one of the latest key enhanced features on our chat platform, aims to provide driver-partners with a one-click chatting experience by offering them the most relevant replies.

To find the best replies to each incoming message, we formulate the task into a machine learning problem with two major components:

(1) Intent Detection
(2) Reply Retrieval

Figure 2 illustrates how OCC works in the real world. Specifically, a driver-partner receives an incoming rider message asking *Where are you right now?*, which is very common during pick-up. In intent detection, the OCC system detects the intent of the message as

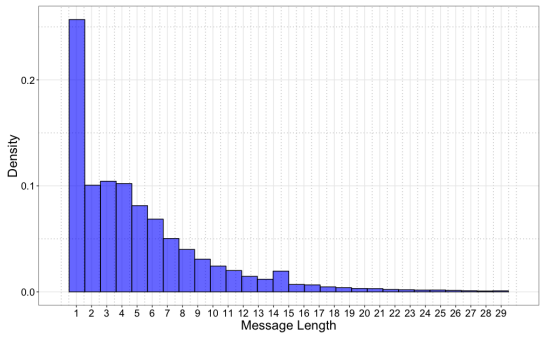

Figure 4: Chat message length frequency, on average 4-5 words per message.

Where are you?. Then in reply retrieval, the system surfaces the top four most relevant replies to the driver-partner, which, in this example, are *Yes, I am omw*, *Sorry, in traffic*, *I am at pick-up address*, and *Call me please*. Now, the driver-partner can select one of these four replies and send it back to the rider with a single tap. The above process finishes one round of communication with smart replies.

OCC system is fully integrated with our in-house chat platform. As depicted in Figure 3, the system architecture follows a five-step workflow:

(1) Sender (rider app) sends a message to driver partner.
(2) Mobile side triggers and sends the message to a back-end service that calls the machine learning model hosted on Uber's in-house machine learning platform Michelangelo [3].
(3) The model preprocesses and encodes the message, generates prediction scores for each possible intent, and sends them back to the back-end service.
(4) Once the back-end service receives the predictions, it follows a predefined reply retrieval policy to find the best replies (in this case, the top four).
(5) Receiver (driver-partner app) receives the smart replies and renders them for the driver-partner to select.

3 MACHINE LEARNING METHODOLOGY

By design, OCC aims to provide an easy chat experience for driver-partners during the pick-up stage for Uber-specific scenarios and topic domains. As a result, it shares a few technical challenges that are unique to mobile messaging systems:

- Messages are short compared to email or other communication channels, $4-5$ words per message on average given it is mostly used during pick-up, see Figure 4 for the message length statistics.
- Messages are non-canonical, containing abbreviations, typos, and colloquialisms. Even for simple message like *Where are you*, there are many variations including *where r u :) ?*, *w Here are you* and more.

We designed our machine learning system with these challenges in mind, and adopted a mixture of unsupervised embedding and supervised classification techniques to tackle them accordingly. This section describes each component of the pipeline shown in Figure 2 in detail.

3.1 Features and Data

We used millions of encrypted and anonymized historical in-app conversation data for our unsupervised embedding model. For supervised classification model, we collected and annotated thousands of conversational messages. Each of which is labeled as one of the intents in our system (such as *I am here*). Here, we assume the messages are all single intent and validate the assumption manually. For the majority of the messages, they are short and convey a single intent in a single exchange of communication, even though there are exceptions. For this version of OCC, we use the text message as the only feature in the modeling process. For future iterations, contextual features such as length of conversation and trip information may be leveraged by the models.

3.2 Intent Detection

Given the nature of our message data (short and non-canonical with typos, etc.), we decide to put the emphasis on intent detection. As we tackle intent detection [1], we encounter several technical challenges due to the complexity of human language itself and the nature of messages exchanged on a mobile platform. For instance, there are many ways to ask the same question, such as *Where are you going?*, *Where are you heading?*, and *What's your destination?*. With typos and abbreviations, chat messages introduce even more permutations. In addition, chat messages are typically very short, which makes distinguishing them from each other very challenging. Creating a system with replies for millions of individual questions does not scale, so we need a system that can identify the intent or topic behind each question, allowing us to provide replies to a finite set of intents.

