DOI: 10.1609/aaai.12023

SPECIAL TOPIC ARTICLE

Large scale multilingual sticker recommendation in messaging apps

Abhishek Laddha | **Mohamed Hanoosh** | **Debdoot Mukherjee** | **Parth Patwa** | **Ankur Narang**

VP-AI, Hike Messenger, New Delhi, India

Correspondence
Ankur Narang, VP-AI, Hike Messenger, New Delhi, India.
Email: ankur@hike.in

Abstract

Stickers are popularly used while messaging to visually express nuanced thoughts. We describe a real-time sticker recommendation (SR) system. We decompose SR into two steps: predict the message that is likely to be sent, and substitute that message with an appropriate sticker. To address the challenges caused by transliteration of message from users' native language to the Roman script, we learn message embeddings by employing character-level CNN in an unsupervised manner. We use them to cluster semantically similar messages. Next, we predict the message cluster instead of the message. Except for validation, our system does not require human labeled data, leading to a fully automatic tuning pipeline. We propose a hybrid message prediction model, which can easily run on low-end phones. We discuss message cluster to sticker mapping, addressing the multilingual needs of our users, automated tuning of the system and also propose a novel application of community detection algorithm. As of November 2020, our system contains 100k+ stickers, has been deployed for 15+ months, and is being used by millions of users.

INTRODUCTION

In messaging apps such as Facebook Messenger, Hike, etc., new modalities are extensively used to visually express thoughts and emotions (e.g., emojis, gifs, and stickers). Emojis are used along with text to convey emotions in messages (Donato and Paggio 2017). Unlike emojis, stickers provide a graphic alternative to text messages. Hike stickers are composed of an artwork (cartoonized characters and objects) and a stylized text (Figure 1). They convey rich expressions along with the message. 100k+ stickers are available for download in popular messaging apps.

Once a user downloads a sticker pack, it gets added to a palette, which can be accessed from the chat box. However, discovering the right sticker while chatting can be cumbersome because it is not easy to think of the best sticker that can substitute your utterance. Apps like Hike and Line offer type-ahead sticker recommendation (SR) while typing (Figure 1) to alleviate this problem. Compared to emoji prediction which predicts a few set of emotions, there are tens of thousands possible utterances in text and their corresponding stickers which makes SR problem more complex (Barbieri, Ballesteros, and Saggion 2017).

The latency of generating such SR should be in tens of milliseconds to avoid perceivable delay during typing. This is possible only if the system runs end-to-end on the mobile without any network calls. Furthermore, a large fraction of Hike users use low-end phones, so we need a solution which is efficient both in terms of CPU load and memory requirements.

Before this work, SR on Hike app used string matching of the typed text to the tags that were manually assigned to each sticker. Recall of string matching is limited by exhaustiveness of tagging. However, there are many ways of

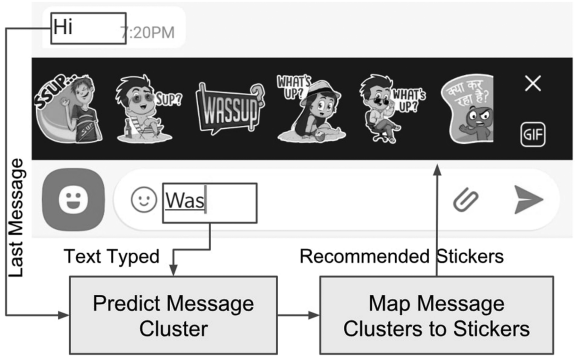

FIGURE 1 Sticker recommendation (SR) UI on hike and a high level flow of our 2 step SR system

expressing same message. For instance, people often skip vowels as they type, for example, "where are you" → "whr r u," "ok" → "k. " This is further exacerbated when people transliterate messages from their native language to Roman script, for example, phrase "acchha" (Hindi for "good") is written in many variants – "accha," "acha," "achha, " etc. such variants proliferate because pronunciation of certain words varies with region. We observe 343 variants of "kya kar raha hai" ("What are you doing") in our dataset. Hence, capturing all variants of an utterance as tags is hard.

Decomposing SR: SR can be formulated as a supervised task by learning the most relevant stickers for a given context defined by the previous message and the text typed by the user. However, due to frequent updates in the set of available stickers and a massive skew in historical usage toward a handful of popular stickers, it becomes difficult to collect unbiased data to train an end-to-end model. Moreover, an end-to-end model requires frequent retraining to support new stickers and the updated model would have to be synced across devices. Such regular updates, which are >10 MB in size, will be prohibitively expensive in terms of data costs. Thus, we decompose the SR task into two steps. First, we predict the message that a user is likely to send based on the chat context and what the user has typed. Second, we recommend stickers by mapping the predicted message to appropriate stickers. We automatically update the mapping frequently based on relevance feedback observed on recommendations and incorporate new stickers as they are launched.

We have launched the SR model in various geographies by taking care of the multi-lingual expression needs of our user base.

In this article, we discuss the major aspects of our sticker recommendation model and how we scale to our diverse user base. We propose the following solutions:

Chat message clustering: As mentioned, many chat messages are simply variants of each other. Similar to SmartReply (Kannan, Kurach, and Ravi 2016) which clusters the short responses having similar intent, we cluster frequent messages which are orthographic variants or semantically similar. Unlike their approach to apply semi-supervised learning for clustering, we learn embeddings of chat messages in an unsupervised manner. Then we cluster the representations with HDBSCAN (McInnes, Healy, and Astels 2017). We investigate various encoders to learn the embeddings and show that the use of charCNN (Kim et al. 2016) with transformer (Vaswani, Shazeer, and Parmar 2017) is highly effective to capture semantics of chat phrases. The clusters obtained are used as classes for our message prediction model. This helps us to drastically reduce the number of classes in the classifier while keeping most frequent message intents of our corpus.

Hybrid message prediction model for low-end smartphones: Running inference with a neural network (NN) model for message prediction is challenging on low-end mobile devices with severe memory limitations. (Gysel, Motamedi, and Ghiasi 2016). The size of an NN model trained for message prediction exceeds the memory limitation even after quantization. We present a novel hybrid model, which runs efficiently on low-end devices without significantly compromising accuracy. Our system is a combination of an NN model (on the server), that processes chat context and predicts message cluster, and a Trie-based model that processes typed text on the client. The first component is not limited by memory and CPU. Trie search is efficient for retrieving message cluster based on typed text and can be executed for each character typed. Hence, the system satisfies the latency constraints. Scores from these two components are combined for final message prediction.

