POLYJUICE: Generating Counterfactuals for Explaining, Evaluating, and Improving Models

Tongshuang Wu[1] **Marco Tulio Ribeiro**[2] **Jeffrey Heer**[1] **Daniel S. Weld**[1,3]
[1]University of Washington [2]Microsoft Research [3]Allen Institute for Artificial Intelligence
wtshuang@cs.uw.edu marcotcr@microsoft.com {jheer,weld}@cs.uw.edu

Abstract

While counterfactual examples are useful for analysis and training of NLP models, current generation methods either rely on manual labor to create very few counterfactuals, or only instantiate limited types of perturbations such as paraphrases or word substitutions. We present POLYJUICE, a general-purpose counterfactual generator that allows for control over perturbation types and locations, trained by finetuning GPT-2 on multiple datasets of paired sentences. We show that POLYJUICE produces diverse sets of realistic counterfactuals, which in turn are useful in various distinct applications: improving training and evaluation on three different tasks (with around 70% less annotation effort than manual generation), augmenting state-of-the-art explanation techniques, and supporting systematic counterfactual error analysis by revealing behaviors easily missed by human experts.

1 Introduction

Counterfactual reasoning — mentally simulating what *would have happened* if conditions were different — is a common tool for making causality assessments (Kahneman and Tversky, 1981), which in turn are crucial for model evaluation, error analysis, and explanation (Miller, 2019). For example, in Figure 1, "It is great for kids" is perturbed into multiple variations, each providing unique insights by simulating what would have happened if the sentence was different.

Applications of counterfactual reasoning to NLP generally specify the relationship $x \nrightarrow \hat{x}$, and then create $\hat{x}$ according to the relationship. As a result, prior work has tailored counterfactual generators for different applications, only collecting subsets of $\hat{x}$ that are useful for the specific task. For example, to support *model training and evaluation*, human annotators create counterfactuals

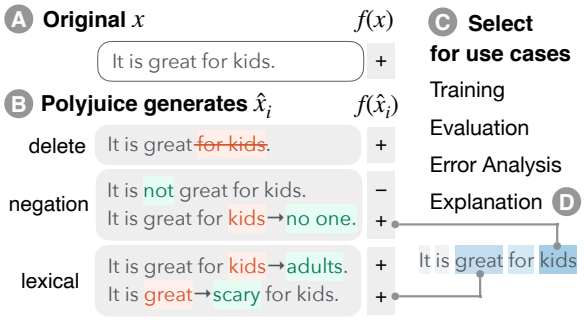

Figure 1: Overview: (A) given a sentiment analysis instance x, POLYJUICE[1] generates (B) various counterfactuals $\hat{x}$, which are then (C) selected for downstream use. *e.g.,* in (D) we select counterfactual explanations that complement a black box explanation: though "great" and "kids" are deemed important, perturbing them may not affect the prediction $f(x) = f(\hat{x}) = positive$, revealing model failures not covered by feature attributions.

that change the groundtruth labels by manually rewriting instances (Gardner et al., 2020; Qin et al., 2019) or defining perturbation functions (Ribeiro et al., 2020). Manual rewrites are costly (*e.g.,* 4–5 minutes per counterfactual (Kaushik et al., 2020)) and susceptible to systematic omissions (*e.g.,* human annotators may cover great ↛ not great, but miss kids ↛ no one in Figure 1B). Meanwhile, automated generators for *model analysis and explanation* usually focus on other relationships, *e.g.,* generating $\hat{x}$ that have different model predictions than x (Ross et al., 2020; Zhang et al., 2019a). As a result, they neglect prediction-preserving counterfactuals that are equally important for understanding or shaping model behaviors, like kids ↛ no one and great ↛ scary linked to Figure 1D.

However, counterfactual generation does not *have* to be task-specific. The same set of counterfactuals in Figure 1 can support a variety of applica-

[1]We open source POLYJUICE at https://github.com/tongshuangwu/polyjuice.

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 6707–6723
August 1–6, 2021.

tions. Moreover, for cases like model explanation and analysis, a general-purpose pool of counterfactuals may be preferable, as the relationship of interest can be more exploratory and user-oriented (Wu et al., 2019). In this work, we formalize the task of *counterfactual generation*, disentangling generation from the application of counterfactuals. Given an input x (Figure 1A), our generator produces a set of counterfactuals $\hat{\mathbf{X}} = \{\hat{x}_1, \hat{x}_2, ...\}$ with *application-agnostic* relationships $x \nrightarrow \hat{x}_i$ (Figure 1B). Afterwards, we use *application-specific* selection methods to find subsets of $\hat{x}$ that are most effective for a given use case (Figure 1C).