We formulate the language understanding task as a classification problem in order to have full control over message replies. We experimented with four different approaches for intent detection.

- Frequency-based, a context-agnostic approach that suggests intents based on their frequency.
- CNN-based deep learning approach, both word and character level [7].
- Embedding [8] plus nearest neighbour classifier (NNC), which is a combination of unsupervised and supervised learning approach. It requires much smaller labeled data and performs on par with deep learning methods on a test dataset.

Since both frequency and CNN-based approaches are relatively straight forward, we focus on the embedding-based NNC for the rest of the section.

3.2.1 Message Embeddings. We embedded messages using the Doc2vec model [8], an unsupervised algorithm proposed by Le and Mikolov (2014), that learns fixed-length feature representations from variable-length pieces of text, such as sentences, paragraphs, and documents. Because our messages are domain-specific and contain a lot of typos and abbreviations, we decided to train our own embedding model using in-house data. Our Doc2vec model was trained on millions of anonymized, aggregated in-app chat messages and was then used to map each message to a dense vector

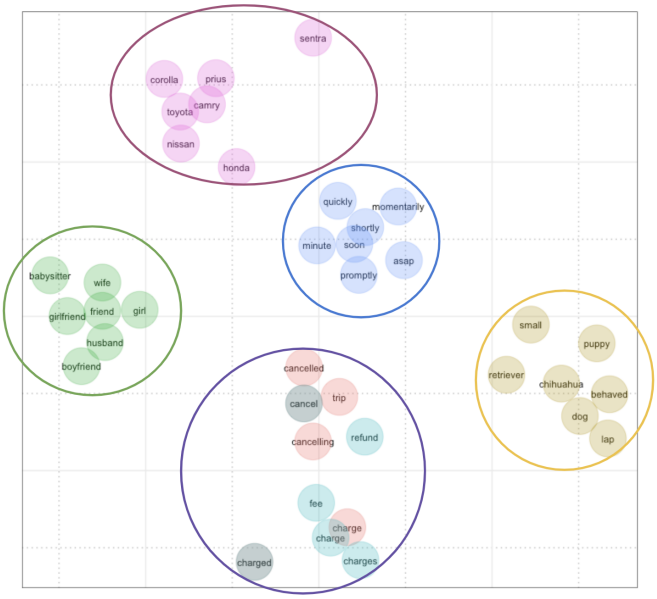

Figure 5: This two-dimensional t-SNE projection of the Doc2vec word embedding illustrates the ability of the model to automatically organize concepts and learn implicitly the relationships between words, clustering them based on semantics.

embedding space. Figure 5 visualizes the word vectors in a two-dimensional projection using a t-SNE plot [14]. Since it captures the semantic meaning of words, the model can cluster similar words together. For example, *fee* is close to *charge* and *refund*, but far away from *friend*.

More formally, given a sequence of training words $w_1, w_2, ..., w_T$ from the document D, the objective of the Doc2vec model is to use a neural network to find the parameter sets θ^* in order to maximize the conditional probability of a target word w_t given k contextual words before and after the target word,

$$\theta^* = \text{argmax}_\theta \prod_{t \in S} P(w_t | w_{t-k}, ..., w_{t+k}; \theta) \quad (1)$$

where S is a sample of words from D and $\theta \in \theta_d, \theta_w, \theta_{\text{softmax}}$, where θ_w are word vectors, θ_{softmax} contains the weights for a softmax hidden layer and θ_d are paragraph vectors. In short, the algorithm itself has two stages: 1) *training stage* to optimize word vectors θ_w, softmax weights θ_{softmax} and paragraph vectors θ_d on already seen paragraphs to maximize the probability of target word given contextual words; and 2) *the inference stage* to compute paragraph vectors d for new documents, which can be never seen before, by gradient descending on d while holding θ_{softmax} and θ_w fixed [8].