Sticker Mapping and Scoring: We effectively manage stickers in the system by mapping and scoring them against message intents. The mentioned approaches help us overcome limitations of tagging process, continuously rank relevant stickers and deal with new stickers.

A graph-based joint message clustering approach: To address expression needs of a multi-lingual user base without introducing complexities in managing message clusters in the system implementation, we need consistent message clustering across models in different geography. We do not have a universal embedding model that appropriately captures the semantics of phrases across geographies, but we have effective embedding models that capture the

semantics of phrases in different geographies. We prepare a unified phrase similarity network from each model's similarity information. Then we use community detection on the graph to build a set of globally relevant message clusters.

Lastly, we demonstrate the efficacy of our system on both offline evaluation and real world deployment performance. We discuss other challenges in deployment such as multiple languages in India, serving and update frequency of models.

In summary, we make the following contributions:

First, a novel system for large-scale multi-lingual type-ahead SR within a messaging application. Our deployed system automatically updates and tunes itself using updated online chat corpus. We decompose the SR task into two steps: message prediction, sticker substitution. Second, we describe an unsupervised approach to cluster semantically similar chat messages. We evaluate different encoders to learn message embeddings. Next, we define a novel hybrid message prediction model, which can run efficiently with low latency and low memory footprint on low-end smart-phones. We demonstrate that this hybrid model has comparable performance with a server-only NN-based message prediction model. Finally, we describe a novel application of community detection on graphs to simplify the joint clustering of messages across geographies.

This article is extension of the work Laddha et al. (2020).

RELATED WORK

Emojis are widely used in social media. Barbieri et al. (2018) predict which of the top-20 emojis are likely to be used in an Instagram post based on the text and image. Emojis are majorly used along with text, whereas stickers are independent messages that substitute text. Thus, to have effective SR, we need to predict the likely utterance, not just the emotion. Since the possible utterances are more than emotions, our problem is harder than emoji prediction.

There exists a large body of research on conversational response generation. Xing, Wu, and Wu (2017) design an end-to-end model leveraging a hierarchical RNN to encode the input utterances and another RNN to decode possible responses. Zhang, Galley, and Gao (2018) describe a model that explicitly optimizes for improving diversity of responses. Yan, Song, and Wu (2016) proposed a retrieval-based approach using a DNN-based ranker that combines multiple evidences around queries, contexts, candidate postings, and replies. Smart Reply (Kannan, Kurach, and Ravi 2016) proposed a system that suggests short replies to e-mails which are high quality as well as diverse.

Akin to our system, Smart Reply generates clusters of responses with same intent. They apply semi-supervised learning to expand the set of responses starting from few manually labeled responses for each semantic intent. However, we follow an unsupervised approach to discover message clusters. A unique aspect of our system is that we update the message prediction by incorporating whatever the user has typed so far.

There is a parallel research thread around learning effective representations for sentences that can capture sentence semantics. Skip Thought (Kiros, Zhu, and Salakhutdinov 2015) is learns to generate the context sentences for a given sentence. It includes sequential generation of words of the target sentences, which limits the target vocabulary and increases training time. Quick Thought (Logeswaran and Lee 2018) circumvents this problem by replacing the generative objective with a discriminative approximation, where the model attempts to classify the embedding of a correct target sentence given a set of sentence candidates. BERT (Devlin et al. 2018) predicts the bidirectional context to learn sentence representation using transformer (Vaswani, Shazeer, and Parmar 2017). Yang, Yuan, and Cer (2018) propose the Input-Response model that we evaluate in this paper. Unlike these works, we add CharCNN (Zhang, Zhao, and LeCun 2015; Kim et al. 2016) in our encoder to learn similar representations for phrases that are orthographic variants of each other.

CHAT MESSAGE CLUSTERING

As mentioned, we cluster frequent messages in our chat and use them as classes in the message prediction model. For covering large fraction of messages in our chat corpus with few clusters, we need to group all messages having same intent into a single cluster. This should done without compromising the semantics of each cluster. For efficiently clustering, obtaining messages embeddings that effectively capture their meaning is critical.

Encoder

The architecture of the encoder is shown in Figure 2. Input to the encoder is a message and the output is a dense vector. To represent a word, we use an embedding composed of two parts; a character-based embedding aggregated from character representations, and a word-level embedding to learn context representation. To generate the character-based embedding, we use a character CNN (Kim et al. 2016) that leverages sub-word information to learn similar representations for orthographic variants of the same word. For a word, the character representation can be obtained by stacking the character-level embeddings in a matrix and

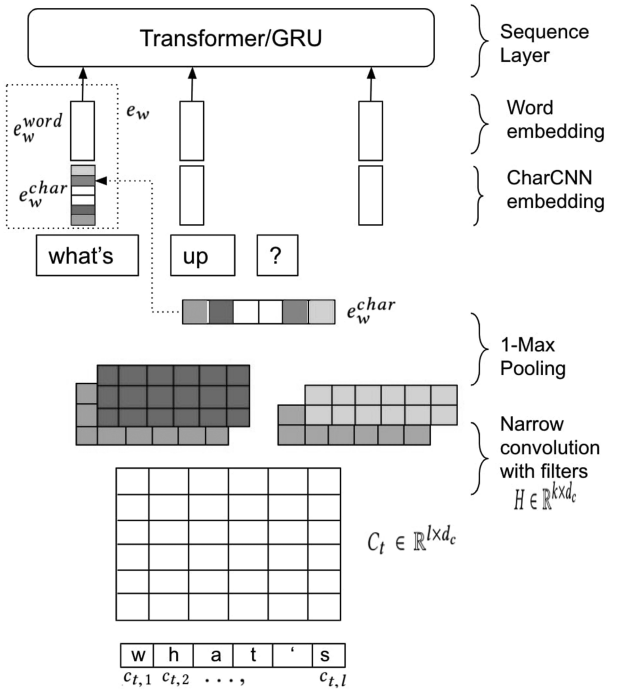

FIGURE 2 Encoder which comprises of a character based CNN layer for each word and GRU/transformer layer at word-level

applying a narrow convolution with a filter of k width. A k width filter is assumed to capture k-gram features of a word. We have multiple filters for particular a width k, that are concatenated to obtain a character-level embedding for a word. The character-level representation of the word is concatenated with the word-level embedding to get the final word representation.

To capture the sequential properties of a message, we explore two architectures:

Gated Recurrent Unit (GRU) (Chung et al. 2014), an improved RNN to solve the vanishing gradient problem. We take the final step representation in the GRU to be the message embedding.