We frame the generation step as conditional text generation, and finetune GPT-2 (Radford et al., 2019) into a generator called POLYJUICE using $(x, \hat{x})$ pairs. To allow for targeted counterfactuals, we also design control codes like `negation` or `delete` (Figure 1B), and adopt fill-in-the-blank structures (Donahue et al., 2020) to specify where the perturbation occurs and how. Intrinsic evaluation shows that POLYJUICE generates $\hat{x}$ that are *fluent*, *diverse*, and *close to* x, and that the *control* mechanisms retrieve perturbations that would likely not be sampled from off-the-shelf language models.

With simple selection heuristics, we show that a single POLYJUICE model can significantly aid humans in diverse downstream applications.[2] For *counterfactual training and evaluation* (§3), humans label POLYJUICE counterfactuals rather than creating them from scratch. They produce training data that significantly improve model generalization, as well as contrast sets that help identify model vulnerabilities (Gardner et al., 2020), with around 70% less annotation effort. In another application, POLYJUICE produces *counterfactual explanations* (§4), providing significant insight on top of state-of-the-art explanation techniques. Finally, POLYJUICE supports counterfactual *error analysis* (§5). It allows users to explore related counterfactuals (*e.g.,* the model responds differently to different negation forms in Figure 1B), and to aggregate individual counterfactuals into patterns in order to gain systematic understanding of model behavior.

2 General-Purpose Counterfactuals

2.1 Definition and Desiderata

Given an instance x, a generator g produces a set of counterfactuals $\hat{\mathbf{X}} = \{\hat{x}_1, \hat{x}_2, ...\}$ with various relationships $x \nrightarrow \hat{x}_i$. For example, great ↛ not great, kids ↛ no one in Figure 1B are both instances of the `negation` relationship. Each $(x, \hat{x})$ pair shares multiple relationships — these two are also instances of the *label flipping* relationship if the task is sentiment analysis (but might not be for other tasks). As illustrated in §1, knowing which relationships apply aids selection for downstream applications.

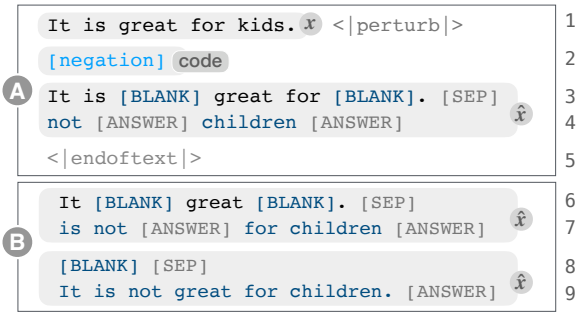

Figure 2: (A) POLYJUICE prompt format, which concatenates the original x, the control code, and the $\hat{x}$ ("It is not great for children" converted to an infilling structure). At *generation* time, POLYJUICE accepts prompts that just include x (Line 1), or optionally with the code and the `[BLANK]`s (Lines 2–3), and fills in the blanks sequentially with spans separated by `[ANSWER]`s (Line 4). (B) POLYJUICE allows blanking at different granularities (even the entire sentence), such that Lines 3–4 in (A) can be replaced by Lines 6–7 or 8–9.

We expect g to produce counterfactuals $\hat{x}$ that are (1) **close** to x, preferably only involving the minimal changes necessary to establish a certain effect (Pearl, 2018), allowing users to make causality assessments. The generated $\hat{x}$ should also be (2) **fluent**, *i.e.,* grammatically correct (Morris et al., 2020) and semantically meaningful (*e.g.,*"Colorless green ideas sleep furiously" is not meaningful (Chomsky, 2002)). Fluency operationalizes "probable" counterfactuals in the context of NLP; as Kahneman and Tversky (1981) stated, humans strongly favor counterfactuals that are close to the original instance, but also prefer those that could have easily happened without assuming rare events or strange coincidences. Further, as a general-purpose generator, g should produce counterfactuals with a measure of (3) **control** over relationships $x \nrightarrow \hat{x}$, such that the counterfactuals can vary with the object-of-attention in each application (the "focus rule" (Kahneman and Tversky, 1981)). Finally, we expect g to output a (4) **diverse** set of $\hat{x}$ in terms of relationships, covering a large variety of "what-ifs" for different applications (Pearl, 2018).

[2] We demonstrate POLYJUICE in semi-automatic settings, but as discussed in §2.2, it can also work automatically.