3.2.2 NNC Approach. We build a nearest-neighbor classifier on top of the distributed representation of labeled messages from document embedding model Doc2vec. The main motivation is that we have a relatively small set of (thousands of) labeled data to train a classification model. Overfitting is a big concern, and hence the non-parametric nearest-neighbor classifier can largely avoid such a problem.

Figure 6: Illustration of the process of nearest neighbor classifier based on document embedding and cosine distance.

Figure 6 illustrates the process of nearest-neighbor classifier using document embedding. First, with the trained Doc2vec model, noted as M, we can obtain document vector d_j^i for any document, which is the dense vectors representing the ith document which belongs to jth intent class. Using labeled data, we then compute the **centroid** D_j of each intent class from the dense vectors as:

$$D_j = \frac{1}{N_j} \sum_{i=1}^{N_j} d_j^i \quad (2)$$

where N_j is the number of labeled messages of jth intent class. Each intent class now is represented by this centroid.

During the inference stage, an inference step is taken to use M to compute the paragraph vector for a new paragraph $m^k = w_1, w_2, ...w_n$, where $w_1, w_2, ...w_n$ are the word tokens. We obtain the corresponding dense vector

$$d^k = M(m^k)$$

Using labeled data, we then compute the vector cosine distance between the message vector and each of the intents' centroids and pick top K closest intents measured by cosine distance as top-K predictions of intent.

$$C_k^j = \frac{d^k \cdot D_j}{\|\mathbf{d}^k\| \cdot \|\mathbf{D}_j\|}$$

Figure 6 illustrates a toy example with only two intent classes. An incoming message is mapped to the embedding space. As it is closer to the *What color is your car?* intent centroid, it would be classified as such rather than *I am here*.

3.3 Reply Retrieval

Once the system detects the intent for the message, reply retrieval becomes relatively straightforward as the topic domain is specific to Uber's in-app communications. For the example shown in Figure 2, the reply to a message with *where are you* intent has only a small set of possible variations given its context as a response on the Uber platform. In this case, there are only a couple of answers such as

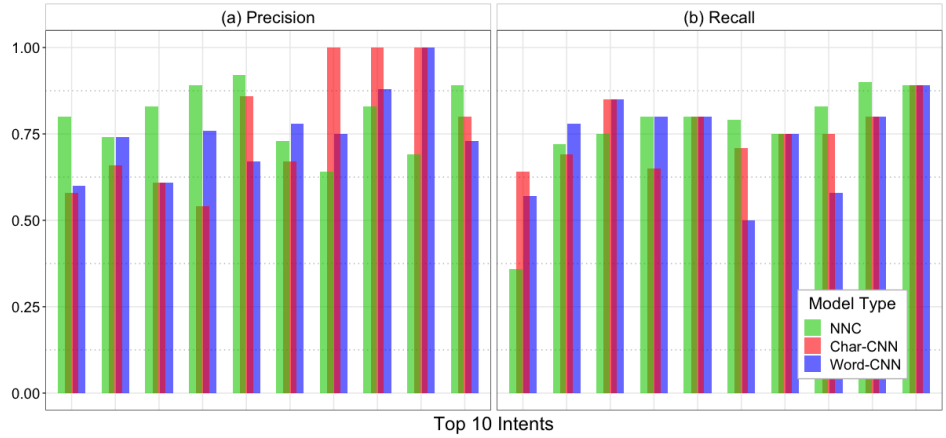

Figure 7: (a) Precision and (b) Recall of the models for top-10 intents.

Model	Accuracy
NNC	0.759
Word-CNN	0.756
Char-CNN	**0.772**
Frequency-based	0.155

Table 1: Model accuracy of intent detection.

