PhotoReply: Automatically Suggesting Conversational Responses to Photos

Ning Ye, Ariel Fuxman, Vivek Ramavajjala, Sergey Nazarov, J. Patrick McGregor, Sujith Ravi

Google

Mountain View, CA

{nye,afuxman,vramavaj,sergeynazarov,mcgregor,sravi}@google.com

ABSTRACT

We introduce the problem of automatically suggesting conversational responses to photos and present an intelligent assistant called PhotoReply that solves the problem in the context of a messaging application. For example, when a user receives a photo showing a dog, PhotoReply suggests responses such as “Aww” and “Cute terrier!” This simplifies composing responses on constrained mobile keyboards, and delights users with uncanny insights into the photos they receive. PhotoReply is an integral part of the Allo chat application, being its predictive assistance feature with highest click-through rate.

We formalize the problem of suggesting responses to images as an instance of multimodal learning which is akin to caption generation models, a topic that has recently received significant attention [10, 21, 24]. We then present a system that “translates” image pixels to text responses. The system includes a conditioned language model, based on an LSTM, which, given an embedding of image pixels and the previous predicted words, calculates the probability of all words in a vocabulary of being the next word in the generated response; and a triggering model trained with image embeddings and concept labels from a large concept taxonomy. We describe training of the models and a thorough experimental evaluation based on crowdsourced datasets and live traffic.

KEYWORDS

image understanding, natural language, generative model, mobile applications, messaging applications

ACM Reference Format:

Ning Ye, Ariel Fuxman, Vivek Ramavajjala, Sergey Nazarov, J. Patrick McGregor, Sujith Ravi. 2018. PhotoReply: Automatically Suggesting Conversational Responses to Photos. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France*. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3178876.3186164>

1 INTRODUCTION

Recent advances in machine learning have led to widespread adoption of “smart personal assistants” such as Amazon’s Alexa, Apple’s Siri, Google’s Assistant and Microsoft’s Cortana [3]. These assistants are designed to help a user complete tasks on a timely basis,

This paper is published under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International (CC BY-NC-ND 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.

WWW 2018, April 23–27, 2018, Lyon, France

© 2018 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC BY-NC-ND 4.0 License.

ACM ISBN 978-1-4503-5639-8/18/04.

<https://doi.org/10.1145/3178876.3186164>

Figure 1: Example responses suggested by PhotoReply.

from setting a timer to placing an online shopping order. In this paper, we describe PhotoReply, a system that gives personal assistants the ability to help users communicate more efficiently. In particular, the system makes it possible for the assistant to automatically suggest responses to photos in a messaging application. For example, consider the screenshot in Figure 1 where a user is sending a photo of their daughter to a friend. When the friend receives the photo, PhotoReply suggests a number of responses such as “Looks fun” and “Aww”. The user who received the photo then has the option of choosing one of PhotoReply’s suggestions or typing their own response.

PhotoReply simplifies composing responses on constrained mobile keyboards, and delights users with uncanny insights into the photos they receive. It was launched last year within Allo [5] and is currently its predictive assistance feature with the highest click-through rate.

PhotoReply is related to SmartReply, a popular feature in Gmail and Inbox that accounts for more than 10% of all mobile responses in Inbox [9]. Given an email thread, SmartReply suggests a text response. The key difference with our work is that, in SmartReply, the input and output both consist of text. In contrast, PhotoReply is an instance of multimodal learning, where the input is an image instead of natural language text. Thus, models that are based on translating natural language text do not apply to PhotoReply.

PhotoReply is akin to caption generation models, a topic that has recently received significant attention ([10, 21, 24]), in the sense that both take an image as input and output a piece of natural language text. However, the kind of text that they output differs drastically. While an image captioning system provides a description of what is depicted on an image, PhotoReply produces text corresponding to the verbal reaction that a human would have when shown an image in the context of a conversation. For example, given an image of a baby sleeping alongside a Teddy bear, a captioning system may generate “a baby sleeping besides a teddy bear”. Clearly, this would be an incorrect response in PhotoReply because it is inappropriate in the context of a conversation. Instead, a response such as “So cute” would be more appropriate – precisely the kind of response that PhotoReply generates. To the best of our knowledge, we are the first to address the problem of automatically generating conversational responses to images.

