Image-Based Joke Generation *

Isidora Jeknic[1], Iulia Zaitova[1], Kirstin Kolmorgen[1], Sharmila Upadhyaya[1], Debanjali Biswas[1]
Department of Language Science and Technology,
Saarland University
[1] {isje00001|s8izait|s8kikolm|shup00001}@stud.uni-saarland.de
[2] dbiswas@coli.uni-saarland.de

Abstract

Humor generation as a sub-field of natural-language generation (NLG) is a notoriously difficult task. In this paper, we present a joke generator that produces question-answer jokes prompted by an input image. We attempted to synthesize three language generation tasks into a single system - image captioning, question generation, and joke generation, whereby the image caption acts as a context from which the question, and, subsequently, the joke ought to be generated. Overall, the model accomplishes the task of generating a *"why"*-joke from the given image, to varying degrees of success in terms of funniness; nevertheless, the system obtains evaluation scores (human and automated) that are on par with previous research.

Keywords: NLG, humor generation, template-based question generation, GPT-2

1 Introduction

Natural-language generation (NLG) is a software process that takes data as input and attempts to generate text that resembles human generated text as much possible. This task has evolved throughout the years from simple language modelling using Markov models to complex transformer models. When boiled down to the basics, text generation is a relatively universal task. It is the usage and purpose of these models that sets them apart.

Humor generation may not be as prolific of a research area as other areas of natural-language generation because of its subjective nature, which makes it particularly difficult. Nonetheless, this should not be taken as an excuse to dismiss automatic humor generation as something too demanding, since it is a vital part of human conversation, and could be used in dialog systems, as well as facilitate human-computer interaction in general. We decided to tackle this task regardless of the challenges it poses, and developed a system that generates question-answer joke pairs, concentrating on *"why"*-questions. Since generating text requires a context, and, while one possibility is to obtain the context from user input (see for example (18)), we decided to get the context from a user-provided image. Thus, we introduce the novelty of synthesizing three different usages of text generation - namely image captioning, question generation and joke generation - into a single joke generation system.

Generally, we generate jokes the following way: a caption is generated based on the input image. From this caption, we then extract entities that are used to generate a *"why"* question, which in turn serves as the input to our joke generator that produces an answer.

2 Related Work

Previous work on image captioning is abundant, with two main approaches being top-down and bottom-up. In the former, the starting point is descriptions of different parts of the image that are then synthesized into a coherent whole. On the other hand, bottom-up image captioning starts with the gist of the image, and adds details subsequently. With the deep learning revolution, an important recent development stems from the paper "Show and Tell: Lessons Learned from the 2015 MSCOCO Image Captioning Challenge" (16). The authors propose a Neural Image Captioning (NIC) encoder-decoder architecture, where the typical Recurrent Neural Network (RNN) encoder is replaced by a deep Convolutional Neural Network (CNN), whereas the typical RNN decoder is preserved in the last hidden layer. You et al. (17) propose a hybrid algorithm based on semantic attention - the caption focuses only on the relevant parts of the image, detected by a bottom-up approach, and at-

*The code can be found here.

tention is allocated by a top-down approach. Using an RNN framework, the initial state is the gist, and the information is updated as attention is allocated and the states change.

When it comes to humor generation, there are two distinct approaches: template-based and neural joke generation. We attempted to synthesize both of these methods in our project. Template-based joke generation systems have a fixed number of open slots and generate text to fit into said slots. Previous work includes the "Light Bulb" generator (13), a system for generating jokes of the format *"How many <group> does it take to screw in a light bulb? - <number1>. One to <activity1> and <number2> to <activity2>"*, as well as the unsupervised joke generation system that produced jokes of the format *"I like my <X> like I like my <Y> - <Z>"* (9). The drawback of these models is their limitation in terms of content generation, due to domain constraints. The latter model included a formula that was meant to maximize the funniness of the generated joke by selecting entities `X` and `Y` that were the least similar synonyms, thus capitalizing off of the surprise factor of a related, but not completely synonymous word.