Yes, I am omw, Sorry, in traffic and so on, depending on the location of the driver partner.

Here, we leverage historical conversation pairs to find the most frequent reply candidates for each intent class. In essence, we perform intent classification on all messages and map out the intents of each message in a conversation. For each turn of the conversation, the intent of the *incoming message* is paired with the intent of the *response message*. For a particular intent of *incoming message*, we measure the frequency of the intents from the *response messages*, and select the most frequent ones as well as all possible variations as candidate replies. After that, our content team performs one more round of augmentation and reordering to make the candidate replies as easily understood and accurate as possible. This whole process creates the intent-reply mapping for reply retrieval.

During serving time, in order to gain more coverage, we pick the top K predicted intents and dynamically de-duplicate repeated reply candidates by order. For instance, when we get predicted intents I_1 and I_2, we first look up the intent-reply mapping

$$I_1 : R_1, R_2; I2 : R_1, R_3, R_4.$$

Instead of providing reply R_1 twice, we merge them and keep their order. So the final smart reply list is

$$[R_1, R_2, R_3, R_4]$$

In addition, corner cases such as extremely short messages (e.g., having only one word) and low confidence predictions (e.g., multi-intent messages) are handled by rules rather than our algorithm.

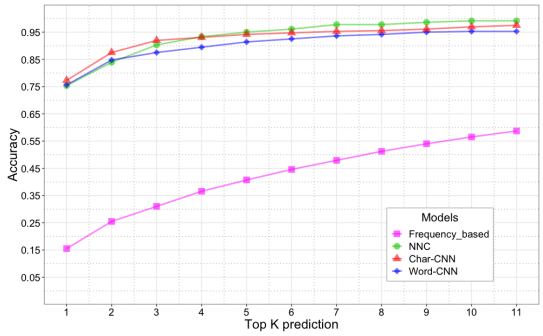

Figure 8: Top-K accuracy of intent detection.

4 EXPERIMENT ANALYSIS

In this section, we evaluate the intent detection task and report overall performance for the approaches described in the above section.

4.1 Results

Model Accuracy: One of the most important metrics for evaluating our system is the overall accuracy for intent detection as we strictly control the number of replies for each intent in the product. Table 1 shows the model accuracy for the four different models we experiment with. The naive frequency-based approach has an accuracy of 15.5% which is simply the population of the top-1 intent classes. The best performing model is Char-CNN model with an accuracy of 77.2%, followed by the NNC with an accuracy of 75.9%. Word-CNN performs slightly (75.6%) worse compared to NNC.

Figure 8 further shows the top-K accuracy of the four approaches. The overall trend with increasing K agrees with the top-1 accuracy except that NNC performs slightly better than Char-CNN after $K = 5$. Except the naive frequency-based approach, all three models reach $> 90\%$ accuracy at $K = 4$. Given the small amount of labeled data (thousands) we have for training, it is not surprising that NNC performs comparable to the two deep learning architectures, as

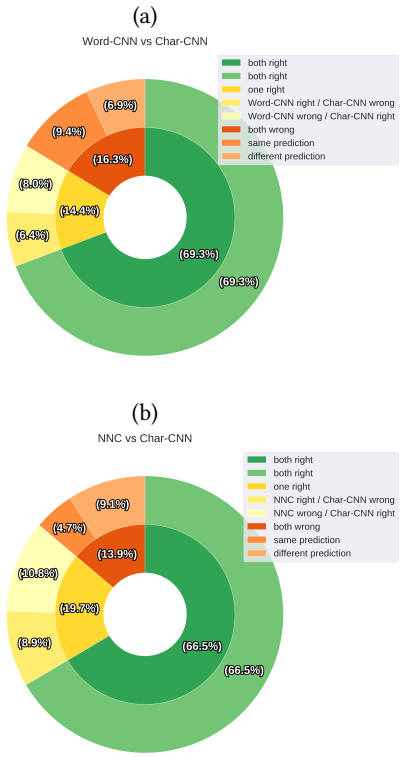

Figure 9: Model prediction comparison: (a) Word-CNN vs Char-CNN models, (b) NNC vs Char-CNN models. The percentage of test samples falling into different buckets are plotted.

typically deep learning approaches start to be advantageous when the training data size is large enough.