Transformer is an architecture solely based on attention to curb the recurrence step in RNN (Vaswani, Shazeer, and Parmar 2017). It uses multi-headed attention to capture various relationships among the words. We compute the message embedding by averaging the representations in the final layer of the transformer.

Model architecture

Akin to Yang, Yuan, and Cer (2018), we use the model architecture shown in Figure 3. The input to our model is a tuple, (m_i, m_{i+1}), extracted from a conversation between

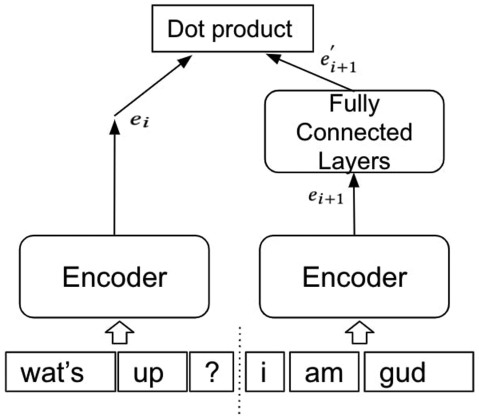

FIGURE 3 Model architecture for learning message embedding

two users. Messages m_i and m_{i+1} are encoded using the encoder and represented as e_i and e_{i+1}, respectively. The parameters of both the encoders share weights. Hence, both e_i and e_{i+1} represent the encoding of message in same space. We transform embedding e_{i+1} to reply space by applying two fully connected layers to obtain the response embedding e_{i+1}. Finally, the dot product of the input and output message embedding is used to score the replies. It maximizes the score of gold reply message. Within a batch, each m_{i+1} serves as the correct response to its corresponding input m_i and all other instances are assumed as negative replies. It is computationally efficient to consider all other instances as negative because we do not have to explicitly encode negative examples for each instance.

Message clustering

We cluster frequent messages using the embeddings learned as above. Our goal is to have a single cluster for all the orthographical variations, acronyms of a message. The variants of a phrase increases with the ubiquity of that phrase. For example, "good morning" has ~300 variants while less frequent phrases have fewer variants. This poses a challenge when applying a standard density-based clustering algorithm such as DBSCAN because it is difficult to decide a single threshold for drawing cluster boundaries. To handle this uncertainty, we choose HDBSCAN to cluster chat messages. It builds a hierarchy of clusters and handles the variable density of clusters in a time efficient manner. After building the hierarchy, it condenses the tree, extracts the useful clusters, and assigns noise to the points that do not fit into any cluster. Further, it does not require parameters such as the number of clusters or the distance between pairs of points to be considered as a neighbor, etc.

The only necessary parameter is the minimum number of points required for a cluster. All the clusters including the noise points are taken to be the different classes in our message prediction step.

MESSAGE CLUSTER PREDICTION

We use previous received message in chat as the only context signal. Upon receiving a message, we predict the likely response. As the user starts typing, we update our message prediction and SR in real time after every character typed. The prediction latency should be in tens of milliseconds so that the user's typing experience is not adversely affected.

We pose message prediction as a classification problem where we train a model to score the response messages. An NN-based classification model that accepts the last received message and the typed text as inputs could be a solution, but running inference on such a model on mobile devices is a challenge (Gysel, Motamedi, and Ghiasi 2016). The size of the NN model that needs to be shipped explodes since we have a large input vocabulary and a large number of output classes. To overcome this, we evaluate two orthogonal approaches: (1) quantization scheme to reduce the model size; and (2) hybrid model, composed of an NN component and a trie-based search.

Quantized message prediction model

We train an NN for message prediction task where the input is the last received message and the typed text, the output is the message cluster scores.

An active area of research is to reduce the model size and the inference times of NN models with minimum accuracy loss so that they run efficiently on mobile devices (Howard, Zhu, and Chen 2017). One approach is to quantize the floating point representations of the weights and the activations of the NN from 32 bits to fewer bits. We use a quantization scheme (Jacob, Kligys, and Chen 2018) that converts both weights and activations to 8-bit integers and use a few 32-bit integers for biases. This reduces the model size by a factor of four. We employ quantize aware training to reduce the effect of quantization on accuracy, that is, we use quantized weights and activation to compute the loss function during training as well. This ensures parity during training and inference. While performing back propagation, we use full precision float numbers because minor adjustments to the parameters are necessary to effectively train the model. Finally, we obtain an ~9 MB model.

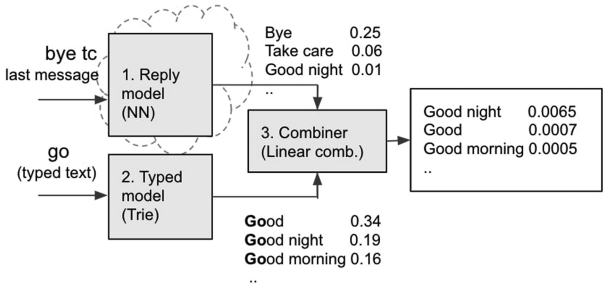

FIGURE 4 Example of hybrid model message cluster prediction. Message prediction from neural network (NN) based reply model on server (1) is combined (3) with trie model prediction on client (2) for final message prediction score

Hybrid message prediction model

We build a hybrid message prediction model, where a resource intensive component is run on the server and its output is combined with a lightweight on-device model to obtain message predictions (Figure 4). The three components in our hybrid model are:

Reply model: We build an NN model which takes the last received message as input and outputs reply probabilities for each message cluster. When a message gets routed to its recipient, the reply model is queried on the server and the response predictions are sent to the client along with the message. The message clusters that have a reply probability above a threshold are sent to the clients.

Typed model: We retrieve the relevant message clusters for a given typed text by querying a trie. It stores <phrase> as key and <(message cluster id, frequency)> tuple as value at leaf nodes. We add the frequency of the phrases from our chat corpus as additional information in the trie, for scoring retrieved entries. We create a trie with top frequent 34*k* phrases and ~7500 message clusters. In the serialized form, the size of the trie is around 700 KB. This is small enough for us to ship to client. Trie is also interpretable and makes it easy for us to incorporate any new phrase.

Combiner: A final score for a message cluster is computed by using a weighted combination of scores from reply model, trie model, and typed string length. The weighting of terms is designed such that as a user types more characters, contribution from the reply model vanishes unless the message cluster is not predicted from the trie. The weights are chosen by trial and error.