Control code	Definitions and POLYJUICE-generated Examples	Training Datasets
`negation`	A dog is not embraced by the woman.	(Kaushik et al., 2020)
`quantifier`	A dog is → Three dogs are embraced by the woman.	(Gardner et al., 2020)
`shuffle`	*To move (or swap) key phrases or entities around the sentence.* A dog → woman is embraced by the woman → dog.	(Zhang et al., 2019b)
`lexical`	*To change just one word or noun chunk without altering the POS tags.* A dog is embraced → attacked by the woman.	(Sakaguchi et al., 2020)
`resemantic`	*To replace short phrases without altering the remaining dependency tree.* A dog is embraced by the woman → wrapped in a blanket.	(Wieting and Gimpel, 2018)
`insert`	*To add short phrases without altering the remaining dependency tree.* A dog is embraced by the little woman.	(McCoy et al., 2019)
`delete`	*To remove short phrases without altering the remaining dependency tree.* A dog is embraced by the woman.	(McCoy et al., 2019)
`restructure`	*To alter the dependency tree structure,* e.g., *changing from passive to active.* A dog is embraced by → hugging the woman.	(Wieting and Gimpel, 2018)

Table 1: We design a list of control codes to guide generation. We show POLYJUICE-*generated* counterfactual examples, and the representative training datasets for each corresponding pattern. Details are in Appendix A.

2.2 Conditional Counterfactual Generation

We frame counterfactual generation as a conditional text generation task using language models (LMs), and train POLYJUICE by finetuning GPT-2 (Radford et al., 2019) using the following prompt design (alternative LMs could also have been used).

Prompt format design. To ensure that $\hat{x}$ is *close* to x rather than arbitrary text, we condition the generation on x, followed by a special token (Line 1 in Figure 2A). In Line 2, we have *control codes* (Keskar et al., 2019) such as `negation`. We design them to specify types of perturbation from among lexical, syntactic, or semantic aspects (see Table 1), inspired by prior work that categorizes manually created counterfactuals (Kaushik et al., 2020; Gardner et al., 2020). As an additional layer of control over $x \rightarrow \hat{x}$, we allow users to specify *where* changes happen by having the LM infill `[BLANK]` tokens (Donahue et al., 2020), rather than generating arbitrary counterfactuals (Lines 3–4).

Finetuning GPT-2 — a causal LM for predicting next tokens — additionally allows us to exercise control at various levels of granularity. At generation time, if the user provides only the original example, POLYJUICE will generate the control code, the blank locations, and the infilling (Lines 2–4). Alternatively, the user can specify the control code, or the control code *and* the blanks, to exercise different degrees of control depending on the application. As later shown in §4 and §5, such control is important for different use cases.

Training data. To train a conditional model, we combine six existing sentence-pair datasets, each containing a subset of the desired phenomena in Table 1. Further, we find naturally occurring sentence pairs (filtered by edit distance to guarantee closeness) in non-paired datasets including CommonGen (Lin et al., 2020), Natural Questions (Kwiatkowski et al., 2019), and SQuAD (Rajpurkar et al., 2016), such that the resulting dataset contains *diverse* counterfactuals.[3]

We translate these sentence pairs into the format given in Figure 2A. For each $(x, \hat{x})$, we compute its primary control code using part-of-speech tags and dependency trees. For example, `negation` occurs when we observe changes to negation modifiers or specific words like "supposedly", and `shuffle` occurs when we have overlap between tokens deleted and added. When multiple changes occur, we label it with the control code which most significantly changes the semantics of the corresponding subphrase as computed by SBERT (Reimers and Gurevych, 2019). For example, in Figure 2A, `negation` (great → not great) is more significant than `lexical` (kids → children). To balance the distribution (Table 7 in Appendix A), for each dataset, we extract control codes from all the $(x, \hat{x})$,[4] and randomly sample up to 10,000 instances per codes.

In order to allow for flexible blanking at generation time, we generate multiple training prompts per pair, covering different dependency tree struc-

[3] We exclude data related to our applications, *e.g.,* PAWS-QQP (Zhang et al., 2019b).

[4] We use sentences in a pair interchangeably as x and $\hat{x}$ to learn the control codes both ways.

Model	Diversity	Closeness	
	Self-BLEU ↓	Levenshtein ↓	Syntactic ↓
POLYJUICE	0.34	**0.25**	**2.13**
GPT-2	**0.18**	0.70	6.35
T5	**0.12**	9,52	3.50
RoBERTa	0.47	**0.14**	**1.32**

Table 2: Intrinsic evaluations: POLYJUICE counterfactuals are *closer* to the original instance than non-fintuned GPT-2 and T5, and more *diverse* than RoBERTa. Computational details are in Appendix A.2.

tures related to the perturbed spans (Figure 2B), including (1) just the changed tokens, (2) the associated parsing structures, (3) the merged changes, and (4) the entire sentence. We eventually obtain 657, 144 prompts from 186, 451 pairs.

