Context-Situated Pun Generation

Jiao Sun[1*] **Anjali Narayan-Chen**[2] **Shereen Oraby**[2] **Shuyang Gao**[2†]
Tagyoung Chung[2] **Jing Huang**[2] **Yang Liu**[2] **Nanyun Peng**[2,3]
[1]University of Southern California
[2]Amazon Alexa AI
[3]University of California, Los Angeles
jiaosun@usc.edu
{naraanja,orabys,shuyag,tagyoung,jhuangz,yangliud}@amazon.com
violetpeng@cs.ucla.edu

Abstract

Previous work on pun generation commonly begins with a given pun word (a pair of homophones for heterographic pun generation and a polyseme for homographic pun generation) and seeks to generate an appropriate pun. While this may enable efficient pun generation, we believe that a pun is most entertaining if it fits appropriately within a given context, e.g., a given situation or dialogue. In this work, we propose a new task, *context-situated pun generation*, where a specific context represented by a set of keywords is provided, and the task is to first identify suitable pun words that are appropriate for the context, then generate puns based on the context keywords and the identified pun words. We collect CUP (**C**ontext-sit**U**ated **P**un), containing 4.5k tuples of context words and pun pairs. Based on the new data and setup, we propose a pipeline system for context-situated pun generation, including a pun word retrieval module that identifies suitable pun words for a given context, and a generation module that generates puns from context keywords and pun words. Human evaluation shows that 69% of our top retrieved pun words can be used to generate context-situated puns, and our generation module yields successful puns 31% of the time given a plausible tuple of context words and pun pair, almost tripling the yield of a state-of-the-art pun generation model. With an end-to-end evaluation, our pipeline system with the top-1 retrieved pun pair for a given context can generate successful puns 40% of the time, better than all other modeling variations but 32% lower than the human success rate. This highlights the difficulty of the task, and encourages more research in this direction.

1 Introduction

Pun generation is a challenging creative generation task that has attracted some recent attention in the research community (He et al., 2019; Yu et al., 2018, 2020; Mittal et al., 2022; Horri, 2011). As one of the most important ways to communicate humor (Abbas and Dhiaa, 2016), puns can help relieve anxiety, avoid painful feelings and facilitate learning (Buxman, 2008). At the same time, spontaneity is the twin concept of creativity (Moreno, 1955), which means the context matters greatly for making an appropriate and funny pun.

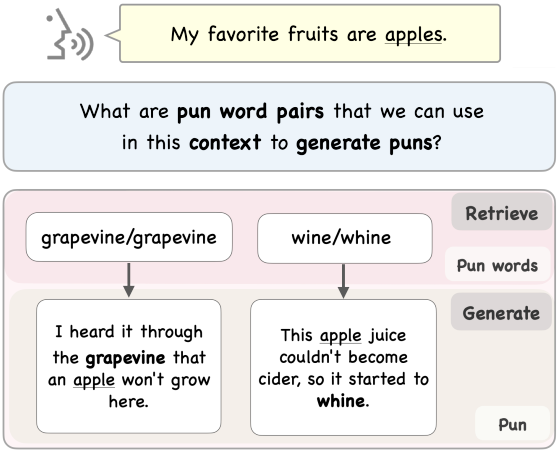


Figure 1: *Context-situated pun generation* aims to find relevant pun words to generate puns within a given context. We propose a unified framework to generate both homographic and heterographic puns; examples shown here are human-written puns from our corpus.

Existing work on pun generation mainly focuses on generating puns given a pair of pun-alternative words or senses (we call it a pun pair). Specifically, in heterographic pun generation, systems generate puns using a pair of homophones involving a pun word and an alternative word (He et al., 2019; Yu et al., 2020; Mittal et al., 2022). Alternatively, in homographic pun generation, systems generate puns that must support both given senses of a single polysemous word (Yu et al., 2018; Luo et al., 2019; Tian et al., 2022). Despite the great progress that has been made under such experimental settings, real-world applications for pun generation (e.g., in dialogue systems or creative slogan

*Work done during Jiao's internship at Amazon.
†Work done while Shuyang was at Amazon.

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 4635 - 4648
December 7-11, 2022

Type	Pun	p_w/a_w	Context C	S_{p_w}	S_{a_w}
het.	Two construction workers had a staring contest.	stair/ stare	construction workers	support consisting of a place to rest the foot while ascending or descending a stairway	look at with fixed eyes
	"I've stuck a pin through my nose", said Tom punctually.	punctually/ puncture	pin, nose	at the expected or proper time	a small hole made by a sharp object
hom.	A new type of broom came out, it is sweeping the country.	sweep/ sweep	broom, nation	sweep with a broom or as if with a broom	win an overwhelming victory in or on
	If you sight a whale, it could be a fluke.	fluke/ fluke	whale	a stroke of luck	either of the two lobes of the tail of a cetacean

Table 1: Two examples each of heterographic puns and homographic puns in the SemEval 2017 Task 7 dataset. We construct context C by extracting keywords from the pun and excluding the pun word p_w. Word sense information S_{p_w} and S_{a_w} are retrieved from WordNet from SemEval annotated senses.

generation) rarely have these pun pairs provided. Instead, puns need to be generated given a more naturally-occurring conversational or creative context, requiring the identification of a pun pair that is relevant and appropriate for that context. For example, given a conversation turn *"How was the magic show?"*, a context-situated pun response might be, *"The magician got so mad he pulled his hare out."*

Motivated by real-world applications and the theory that the funniness of a pun heavily relies on the context, we formally define and introduce a new setting for pun generation, which we call *context-situated pun generation*: given a context represented by a set of keywords, the task is to generate puns that fit the given context (Figure 1). Our contributions are as follows:

- We introduce a new setting of *context situated pun generation*.
- To facilitate research in this direction, we collect a large-scale corpus called CUP (**C**ontext-sit**U**ated **P**un), which contains 4,551 tuples of context keywords and an associated pun pair, each labelled with whether they are compatible for composing a pun. If a tuple is compatible, we additionally collect a human-written pun that incorporates both the context keywords and the pun word.[1]
- We build a pipeline system with a *retrieval module* to predict proper pun words given the current context, and a *generation* module to incorporate both the context keywords and the pun word to generate puns. Our system serves as a strong baseline for *context situated pun generation*.