Neural joke generation systems train a neural network on a joke corpus to generate jokes based on different contexts: e.g. Ren and Yang trained an LSTM to generate a joke that contains concepts provided by the user (14), whereas Yu et al. tried to use the prevalence of polysemy in comedy (e.g. puns, word play) when generating jokes (18). The benefits of such approaches are the vastness and creativity displayed by the models, since they are not confined to a particular format; however, this freedom also makes for the biggest drawback is that often times the generated jokes make no sense or are unfunny, as the only part of the system that can be controlled are the parameters and input.

In our system, we synthesized these two approaches: question generation was fully template-based, whereas the joke generation was neural, done using the GPT-2 model.

3 Methodology

To train the image captioning model, we used the COCO data set (5) consisting of 164 thousand colored images.

The two main data sets used for question and joke generation were the `short_jokes` (7) and `plaintext_jokes` (10). For joke generation, *"why"*-jokes were separated from both sets and parsed to include tags for the beginning and end of the question and joke (`<soq>`,`<eoq>`, `<eoj>`, `<eoj>`) before being passed to the GPT-2 model for fine-tuning.

For question generation, only the former data set was used; namely, generation of the questions was facilitated using templates that were obtained from the *"why"*-jokes subset, whereas object generation was made possible using the remaining jokes (from the same data set - `short_jokes`).

Data preprocessing consisted of scripts being written to extract all jokes that were of the aforementioned *"why"* format. Subsequently, *"how"*-jokes and *"what"*-jokes were collected, in order to be used in object generation. The final data set consisted of 40 thousand *"why"*-jokes. Moreover, as the jokes were collected from the internet, they needed to be stripped of offensive and profane content before being used. This was done using the `better-profanity` Python library, as well as a custom script including particular words, such as races or ethnicities, since jokes containing those entities mostly had a negative connotation and were rather offensive.

4 Architecture

The architecture of our joke generator is made up of three different modules - an image captioning module, a question generation module, and lastly, a joke generation module. An image of the whole architecture can be found in Figure 1. We will discuss each module in the following.

4.1 Image Captioning Module

The initial part of the pipeline is a feature-based image captioning module, for which we used a pre-trained model (6). This repository references to Andrej Karpathy's NeuralTalk2 model (4). The architecture is made up of a pre-trained RESNET 101 layer followed by an RNN model which is fine-tuned on the COCO Caption data set (5) for the caption generation. We used the pre-trained model comprising of RESNET and RNN only for the inference.

4.2 Question Generation Module

The next part of the pipeline consists of a template-based question generation module. In order to generate a question from the caption provided by the image captioning module, we created question tem-

plates that were then filled with verb(s) and noun phrase(s) extracted from the caption, as well as with additionally generated object noun phrases. With this template-based approach, we hoped to capture joke-specific question structures, while still being able to use as much information from the caption as possible. The whole process is described in the following subsections.

4.2.1 Template Creation

The templates were created from the question part of the jokes in the `short_jokes` data set. The questions were parsed using Python's spaCy library (3), which was considerably faster than its Stanza library (11), while producing comparable results.

Parsing the questions allowed us to locate (complex) noun phrases (NP), verbs, and prepositions with the purpose of replacing them with a label of the form `<POS tag>_dependencylabel`, with the POS tag and dependency label referring to the word (or head noun in case of NPs) it replaces. This label provides information as to what kind of word or phrase fits into a particular slot, which is necessary to ensure that the generated question is as grammatically (due to matching POS tags) and semantically (due to matching dependency labels) correct as possible. The decision to replace prepositions as well was mainly motivated by the need for placing prepositions that match with prepositional object NPs later in the process. An example of a template is presented below:

Example:
Question: Why do seagulls live by the sea
Template:
`Why <VBP>_aux <NNS>_NP_nsubj <VB>_ROOT <IN>_prep <NN>_NP_pobj`

We excluded templates with more than three slots to keep the question simple and minimize the chance of grammar mistakes. The lookup process is done by using the verb POS tag(s), and obtaining another dictionary with the POS tags and dependency labels of all NP heads in the template. Subsequently, the NPs obtained from the captions are used for lookup, and the obtained values are sets of all templates providing slots for verbs and NPs with the tags and labels of the two keys (see example entry below). Auxiliary verbs are stored in a separate dictionary (`POS tag : {verb1, verb2}`).