MESSAGE CLUSTER TO STICKERS MAPPING

We use a message cluster to sticker mapping to suggest suitable stickers from the predicted message clusters. Quite a few stickers are applicable to a message cluster. For popular message clusters, there are many of relevant stickers. At a time users can only view four stickers in the recommendation panel until they scroll. So, we need a ranking system to keep appropriate stickers on the top. The ranking should take care of discoverability of new and old stickers, overall preference of the stickers by users over time. We need a way to score stickers in a message cluster initially when they are produced as and to update the score based on user feedback.

Computing initial sticker mapping score

During production, stickers are tagged with phrases which can replace a corresponding sticker (e.g., "good morning") and with high level tags (e.g., "emotion lol"). The high level tags are less precise then tag phrases but represent a larger meaning associated with the sticker. We use both for mapping of message clusters to stickers.

The tags capture necessary sticker intent but do not cover all the message variations. Example, "good morning" may be a tag of a sticker, but "gm" may not be there as a tag. Also, as the stickers are produced at different times by different people, the tagging does not need to be consistent.

We can map stickers to message clusters by matching their tags against phrases in these message clusters. To overcome limitations of the tagging, while matching the tags, we use phrase embedding computed earlier. So a sticker S can be mapped to message cluster M, with score $s = \max_{t \in \text{tags}(S), p M} e_t.e_{p\in}$ where e_t and e_p are embedding of tag t and phrase p, respectively.

We learn embeddings of higher level attributes of stickers, just like embeddings of phrases, by keeping those attributes as a message in the place of sticker. We also use this to compute the sticker mapping score $s = \max_{t \in \text{tags}(S), p \in M} e_t.e_p$ where e_t and e_p are embedding of a higher level attribute t of the sticker S and phrase p, respectively. This score is not as accurate as above one but it helps map stickers which are not well represented using their phrase tags.

We can also use embedding of a sticker directly to match it against the message clusters, with score $s = \max_{p \in M} e_s.e_p$ where e_s and e_p are embedding of sticker S and phrase p, respectively. But this approach works only for stickers that have a large historical data available.

Updating sticker mapping score from usage feedback

The sticker mapping score should reflect the users' sticker preference. We can pose the problem as computing expected reward for pulling levers in a multi-arm bandit problem (Agrawal and Goyal 2012) (MAB), where different stickers in a message cluster represent different arms. For each message cluster, the reward distribution from each arm should be computed separately. We use Thompson sampling (Agrawal and Goyal 2012) on each message cluster to compute scores for each sticker mapped to it. For this, we collect the data < predicted message cluster, stickers recommended, sticker selected by the user>. From this we can derive # times sticker was shown and # times sticker was used, for each message cluster. Thompson sampling updates the reward distribution based on this data.

We make the sticker ordering more personalized by maintaining the score distribution at a fine granularity. We identify user segments having varying sticker taste. Then we run the MABs per message cluster, per user segment, per geography.

To introduce a new sticker, first, we find relevant the message clusters to assign it into. Then we keep a prior score in the reward distribution in those message clusters. After a decent amount of exposure their scores will get updated to required values.

Addressing multi-lingual user base

Many of our users speak more than one language. Though they speak multiple languages, extent of usage of each language is different. First language may be used to convey all types of messages, but second language may be used in few contexts like greetings, usage of popular phrases, formal talk, etc. The first language, second language (and third) varies from user to user in different geographies. Our embeddings and recommendation should honor these mixed usage. We ensure this by preparing models for each geography instead of for each language. The data observed from individual users of a geography actually shows prevailing usage of multiple languages in that region. So all the models trained on this data invariably capture the required mixing of languages in those geographies.

There are many users who in addition to the languages prevailing in their preferred geography use phrases from languages that are less used in that geography but common in another geography. For example, there are users who like to speak both Marathi and Bengali but such users are not prominent in either of corresponding localities. If such users were given the model from one geography, it

would not complete their expression needs. If there were models for each potential combination of languages used by a user, we could have used that. However in practice, it is difficult to train and manage so many models due to the large number of such potential combinations, challenges in segregating required corpora and availability of enough data per combinations.

A potential approach is to ship multiple models to a user, depending on the user's language preferences. This will significantly increase the size of models shipped. As multiple models become active in SR, the recommendations may get mixed up inappropriately many times.

Another approach is to ship a single model to the user but augment it by the essential message clusters used by the user from the second model. The trie and sticker mapping components used in our algorithms can be extended with new message clusters by adding appropriate entries in trie and by adding new entries in sticker mapping. This will help the users send additional stickers through SR which were not discoverable earlier. We can figure out the relevant message clusters that need to be extended with existing model from the users' sticker usage (through some other sticker discovery channel) that were not present in their model.

When we augment message clusters from one model to another, there will be duplication of message clusters. For example, top phrases (e.g., "hello") used in one geography will be present in another geography with high frequency of usage, due to which the phrase has to be now assigned with two message clusters. This will result in unnecessary duplication of phrases to different message clusters, affecting score computation and size. It is difficult to avoid such duplicates. As different models are prepared independently, there will be significant difference between message clusters observed across models. For example, message cluster of "hello" obtained from one geography could contain the phrase "*vanakkam*," whereas its message cluster obtained from another geography need not have that phrase. Keeping only one of these message will cause losing some phrases, but keeping both will introduce duplicate entries. Managing message cluster ids from multiple models requires that we have to encode the model specifier along with each message cluster entry in the required data structures. Apart from the extra memory needed to store this, it brings additional complexities in deploying, maintaining the system in the client and the server.

If the message clusters across models were consistent (i.e., if a phrase p is present in two clusters in the two models, then other members of the two clusters should be same), then we need not worry about such complexities and we can avoid potential duplication of phrases. So, we need a way to make the message clusters across models to share a consistent grouping. To achieve this consistency either we should adjust the message clusters across models post the individual model's message cluster preparation or prepare the message clusters by clustering all phrases across geographies together. In the overall flow, we do this before training the message cluster prediction models. We briefly cover these two approaches we tried to consolidate clusters.

Post clustering consolidation

We developed a cluster merge algorithm to combine all the clusters of different demographics. We chose one geography as target and tried to merge clusters from each of the other geographies. For each cluster built in one demography, we check the nearest cluster in target global clusters. If the similarity (number of common phrases) is above a threshold, we merge the two cluster (threshold is tuned for each demographic). This algorithm gives us good consolidated clusters with reduced redundant words across all languages and good quality clusters for both native and global language. This approach works dealing with one or a few additional geography, but the quality degrades when working with more geographies.