[1]Resources will be available at: https://github.com/amazon-research/context-situated-pun-generation

2 Task Formulation

Preliminaries. Ambiguity is the key to pun generation (Ritchie, 2005). First, we define the term *pun pair* in our work. For heterographic pun generation, there exists a pair of homophones, which we call *pun word* (p_w) and *alternative word* (a_w). While only p_w appears in the pun, both the meaning of p_w and a_w are supported in the pun sentence. Therefore, the input of heterographic pun generation can be written as (p_w, S_{p_w}, a_w, S_{a_w}), where S_{p_w} and S_{a_w} are the senses of the pun word and alternative word, respectively. We refer to these as ***pun pairs***, and use the shorthand (p_w, a_w) for simplicity. For homographic pun generation, the pun word is a polyseme that has two meanings; here, we can use the same representation, where $p_w = a_w$ for homographic puns.

Formulation. Given the unified representation for heterographic and homographic puns, we define the task of context-situated pun generation as: *Given a context C, which can be a sentence or a list of keywords, find a pun pair (p_w, S_{p_w}, a_w, S_{a_w}) that is suitable to generate a pun, then generate a pun using the chosen pun pair situated in the given context*. In this work, we assume we are given a fixed set of pun pair candidates (P_w, A_w) from which (p_w, a_w) are retrieved. The unified format between heterographic and heterographic puns makes it possible for us to propose a unified framework for pun generation.

p_w / a_w	L	**Context-Situated Pun** for *hunts, deer*
hedges/ edges	1	Why is the hunter so good at hunting deer? Because he hunts life on the hedges
husky/husk	0	-
catch/ catch	1	He hunts deer but the catch is that they rarely show up.
pine/ pine	1	Hunting deer in the forest always makes him pine for the loss.
boar/ bore	1	He is so mundane about hunting deer, but it is hardly a boar.
jerky/ jerky	1	What do you call an erratic deer that is being hunted? Jerky

Table 2: Example annotations from the ☕ CUP dataset. Labels L indicate whether the annotator was able to write a pun given the context and pun pair.

3 ☕ CUP Dataset

Motivation. The largest and most commonly-used dataset in the pun generation community is the SemEval 2017 Task 7 dataset (Miller et al., 2017).[2] Under our setting of context-situated pun generation, we can utilize keywords from the puns themselves as context. However, the majority of pun pairs only occur once in the the SemEval dataset, while one given context could have been compatible with many other pun pairs. For example, given the context *beauty school, class*, the original pun in the SemEval dataset uses the homographic pun pair *(makeup, makeup)* and says: *"If you miss a class at beauty school you'll need a* makeup *session."* At the same time, a creative human can use the heterographic pun pair *(dyed, die)* to instead generate *"I inhaled so much ash from the eye shadow palette at the beauty school class – I might have* dyed *a little inside."* Because of the limitation of the SemEval dataset, we need a dataset that has a diverse set of pun pairs combined with given contexts. Furthermore, the dataset should be annotated to indicate whether the context words and pun pair combination is suitable to make context-situated puns.

Data Preparation. We sample puns that contain both sense annotations and pun word annotations from SemEval Task 7. We show two examples of heterographic puns and homographic puns and their annotations from the SemEval dataset in Table 1. From this set, we sample from the 500 most frequent (p_w, a_w) pairs and randomly sample 100 unique context words C. [3] Combining the sampled pun pairs and context words, we construct 4,552 (C, p_w, a_w) instances for annotation.

Annotation. For our annotation task, we asked annotators to indicate whether they can come up with a pun, using pun pair (p_w, a_w), that is situated in a given context C and supports both senses S_{p_w} and S_{a_w}. If an annotator indicated that they could create such a pun, we then asked the annotator to write down the pun they came up with. Meanwhile, we asked annotators how difficult it is for them to come up with the pun from a scale of 1 to 5, where 1 means very easy and 5 means very hard. [4] To aid in writing puns, we also provided four T5-generated puns as references. [5]

We deployed our annotation task on Amazon Mechanical Turk using a pool of 250 annotators with whom we have collaborated in the past, and have been previously identified as good annotators. Each HIT contained three (C, p_w, a_w) tuples and we paid one US dollar per HIT.[6] To ensure dataset quality, we manually checked the annotations and accepted HITs from annotators who tended not to skip all the annotations (i.e., did not mark everything as "cannot come up with a pun"). After iterative communication and manual examination, we narrowed down and selected three annotators that we marked as highly creative to work on the annotation. To check inter-annotator agreement, we collected multiple annotations for 150 instances and measured agreement using Fleiss' kappa (Fleiss and Cohen, 1973) ($\kappa = 0.43$), suggesting moderate agreement.

Statistics. After annotation, we ended up with 2,753 (C, p_w, a_w) tuples that are annotated as compatible and 1,798 as incompatible. For the 2,753 compatible tuples, we additionally collected human-written puns from annotators. The number of puns we collected exceeds the number of annotated puns in SemEval 2017 Task 7 which have annotated pun word and alternative word sense annotations (2,396 puns). The binary compatibility labels and human-written puns comprise our resulting dataset, ☕ CUP (**C**ontext Sit**U**ated **P**uns). Table 2 shows examples of annotations in CUP.

[2] https://alt.qcri.org/semeval2017/task7/. The data is released under CC BY-NC 4.0 license (https://creativecommons.org/licenses/by-nc/4.0/legalcode).

[3] We sample a limited number of context words to keep the scale of data annotation feasible.

[4] Full annotation guidelines in Appendix D.

[5] Annotators find it extremely hard to come up with puns from scratch. Generated texts greatly ease the pain.

[6] This translates to be well over $15/hr.