Figure 7 shows the precision and recall of NNC, Word-CNN and Char-CNN on the top-10 intents. Specifically, for precision, NNC model shows relatively even performance across all top-10 classes. While deep learning models (Char-CNN) in particular show lower precision for the top-5 intents compared to the remaining ones. This highlights a key difference between the NNC model and deep learning architectures. Because NNC is non-parametric, it is less biased towards popular classes. The pattern observed above for precision is reversed when considering recall: deep learning models are performing better than NNC for top classes (top-3 in particular) as shown in Figure 7(b) due to the same biased towards predicting popular classes compared to NNC.

Model Complementary: Next, we look at how complementary the predictions are between NNC and deep learning models using Ludwig [9]. As shown in Figure 9(a), Word-CNN has a rather large overlap with Char-CNN in predictions: they have 69.3% predictions being correct at the same time, and 9.4% predictions being the same but wrong at the same time. Together, they made the same predictions on 78.7% test samples. In contrast, Figure 9(b) shows that NNC and Char-CNN have less overlap in their predictions: 66.5% being right and 4.7% being the same but wrong at the same time. Looking at the portion of predictions where one model being right and the other being wrong, we find that NNC and Char-CNN have 19.7% such predictions compared to 14.4% from Word-CNN and Char-CNN. It confirms that non-parametric model NNC is indeed more complementary to Char-CNN than Word-CNN.

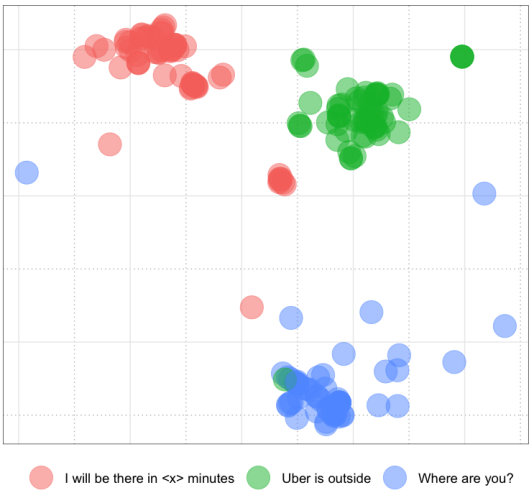

Figure 10: In this two-dimensional t-SNE projection of sentence embedding, the model clusters messages around intent.

Analysis of NNC Model: Finally, in order to better understand the underlying mechanism of such a good performance for a simple nearest-neighbor classifier, we look at the embedding representation of the messages and their corresponding labels. Figure 10 shows examples of three different intent classes in a t-SNE plot. Surprisingly, the message embedding vectors are clustered for each intent even after projected down to $2d$ space. This confirms the high quality of the message embeddings and its capability to capture semantic meanings of different intents even though most messages are rather short and can contain various typos and abbreviations. The separation between the three classes is also rather pronounced. As a result, it is expected that a very simple non-parametric nearest-neighbor classifier can perform on par with sophisticated deep learning architectures such as Char-CNN.