Combined message clustering

If embedding of phrases learned from multiple models were comparable, then we could have jointly clustered the phrases. Unfortunately, embeddings obtained from models from different geographies may not lie in the same space, that is, embeddings of common phrases tend to have different representations. Building a single embedding model on the combined corpus fails to capture semantics of regional phrases as they get under represented in the combined corpus.

Translation-based approach: Using translation approach mentioned in (Artetxe, Labaka, and Agirre 2016; Joulin, Bojanowski, and Mikolov 2018), we tried to align embeddings from different models to a single target embedding. Common words present across geographies help us in this regard. We treated the common words across geographies as equivalence words, which will be used to learn the transformation. We kept one location embedding to be the target domain and transformed all other location embedding to the target domain. We used training objective described in (Joulin, Bojanowski, and Mikolov 2018) to train the transformation matrix.

We used (Joulin, Bojanowski, and Mikolov 2018) training objective which is inspired by cross-domain similarity local scaling (CSLS) retrieval criteria to reduce the "hubness problem" (Artetxe, Labaka, and Agirre 2016). We

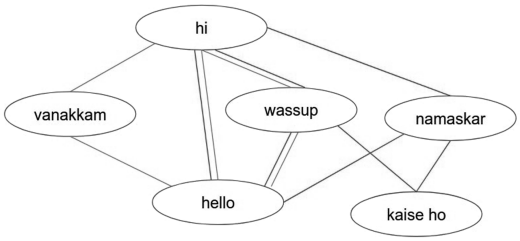

FIGURE 5 A schematic graph representing universal phrase similarity. Blue edges indicate that corresponding phrases are similar in embedding model trained on one geography. Red edges indicate phrase similarity in another model trained on another geography. When we combine the edge scores across models and put thresholds on them, only commonly agreeing edges will remain

transformed different geographically learned embeddings to the same target global space. Although phrase embeddings are comparable and k nearest neighbors are similar, the scale of similarity scores of semantically similar phrases are still different (similarity of "hey" and "hi" is 0.98 while similarity of "hey" from source language to "hi" is 0.8). It caused difficulties in clustering them, so we employ an alternate approach.

Graph-based approach: We create a graph (Figure 5) where nodes are union of all phrases across all geographies. Two nodes are connected with an edge if they are similar to each other in one or more embedding models. These edges are filtered by applying some threshold on total weighted mean similarity score computed across different models. The weighing is based on frequency of the corresponding nodes in corresponding geography to ensure that importance of the score increases with increase in data points. If a set of messages are similar to each other, then there is strong connectivity among corresponding nodes in this graph. These set of message should constitute a message cluster. So, we can pose message clustering problem as community detection problem in this graph.

There are several community detection techniques (Clauset, Newman, and Moore 2004; Raghavan, Albert, and Kumara 2007) available for graphs. We find k-clique-community detection algorithm (Palla et al. 2005) useful. As it discovers overlapping communities, it causes duplicate entries for few phrases. But instead of severely affecting the performance, overlapping nature helps us properly capture the cluster semantics. By varying k from 5 to 3, we capture highly precise followed by loosely precise clusters. Clusters prepared this way on universal list of phrases are comparable to those prepared in individual model.

EXPERIMENTS

We describe the dataset used for training the SR system. Next, we quantitatively evaluate the message embeddings on manually curated datasets. We show qualitative results after clustering to show the effectiveness of the message embeddings. Then, we present a comparison of the NN model and the hybrid model for the message cluster prediction.

Dataset and pre-processing

To ensure user privacy in data collection, we strip user identity and replace it with anonymous ids. We randomly sample 10 percent of anonymous ids to collect the dataset for a period of 5 months, from a particular geography for which we need to build a model. This will have a mixture of languages used in that region. This dataset is then pre-processed to extract useful conversations from the chat corpus. After pre-processing the data, we create tuples of the current and the next message for training the message embedding models. We get 27million tuples. Since stickers are used to convey short messages, we remove all the tuples having at least one message with more than five words. Our input vocabulary consist of top frequent 50k words in our corpus. The dataset is randomly split into training and validation sets with 520k examples in validation set.

Message embedding evaluation

We evaluate the embeddings generated from different architecture on manually labeled data. First, we describe the baselines.

Message embeddings models

We compare the embeddings obtained from architecture (Figure 3) and its variants which are as follows: (1) *Trans+CharCNN* – encoder as shown in Figure 2 with transformer; (2) *Trans* – uses only word-level embedding e_m^{word} as input to transformer; (3) *GRU+CharCNN* – encoder as shown in Figure 2 with GRU; and (4) *GRU* – uses only word-level embedding as input to GRU.

Phrase similarity task

We create a dataset which consists of phrase pairs labeled as similar or not-similar. For example ("ghr me hi," "room me h"), ("majak kr rha hu," "mjak kr rha hu") are labeled as similar while ("network nhi tha," "washroom gyi thi"), ("it's normal," "its me") are labeled as not-similar. Possible similar examples are sampled from top 50k frequent phrases using two methods: (a) Pairs close to each other

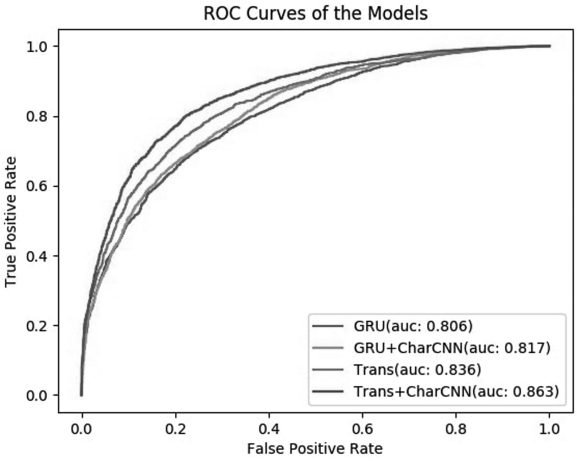

FIGURE 6 Performance of the message embedding models on phrase similarity task

in terms of minimum edit distance. (b) Pairs having words with similar word embeddings. Possible negative examples are sampled randomly. These pairs were annotated by three annotators. Pairs with disagreement within the annotators were dropped. Finally, the dataset has 3341 similar pairs and 2437 non-similar pairs.

To evaluate the models, we calculate the cosine similarity between the embeddings of phrases in a pair. The ROC curves of the models are shown in Figure 6. We observe that transformer has significant improvement over GRU. This is due to better capturing of semantics of phrases using self-attention. CharCNN improves the performance of GRU and transformer. It is able to capture chat characteristics which occur due to similar sounding sub-words. Trans+CharCNN model performs the best and shows ≈6 percent absolute improvement in terms of AUC over the GRU baseline.