Example template dictionary entry:

```
{'VB' :
    {
    ('NNS_nsubj','NN_pobj') :
        {template1,template2,...},
    ('NN_nsubj','NNS_dobj') :
        {template1,template2,...},
    [...]
    }
}
```

Problems that arose in the template creation process were mainly caused by parsing errors, where, for example, the head of the VP gets interpreted as a noun, and the actual head of the NP as a pre-modifier.

Example:
Question: Why wasn't the Canadian scared at the movie theater?
Template:
`Why <VBD>_ROOT n't` **`the Canadian scared`** `<IN>_prep <NN>_NP_pobj`

These errors were taken care of by the probabilistic nature of the model, as we only stored the ten most frequent templates in dictionaries, and the errors were rare enough to not impact the overall probability distribution of the templates.

4.2.2 Caption Parsing

In order to obtain verb(s) and NPs from the caption to fill a template, we used spaCy (3) to parse the caption the same way we parsed the joke questions. Unfortunately, parsing the caption was not always a straightforward task. Complications mainly arose, because unlike the joke questions, the captions were usually not full sentences. Often, the caption consisted only of one complex NP. In these cases, we try to find a verb inside the NP, and use basic NPs instead. For example, from the caption *a woman sitting on a bed with a cat*, we would extract the verb *sitting* and the basic NPs *a woman*, *a bed*, and *a cat*. However, if there is no verb in the caption, we resort to using default verbs instead. Our transitive default verbs include: *ignore, destroy, become, watching, inviting, hating, angered, befriended, admired*; moreover, the ditransitive default verbs we chose were: *sell, giving, bought, put*. The default verb is chosen randomly. In this case, the complex NP is not split up and used as is. To illustrate, if we obtain the caption *a bedroom with*

a bed in a room with a desk, we could choose the default verb *watching* and generate the question *Why was a bedroom with a bed in a room with a desk watching people*. In this example, another NP (*people*) has been generated. This process of object generation is explained in the next section.

4.2.3 Object Generation

Inspired by Petrović and Matthews (9) who concluded that the more dissimilar the nouns (X and Y), the funnier the joke, we decided to take the subject NP from the caption, but generate a new object NP randomly. In doing so, we hoped to increase the chance of creating a funny joke.
In order to generate an object, we needed the transitivity information of the verb. This information was obtained from the Google Syntactic N-grams corpus (English 1 Million subcorpus) (**?**), and include the possibility of a verb being transitive, intransitive or ditransitive. Furthermore, NPs acting as direct and prepositional objects needed to be obtained. This was achieved by considering the remaining jokes from the `short_jokes` data set, such as *how*-jokes and *what*-jokes, and extract the respective objects using spaCy (3), which were then stored in two separate dictionaries together with the head noun POS tags. In case of a prepositional object, its head preposition was stored in the dictionary as well. In order to generate objects, we first determine the transitivity of the verb that we either obtained from the caption or chose from the list of default verbs. If the verb is classified as intransitive, we do not generate an object and use the NPs we extracted from the caption instead. If, however, the verb is transitive, we randomly choose a direct object NP out of its dictionary (with the verb being the key). In cases where the verb is classified as being ditransitive, we additionally choose a prepositional object in an analogous manner. If an object NP was generated, we only keep the subject NP from the caption. Together with the already determined verb(s), the NPs selected after this step form the basis for choosing a template.