In order to implement PhotoReply, we faced a number of challenges, including the following:

- As we show in Section 6.1, the architectures used in the image captioning literature (e.g., CNN-LSTM networks as proposed in [21]) produce confidence scores (e.g., sum of word log-likelihood in [21]) that do not correlate well with human ratings. To tackle this, we incorporated a triggering model, based on a feed-forward network, that thresholds the responses produced by an LSTM in such a way that it enables a coverage-quality tradeoff and leads to drastic performance improvements.
- Existing image captioning systems take raw image pixels as input. For privacy reasons, this cannot be done in PhotoReply. Instead, we only had access to image embeddings computed by a pre-trained Inception network [20] with batch normalization [8] and triplet loss [7, 16, 22], which had significant implications in system design.
- While many datasets are available for training and testing captioning systems (e.g., COCO [2]), there were none for generation of conversational responses prior to our work. This required us to build our own training pipeline based on properly anonymized image-response pairs from a chat application; and to evaluate the system via crowdsourcing and live traffic on the chat application.

The contributions of this paper include:

- We formalize the problem of suggesting conversational responses to images. To the best of our knowledge, we are the first to tackle this problem.
- We describe a model to predict conversational responses in chat applications given an embedding (i.e., a feature vector) of the input image.

- We present the implementation of PhotoReply, the system that powers suggestion of conversational responses in the Allo chat application. PhotoReply is the predictive assistance feature with highest click-through rate in Allo.
- We thoroughly evaluate the system using both crowdsourced datasets and live traffic.

The rest of the paper is organized as follows. In Section 2, we present related work. In Section 3, we provide an overview of the system, and in Section 4 we describe the machine learned models. In Section 5, we describe the implementation and deployment of the system; in Section 6 we provide an experimental evaluation of the approach; and we close in Section 7 with concluding remarks.

2 RELATED WORK

PhotoReply is related to the image captioning problem ([10, 21, 24]) in the sense that both have images as input and generate natural language sentences. There has recently been great progress in image captioning with the adoption of deep convolutional neural networks (CNN) and recurrent neural networks (RNN). Karpathy *et al.* [10] proposed a novel way of aligning image regions with text fragments using a bi-directional RNN, given an image and a complete natural language sentence. Then they broke down the image-sentence pairs into corresponding image region and text fragment pairs using the alignment model. Those fragment pairs were further used to train a CNN + RNN architecture for generating texts conditioned on an input image. The trained generative model was capable of describing both image regions as well as full frame images. Vinyals *et al.* [21] proposed a generative network of similar architecture, but LSTM (Long-Short Term Memory [6]) was used in place of vanilla RNN, which led to better performance. Xu *et al.* [24] further developed the idea of [21] and introduced the notion of “attention” into their neural networks, which was able to demonstrate where the model was “looking” at in the image when generating a word.

Over the years, novel approaches were proposed to address limitations researchers have observed with CNN + RNN architectures. Samy *et al.* [1] introduced scheduled sampling to address the “exposure shift” problem, i.e., a model feeds on groundtruth words during training yet on self-generated words during inference, the distributions of which were likely different. The problem of word-level objective function also attracted attention. Specifically, models were trained to predict each “next word” correctly rather than evaluating the quality of the generated sentence as a whole. Ranzato *et al.* [14], Rennie *et al.* [15] and Liu *et al.* [12] all proposed using reinforcement learning with text similarity metrics (similarity to groundtruth sentences as whole) as rewards to model the problem. Their approaches were similar in that all were based on policy gradient [19, 23] and required a “warm-up” by training the model using word-level objectives first. Their methods differed in the way of estimating the expected return of a given model as well as estimating the baseline of the expected return. Ding *et al.* [4] proposed to avoid “warm-up” training by mixing a word-level objective function with a sentence-level objective function and gradually shifting towards the sentence-level end as training proceeds.

As we explain in the Introduction, the conversational responses produced by PhotoReply are very different in nature from the descriptions produced by image captioning systems (e.g., “So cute” vs.