Chat slang task

We manually labeled a set of words which are common slangs or variants used in conversation and mapped them to their corresponding correct form. We considered two types of errors. (1) Spelling variants (spell) – in this class we considered words which can be converted to their correct form by either replacing the characters nearby in keyboard ("hsn" → "han") or addition of vowels ("phn" → "phone," "yr" → "yaar"). (2) Synonym words (syn) – we considered words which have same meaning and are phonetically similar but cannot be converted into their correct form by spelling correction. For example ("plzz" → "please," "n8" → "night"). Most users widely use such transformations of words. We labeled 213 words for spelling variants and 78 synonym words.

To evaluate the message embedding, we retrieve top 10 nearest neighbors from top frequent phrases for each query of labeled dataset and use two evaluation metrics: (1) Precision@10 (P@10) counts the number of correct phrases in top 10 (retrieved phrases that are semantically similar to query); and Recall@10 (R@10) calculates whether labeled phrase of query is present in top 10 or not. Two human judges did the annotation, disagreements were resolved upon consensus.

Table 1 shows the performance of different variants of encoder to learn message embedding. Transformer and GRU have comparable recall on spell data because it contains single words. Hence, transformer does not have much advantage over GRU. CharCNN improves the recall performance on synonym word variants for both the architectures. CharCNN is able to capture certain chat characteristics occurring due to similar sounding sub-words. For example. "night" and "n8," "see" and "c," "what" and "wt" are used interchangeably. CharCNN learns such nuances and performs well in matching those words to semantically similar words.

Message clustering evaluation

Figure 7 shows a qualitative evaluation of clusters obtained from HDBSCAN algorithm; phrases from the same cluster are represented with the same color. We select the top 100 clusters based on the number of phrases present in cluster. We project the phrase embeddings learned from our model in 2-dimension using TSNE (Maaten and Hinton 2008) for visualization. We observe that various spelling variants and synonyms of a phrase are grouped in a single cluster. For example, "good night" cluster includes phrases like "good nighy," "good nyt," "gud nite." Our clustering algorithm captures fine-grained semantics of phrases. For instance, "i love u," "i luv u" phrases belong to one cluster while "love u too," "love u 2" belong to a different cluster which helps us show accurate SR for both clusters. If a user messages "i love u" then showing "i love u too" stickers as reply is more relevant than showing "i love u" stickers. Our message embedding is able to cluster phrases like "i love u baby" and "i luv u shona" in a cluster distinct from the "i love u" cluster. This helps us deliver more precise SR since we have different stickers in our database for phrases like "i love u" and "i love u baby."

Message cluster prediction evaluation

We compare our hybrid model and the quantized NN model on the following metrics: (1) number of characters that a user needs to type for seeing the correct message

TABLE 1 Evaluation of different architecture of message embedding model on chat slang task

	Spell		Syn		Overall	
Models	**P@10**	**R@10**	**P@10**	**R@10**	**P@10**	**R@10**
Trans + CharCNN	73.3	67.1	89.2	**51.3**	**77.6**	**62.9**
Trans	70.8	67.6	86.7	47.4	75.0	62.2
GRU + CharCNN	71.6	66.7	86.6	48.7	75.6	61.9
GRU	70.7	67.6	86.2	44.9	74.8	61.5

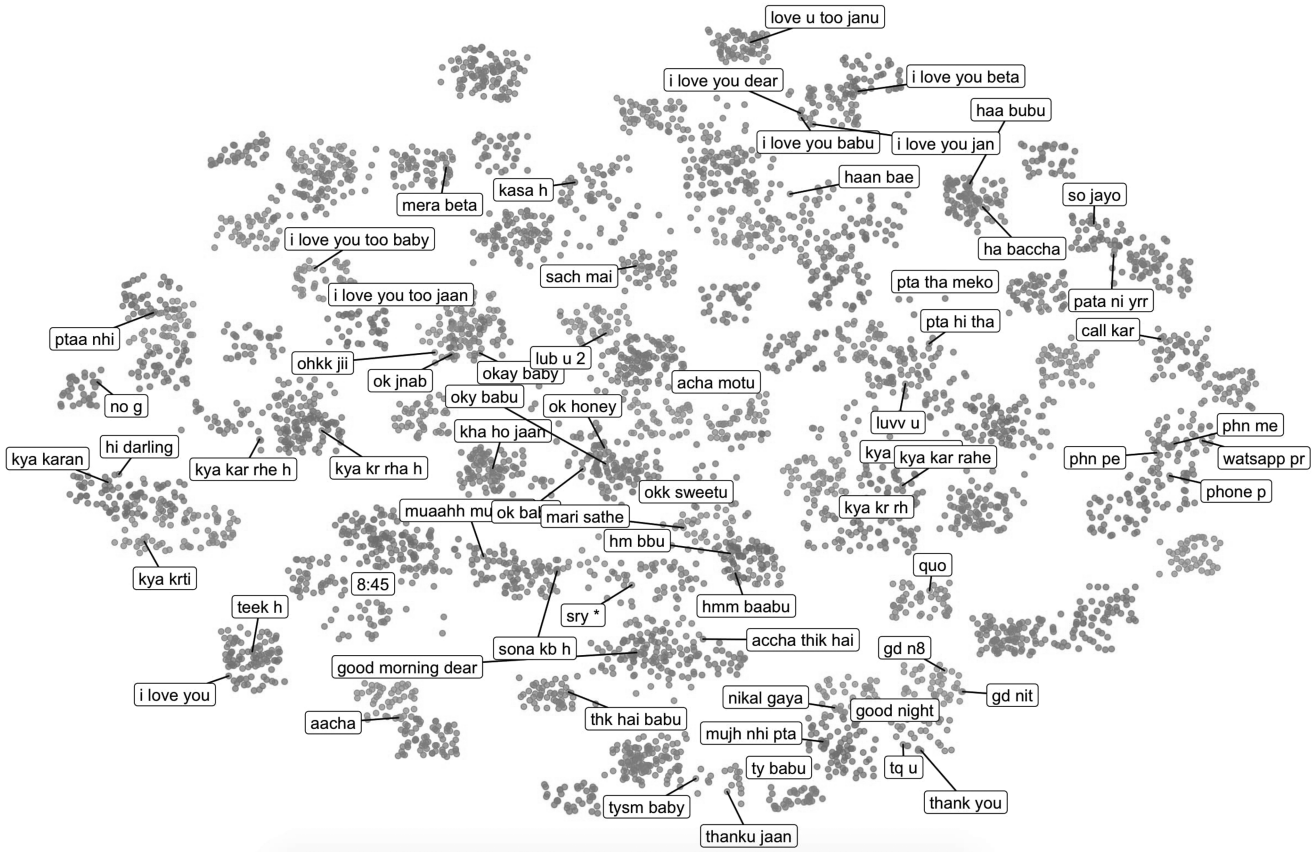

FIGURE 7 HDBSCAN clusters of top frequent phrases using message embedding produced by GRU-CharCNN model. The points represents chat message vectors projected into 2D using TSNE

cluster in top three positions (# of Character to be typed); (2) how many times the model shows a wrong message cluster before predicting the correct one in first three positions (# of times inaccurate prediction); and (3) fraction of messages that could be retrieved by a model in first three positions with a prefix of that message (fraction of msg retrieved). Lower the number for the first two metrics, the better the model is; while the third metric should be high.