4.2.4 Template Selection and Filling

Templates are selected by using the POS tag and dependency information of the extracted/generated verb(s) and NP(s) as lookup keys in the corresponding template dictionaries. If there is no match, we dismiss the NP that was added last and start the matching process once more. In the case of more than one template match, we choose one randomly. Once a template is selected, the filling of its slots is fairly straightforward: The labels currently occupying the slots are replaced with the NP, verb or preposition that matches its POS tag and dependency information. However, we encountered some difficulties with choosing an auxiliary verb that is compatible with both the tense of the main verb and the number information (singular/plural) of the subject NP, as well as choosing the right form in cases where the template contains a negation (e.g. selecting *wo* instead of *will* if the auxiliary verb slot is followed by *n't*). This was taken care of by implementing detailed rules ensuring grammatical correctness. The whole process takes approximately three seconds to complete. An example of the steps is illustrated below:

Example:

```
Caption: a kitchen with a stove and a sink
Extracted NPs: a kitchen with a stove, a sink
Verb(s): hating (default verb)
Objects generated: a turkey
Possible templates:
Why <VBD>_aux <NN>_NP_nsubj <VBG>_ROOT <NN>_NP_dobj,
Why <VBD>_aux n't <NN>_NP_nsubj <VBG>_ROOT <NN>_NP_dobj
Chosen template:
Why <VBD>_aux n't <NN>_NP_nsubj <VBG>_ROOT <NN>_NP_dobj
Filled template/generated question:
Why wasn't a kitchen with a stove hating a turkey
```

4.3 Joke Generation Module

For joke generation, we fine-tuned the GPT-2 model (12) using *"why"*-jokes from the extended data set. GPT-2 is a transformers model that was pre-trained (self-supervised) on 40GB of English text (12). We used the default ("small") model with 124 million parameters, with the default learning rate of $1^{e(-5)}$. The model was trained for 1000 steps on 16GB RAM with 1 GPU core using Google Colaboratory, which took approximately 20 minutes for around 40 thousand inputs.
The main benefit of the GPT-2 model is how resource light it is, making it possible to train without requiring GPU. The resulting drawback is that there are very specific dependencies needed to get the model to run, such as TensorFlow version 1.14, which requires Python version 3.7.5 or less.

4.4 Front End

The front end consists of a small web app made with the Python library Flask (2). The user interface contains a field where the user inputs an image, and a button that is used to get the caption, and generate a question and joke. After the joke is generated, it is displayed below the input image

(see Figure 2 and Figure 3 in the Appendix).
The current run-time of the entire system, including the web app, on an Ubuntu 20.04 machine with 16GB RAM is 20 seconds. Our system's minimal requirement for running is Python 3.7.5, approximately 4 GB RAM and 2.5 GB of disk space.

5 Evaluation

To evaluate our model's performance, we used both automated metrics and human evaluation for different parts of the pipeline. Initially, BLEURT and answerability scores were computed for the generated questions, and human evaluators were subsequently asked to rate the generated jokes and the captions.

5.1 Automated Metrics

5.1.1 BLEURT

BLEURT (15) is an evaluation metric for language generation that takes two sentences as inputs: a reference and a candidate, and computes a score which reflects the degree to which the candidate conveys the meaning of the reference, while also taking grammaticality into account. The authors describe it as "a novel, machine learning-based automatic metric that can capture non-trivial semantic similarities between sentences".
The score ranges from -2 to 1, with 1 demonstrating perfect similarity, and -2 complete lack of overlap. In our case, the reference sentence is the generated image caption, and the candidate sentence is the generated question, considering the fact that we do not need a 1:1 comparison between entities (due to the object generation aspect of the system which may generate related, but not entirely the same entities). Because of the different use, it is natural to expect that the score will not be as optimal as with two synonymous declarative sentences.
The authors also claim that the score should primarily be used as a point of comparison to observe the trajectory of the model's performance.
Before extending the jokes data set to include `plaintext_jokes`, our mean BLEURT score for 100 random jokes was -0.781755. After introducing the data set extension, the BLEURT score was -0.679011.