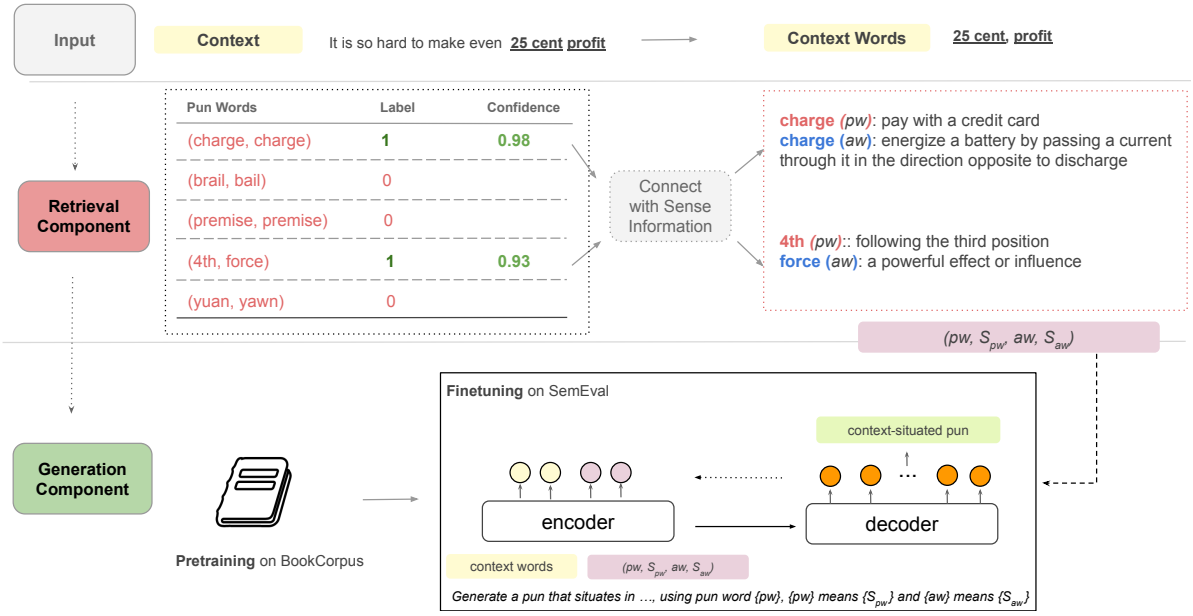


Figure 2: Our framework contains two components: (i) a retrieval component (top) that identifies relevant pun words for a given context, and (ii) a generation component (bottom) that takes the context and retrieved pun words and generates context-situated puns.

4 Context-Situated Pun Generation

We propose a pipeline framework to generate context-situated puns, shown in Figure 2. It consists of: (i) a retrieval-based module that selects a set of relevant pun word pairs, and (ii) a generation module that takes the context words and retrieved pun word pairs as input to generate puns. In this section, we briefly describe each component.

Pun Word Pair Retrieval. We propose a retrieve-and-rank strategy to select k relevant pun word pairs (p_w, a_w) from a large, fixed set of pun word pairs (P_w, A_w) for a given context C. C should be a list of keywords describing the context. If the context is given as a sentence, we use RAKE (Rose et al., 2010) to automatically extract a list of keywords from the context to construct C. For each context C, we apply a classifier to all available pun word pairs in our data (P_w, A_w) and retrieve pairs classified as suitable. Then, we rank the suitable instances according to the model's confidence and take the top k instances as the final retrieved (p_w, a_w) pun word pairs. We experiment with both supervised and unsupervised approaches to build the retrieval module in Section 5.1.

Pun Generation. Given pun word pair (p_w, a_w), pun word senses S_{p_w} and S_{a_w}, and context C, the pun generation module generates puns that relate to C, incorporate pun word p_w, and embody the meanings S_{p_w} and S_{a_w} of the pun word pair. Since there are limited pun datasets available for model training, we adopt a two-stage strategy that involves pretraining a T5-base (Raffel et al., 2020) model on non-pun text to learn to incorporate words and their senses in generations, then finetuning the model on pun data to learn the structure of puns. We describe our pun generation models in Section 5.2.

5 Experiments

We design our experiments to answer the following three research questions:

Q1. What is the performance of the pun word pair retrieval module? (Section 5.1)

Q2. What is the performance of the pun generation module? Is the pretraining stage necessary? (Section 5.2)

Q3. How well does the pipeline system perform in an end-to-end evaluation? Is the context-situated pun generation task plausible for humans? (Section 5.3)

5.1 Pun Word Pair Retrieval

In this task, for a given context C of keywords, the goal is to select k relevant pun word pairs (p_w, a_w) from a large, fixed set of pun word pairs (P_w, A_w).

Approaches. We experiment with two approaches to building pun word pair retrieval systems, including supervised neural modeling and unsupervised embedding-based approaches.

Neural. We finetune BERT-base (Devlin et al., 2019), RoBERTa-base (Liu et al., 2019) and DeBERTa-base (He et al., 2021) models on the CUP dataset for pun word pair classification. The input is formatted as sentence matching, where, given the context C as sentence 1 and the pun word pair as sentence 2, the output label indicates if the two sentences are compatible. Additionally, we experiment with finetuning natural language inference (NLI) models, RoBERTa-large-NLI (Liu et al., 2019) and BART-large-MNLI (Lewis et al., 2020). We use the context words as premise and the pun word pair as hypothesis, with *entailment* and *contradiction* labels as outputs. For each context, we retrieve all pun pairs classified as suitable by the model, then rank the instances according to the model's confidence (i.e., output from the last layer after softmax) to retrieve the top-k pun pairs.

Unsupervised. The key idea behind the compatibility classification is to find pun word pairs that are semantically close to the context. Therefore, a natural question to ask is, "Can an unsupervised method that measures semantic similarity can perform as well as the neural method?" Here, we use Euclidean distance between Glove embeddings (Pennington et al., 2014) of pun and context words to measure the semantic similarity. Formally, for a context C consisting of a list of context words $c_1, c_2, ..., c_n$, we calculate the average Euclidean distance between the Glove representation of p_w, a_w and the embedding of each of context word c_i:

$$\sum_{i=1}^{n} d(\vec{p_w}, \vec{c_i}) + \sum_{i=1}^{n} d(\vec{a_w}, \vec{c_i}). \quad (1)$$

Then we rank all 500 possible (p_w, a_w) candidates using the distance score above, retrieving the k pairs with the smallest distance as the top-k retrieved pun word pairs.