Fluency filtering. While the original GPT-2 produces *fluent* text, some combinations of control codes and blanks cause POLYJUICE to generate nonsensical results. Following Morris et al. (2020), we score both x and $\hat{x}$ with GPT-2, and filter $\hat{x}$ when the log-probability (on the full sentence or the perturbed chunks) decreases by more than 10 points relative to x. Fully automated uses of POLYJUICE (*e.g.,* adversarial attacks) may benefit from stricter constraints, at the cost of diversity (as surprising changes may be filtered even if they are fluent).

2.3 Intrinsic Evaluation

We evaluate POLYJUICE on *closeness* and *diversity* by comparing its perturbations on 300 randomly selected sentences with baselines that use more or less context from x: (1) non-finetuned GPT-2, (2) token-infilling RoBERTa (Liu et al., 2019) and (3) span-infilling T5 (Raffel et al., 2020).

As shown in Table 2, POLYJUICE generates counterfactuals that are close to the original instance, measured by syntactic tree (Zhang and Shasha, 1989) and Levenshtein edit distance (Levenshtein, 1966). In contrast, non-finetuned GPT-2 generates arbitrary text instead of perturbations when given the starting tokens of a sentence, as it only leverages context in a single direction. As for infilling models, POLYJUICE counterfactuals are more diverse (measured by self-BLEU (Zhu et al., 2018)) than RoBERTa ones, which is restricted to word substitution. Meanwhile, T5 displays higher diversity but less closeness, probably due to the fact that it does not consider the original masked tokens when generating $\hat{x}$. For example, in Figure 1 "It is great for kids," T5 replaces "for kids" with "idea", "to meet you," whereas POLYJUICE generates "for kids yet adults can enjoy," "for any audience."

We evaluate *controllability* by comparing POLYJUICE with T5 as well as with GPT-2 finetuned on prompts *without* codes. We verify that the codes improve the success rate of generating counterfactuals with the desired perturbation types set out in Table 1 by as much as 42% for perturbations such as `negation` and `insert`. For example, given "It is `[BLANK]` great for kids," baselines generate "also," "fun and," rather than "not" (`negation`).

We further verify the *fluency* for POLYJUICE counterfactuals in three tasks/datasets: (1) *Sentiment* Analysis, SST-2 (Socher et al., 2013), (2) Natural Language Inference (*NLI*), SNLI (Bowman et al., 2015), and (3) Duplicate Question Detection (QQP) (Wang et al., 2019). We randomly select 100 sentences per dataset, generate 3 $\hat{x}$ per x, and ask crowd workers to rate whether they are *"likely written by native speakers."* The workers rated most counterfactuals as fluent: 78% in SST-2, 76% in QQP, and 86% in SNLI. In subsequent sections, we show these rates are suitable for applications where people "team up" with POLYJUICE.

3 Counterfactual Evaluation & Training

We ask crowdworkers to label POLYJUICE-generated counterfactuals for *Sentiment*, *NLI*, and *QQP*, for the purposes of evaluation and training.[5] In each labeling round, the worker is presented with an original x and its label, and asked to annotate the groundtruth for three $\hat{x}$, rejecting non-fluent ones (details and interface in Appendix B.1).

We use a simple heuristic to select which counterfactuals are presented for labeling, aimed at increasing diversity. Representing each $\hat{x}$ by its token changes, control code, and dependency tree structure, we greedily select the ones that are least similar to those already selected for labeling. This avoids redundancy in the labeling set, *e.g.,* common perturbation patterns such as black → white.

3.1 Evaluation with Contrast Sets

We verify whether POLYJUICE counterfactuals can be used to create *contrast sets* (Gardner et al., 2020), *i.e.,* evaluation sets where each instance has a nearby counterfactual with a *different* groundtruth, to better evaluate model decision boundaries. We

[5] We collect *asymmetric counterfactuals* (Garg et al., 2019) by sampling more *Duplicate* and *Entailment* examples in *QQP* and *NLI* to perturb, due to the difficulty of flipping other labels.

Task	Dev.	Orig. set	Contrast set ↓	Consistency ↓
Sentiment	94.3	93.8	84.9 (-8.9)	76.1
NLI	86.5	91.6	72.3 (-19.3)	56.4
QQP	91.7	87.5	75.3 (-12.2)	61.1

Table 3: POLYJUICE $\hat{x}$ as contrasts sets, with model accuracy on the development set, the original set of x, the contrast sets, and consistency (cases where the model predicts both x and $\hat{x}$ correctly). The performance drops are similar to that of expert-created sets (Gardner et al., 2020), on which the accuracy of all classification models decreases by 9.8 on average, with a consistency of ≈64.1. This indicates POLYJUICE can be used to create such sets without expert annotators and at less cost.

construct these sets by simply filtering out counterfactuals that are labeled the same as their original instances (40%–63% depending on the task).