“A baby sleeping besides a teddy bear”). Furthermore, in PhotoReply we faced challenges that are not addressed by existing image captioning because we are more constrained by practical issues, such as user privacy and coverage- quality balancing. For example, for balancing between coverage and quality, we showed that the output scores of generative models are not appropriately correlated with human quality scores (see Section 6.1 for details), which led us to incorporate a set of triggering models. As another example, to protect user privacy, raw image pixels were never accessible for training our models in PhotoReply. Instead, we modified the architecture of Vinyals *et al.* [21] to feed on the 128-dimensional image feature vectors that were available from the production system.

The coverage-quality balancing issue also reveals an important aspect in which PhotoReply differs from existing image captioning systems: PhotoReply needs to deal with images for which a proper response may not exist (e.g., an image for which there is not much to say about). In comparison, none of the existing image captioning systems address this issue because they have been mostly benchmarked on test datasets (e.g., COCO [2], Flickr30K [13]) in which each image is guaranteed to have meaningful captions and must be provided with one during evaluation.

Finally, to the best of our knowledge, no image captioning system has been productionized and used in a real application to date. PhotoReply is thus the first application that has productionized technology that produces natural language text in response to images.

3 SYSTEM OVERVIEW

In Figure 2, we give a high-level overview of the system design. The input to the generative and triggering models consists of pre-computed image metadata that is available to Allo’s production systems. In particular, we use the following: a 128-dimensional embedding of the image pixels, computed by a pre-trained Inception network [20] with batch normalization [8] and triplet loss [7, 16, 22]; and a set of concept labels produced by an image classifier that maps images to concepts from a very large taxonomy consisting of more than 20,000 concepts and is based on an Inception network model [17].

The image embedding is first fed into a model that generates candidate responses. The model is trained using properly anonymized image-response pairs from a chat application (see Section 5.1) and produces a ranked list of responses. It also produces scores based on sum of word log-likelihood. Unfortunately, as we show in Section 6, these scores are poorly correlated with human ratings (0.2 Spearman’s rank correlation). As a result, we added a set of triggering models to PhotoReply that decide which of the generated responses (if any) should be shown to the users. The triggering models take image embeddings and concepts as input; and lead to a much better correlation with human ratings (0.4 Spearman’s rank correlation). In the next section, we explain in detail both the response generation and the triggering models.

Figure 2: Overall workflow of PhotoReply.

4 MODELS

4.1 Response Generation Model

The purpose of the response generation model is to model the following conditional word probability distribution:

$$p(w_t | u, w_0, \dots, w_{t-1}), \quad (1)$$

where w_t denotes the word at location t , w_0 to w_{t-1} denote all previous words and u is a 128-dimensional pre-computed image feature vector (embedding) as described in Section 3.

Recurrent Neural Networks, particularly Long-Short Term Memory networks (LSTM) have been found to be very effective for this kind of language modeling task in applications such as text summarization [14], language translation [18] and image captioning [21]. Compared with Hidden Markov models, LSTMs seem to better capture long-range relations between words [11]. An LSTM is a stateful network. It has a memory cell that encodes the knowledge it has seen so far in an input sequence. The key difference between an LSTM and a vanilla RNN is that LSTM has a set of “gates” which help regulate reading and writing of the memory cell [6]. Those gates make LSTM suffer less from gradient vanishing and explosion problems compared to RNN and thus more capable of dealing with long sequences (for a detailed discussion, please see [6]). The particular implementation of the LSTM we used is the one described in [25] (Fig. 4). Mathematically, at each step, an LSTM unit computes the following:

$$\begin{aligned} i &= \sigma(W_{ix}x_t + W_{ih}h_{t-1}) \\ f &= \sigma(W_{fx}x_t + W_{fh}h_{t-1}) \\ o &= \sigma(W_{ox}x_t + W_{oh}h_{t-1}) \\ c_t &= f \odot c_{t-1} + i \odot g(W_{cx}x_t + W_{ch}h_{t-1}) \\ h_t &= o \odot c_t \\ p &= \text{Softmax}(h_t) \end{aligned} \quad (2)$$

where W_* are variables to learn during training; i , f and o denote the input gate, forget gate and output gate, respectively; x_t and h_t denote the input and output of LSTM at step t ; c_t denotes the memory cell; $\sigma(\cdot)$ and $g(\cdot)$ denote two non-linear functions, sigmoid and hyperbolic tangent; $\odot$ denotes element-wise product; p denotes the probability distribution over words in the vocabulary (we used a vocabulary of about 35K English words). The $\text{Softmax}(\cdot)$ operator encapsulates a fully-connected layer that maps from LSTM output

Figure 3: Illustration of a conditioned text generation model.