Experiment Setup. We split CUP into 70% training, 10% validation and 20% test data. Table 3 shows the distribution of pun word compatibility labels in our data splits. For each context word, we use our models to retrieve pun word pairs from 500 candidate pairs for making context-situated puns. [7]

[7] Further experimental details in Appendix A.

	train	dev	test	total
pos	1,873	290	590	2,753
neg	1,282	175	341	1,798
all	3,155	465	931	4,551

Table 3: CUP data splits for the pun word pair retrieval task. We show the distribution of (C, p_w, a_w) tuples labeled as suitable or unsuitable in each split.

Evaluation Metrics. For neural models, we first benchmark the accuracy, precision, recall, and F1 of the model's predictions for the pun word pair classification task on the CUP dataset. Additionally, for both approaches, we use the True Positive rate (TP@N) to evaluate the performance of our pun word retrieval module. It measures the percentage of top-k retrieved pun word pairs that can be used to generate puns for a given context. The higher the TP@N is, the stronger the retrieval module is in terms of retrieving appropriate pun word pairs.

Results. We show results of our supervised pun word classifiers in Table 4. Our results show that the task of classifying whether a context word is compatible with a pun word pair is challenging for current pretrained LMs, with a best F1 of 64.72 from RoBERTa-large-NLI. Table 5 shows the TP@N evaluation of pun word pairs retrieved by our best neural model, finetuned RoBERTa-large-NLI, and our unsupervised method. In general, the supervised neural model outperforms the unsupervised method. TP@1 shows that 69% of pun word pairs retrieved by the neural model are compatible with their given context, showcasing the effectiveness of our retrieval module. We provide additional qualitative analysis in Appendix C, Table 9.

5.2 Pun Generation

Given pun word pair (p_w, a_w), pun word senses S_{p_w} and S_{a_w}, and context C, the task is to generate a pun that relates to C, incorporates pun word p_w, and utilizes both pun word senses S_{p_w} and S_{a_w}.

Approach. For the novel task of context-situated pun generation, we establish a baseline model that uses a combination of pretraining on non-pun text and finetuning on pun text to generate both homographic and heterographic puns. Our unified framework for homographic and heterographic pun generation is also new to the community. We evaluate the following model variants:

AmbiPun (Mittal et al., 2022). Previous systems

	dev				test			
	F1	**Precision**	**Recall**	**Acc**	**F1**	**Precision**	**Recall**	**Acc**
bert-base (Devlin et al., 2019)	$62.29_{1.18}$	$62.23_{1.16}$	$62.58_{1.28}$	$64.02_{0.96}$	$62.39_{0.34}$	$62.30_{0.31}$	$62.56_{0.38}$	$64.70_{0.28}$
roberta-base (Liu et al., 2019)	$63.91_{0.79}$	$63.88_{0.72}$	$64.57_{0.71}$	$65.09_{0.90}$	$61.85_{0.17}$	$61.73_{0.17}$	$62.14_{0.09}$	$63.91_{0.32}$
deberta-base (He et al., 2021)	$63.93_{0.62}$	$63.84_{0.58}$	$64.14_{0.70}$	$65.73_{0.54}$	$62.55_{1.49}$	$62.48_{1.47}$	$62.70_{1.59}$	$64.91_{1.28}$
roberta-large-nli (Liu et al., 2019)	$67.25_{0.69}$	$67.13_{0.70}$	$67.45_{0.65}$	$68.96_{0.71}$	$64.72_{0.42}$	$64.96_{0.63}$	$64.60_{0.30}$	$67.60_{0.73}$
bart-large-nli (Lewis et al., 2020)	$67.33_{0.74}$	$67.28_{0.82}$	$67.54_{0.52}$	$69.03_{1.05}$	$63.81_{0.39}$	$63.83_{0.53}$	$63.87_{0.20}$	$66.31_{0.79}$

Table 4: Pun word classification performance of neural models on CUP, showing that our task is challenging for pretrained LMs. We report models' performance across three random seeds with standard deviation as subscripts.

	TP@1	**TP@5**	**TP@10**	**TP@20**
Unsupervised	64.0	59.4	60.2	**61.5**
🏆 **Neural**	**69.0**	**63.2**	**61.7**	59.3

Table 5: TP@N results for supervised (neural) and unsupervised approaches for pun word retrieval. `TP` stands for True Positive rates.

for heterographic pun generation explicitly require homophones, making it hard to adapt them to homographic puns (Yu et al., 2020; He et al., 2019). Therefore, we use AmbiPun, a the state-of-the-art homographic pun generation model, to generate both homographic and heterographic puns without further finetuning. Following their prompt format, we use "generate sentence: $\{C\}$, $\{p_w\}$, $\{a_w\}$" for homographic puns and "generate sentence: $\{C\}$, $\{p_w\}$" for heterographic puns.

Finetuned T5 (T5_{FT}). We finetune T5-base (Raffel et al., 2020) on the SemEval 2017 Task 7 dataset (Miller et al., 2017), in which puns are annotated with pun word pairs p_w and a_w along with their sense information S_{p_w} and S_{a_w}. We construct C using the RAKE (Rose et al., 2010) keyword extraction algorithm on the pun text, and further verify them against human-annotated keywords from an augmentation of the SemEval dataset we designed to enable keyword-conditioned pun generation (Sun et al., 2022). During finetuning, we use the input prompt: "*generate a pun that situates in* $\{C\}$, *using the word* $\{p_w\}$, $\{p_w\}$ *means* $\{S_{p_w}\}$ *and* $\{a_w\}$ *means* $\{S_{a_w}\}$". The goal of finetuning is to teach the model to incorporate both word senses in the final generated puns.