For each task, we test multiple classifers open-sourced by Huggingface (Wolf et al., 2020), and report the best performing model for each[6] in Table 3 (results for other models are analogous). POLYJUICE contrast sets display performance gaps consistent with those of Gardner et al. (2020), where the sets are constructed manually by NLP researchers, even though we use non-expert annotators who only *label* examples rather than creating them.

3.2 Training with Counterfactuals

Following Kaushik et al. (2020), we augment training sets with counterfactual examples. In all experiments, we finetune `roberta-base` on datasets of n original examples and m counterfactuals, which are generated by POLYJUICE (`m-polyjuice`) or crafted from scratch by humans (`m-CAD` from Kaushik et al. (2020), only available for *NLI*). To distinguish the benefit of counterfactuals from that of just adding more data, we further add a baseline that uses $n+m$ original examples (`m-baseline`). In addition to in-domain test set accuracy, we measure models' generalization on out-of-domain datasets, as well as contrast sets and challenge sets. We also evaluate model capabilities with CheckList (Ribeiro et al., 2020) for *Sentiment* and *QQP*. Reported model performances are averaged across multiple data samples and random seeds (Appendix B.2).

For *Sentiment*, we select random POLYJUICE counterfactuals regardless of their labels, as long as an original x has at least one $\hat{x}$ that flips the label. For *NLI* and *QQP*, we observed in a pilot study that randomly chosen counterfactuals may not be more effective than the same amount of additional data. We suspect that POLYJUICE lacks domain knowledge and context for identifying critical perturbations, and therefore brings benefits redundant with pre-training (Longpre et al., 2020). Thus, we use the slicing functions of Chen et al. (2019) to find patterns of interest (*e.g.,* prepositions in *NLI*), and perturb those patterns by placing `[BLANK]`s on the matched spans. For example, "His surfboard is beneath him" becomes "His surfboard is `[BLANK]` him", and POLYJUICE generates counterfactuals such as "His surfboard is beneath → next to him."

Results. Tables 4–6 indicate that POLYJUICE augmentation is effective in all tasks: `m-polyjuice` maintains in-domain accuracy while *consistently* improving or maintaining generalization accuracy in various out-of-domain and challenge sets. On *NLI*, POLYJUICE counterfactuals are as effective or more effective than counterfactuals created from scratch (`m-CAD`). Notably, we obtain the largest gains on challenge and contrast sets (*e.g.,* `Break` and `DNC` in Table 5) or when the out-of-domain dataset is sufficiently different from the training domain (*e.g.,* Senti140 and SemEval in Table 4). POLYJUICE also improves results on CheckList tests that previously had high error rates: it significantly lowers the error rates on 11 out of 27 *QQP* tests,[7] making 2/27 tests worse. For *Sentiment*, it improves the model on 5 out of 15 tests, hurting 1. Here, we only report a low m/n ratio (<10% for NLI and QQP) to show that a small amount of augmentation is already beneficial. The results are similar for other combinations we explored (see Appendix B.2), except when the ratio of counterfactual to original data was too high (*e.g.,* $m = n$ may decrease vocabulary diversity or induce additional data bias, echoing (Khashabi et al., 2020)).

3.3 Discussion

We show that POLYJUICE counterfactuals are useful for evaluation, and more effective than additional (non-counterfactual) data for training in a variety of tasks. In contrast to prior work where humans generate counterfactuals from scratch, we only ask them to *label* automatically generated ones, while still achieving similar or better results.

We believe our approach is more effective than manual creation (although both are beneficial): in

[6] `huggingface.co/{roberta-large-mnli, textattack/roberta-base-SST-2, ji-xin/roberta_base-QQP-two_stage}`

[7] The absolute error rate drops for at least 5 points, with a relative difference of more than 10%.

Model	SST-2	Senti140	SemEval	Amzbook	Yelp	IMDB	IMDB-Cont.	IMDB-CAD
`m-baseline`	92.9 ± 0.2	88.9 ± 0.3	84.8 ± 0.5	85.1 ± 0.4	90.0 ± 0.3	90.8 ± 0.5	92.2 ± 0.6	86.5 ± 0.2
`m-polyjuice`	92.7 ± 0.2	**90.7 ± 0.4**	**86.4 ± 0.1**	85.6 ± 0.8	90.1 ± 0.0	90.6 ± 0.3	**94.0 ± 0.3**	**89.7 ± 0.5**

Table 4: *Sentiment* model performance, with n=4,000 and m=2,000. **Bolded** cells highlight significant improvements. `m-polyjuice` maintains the in-domain and out-of-domain accuracies on reviews (SST-2, Amzbook, Yelp, IMDb Movie Review (Ni et al., 2019; Asghar, 2016; Maas et al., 2011)), improving it on Twitter data (Senti140 and SemEval 2017 (Go et al., 2009; Nakov et al., 2013)) and contrast sets (Gardner et al., 2020; Kaushik et al., 2020), likely because their distributions are less similar to the original SST-2 training data.