Figure 4: Illustration of an LSTM unit.

dimension to the size of vocabulary, followed by a softmax function,

$$\text{Softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}. \quad (3)$$

Following the idea proposed in [21], we build a conditioned generative model using LSTM, as illustrated in Fig. 3. The image feature vector is fed into the first LSTM, whose internal state is passed along to the next one. The output from the first LSTM is ignored. From the second LSTM, at each step, the embedding of the previous word is fed into the current LSTM, which produces a vector for softmax operator. During training, the embeddings of the ground-truth word sequence (*i.e.*, the ground-truth response) is fed into LSTMs. During inference, the embedding of a word sampled from previous step is fed into the current LSTM. All LSTM cells share the same variables. The word embeddings are trained during training as well.

During inference, we run beam search to sample sequences of words (*i.e.*, responses) with high overall probabilities. A beam size of one corresponds to a greedy sampling schema of taking the word of the highest probability at each step. The pseudocode for inference is sketched in Algorithm 1.

Algorithm 1: Response sampling with beam search.

Input : $\vec{u}$, an image feature vector
Param : k , beam size
Param : n , max number of responses
Model : f , which produces probabilities of words and new internal state of the model
Output : R , a queue of sampled responses

```

1  $\vec{s} \leftarrow \vec{0}$  // initial state;
2  $\vec{p}, \vec{s} \leftarrow f(\_, \vec{u}, \vec{s})$  //  $\vec{p}$  from this first step is not used and  $\_$ 
   denotes an empty word input;
3  $B \leftarrow ([\langle \_, \vec{s} \rangle], \vec{s}, 1.0)$  // initial beam, a 3-tuple of word
   sequence, model state and sequence probability (priority);
4  $Q \leftarrow \text{CreatePriorityQueueOfMaxSize}(k, [B]);$ 
5  $R \leftarrow \text{CreatePriorityQueueOfMaxSize}(n, []);$ 
6 while  $Q$  is not empty do
7   for  $B$  in  $\text{PullAll}(Q)$  do
8      $\vec{s} \leftarrow \text{GetModelState}(B);$ 
9      $w \leftarrow \text{GetLastWord}(B);$ 
10     $q \leftarrow \text{GetSequenceProbability}(B);$ 
11     $\vec{p}, \vec{s} \leftarrow f(w, \_, \vec{s})$  //  $\_$  denotes an empty image input;
12    for  $w$  in top  $k$  words according to  $\vec{p}$  do
13       $E \leftarrow \text{UpdateBeam}(B, w, \vec{s}, q \cdot \vec{p}(w))$  // expanding
         $B$  by one word;
14      if  $w$  is " $\langle \_, \vec{s} \rangle$ " or  $E$  has reached maximum
        number of words allowed then
15         $R \leftarrow \text{InsertWithPriority}(R, E, q \cdot \vec{p}(w))$ 
16      else
17         $Q \leftarrow \text{InsertWithPriority}(Q, E, q \cdot \vec{p}(w))$ 
18      end
19    end
20  end
21 end

```

Given a trained model, we can sample an arbitrary number of free-form responses by using the beam search algorithm. Unfortunately, "free-form" also means that we can end up with ungrammatical responses (*e.g.*, "Nice so" instead of "So nice") or responses that are inappropriate from a product perspective (*e.g.*, "It looks ugly"). To avoid such responses, a group of policy experts went through the most common responses generated by the model and editorially produced a whitelist of acceptable responses. Notice that while the whitelist is small enough to be incorporated into a single n -way classifier, using a generative model gives the flexibility to adapt and extend the whitelist on an ongoing basis.