5.1.2 Answerability

Another metric that was used to evaluate model performance was answerability, as introduced by Nema and Khapra (8). The premise is to modify an existent metric (BLEU, ROGUE, NIST, and METEOR) to include answerability as a sub-metric, which is proportional to relevant features such as type of question, entities, and relations. The score ranges from 0 to 1, with 0 being a poorly constructed, and 1 signifying a well-formed, answerable question.
Before presenting the model with the extended data set, the answerability score was 0.721 and the N-gram 0.646. After including the extended data set, the mean answerability score was 0.719, and the N-gram score was 0.657. This slight decrease in score is not significant enough to draw any meaningful conclusions with respect to model performance.
A summary of the results obtained through the automated metrics can be found in Table 1.

Table 1: Question Evaluation - Automated Metrics

	Original	**Extended**
BLEURT	-0.781755	-0.679011
Answerability	0.721	0.719
N-Gram	0.646	0.657

5.2 Human Evaluation

Our aim was to generate a high proportion of funny jokes. Due to the fact that jokes and humor are highly subjective, there does not exist a simple metric to quantify funniness. Even though this was previously attempted by Petrović and Matthews (9), the overall result was not reliable. In order to judge this aspect of our model's performance, we followed other previous works (see (14), (1)) and asked human evaluators to rate the generated captions and jokes. This was achieved by creating five Google Forms[1] each of them containing twenty captions and jokes to be evaluated on a Likert scale from 1 (totally incorrect) to 3 (correct) for the captions, and from 1 (not funny at all) to 5 (very funny) for the jokes. The selected jokes were the same for which the BLEURT and answerability scores were computed.
The 150 human evaluators were comprised of 94 unpaid acquaintances and users of reddit.com, as well as 56 people sourced from Amazon's Mechanical Turk. Overall we gathered 3000 responses, 1680 of which came from Amazon's Mechanical Turk respondents. The results of human evaluation are provided in tables 2, 3, and 4.

[1]The evaluation surveys can be found under this link.

Table 2: Caption Evaluation

Image data set	Mean Caption Evaluation
COCO	2.50
Flickr	1.96
All	2.18

Table 3: Image Evaluation

Image data set	Mean Joke Evaluation
COCO	2.17
Flickr	2.96
All	2.65

Table 4: Percentage of Good Jokes

Image data set	4+ ratings
COCO	19.33%
Flickr	40.72%
All	32.16%

6 Discussion

Image Captioning:
We reflected through the pre-trained image captioning model trained in COCO data set which is huge and should suffice our purpose but we realized the model is not robust and the jokes were not much diverse. Hence, with an intent to create more broad evaluation we used flicker data set as the 60 percent of our evaluation set. The change from COCO data set, on which the Image Captioning model was trained (in-domain evaluation) to Flickr data set (out-of-domain evaluation) presents us with a drastic variation in the results in both caption and joke human evaluation. The mean caption evaluation score is logically 0.56 (out of 3) points lower for captions generated using Flickr data set, which is a typical behaviour for the change from in-domain to out-of-domain evaluation.