Model	SNLI	MNLI-m	MNLI-mm	SNLI-CAD	break	DNC	stress	diagnostic
`m-baseline`	85.7 ± 0.4	86.1 ± 0.2	**86.6 ± 0.2**	72.8 ± 0.3	86.4 ± 1.5	54.5 ± 0.6	65.1 ± 0.6	56.0 ± 0.8
`m-CAD`	85.8 ± 0.6	**86.6 ± 0.1**	85.6 ± 0.3	**73.8 ± 0.2**	89.4 ± 2.9	55.8 ± 0.9	65.5 ± 0.5	56.4 ± 0.4
`m-polyjuice`	85.3 ± 0.3	86.0 ± 0.1	**86.4 ± 0.0**	**73.6 ± 0.2**	**89.1 ± 1.2**	**57.7 ± 0.3**	65.1 ± 0.2	**57.5 ± 0.5**

Table 5: *NLI* models, with n=20,000 and m=1,574. `m-polyjuice` improves accuracy on contrast and challenge sets (Kim et al., 2019; Naik et al., 2018; Glockner et al., 2018; Wang et al., 2019); it exhibits comparable (or better) gains than `m-CAD` (manual counterfactuals) with less implementation and annotation effort.

Model	QQP	PAWS-QQP
`m-baseline`	84.5 ± 0.6	37.0 ± 0.5
`m-polyjuice`	84.7 ± 1.0	**38.7 ± 0.4**

Table 6: PolyJuice with n=20,000 and m=1,911 improves accuracy on PAWS-QQP (Zhang et al., 2019b).

terms of implementation effort, the process of just labeling counterfactuals is the same as labeling original examples, such that no additional annotator training or separate pipelines are required; in contrast, Kaushik et al. (2020) set up two separate crowdsourcing tasks for creating and labeling the counterfactuals. Further, annotator effort is much lower, as evaluating examples is easier than creating them — Kaushik et al. (2020) report an average of ≈2 minutes per *NLI* counterfactual *prior* to quality validation, while our median time was 10 seconds per counterfactual. Even after our quality validation (removing noisy annotators, disregarding non-fluent counterfactuals), our rate for *NLI* is ≈36 seconds per counterfactual (used in Table 5).

In terms of the utility per counterfactual, manual creation and PolyJuice may be complementary. Manual annotation may be unreliable or incomplete for certain forms of counterfactuals (Ribeiro et al., 2018), whereas PolyJuice can miss more complex or context-dependent changes, and could benefit from target perturbations that compensate for its lack of domain knowledge (targeted guidance is also helpful for human annotators (Huang et al., 2020)). Thus, it may be important to mix both approaches (Khashabi et al., 2020). PolyJuice's flexibility opens up possibilities for hybrids between human creation and human verification of targeted, machine-generated counterfactuals.

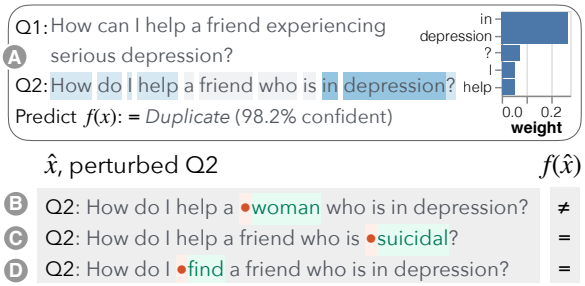

Figure 3: (A) An instance in *QQP* where the model prediction $f(x)$ is *Duplicate* (=) at 98.2% confidence, with SHAP importance weights for tokens in Q2. Counterfactual explanations complement SHAP with concrete examples and surprising behaviors, *e.g.,* (B) shows that friend → woman surprisingly flips the prediction to *Non-Duplicate* (≠), despite the low weight on "friend."

4 Counterfactual Explanations

A popular way of explaining NLP models is to attribute importance weights to the input tokens, either using attention scores (Wiegreffe and Pinter, 2019) or by summarizing the model behavior on perturbed instances (*e.g.,* LIME (Ribeiro et al., 2016) and SHAP (Lundberg and Lee, 2017)). Though ubiquitous, token scores may not always reflect their real importance (Pruthi et al., 2020). Popular packages like LIME or SHAP estimate scores by *masking* words, and therefore may not reflect model behavior on natural counterfactual cases. For example, the token "friend" in Figure 3A is not considered important even though a natural substitution in Figure 3B flips the prediction. The opposite happens to "in depression," where a significant change makes no difference to the model's prediction (Figure 3C). Even perfect importance

scores may be too abstract for users to gain real understanding (Miller, 2019), *e.g.,* users may not grasp the significance of a low importance score for the token "help" without concrete examples such as the one in Figure 3D.