Finetuned T5 with pretraining ($\text{T5}_{\text{PT+FT}}$). Here, we investigate whether the model can learn to incorporate words and their senses into the generated sentences by pretraining on non-pun text. To this end, we automatically construct a pretraining dataset from BookCorpus (Zhu et al., 2015). For each word $w \in \{p_w, a_w\}$ in a given pun word pair, [8] we mine 200 sentences that contain w from BookCorpus. [9] We extract keywords from a given BookCorpus sentence containing w using RAKE to construct context C. We retain noun and verb keywords, as they are more likely to have significant impact at the sentence level (Kim and Thompson, 2000; Cutler and Foss, 1977), and exclude pun word w from the keyword list. Using these automatically-constructed samples, we finetune T5 (Raffel et al., 2020) to generate sentences situated in C that incorporate w, using the input prompt: "*generate a sentence that situates in* $\{C\}$, *using the word* $\{w\}$, $\{w\}$ *means* $\{S_w\}$ *and* $\{w\}$ *means* $\{S_w\}$", the output of which is the retrieved sentence from BookCorpus that uses C and w.

Experiment Setup. We finetune our T5 models on 1,382 training samples from SemEval Task 7 that contain both pun word and sense annotations. For testing, we randomly sample 200 (C, p_w, a_w) tuples from CUP that annotators marked as compatible. We use each model to generate puns for this set and compare their performance. [10]

Evaluation Metrics. We report *pun word incorporation rate* as the automatic evaluation metric to measure the model's ability to incorporate pun words in the final generation. We also conduct human evaluation on Amazon Mechanical Turk to judge whether the generated puns are successful. [11]

[8] We select heterographic pun pairs ($p_w \neq a_w$) to avoid introducing polysemic ambiguity in the pretraining stage.

[9] We lemmatize w and the sentence using Spacy (`https://spacy.io/`) so grammatical features will not have impact on our mining.

[10] Further experimental details in Appendix B.

[11] Turkers had to pass a qualifier by correctly labeling >= 80% of 20 samples that we manually annotated. Success is defined as whether the text supports both senses of the pun word. We measure inter-annotator agreement among 3 annotators using Fleiss' kappa ($\kappa = 0.49$), showing moderate agreement.

Model	p_w Incorp. %	Success %
AmbiPun	**97.22**	11.11
$T5_{FT}$	96.67	23.89
🏆 **$T5_{PT+FT}$**	**97.22**	**31.11**

Table 6: Pun generation results using automatic (pun word incorporation) and human (success rate) evaluation. We compare our finetuned T5 models to a state-of-the-art baseline, AmbiPun (Mittal et al., 2022). `PT` stands for Pre-Training and `FT` stands for Fine-Tuning.

Results. Pun generation results are shown in Table 6. We find that: (1) adding the pretraining stage helps our model better incorporate pun words, and (2) our generation module can generate successful puns at almost triple the rate of the current state-of-the-art framework AmbiPun (examples in Table 7). We hypothesize that this is because AmbiPun is a completely unsupervised approach in which the pun generator is not finetuned on any pun data, and because our models additionally benefit from rich word sense information in the input.

5.3 End-to-end Evaluation

Finally, we evaluate how well our pipeline retrieves relevant pun word pairs and generates novel puns given a context of keywords in an end-to-end fashion, and compare our pipeline's performance to human-standard annotations from CUP.

Experiment Setup. We randomly choose 60 context words to conduct the end-to-end evaluation. For each context, we use both unsupervised and neural pun word retrieval modules from Section 5.1 to retrieve the *top-1* predicted pun word pair, then use each of the pun generation modules from Section 5.2 to generate puns using the retrieved pun word pair. We also compare with human performance. For each context, we find the human-written pun in CUP that annotators indicated was least difficult to write, randomly sampling one pun in case of ties. We use annotation difficulty as a proxy for ranking human context-situated puns, assuming more natural puns are easier to write.

Evaluation Metrics. We measure the incorporation rate of context words C and pun words p_w as automatic evaluation metrics. In addition, similar to standalone pun generation evaluation, we conduct human evaluation to judge whether the generated puns are successful.

Results. We report results of combinations of our retrieval and generation modules in Table 8. We show that: (i) our pretraining step is helpful in terms of both improving the keyword incorporation rate and pun success rate of the generation module, despite using retrieved pun words as input. (ii) Our pipeline system performs the best among all model variations, yielding a success rate of pun generation of 40%. This success rate improves over the best reported in Section 5.2 (31%), showcasing the benefit of using our neural pun word retrieval module over randomly sampling pun word pairs for a given context. However, (iii) the best model performance is still about 32% lower than the human success rate, indicating that humans can complete the context-situated pun generation task plausibly and much more successfully, indicating large room for improvement.

6 Related Work

Our work proposes an approach for conditional generation of humor using a retrieve-and-generate framework. More specifically, our work enables a constrained type of pun generation. We briefly summarize existing work in these directions.

Humor generation. With the recent advent of diverse datasets (Hasan et al., 2019; Mittal et al., 2021; Yang et al., 2021), it has become easier to detect and generate humor. While large pretrained models have been fairly successful at humor detection, humor generation still remains an unsolved problem, and is usually studied in specific settings. Petrović and Matthews (2013) generate jokes of the type 'I like my X like I like my Y, Z'. Garimella et al. (2020) develop a model to fill blanks in a Mad Libs format to generate humorous sentences, and Yang et al. (2020) edit headlines to make them funny. More research is required to generate humorous sentences that are not constrained by semantic structure.

Retrieve and generate. Our work proposes a retrieval and generation pipeline for generating context-situated puns. The retrieval component finds proper pun word pairs for the current context, and the generation component generates puns utilizing context words and pun word pairs. Similarly, Yu et al. (2020) adopt a pair of homophones, retrieve sentences that contain either word from a large corpus then edit the sentence into a pun sentence. Sun et al. (2021) first retrieve syntac-

Context	p_w/a_w	Generated Pun	
scientist, liquid chemicals, problem	assay/ say	*Ours:* *Ambi.:*	A scientist who is a liquid chemical expert can't assay the problem. What do you call a scientist with a liquid chemicals problem? an assay-ist.
fruit vendor, daughter	yammered/ yam	*Ours:* *Ambi.:*	She was only a Fruit Vendor's daughter, but she yammered. My daughter yammered at the fruit vender... she said i'm not a fruit vender.
opera, orchestra conductors	pitch/ pitch	*Ours:* *Ambi.:*	Conductors of the opera had to make a good pitch. Why do opera and orchestra conductors pitch their voices? because they can't sing.
company football team, meeting, get together	kickoff/ kickoff	*Ours:* *Ambi.:*	A football team's meeting was about to kick off. I'm going to get together for a company football team meeting before kickoff.