Question Generation:
As previously mentioned, we chose a template-based approach for generating questions in hopes of capturing joke-specific question structures that would be a good prompt for a funny answer. When looking at the most frequent templates, however, they seem to be very generic, not displaying many elements that could be considered joke-specific. For example, the most frequent templates for one, two and three open NP slots are *Why AUX NP VERB*, *Why AUX NP VERB NP*, and *Why AUX NP VERB NP PREP NP* respectively. Compared to these templates, the ones that do display some specifics such as *Why AUX NP [always|never|only] VERB NP* are relatively rare. This seems to render a template-based approach unnecessary. Instead, we could have opted for an entirely rule-based approach turning the captions into questions directly, without having to use templates and complex matching rules that might not find a match and leave us without a question. Another option would be to train a separate model for automatic question generation (e.g. a next word prediction model) and dismiss rule-based approaches completely. This would eliminate the necessity of implementing rules for countless cases, many of which rare and/or unforeseeable. Due to time constrains and an already complex architecture, we did not pursue such an approach any further. It could, however, be integrated in the future.
Another feature of the question generation module that did not work as intended was the generation of object NPs. This was mainly caused by many captions containing intransitive verbs, while, on the other hand, many transitive and ditransitive verbs being classified as intransitive by the transitivity detection model. Both of these cases led to no objects being generated, which did prevent the model from realizing one of our means of increasing humor.

Automatic Evaluation
Juxtaposing the two obtained BLEURT scores seems to indicate that the model's performance got slightly better with the introduction of the extended data set, which implies that extending the data set even further would result in even better performance. Furthermore, the answerability score difference was not large enough to draw conclusions about the improvement of the generated questions in terms of joke setup appropriateness. In humor and comedy, a balance needs to be struck between answerable and sensible, on the one hand, and non-answerable and surprising on the other. Therefore, we argue that this metric could signal potentially good joke set-ups, reflected in answerability scores of above 0.5 and below 0.9. In future work, this metric could potentially be utilized to prune questions with too high or too low of an answerability score as poor joke set-ups.

Human Evaluation

Compared to previous works on humor generation (e.g. (14), (1)), we managed to improve the subjective funniness scores. There are several characteristics of our model that could have contributed to such a result. Namely, our model limits the scope of jokes to only *"why"*-type of jokes, which might potentially be less confusing to the neural joke generation model. Furthermore, in addition to the deep learning neural model, we used a template-based question generation model with certain linguistic constraints. Provided the generated question was grammatically and logically correct, the neural model only had to generate the answer to the given question, which again facilitates the neural model's task.

Technical Difficulties

There were many technical glitches during the implementation of the software. The sequential nature of the architecture made it difficult to run within the short period of time. As various models are to be loaded and the pipeline follows the running of one component after another approach, waiting for the response of one module to execute another made the run time slow. Similarly, the session conflicts between two different models further prevent the loading of a checkpoint beforehand. Furthermore, training the GPT-2 model was carried out using the Google Colaboratory platform as our systems and resources were not sufficient. At last, we switched to local resources for the inference. The technical part of the implementation made us discern deployment and set up of our system in the different devices. We encountered various hardware problems caused by libraries like PyTorch and TensorFlow in low-specified devices. Furthermore, the dependencies of one model conflicting with others was observed and as a result we settled for TensorFlow 1.14 and Python 3.7.5. This trade-off resulted from the lack of resources, as the simple GPT-2 model only ran in the version of TensorFlow under 1.15. This resulted in the use of Python 3.7, as Python 3.8 does not support a version of TensorFlow lower than 2.0.

7 Future Work

Each module of the project still has room for improvement. As the final result depends on three different models, optimizing these could help increase the quality of jokes in terms of both linguistic structure and humor. Firstly, using a robust model for image captioning can provide solid ground for question generation. Secondly, introducing linguistic constraints, such as considering the animacy of NPs during question generation, has the potential to further improve the results, as it would permit us to choose linguistic constructions that are semantically compatible (e.g. templates with passive constructions for inanimate subject NPs). Lastly, experimenting with the joke generation module by using larger and better GPT models would help obtain more coherent texts. Similarly, we could extend the type of jokes used to include *"what"*-jokes and *"how"*-jokes. All in all, humor generation based on image content is a prolific area with respect to research and implementation that could be explored further.

8 Conclusion

In conclusion, we successfully implemented an application version of our image-based joke generation project idea. This process included the synthesized usage of three different uses of language generation, namely mapping image features to text (image captioning), template-based question generation, and neural joke generation.