We then slightly modified the beam search algorithm to restrict the generated responses to those existing on the whitelist. Specifically, we build a trie of words based on the whitelist. During beam search, at each step of expanding a sequence of words $(w_0, w_1, \dots, w_{t-1})$, we require that w_t must be picked from the set of words that have $(w_0, w_1, \dots, w_{t-1})$ as their prefix in the trie.

Cluster for “cute”	Cluster for “fun”
So cute	Looks like fun
That’s really cute	Looks fun
Those are so cute	Fun
That is adorable	Looks like you had fun
That’s so adorable	Looks like fun times
That’s pretty cute	Looks like a lot of fun
What a cutie	Looks like a fun time
What a cutie pie	Looks like a fun day

Table 1: Examples of response clusters.

4.2 Triggering Models

As we show in Section 6.1, the scores produced by the generation model are poorly correlated with human ratings (0.2 Spearman’s rank correlation). As a result, we added a set of triggering models to PhotoReply that decide which of the responses from the response generation model (if any) should be shown to the users; and lead to a much better correlation with human ratings (0.4 Spearman’s rank correlation).

Triggering is done based on *clusters* of semantically-related responses. For example, as shown in Table 1, “Fun” and “Looks like a fun day” are assigned to the same cluster. We train a binary model for each cluster, using image-response pairs for the responses in the cluster as positive examples, and randomly-sampled responses as negative examples. At inference time, given pair (i, r) , where i consists of features for an image and r is a response generated by the response generation model, we map r to its corresponding cluster c and then feed the features i to the binary classifier for c . If the response is above a threshold, we allow response r to be shown; otherwise, r is discarded.

The features used by each triggering model consist of the the 128-dimensional image feature vector and the concept labels produced by the image classifier described in Section 3. The model for each cluster c has a single fully connected layer with a logistic unit that outputs the probability of the image being appropriate for the responses in cluster c . (We use a single layer for each binary model for efficiency reasons.)

The clusters are obtained as follows. As mentioned in Section 4.1, a group of policy experts editorially created a whitelist for allowed responses. In addition, they also grouped semantically-related responses into clusters. The final set consisted of over one hundred clusters. Notice that this clustering was done editorially due to product requirements but it could have easily been done automatically through existing clustering techniques.

5 IMPLEMENTATION AND DEPLOYMENT

In this section, we provide details about the training data collection pipeline and the model serving stack.

5.1 Training Data Collection Pipeline

The training data used by PhotoReply consists of image-response pairs from the chat application. We implemented a pipeline to extract these pairs from the application logs. We restrict to pairs that belong to the same chat session and correspond to different users

(i.e., one user is sending the photo and another is responding). All personally identifiable information is removed from the responses. Furthermore, the pipeline does not have access to the raw pixels of the images. Instead, we use pre-computed 128-dimensional embeddings and image concepts that are readily available to the production system. We described the computation of these image features in Section 3.

As described in Section 4.2, we train one triggering model per response cluster. We retrieve all image-response pairs from the response generation model training data whose responses belong to the target cluster. We use those image-response pairs as positive samples. Negative samples are image-response pairs randomly picked from the text generation model training data whose responses do not belong to the target cluster. We keep the number of positive and negative samples balanced for each cluster.

5.2 Serving Stack

The PhotoReply serving stack is used by the chat application to fetch suggested responses for images. The serving stack consists of the following parts:

- (1) The image feature extraction model, which produces an embedding for an image.
- (2) The image classification model, which categorizes images into concepts.
- (3) The response generation (LSTM) model that scores responses using the image embedding, and returns 20 responses that are most likely to be responses to that image.
- (4) The triggering model that determines which of the top 20 responses should be shown.

Since the suggestions are required in real-time, it was necessary to minimize end-to-end latency. To do so, the first two models (image feature extraction and image classification) are executed in parallel. The implementation of these models, based on Inception models, is not described here.

Text Generation Model. To minimize latency due to LSTM scoring, the responses are stored as a trie and beam search is used to find the 20 most likely suggestions. The LSTM is initialized with the image embedding and 20 partially complete responses, which are initially empty. Each partial response represents a path starting at the root of the trie, and each partial response can be extended through its children in the trie.