Table 7: Examples of generated context-situated puns from our system and AmbiPun (Mittal et al., 2022).

Retrieval Sec 5.1	Generation Sec 5.2	Incorp. % C	Incorp. % p_w	Success %
Human		81.94	75.67	**71.67**
Unsup.	AmbiPun	**100.00**	**92.66**	10.00
	$T5_{FT}$	91.67	80.76	26.67
	$T5_{PT+FT}$	97.22	80.74	26.67
Neural	AmbiPun	98.51	97.34	21.67
	$T5_{FT}$	91.04	78.08	23.33
	$T5_{PT+FT}$	97.01	79.83	**40.00**

Table 8: End-to-end evaluation of our system against AmbiPun and human baselines.

tic parses and then generate paraphrases that keep the semantic meaning while conforming to the retrieved syntactic parses.

Pun generation. Previous work on pun generation has focused on heterographic pun generation or homographic pun generation (Miller and Gurevych, 2015; Hong and Ong, 2009; Petrović and Matthews, 2013; Valitutti et al., 2013). At the same time, all of them require an input of pun words and assume pun words are given. Heterographic pun generation requires a pair of homophones, and homographic pun generation requires a polyseme, i.e., a pun word that has more than one meaning. He et al. (2019) make use of local-global surprisal principle to generate heterographic puns and Yu et al. (2020) use constrained lexical rewriting for the same task. Hashimoto et al. (2018) use a retrieve and edit approach to generate homographic puns and Yu et al. (2018); Luo et al. (2019) propose complex neural model architectures such as constrained language model and GAN. Mittal et al. (2022) generate homographic puns given a polyseme and try to incorporate the multiple senses of the polyseme. Tian et al. (2022) proposed a unified framework to generate both homographic and homophonic puns. Our setting is different from all previous work, first asking what pun words we should use for generating a pun in a given context. Meanwhile, our work finds the connection between heterographic pun generation and homographic pun generation: both types must utilize the two meanings of a pair of words. For heterographic pun generation, the two meanings come from the pair of homophones, while for homographic pun generation, the two meanings come from the polyseme itself. Motivated by this, we propose a unified framework that can generate both heterographic puns and homographic puns adaptively.

7 Conclusion and Future Work

We propose a new setting for pun generation: *context-situated pun generation*. As a pioneering work, to facilitate future research in this direction, we first collect a large-scale corpus, CUP, which contains 4,551 annotated context and pun word pairs annotated for compatibility, along with 2,753 human-written puns for the compatible pairs, which is of an even larger size than the current most commonly-used pun dataset, SemEval 2017 Task 7 (Miller et al., 2017). To benchmark the performance of the state-of-the-art NLG techniques on the proposed task, we build a pipeline system composed of a pun pair retrieval module that identifies suitable pun pairs for a given context, and a generation module that generates context-situated puns given the context and compatible pun pairs. Human evaluation shows that the best model achieves 40% success rate in end-to-end evaluation, trailing behind human performance by almost 32%, highlighting the challenge of the task and encouraging more future work in this direction.

Our work introduces the concept of *situating in context* to pun generation. However, future

work can easily extend the idea and framework to other areas of creative generation, such as metaphor generation, lyric generation, and others. Another promising future direction is to integrate the generated puns into the original conversational or situational context to improve the interestingness and engagingness of the downstream applications. We hope our work can inspire more innovations on context-situated creative generation.

Acknowledgements

We thank PlusLab members for the discussion and early feedback about this new setup. We also thank anonymous reviewers for their constructive feedback and suggestions that helped improve the draft.

Limitations

In this work, we focus on the task of pun generation, a specific area of creative language and humor generation. We acknowledge that humor is a highly subjective area, i.e., what might be perceived as humorous may differ greatly from one person to another depending on their unique backgrounds and experiences. We hope this work and dataset can be used more broadly to give insight into how humor can differ based on contextual nuances and personal characterizations.

Ethics

We hereby acknowledge that all of the co-authors of this work are aware of the provided *ACL Code of Ethics* and honor the code of conduct.

Since we use pretrained language models for our generation tasks, we note that this makes our models susceptible to generating biased or sensitive content. While we do not explicitly address concerns around bias/sensitive content within our framework to date, we aim to incorporate these considerations into pun generation as we develop new models, including methods to filter our inputs and generated data for toxicity and biased references that may be deemed offensive.

References

Nawal Abbas and Sura Dhiaa. 2016. Pun and (un) intentional humor. *Journal of American Academic research*, 4:1–18.

Karyn Buxman. 2008. Humor in the OR: A stitch in time? *AORN journal*, 88(1):67–77.

Anne Cutler and Donald J Foss. 1977. On the role of sentence stress in sentence processing. *Language and speech*, 20(1):1–10.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Joseph L. Fleiss and Jacob Cohen. 1973. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. *Educational and Psychological Measurement*, 33:613 – 619.

Aparna Garimella, Carmen Banea, Nabil Hossain, and Rada Mihalcea. 2020. "judge me by my size (noun), do you?" YodaLib: A demographic-aware humor generation framework. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2814–2825, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Md Kamrul Hasan, Wasifur Rahman, AmirAli Bagher Zadeh, Jianyuan Zhong, Md Iftekhar Tanveer, Louis-Philippe Morency, and Mohammed (Ehsan) Hoque. 2019. UR-FUNNY: A multimodal language dataset for understanding humor. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2046–2056, Hong Kong, China. Association for Computational Linguistics.

Tatsunori B. Hashimoto, Kelvin Guu, Yonatan Oren, and Percy S. Liang. 2018. A retrieve-and-edit framework for predicting structured outputs. In *NeurIPS*, pages 10073–10083.

He He, Nanyun Peng, and Percy Liang. 2019. Pun generation with surprise. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1734–1744, Minneapolis, Minnesota. Association for Computational Linguistics.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with disentangled attention. In *International Conference on Learning Representations*.