Since presenting a large number of concrete counterfactuals would be overwhelming, we propose a hybrid approach, displaying feature attributions as a high-level summary, together with a judicious selection of POLYJUICE counterfactuals that make behaviors concrete and highlight potential limitations. Following Miller (2019)'s observation that people look for explanations revealing *unexpected* behavior, we select *surprising* counterfactuals.[8] That is, we estimate the expected change in prediction with feature attributions, and select counterfactuals that violate these expectations, *i.e.,* examples where the *real* change in prediction is large even though importance scores are low (Figure 3B), and examples where the change is small but importance scores are high (Figure 3C). Of course, users can also view additional counterfactuals that perturb tokens of particular interest, a technique that we explore in the next section.

User evaluation. We study the scenario where an expert has access to a model and local explanations, and evaluate the *additional* benefit of showing counterfactuals, *i.e.,* whether they bring *new* insights. We evaluate three ways of generating counterfactuals: (1) POLYJUICE-*random*, a baseline where we show random POLYJUICE counterfactuals, (2) *Expert-surprise*, where two graduate students (non-participants) were given access to the model and instructed to create counterfactuals that are surprising given the associated SHAP scores, and (3) POLYJUICE-*surprise*, which uses the selection procedure described in the previous paragraph.

We recruited 13 participants (graduate students with experience in model explanation), and had them analyze the aforementioned *QQP* model. In each round, users were shown an example, the model prediction, and a SHAP explanation, as in Figure 3A. Users were instructed to create up to 10 counterfactuals in order to better understand model behavior around the example, for which model predictions were given (users created 6 on average). Finally, users simulated what the model would do on six counterfactuals (Hase and Bansal, 2020), two from each condition (in random order). Counterfactuals where users make mistakes are preferable, as displaying these would add information that users do not already have.

[8] Details in Appendix C.1.

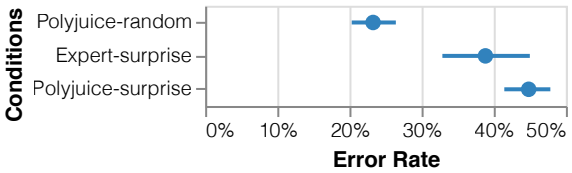

Figure 4: Simulation error rates per condition (higher the better). POLYJUICE-*surprise* has the highest error rate, indicating these counterfactuals would add the most information to users if displayed.

As shown in Figure 4, humans simulated model behavior on POLYJUICE-*surprise* counterfactuals only slightly better than random guessing (45% ± 6%), *i.e.,* these examples display model behavior that is surprising to users even after seeing explanations and creating their own counterfactuals. *Expert-surprise* also had a high error rate, but at a much higher cost: generating these for just 20 original instances took 1.5–2 hours of expert labor.

While high error rates could be achieved with unrelated or nonsensical examples, all counterfactuals under evaluation were close to the original examples, when measured by syntactic tree edit (≈1.0) or Levenshtein distance (≈0.2), POLYJUICE-*surprise* being the closest on both. An independent rater labeled 95% of POLYJUICE-*surprise* counterfactuals as "likely written by a native speaker," in contrast to 85% for *Expert-surprise*, indicating that experts sometimes resorted to ungrammatical or nonsensical sentences to find surprising behaviors.

Qualitatively, the study participants tended to create counterfactuals by perturbing the token with the highest weights (84% of their $\hat{x}$ perturbed tokens in the top 15% quantile of weights), not gaining a real understanding of how the other tokens impact predictions. Participants also made a significant number of mistakes even for tokens they had inspected, *e.g.,* a participant perturbed the example in Figure 3A by replacing help → play with, yielding a *Non-Duplicate* model prediction. When faced with help → find in Figure 3D, they incorrectly assumed the behavior would be the same.

These results indicate that POLYJUICE counterfactuals complement feature attribution explanations by displaying information that users often miss, even after they have manually explored the model behavior *beyond* explanations. Moreover, POLYJUICE counterfactuals for this application were more surprising and fluent than *Expert-surprise*, despite being computed automatically.

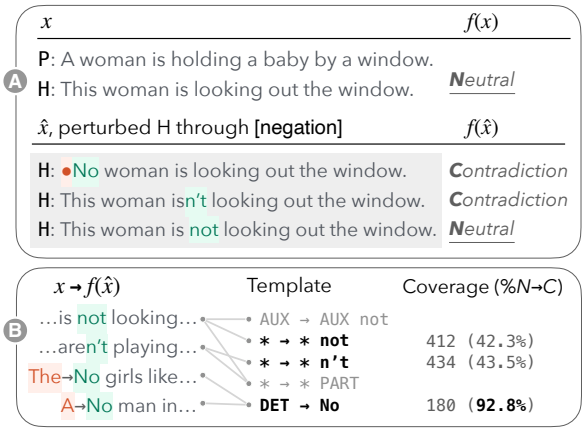

Figure 5: (A) An *NLI* case with a *Neutral* prediction (underlined $f(\hat{x})$ are correct). Polyjuice generates counterfactual hypotheses conditioned on the `negation` control code. (B) Generalizing perturbations into patterns (Wu et al., 2020). The change DET → no flips 92.8% of predictions from *N*eutral → *C*ontradiction.