For training the message prediction NN model, we collect pairs of current and next messages from complete conversation data. Our training data had 10M such pairs. We randomly sample 38k pairs for testing. We curate the training data by treating all prefixes of the next message as typed text and its message cluster as the class label. We consider only top 7500 message clusters based on the total frequency of their phrases. Selected clusters cover 34k top frequent reply messages in our dataset. The hybrid model is a combination of two models. The first model predicts the next message given the current message. It is trained directly from pairs of consecutive messages, (current, next), in our corpus, where the next message was mapped to its corresponding message cluster. In the second model, we build a trie based on the typed message.

Results of the various models are shown in Table 2. The quantized NN model performs better in terms of # of characters to be typed. This is expected because the NN model learns to use both inputs simultaneously whereas the combiner (last function in hybrid model) used is a linear combination of just three features. The hybrid model performs slightly better in terms of # of times inaccurate

TABLE 2 Performance of the message prediction models

Method	# of character to be typed	# of times inaccurate predictions shown	Fraction of msg retrieved
Quantized NN (100d)	**1.58**	1.47	0.978
Hybrid	2.84	**1.22**	**0.991**

recommendations shown and fraction of messages retrieved. Compared to the quantized model, the hybrid model needs approximately one more character to be typed on an average to get the required predictions.

However, the quantized NN is ∼9.2 MB in size, which goes beyond the permissible memory limits set in our use case. The on-device foot print for our hybrid model is below 1 MB. Also, the trie-based model can guarantee retrieval of all message clusters, with some prefix of the message. If the message is not interfered by another longer but more frequent message, the message class can be featured in top position itself, with some prefix of the message as input or with full message as input. A pure NN-based model cannot make such guarantees. Hence, more than 2 percent improvement in the third metric shown in Table 2. Given that this SR interface is a heavily used interface for sticker discovery, if a sticker is not retrieved through this recommendation, the user might think that the app does not support the message class or the app does not have such stickers. So, making the retrieval of message cluster close to 100 percent is critical for our SR models.

DEPLOYED SYSTEM

There are multiple languages spoken in India. We segregate our data based on state and train message embedding models on each geography. Then we jointly cluster all messages using the graph-based clustering algorithm. Then we curate relevant message clusters for each geography based on their presence in those geographies and train separate prediction models for each state. It ensures that frequent chat phrases distribution does not get skewed toward any single languages. We obtain the primary language of users from the preferences set by them or infer it from their sticker usage. We ship corresponding language's model to the user.

We decompose our message prediction system in two steps. It helps reduce the size of the model shipped to client. It also reduces the download failures on sketchy network. When a message is being sent from user A to user B, server fetches the response message cluster scores corresponding to the model assigned to user B. To maintain high speed of message delivery, we serve the reply model by caching the top 300k message in each geography. For running typed model based on trie, we ship the trie and message cluster to sticker mapping files to client. When a user logs in, client checks if an updated model is available. If trie or sticker mapping file has been updated, client downloads those files.

We observe that the top phrases in a corpus rarely change. Only event specific (e.g., "cricket world cup," "movies") or festival specific phrases get updated which depend on seasonality. So, we freeze our message clusters for each geography and do not retrain the message embedding model. We only need to update frequency of phrases for message cluster prediction based on seasonality which we fetch from analytics. Model maintenance is not required since we are using unsupervised approach to generate clusters and only need the frequency of phrases for scoring the message cluster in trie. Complete pipeline for message prediction has been automated with anonymous chat corpus. It helps us extend to more languages and requires minimal manual efforts.

We can add new stickers using limited tags decided while creation. The sticker mappings thus created are later scored and ranked using the sticker usage data across users.

User impact

The proposed system has been deployed in production on Hike for 15+ months. Before a full rollout, we conducted state wise A/B tests. We chose user sets of size of the order of 10k as control and test in each experiment group. A/B tests to compare this system with a previous implementation showed an average ∼8 percent relative improvement in the fraction of users who send stickers, among those who send a message. Compared to legacy system, proposed SR system also increases the volume of stickers exchanged.

CONCLUSION AND FUTURE WORK

We present our deployed system for deriving contextual, type-ahead SR within a chat application. We decompose the task into two steps. First, we predict the next message that a user is likely to send based on the last message and the typed text. Second, we substitute the predicted message with relevant stickers. We discuss how numerous

orthographic variations for the same utterance exist in Hike's chat corpus, which mostly contains messages in a transliterated form. We describe a clustering solution that can identify such variants in message embeddings which learn similar representations for semantically similar messages. Message clustering reduces the complexity of the classifier used in message prediction. For example, by predicting one of 7500 message clusters, it is able to cover intents expressed in ≈34k frequent messages. For message prediction on low-end mobiles, we propose a hybrid model that combines an NN on the server and a memory efficient trie search on the device for low latency SR. We show experimentally that the hybrid model is able to predict a higher fraction of overall messages clusters compared to a quantized NN. This model also helps in better sticker discovery for rare message clusters. We discuss ways to introduce new stickers to the SR system and maintain their ordering by capturing relevant user feedback. We also discuss special steps we had to take to deal with language diversity and multi-lingual expression needs of our user base. As of November 2020, our described system has been deployed for eight Indian languages and serving millions of users daily with ∼8 percent relative increase in the fraction of users sending stickers.

In the future, we plan to add character-level CNN for message prediction on client which is memory efficient compared to current quantized NN. Although we construct the message clusters to reduce complexity of the message prediction task, we observe that these message clusters are generic enough to be used in other application such as conversational response generation, intent classification, etc. (Serban, Sordoni, and Bengio 2016). We also plan to incorporate federated learning into our system.