At each time step, the LSTM queries the trie for all possible children for all partial responses. The LSTM scores these possible extensions and retains the top 20 as the most likely partial responses for the next time step. When all partial responses have reached leaf nodes in the trie, the beam search is concluded. The LSTM scores the whitelist and returns the top 20 responses in 300ms.

Triggering Model. After the LSTM finds the 20 most likely responses, the triggering model retrieves the corresponding cluster for each response. The triggering model produces a score for each cluster using the image embedding and the concepts from the image classification model. A global threshold is used to filter out clusters that should not trigger, and only responses belonging to the remaining clusters are finally returned as valid suggestions.

Threshold	Excellent	Good	Okay	Bad
0.00 (baseline)	22.2%	58.1%	5.0%	14.6%
0.48	24.6%	61.5%	3.9%	10.0%
0.69	28.6%	65.1%	1.3%	5.0%

Table 2: Offline rating distribution changes by thresholding.

6 EXPERIMENTS

In this section, we evaluate PhotoReply using offline metrics on crowdsourced datasets (ratings on response quality from a group of human raters) and online metrics (live traffic clickthrough rate from real users of the chat application).

6.1 Offline Evaluation

Test set. The test set consists of 12K images. For each image, we ran PhotoReply in order to generate a response. We then sent the image-response pairs to a group of human raters using an in-house crowdsourcing tool. Each image-response pair was rated by three raters independently and the final rating was taken by majority voting (discarding pairs for which there as no majority). We used a four-grade rating scale: Excellent, Good, Okay, Bad. The definition of the grades is as follows:

- Excellent: The response makes total sense for this photo
- Good : The response is reasonable but is not a very natural way to respond to a photo in a chat application or it is too generic and would apply to pretty much any photo.
- Okay: The response is not directly related to the photo, but it would be an acceptable response for a user in a chat application.
- Bad: The response has nothing to do with the photo.

Results. Table 2 shows the rating distribution for the results on the test set. The distribution is over the images that receive a response (we will discuss result coverage momentarily). We can see that as we increase the triggering threshold, the fraction of Good and Excellent results increases and the fraction of Okay and Bad results decreases. For example, when we compare the model without a triggering threshold and with a 0.69 threshold, the fraction of Excellent results increases from 22.2% (no thresholding) to 28.6%; and the fraction of Bad results decreases from 14.6% to 5.0%. This shows that the triggering component effectively improves the quality of results.

We now study the trade-off between result quality and coverage as a function of the triggering threshold. We use precision and recall as metrics. Since precision and recall require binary ratings (correct/incorrect), we consider Bad results as incorrect; and Okay, Good, and Excellent results as correct. The precision-recall curve is shown in Figure 5. We can see that, for example, we can achieve 90% precision with 91.7% recall.

A key finding in our work was that while the LSTM generates appropriate responses, its confidence scores are not appropriate for thresholding. This in turn led us to incorporate a triggering model to PhotoReply. To understand why this triggering model is fundamental, we now compare its performance against an ablated system where thresholding is done directly on the scores produced

Figure 5: Precision-recall curves.

by the LSTM. We consider two versions of the LSTM output scores: one corresponds to the sum of the word log-likelihood; the other corresponds to the sum of log-likelihood averaged by the number of words in the response. The results are shown in Figure 5. As we can see, thresholding based on the triggering model drastically outperforms thresholding based on the generative model for every recall value greater than 20%. For example, at precision of 85%, using the triggering score leads to 63% recall, while using response probability from the generative model leads to 34% recall.

We also computed Spearman’s rank correlation between the scores produced by the two models (generative and triggering model) and the human ratings for the 12K images in the test set. The Spearman’s rank correlation between the generative model scores and human ratings was 0.2 (0.14 for averaged version), compared to 0.4 for the scores from the triggering model. This further justifies the need for PhotoReply’s triggering model.

6.2 Online Evaluation

We now present the evaluation of PhotoReply on live traffic within the chat application. The results are from A/B tests performed on the launched system.