Bryan Anthony Hong and Ethel Ong. 2009. Automatically extracting word relationships as templates for pun generation. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 24–31, Boulder, Colorado. Association for Computational Linguistics.

Abolfazl Horri. 2011. Linguistic mechanisms of humor: Pun and/or ambiguity. *Language Related Research*, 2(2):19–40.

Mikyong Kim and Cynthia K Thompson. 2000. Patterns of comprehension and production of nouns and verbs in agrammatism: Implications for lexical organization. *Brain and language*, 74(1):1–25.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Fuli Luo, Shunyao Li, Pengcheng Yang, Lei Li, Baobao Chang, Zhifang Sui, and Xu Sun. 2019. Pun-GAN: Generative adversarial network for pun generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3388–3393, Hong Kong, China. Association for Computational Linguistics.

Tristan Miller and Iryna Gurevych. 2015. Automatic disambiguation of English puns. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 719–729, Beijing, China. Association for Computational Linguistics.

Tristan Miller, Christian Hempelmann, and Iryna Gurevych. 2017. SemEval-2017 task 7: Detection and interpretation of English puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 58–68, Vancouver, Canada. Association for Computational Linguistics.

Anirudh Mittal, Pranav Jeevan P, Prerak Gandhi, Diptesh Kanojia, and Pushpak Bhattacharyya. 2021. "so you think you're funny?": Rating the humour quotient in standup comedy. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10073–10079, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Anirudh Mittal, Yufei Tian, and Nanyun Peng. 2022. AmbiPun: Generating humorous puns with ambiguous context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1053–1062, Seattle, United States. Association for Computational Linguistics.

Jacob L Moreno. 1955. Theory of spontaneity-creativity. *Sociometry*, 18(4):105–118.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Saša Petrović and David Matthews. 2013. Unsupervised joke generation from big data. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 228–232, Sofia, Bulgaria. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Graeme Ritchie. 2005. Computational mechanisms for pun generation. In *Proceedings of the Tenth European Workshop on Natural Language Generation (ENLG-05)*, Aberdeen, Scotland. Association for Computational Linguistics.

Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic keyword extraction from individual documents. *Text mining: applications and theory*, 1(1-20):10–1002.

Jiao Sun, Xuezhe Ma, and Nanyun Peng. 2021. AESOP: Paraphrase generation with adaptive syntactic control. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5176–5189, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jiao Sun, Anjali Narayan-Chen, Shereen Oraby, Alessandra Cervone, Tagyoung Chung, Jing Huang, Yang Liu, and Nanyun Peng. 2022. ExPUNations: Augmenting puns with keywords and explanations. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yufei Tian, Divyanshu Arun Sheth, and Nanyun Peng. 2022. A unified framework for pun generation with humor principles. In *Proceedings of the Findings of ACL at 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP-findings)*.

Alessandro Valitutti, Hannu Toivonen, Antoine Doucet, and Jukka Toivanen. 2013. "let everything turn well in your wife": Generation of adult humor using lexical constraints. volume 2.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Ziqing Yang, Yiming Cui, Zhipeng Chen, Wanxiang Che, Ting Liu, Shijin Wang, and Guoping Hu. 2020. TextBrewer: An Open-Source Knowledge Distillation Toolkit for Natural Language Processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 9–16, Online. Association for Computational Linguistics.

Zixiaofan Yang, Shayan Hooshmand, and Julia Hirschberg. 2021. CHoRaL: Collecting humor reaction labels from millions of social media users. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4429–4435, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhiwei Yu, Jiwei Tan, and Xiaojun Wan. 2018. A neural approach to pun generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1650–1660, Melbourne, Australia. Association for Computational Linguistics.

Zhiwei Yu, Hongyu Zang, and Xiaojun Wan. 2020. Homophonic pun generation with lexically constrained rewriting. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2870–2876, Online. Association for Computational Linguistics.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

A Classifier Implementation Details

We finetune five pretrained language models for classifying whether context words and pun word pairs are compatible in Table 4, and we use HuggingFace (Wolf et al., 2020) throughout our implementation for accessing model checkpoints and modeling. For hyper-parameter search, we tried the combinations of learning rate {$1e^{-4}$, $3e^{-4}$, $1e^{-5}$, $3e^{-5}$} * training epoch {3, 10, 20}. The final hyper-parameters for bert-base, roberta-base and deberta-base are: learning rate $1e^{-5}$, training epoch 20 and training batch size 32. For roberta-large-mnli and bart-large-mnli models, we reduce the training epochs to 3 and training batch size to 8. We choose the checkpoint with the best accuracy on the dev set for inference.

B T5 Implementation Details

We finetune multiple T5 models (Raffel et al., 2020) in our work, and we use T5-base from SimpleT5 [12] throughout our implementation. We use 512 and 256 for the maximum source length and the maximum target length respectively. As the optimizer, we use AdamW (Loshchilov and Hutter, 2019) with a learning rate of 0.0001. For the pretraining stage, we finetune T5 for 3 epochs on retrieved BookCorpus data. During the finetuning stage, we train each model on a Tesla V100 with a batch size of 8 for 30 epochs. During inference, we use beam search as the decoding method with a beam size of 2. We terminate decoding when the EOS token is generated or the maximum target length is reached.

C Retrieved Pun Word Pair Examples

Table 9 shows examples of retrieved pun word pairs from both the unsupervised and neural methods.