REFERENCES

Agrawal, S., and N. Goyal. 2012. "Analysis of Thompson Sampling for the Multi-armed Bandit Problem." In *Proceedings of the 25th Annual Conference on Learning Theory*. Edinburgh, Scotland: PMLR.

Artetxe, M., G. Labaka, and E. Agirre. 2016. "Learning principled bilingual mappings of word embeddings while preserving monolingual invariance." In *EMNLP*. Austin, Texas: Association for Computational Linguistics.

Barbieri, F., M. Ballesteros, and H. Saggion. 2017. "Are emojis predictable?" In EACL. *Valencia, Spain: Association for Computational Linguistics*.

Barbieri, F., M. Ballesteros, F. Ronzano, and H. Saggion. 2018. "Multimodal emoji prediction." In *NAACL*. New Orleans, Louisiana: Association for Computational Linguistics.

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio. 2014. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." *NIPS 2014 Deep Learning and Representation Learning Workshop*.

Clauset, A., M. E. J. Newman, and C. Moore. 2004. "Finding Community Structure in Very Large Networks." *Physical Review E* 70(6).

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova. 2019. "BERT: Pre-training of deep bidirectional transformers for language understanding." In *NAACL*. Minneapolis, Minnesota: Association for Computational Linguistics.

Donato, G., and P. Paggio. 2017. "Investigating redundancy in emoji use: Study on a Twitter based corpus." In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Copenhagen, Denmark: Association for Computational Linguistics.

Gysel, P., M. Motamedi, and S. Ghiasi. 2016. "Hardware-oriented Approximation of Convolutional Neural Networks." *iclr workshop*.

Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, et al. 2017. "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications." *arXiv preprint arXiv:1704.04861*.

Jacob, B., S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, et al. 2018. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference." In *CVPR*. Utah, USA: IEEE.

Joulin, A., P. Bojanowski, T. Mikolov, H. Jeťgou, and E. Grave. 2018. "Loss in translation: Learning bilingual word mapping with a retrieval criterion." In EMNLP. *Brussels*, Belgium: Association for Computational Linguistics.

Kannan, A., K. Kurach, S. Ravi, T. Kaufmann, A. Tomkins, B. Miklos, G. Corrado, et al. 2016. "Smart Reply: Automated Response Suggestion for Email." In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, USA: ACM.

Kim, Y., Y. Jernite, D. Sontag, and A. M. Rush. 2016. "Character-Aware Neural Language Models." In *AAAI*. Phoenix, USA: AAAI press.

Kiros, R., Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler. 2015. "Skip-thought Vectors." In *Advances in neural information processing systems*. Curran Associates, Inc.

Laddha, A., M. Hanoosh, D. Mukherjee, P. Patwa, and A. Narang. 2020. "Understanding Chat Messages for Sticker Recommendation in Messaging Apps." In *AAAI*. AAAI Press.

Logeswaran, L., and H. Lee. 2018. "An Efficient Framework for Learning Sentence Representations." *ICLR*.

Maaten, L. v. d., and G. Hinton. 2008. "Visualizing Data using t-SNE." *Journal of machine learning research* 9.

McInnes, L., J. Healy, and S. Astels. 2017. "hdbscan: Hierarchical Density Based Clustering." *The Journal of Open Source Software* 2(11).

Palla, G., I. Dereťnyi, I. Farkas, and T. Vicsek. 2005. "Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society." *Nature* 435.

Raghavan, U. N., R. Albert, and S. Kumara. 2007. "Near Linear Time Algorithm to Detect Community Structures in Large-scale Networks." *Physical Review E* 76(3).

Serban, I. V., A. Sordoni, Y. Bengio, A. Courville, and J. Pineau. 2016. "Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models." In *AAAI*. Phoenix, USA: AAAI Press.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, et al. 2017. "Attention is All you Need." In

Advances in neural information processing systems. *Long beach*, ASA: Curran Associates, Inc.

Xing, C., W. Wu, Y. Wu, M. Zhou, Y. Huang, and W.-Y. Ma. 2017. "Hierarchical Recurrent Attention Network for Response Generation." In *AAAI*. AAAI Press.

Yan, R., Y. Song, and H. 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: ACM.

Yang, Y., S. Yuan, D. Cer, S. yi Kong, N. Constant, P. Pilar, H. Ge, et al. 2018. "Learning semantic textual similarity from conversations." In *Proceedings of The Third Workshop on Representation Learning for NLP*. Melbourne, Australia: Association for Computational Linguistics.

Zhang, Y., M. Galley, J. Gao, Z. Gan, X. Li, C. Brockett, and B. Dolan 2018. "Generating Informative and Diverse Conversational Responses via Adversarial Information Maximization." *Neurips*.

Zhang, X., J. Zhao, and Y. LeCun 2015. "Character-level convolutional networks for text classification." In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume* 1, NIPS'15, 649–57. Cambridge, MA, USA: MIT Press.

AUTHOR BIOGRAPHIES

Abhishek Laddha is a Data Scientist at Flipkart. Previously he worked at Hike messenger. He has worked on various problems related to question answering, knowledge graphs, and conversation modelling. He received Bachelors, Masters in Mathematics and Computing from IIT Delhi.

Mohamed Hanoosh is a Data Scientist at Gojek. Previously he worked at Hike. He has worked on machine learning (ML) problems and Deep Learning (DL) in Natural Language Processing (NLP), Social Networking, and Digital Advertising. He received Masters in CSE from IIT Bombay.

Debdoot Mukherjee is VP, AI at ShareChat. He leads teams working in feed relevance and multimodal content understanding. He has 12+ years of experience in building AI products. Debdoot was leading the AI team at Hike Messenger. He completed Masters in CSE from IIT Delhi. He has 25+ patents, 25+ publications.

Parth Patwa is a senior CSE undergrad at IIIT Sri City. He was a data science intern at Hike Messenger. He is interested in ML, DL and Natural Language Processing. He has authored 5+ publications in top venues.

Dr. Ankur Narang is the VP, AI at Hike Messenger. He has 25+ years of experience in Leadership roles, along with 40+ publications and 15 patents. His areas of interest and expertise include: AI and ML, Big Data Analytics, and High Performance Computing. He has B.Tech and PhD in CSE from IIT Delhi.

How to cite this article: Laddha, A., M. Hanoosh, D. Mukherjee, P. Patwa, and N. Ankur. 2021. "Large scale multilingual sticker recommendation in messaging apps." *AI Magazine* 42: 16–28. https://doi.org/10.1609/aaai.12023