The above analysis demonstrates that the quality of document embedding is critical to the good performance of NNC. Furthermore, we perform a hyperparameter search to understand the correlation between the hyperparameters of the Doc2vec model and the intent detection performance using *gensim* [11]. Empirically, we find that document vectors with distributed bag of words (DBOW) work better than those obtained with distributed memory (DM) for intent detection. Figure 11 shows the hyperparameter search for DBOW and its impact on the downstream classification accuracy. It is clear that the accuracy is rather sensitive to the parameters *alpha* and *sample* but varies very little for all the other parameters. Specifically, *alpha* is the initial learning rate of DBOW and we found that a moderate learning rate around 0.02 is optimal. *sample* determines the amount of down-sampling on high-frequency words. Empirically, we observed that a small *sample* gives the best

Message content	First prediction	Second prediction	Label
This Uber is for my daughter. She's going to school and coming right back. Thanks	You are picking up <person>	I'm going to <loc>	I'm going to <loc>
Ok I'll drive on the Main Street	Wrong side	I am at <loc>	I'm going to <loc>
I will come to 51 and 6	Come to <loc>	Can we meet at <loc>?	I'm going to <loc>

Table 2: Examples of prediction errors by NNC model on *I am going to <loc>* intent.

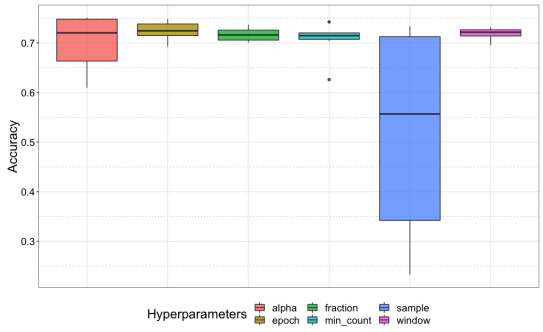

Figure 11: Hyper-parameter search for Doc2vec model with distributed bag of words (DBOW). The intent detection accuracy is plotted against several parameters showing both the average and standard deviation.

performance, implying that little or no down-sampling is necessary. It is therefore reasonable to conclude that high-frequency words are key to capture the semantic meaning of the short chat messages, and thus crucial for our intent detection task.

Model Deployment: Due to its simplicity to develop and deploy, its requirement of small amount of labeled data, and its advantage of speedy inference, we decided to deploy the NNC model in production for our smart reply OCC system. Through experimentation, we observed that 23% of trips in English-speaking countries on the Uber platform involved two-way in-app communications between riders and driver-partners. Among these in-app communications, over 71% of them adopted the smart replies suggested by OCC to speedup and smooth out the process. Such an adoption rate is consistent with the system accuracy on intent detection.

4.2 Error Analysis of NNC Model

The NNC model has relatively high performance on most classes, as discussed above. In this section, we conduct an error analysis on intents where the NNC model doesn't perform very well to understand the weakness of the model. One such intent is *I am going to <loc>*. Table 2 shows the raw message and its top 2 predictions for this intent class in the test set. For the first message, it contains dual intents. The first part of the message informs the driver-partner that *it is for another person*. The second half tells the driver-partner *where she is going*, which matches the model prediction. Multi-intent issues are challenging and beyond the current design of our system. Regarding the second and third messages, the algorithm misclassifies the intents but was able to capture the *location* information correctly. This analysis points out directions for further improvements of our system in the future.

5 CONCLUSIONS AND FUTURE WORK

In conclusion, we introduce a novel smart reply system designed to speed up in-app communication between riders and driver-partners on Uber's platform. Our smart reply system is unique in handling short chat messages with various non-canonical text data. The algorithm we adopt also has the advantage of requiring a rather small amount of labeled data to achieve a relatively high performance. In contrast to existing smart reply systems, we break the task down into two steps of intent detection and reply retrieval instead of directly predicting reply. For the task of intent detection, we experimented with four models, and showed that a simple approach of nearest-neighbor classifier together with document embedding proves to be powerful enough to achieve an accuracy of $\geq 75\%$, which is comparable with the two deep learning models. Further analysis reveals that the non-parametric NNC model is more complementary to Char-CNN than Word-CNN, as Char-CNN and Word-CNN belong to the same class of deep learning architecture. Finally, we analyze the document embeddings and uncover that the key to the success of NNC model is its high quality embeddings which provides clear separations between different intent classes. Due to the advantages of easy development and deployment, and the fast inference, the NNC model is deployed in production to serve the traffic of Uber's smart reply system. Through experimentation, we observed that $\geq 71\%$ of all two-way communications in English on Uber's in-app messaging platform adopted the smart replies recommended by our OCC system to speed up the communication process.