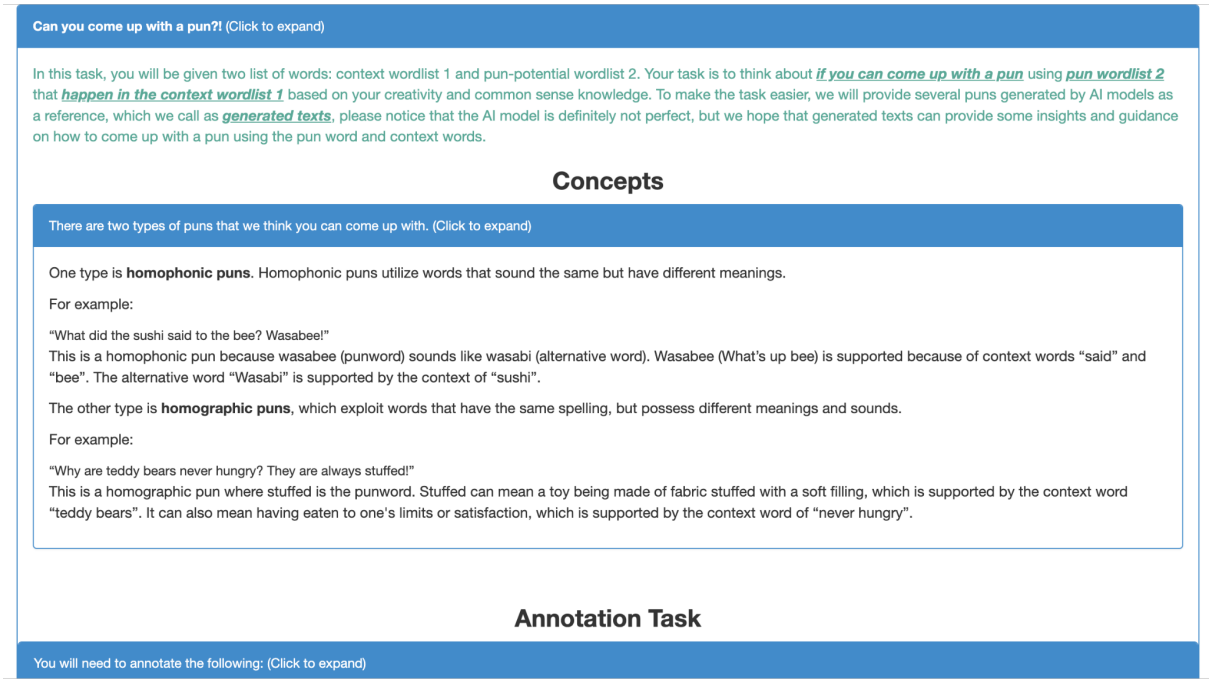
D Annotation Guidelines

Figure 3 shows our annotation interface for collecting the CUP dataset.

[12] https://github.com/Shivanandroy/simpleT5

Context	SemEval Annot.	Modeling	Retrieved Pun Word Pairs
einstein, parents	relatively, relativity	Unsupervised	(kid, kid), (father, feather), (allow, aloud), (census, sense), (throng, wrong)
		neural	**(relatively, relativity)**, (pinch, pinch), (pupil, pupil), (father, feather), (kid, kid)
bright star	seriously, sirius	Unsupervised	(bright, bright), (guess, guest), (limelight, lime), (father, feather), (mist, miss)
		neural	(bright, bright), (light, light), (constellation, consolation), **(seriously, sirius)**, (serious, sirius)
interpreters, die	sign, sign	Unsupervised	(go, go), (turn, turn), (dye, die), (throng, wrong), (get, get)
		neural	(connection, connection), (dye, die), **(sign, sign)**, (sentence, sentence), (fluently, flue)

Table 9: Examples of retrieved pun word pairs from both the *unsupervised* and the *neural* method. We highlight the annotated pun word pairs from the SemEval dataset in the prediction list in **bold**. However, using the annotated pun word pairs as the only ground truth underestimates the pun word retrieval module. As shown here, both methods can retrieve pun word pairs that are related to the context. However, these (context words, p_w, a_w) combinations are missing from the original SemEval annotations. This again highlights the importance of collecting ☕ CUP that includes (context words, p_w, a_w) pairs to facilitate future studies in the context-situated pun generation domain.



- **can you make a pun? what is the pun?** the pun needs to situate in the context wordlist 1 using one of pun words from pun wordlist 2. you need to include as many context words in the context wordlist 1 as possible.
- **what is the pun?** write the pun that you come up with. Please note that **only pun word** should appear in the pun (not both pun word and alternative word!!), but it MUST contain both the meaning of two senses we are giving.
  - **you can put NA if you really cannot come up with a pun, but if you put less than 5 NAs every 50 HITs, we will send you a $10 bonus**
  - we emphasize that you CAN manipulate and play with the referred pun, revise them for your own pun
  - for the context words, it is the best that you can utilize all the context words, but it is fine that you only utilize a couple of them (at least one) though!!
- **Rate how hard it was for you to make the pun on a Likert scale of 1-5 (1 very easy, 5 extremely hard; choose NA if you cannot write a pun)?** (please choose "NA" for this question if you put "NA" in the last question)
  - [score of 1] the given context words **relate well** to the pun word and both senses that you can **easily** come up with a **very good** pun, which can be very **easily understood and interpreted** by others!
    [Example] context words: desert, people; pun word: dry; two senses: dry can mean "lack of moisture", it can also mean "does not contain too much emotion". **pun**: people in the desert have a very dry sense of humor.
    [Explanation] it is easy to incorporate both senses of "dry" because od desert and people are both closely related to the two senses. The pun by self evaluation is a good .
  - [score of 3] the given context words still **relate well** to the pun word and **at least one sense** in two provided senses. After putting a good effort, we can come up with a pun. The pun can be interpreted by yourself easily, but it will take a good amount of time for others to interpret the sentence as a pun: they can still recognize it as a pun by looking at both senses.
    [Example] context words: coins,to be; pun word: handy; handy means "useful and convenient", the other word is "hand. **pun** : I like my coins to be at handy when I need them.
    [Explanation] handy and coins are definitely related. the pun here can be interpreted as a pun, but other people need to interpret carefully and look at provided senses to be able to understand the pun.
  - [score of 5] the given context words **only loosely relate** to one of the two senses. one has to try extremely hard to come up with a pun using the context words and the pun word. It is **impossible** for you or others to understand the pun **without** knowing the two senses beforehand. It is still a pun, but most people will see it as a bad pun.
    [Example] context words: condition,mouthwash bottles; pun word: handy; handy means "useful and convenient", the other word is "hand".
    pun: I keep my mouthwash bottles in good condition so they don't slip off my handy
    [Explanation] "hand" loosely relates to mouthwash bottles, and people will unlikely recognize this as a pun.

Figure 3: The annotation interface for collecting CUP dataset.