Using our web application, users can upload an image, and receive a response from the system in the form of a *"why"* question-answer joke related to said image. The generated jokes achieved evaluation scores that are comparable to those of previous works on humor generation.

References

[1] AMIN, M., AND BURGHARDT, M. A survey on approaches to computational humor generation. *ACL Anthology* (2020).

[2] GRINBERG, M. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[3] HONNIBAL, M., MONTANI, I., VAN LANDEGHEM, S., AND BOYD, A. spaCy: Industrial-strength Natural Language Processing in Python, 2020.

[4] KARPATHY, A. Neuraltalk2. https://github.com/karpathy/neuraltalk2, 2015.

[5] LIN, T., MAIRE, M., BELONGIE, S. J., BOURDEV, L. D., GIRSHICK, R. B., HAYS, J., PERONA, P., RAMANAN, D., DOLLÁR, P., AND ZITNICK, C. L. Microsoft COCO: common objects in context. *CoRR abs/1405.0312* (2014).

[6] LUO, R., PRICE, B., COHEN, S., AND SHAKHNAROVICH, G. Discriminability objective for training descriptive captions. *arXiv preprint arXiv:1803.04376* (2018).

[7] MOUDGIL, A. A dataset of english short jokes., 2017.

[8] NEMA, P., AND KHAPRA, M. M. Towards a better metric for evaluating question generation systems. *arXiv preprint arXiv:1808.10192* (2018).

[9] PETROVIĆ, S., AND MATTHEWS, D. Unsupervised joke generation from big data. In *Proceedings of the 51st annual meeting of the association for computational linguistics (volume 2: Short papers)* (2013), pp. 228–232.

[10] PUNGAS, T. A dataset of english plaintext jokes., 2017.

[11] QI, P., ZHANG, Y., ZHANG, Y., BOLTON, J., AND MANNING, C. D. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations* (2020).

[12] RADFORD, A., WU, J., CHILD, R., LUAN, D., AMODEI, D., AND SUTSKEVER, I. Language models are unsupervised multitask learners.

[13] RASKIN, J. D., AND ATTARDO, S. Non-literalness and non-bona-fide in language: An approach to formal and computational treatments of humor. *Pragmatics & Cognition 2*, 1 (1994), 31–69.

[14] REN, H., AND YANG, Q. Neural joke generation. *Final Project Reports of Course CS224n* (2017).

[15] SELLAM, T., DAS, D., AND PARIKH, A. P. Bleurt: Learning robust metrics for text generation. In *Proceedings of ACL* (2020).

[16] VINYALS, O., TOSHEV, A., BENGIO, S., AND ERHAN, D. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE transactions on pattern analysis and machine intelligence 39*, 4 (2016), 652–663.

[17] YU, J., LI, J., YU, Z., AND HUANG, Q. Multimodal transformer with multi-view visual representation for image captioning. *IEEE transactions on circuits and systems for video technology 30*, 12 (2019), 4467–4480.

[18] YU, Z., TAN, J., AND WAN, X. A neural approach to pun generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2018), pp. 1650–1660.