Metrics. The metrics are computed over all shared images that get a response. This includes both the images for which users type a response on the mobile device keyboard and the ones for which they choose one of the responses suggested by PhotoReply. We report two metrics: click-through rate (CTR) and impression rate. CTR is computed as the ratio between the number of shared photos for which users choose a response suggested by PhotoReply and the total number of shared photos. Impression rate is computed as the ratio between the shared images for which PhotoReply produces at least one suggestion (the system suggests up to three responses) and the total number of shared images. Due to policy restrictions, we do not report absolute numbers for the live metrics. Instead, we report numbers that are relative with respect to the baseline which in this case is the system without any thresholding from the triggering model.

Results. Table 3 shows the live traffic results for different triggering model thresholds¹. We can see that increasing the triggering

¹The thresholds were chosen to correspond with 85%, 90%, and 95% precision according to the offline evaluation in Section 6.1.

Threshold	Δ CTR (relative to baseline)	Δ Impression Rate (relative to baseline)
0.00	baseline	baseline
0.48	17.2%	-9.3%
0.69	25.8%	-39.0%

Table 3: Live metrics.

threshold leads to higher CTR. For example, a threshold of 0.48 increases CTR by 17.2% with respect to a baseline where no triggering threshold is used. As a result, the triggering threshold can be used as a knob to achieve the desired balance of CTR and impression rate for the production system.

7 CONCLUSIONS

In this paper, we have introduced PhotoReply, a system that suggests conversational responses to photos. We have presented the design and underlying models of the system and discussed deployment within the Allo chat application. PhotoReply is currently the predictive assistance feature with the highest click-through rate in Allo. We have presented a number of challenges faced in building the system such as the need for triggering models in order to increase the correlation between confidence scores and human ratings. We have also presented a thorough evaluation of the system using both crowdsourced human ratings and live traffic.

REFERENCES

- [1] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS*. 1171–1179.
- [2] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. 2015. Microsoft COCO captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325* (2015).
- [3] Barb Darrow. 2016. Why Smartphone Virtual Assistants Will Be Taking Over for Your Apps Soon. (December 2016). <http://fortune.com/2016/12/21/virtual-assistant-usage-gartner/>
- [4] Nan Ding and Radu Soricut. 2017. Cold-Start Reinforcement Learning with Softmax Policy Gradients. In *NIPS*. 2814–2823.
- [5] Amit Fulay. 2016. Say Hello to Google Allo: A Smarter Messaging App. (September 2016). <https://blog.google/products/allo/google-allo-smarter-messaging-app/>
- [6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
- [7] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*. 84–92.
- [8] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*. 448–456.
- [9] Anjali Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young, and Vivek Ramavajjala. 2016. Smart Reply: Automated Response Suggestion for Email. In *KDD*. 955–964.
- [10] Andrej Karpathy and Li Fei-Fei. 2015. Deep Visual-Semantic Alignments for Generating Image Descriptions. In *CVPR*. 3128–3137.
- [11] Henry W Lin and Max Tegmark. 2016. Critical Behavior from Deep Dynamics: A Hidden Dimension in Natural Language. *arXiv preprint arXiv:1606.06737* (2016).
- [12] Siqi Liu, Zhenhai Zhu, Ning Ye, Sergio Guadarrama, and Kevin Murphy. 2017. Improved Image Captioning via Policy Gradient optimization of SPIDER. In *ICCV*. 873–881.
- [13] Bryan A Plummer, Liwei Wang, Chris M Cervantes, Juan C Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. 2015. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models. In *ICCV*. 2641–2649.
- [14] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732* (2015).
- [15] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *CVPR*. 1179–1195.
- [16] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *CVPR*. 815–823.
- [17] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *ICCV*. 843–852.
- [18] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*. 3104–3112.
- [19] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*. 1057–1063.
- [20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *CVPR*. 1–9.
- [21] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and Tell: A Neural Image Caption Generator. In *CVPR*. 3156–3164.
- [22] Jiang Wang, Yang song, Thomas Leung, Chuck Rosenberg, Jinbin Wang, James Philbin, Bo Chen, and Ying Wu. 2014. Learning Fine-grained Image Similarity with Deep Ranking. In *CVPR*. 1386–1393.
- [23] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.
- [24] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*. 2048–2057.
- [25] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* (2014).