In the future, there are several areas where we can further improve the system. First, the current system only uses the message itself as a feature for intent detection. Including additional features around the trip can certainly provide better intent modeling. Second, the reply retrieval is done using static mapping from intent to replies. Dynamic reply retrieval holds great promise for providing more context-aware replies. This can be achieved by ranking the replies dynamically using a ranking algorithm by taking into account the contextual information. Lastly, active learning feedback loop can be another avenue to steadily correct the errors made by the models and improve the system performance.

6 ACKNOWLEDGMENTS

The authors wish to thank Uber's Conversational AI, Applied Machine Learning, Communications Platform, and Michelangelo teams, Anwaya Aras, Chandrashekar Vijayarenu, Bhavya Agarwal, Lingyi Zhu, Runze Wang, Kailiang Chen, Han Lee, Molly Vorwerck, Jerry Yu, Monica Wang, Manisha Mundhe, Shui Hu, Zhao Zhang, Hugh Williams, Lucy Dana, Summer Xia, Tito Goldstein, Ann Hussey, Yizzy Wu, Arjun Vora, Srinivas Vadrevu, Huadong

Wang, Karan Singh, Arun Israel, Arthur Henry, Kate Zhang, and Jai Ranganathan.

REFERENCES

[1] David J. Brenes, Daniel Gayo-Avello, and Kilian Pérez-González. 2009. Survey and evaluation of query intent detection methods. In *Proceedings of the 2009 workshop on Web Search Click Data, WSCD@WSDM 2009, Barcelona, Spain, February 9, 2009*. 1–7. https://doi.org/10.1145/1507509.1507510

[2] Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient Natural Language Response Suggestion for Smart Reply. *CoRR* abs/1705.00652 (2017). arXiv:1705.00652 http://arxiv.org/abs/1705.00652

[3] Jeremy Hermann and Mike Del Balso. 2018. Meet Michelangelo: Uber's Machine Learning Platform. http://eng.uber.com/michelangelo/. (2018).

[4] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735-1780 (1997).

[5] Nimesh Chakravarthi Jeff Pasternack. 2017. Building Smart Replies for Member Messages. Press Release. https://engineering.linkedin.com/blog/2017/10/building-smart-replies-for-member-messages. (2017).

[6] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, László Lukács, Marina Ganea, Peter Young, and Vivek Ramavajjala. 2016. Smart Reply: Automated Response Suggestion for Email. *CoRR* abs/1606.04870 (2016). arXiv:1606.04870 http://arxiv.org/abs/1606.04870

[7] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 1746–1751.

[8] Quoc V. Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. *CoRR* abs/1405.4053 (2014). arXiv:1405.4053 http://arxiv.org/abs/1405.4053

[9] Piero Molino. [n. d.]. Ludwig. http://ludwig.ai.

[10] Uber Newsroom. 2017. Connect Ahead of the Pickup with In-App Chat. Press Release. https://www.uber.com/newsroom/in-app-chat/. (2017).

[11] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50. http://is.muni.cz/publication/884893/en.

[12] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 3104–3112.

[13] Suzie Lee et al. Thomas A. Dingusa, Feng Guo. 2016. Driver crash risk factors and prevalence evaluation using naturalistic driving data. *PNAS* 13, 10 (2016).

[14] Laurens van der Maaten and Geoffrey E. Hinton. 2008. Visualizing Data using t-SNE.

[15] Anbang Xu, Zhe Liu, Yufan Guo, Vibha Sinha, and Rama Akkiraju. 2017. A New Chatbot for Customer Service on Social Media. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017*. 3506–3510. https://doi.org/10.1145/3025453.3025496