Appendix

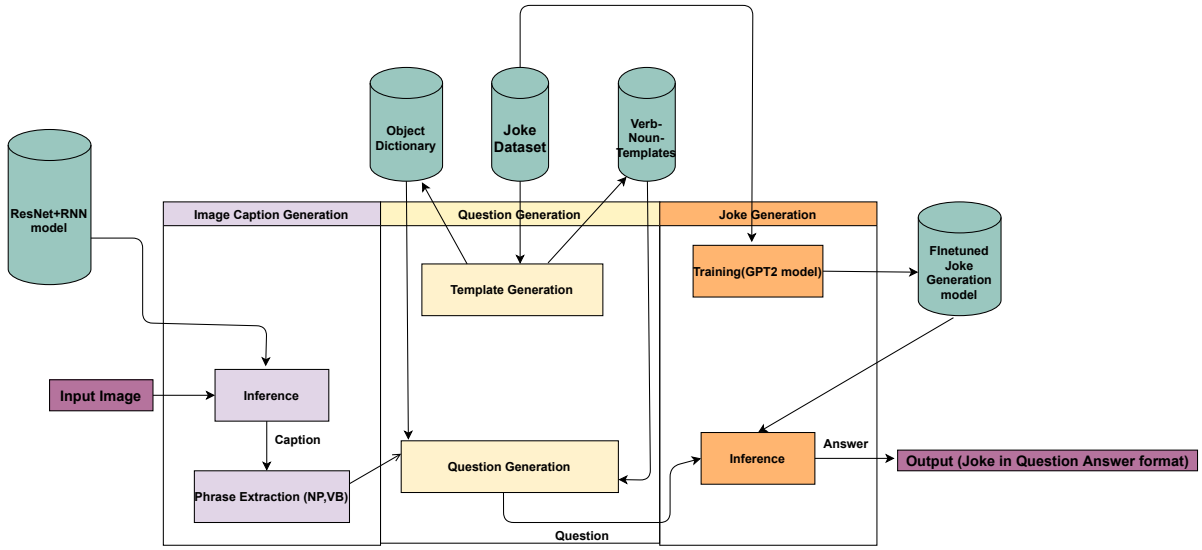

Figure 1: Joke Generator Architecture

Figure 2: Web Application Interface

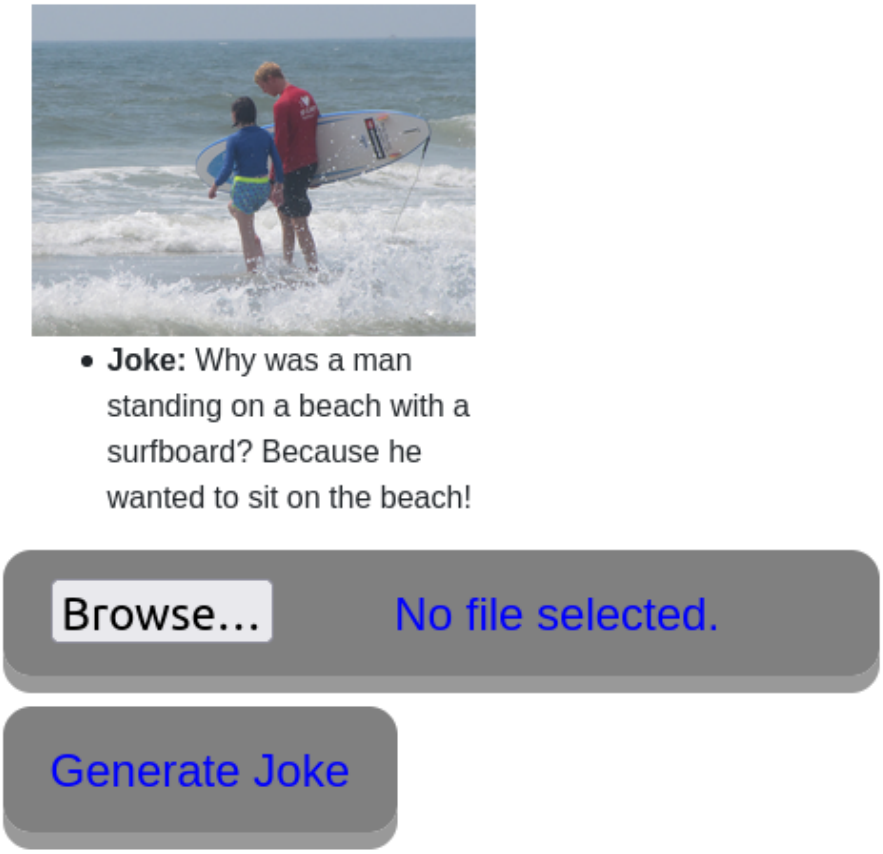

Figure 3: Web Application Interface