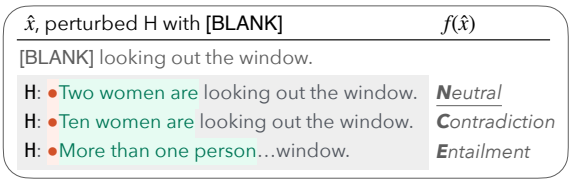

Figure 6: Perturbing the subject of x in Figure 5A through `[BLANK]`, resulting in erroneous predictions for different *quantifiers* (all should be *Neutral*).

5 Interactive Analysis

While our use of Polyjuice has so far relied on automatic selection of counterfactuals, we show in this section how an analyst can benefit from *multiple* counterfactuals per x, make use of controlled generation for more advanced analysis, and extract general patterns from individual observations. Our use case is counterfactual error analysis (Wu et al., 2019) of RoBERTa finetuned on *NLI* (used in §3.1), although the techniques are generally applicable.

There is a known correlation between the label *Contradiction* and hypotheses with negation in *NLI* datasets (Gururangan et al., 2018), which may cause models to fail on non-contradiction negations. We explore this in Figure 5A by generating counterfactual hypotheses for a random *Neutral* instance, conditioning only on the original x and the `negation` control code. While the first two counterfactuals display this failure mode, there is a surprising inconsistency in model behavior between "not" and "n't". We note that manual analysis may not explore these three negation forms, and thus not surface this puzzling behavior.

To verify if the pattern is widespread, we generate counterfactuals with the `negation` control code for a random set of instances correctly predicted as *Neutral* (n = 895). To generalize individual changes into patterns, we extract frequent *counterfactual templates* with Tempura (Wu et al., 2020) (details in Appendix D.2), shown in Figure 5B. The top templates (in bold) show that the model flips its prediction from *Neutral* to *Contradiction* with roughly the same frequency (≈43%) whether the negation word is "not" or "n't", but flips much more frequently with a different negation pattern where a determiner is replaced with "no" (92.8%). While these behaviors may be correct in some instances, they often are not (*e.g.,* Figure 5A), and thus would warrant further exploration, and potential mitigation strategies (*e.g.,* counterfactual training, §3). Tangentially, the impact of DET → no might lead the analyst to explore the impact of perturbing the *subject* of hypotheses, which we do in Figure 6 by placing a `[BLANK]` on the subject rather than using a control code. This leads to the discovery of unstable and erroneous behaviors regarding *quantifiers*, which we analyze in more detail in Appendix D.1.

Discussion. Polyjuice is a powerful tool for interactive analysis. Generating multiple counterfactuals per instance leads to insights that might be missed by manual analysis, and the steering provided by control codes and `[BLANK]`s allow for analyses that would be non-trivial to do manually (Wu et al., 2019) or with masked language models (*e.g.,* Figure 5B places negations in various parts of sentences, and Figure 6 replaces spans with other spans of varying lengths). Besides error analysis, an analogous interactive use of Polyjuice may be suitable for test creation (Ribeiro et al., 2020) and forms of data augmentation that are more controlled than what we presented in §3.

6 Related Work

Some prior work in training and evaluation relies on humans to generate counterfactuals from scratch (Gardner et al., 2020; Teney et al., 2020; Kaushik et al., 2020). Our experiments in §3 indicate that asking humans to *label* Polyjuice counterfactuals yields similar or better results at a lower cost, which motivates an exploration of a mixture of manual and semi-automated generation. Similarly, prior work on analysis relies on experts to

create individual counterfactuals or perturbation functions (Wu et al., 2019; Ribeiro et al., 2020). In §5, we show that POLYJUICE enhances current practice by generating multiple counterfactuals that might have been overlooked, and by providing abstractions that allow for new kinds of analyses.

Prior work on automatically generating counterfactuals typically has a narrower scope in terms of the relationships $x \nrightarrow \hat{x}$. For example, adversarial generators aim to maintain semantics while changing model predictions (Ribeiro et al., 2018; Iyyer et al., 2018; Li et al., 2021), whereas concurrent work to our own (Madaan et al., 2021; Ross et al., 2020) automatically generates $\hat{x}$ that change predictions for explanation or analysis, with no constraints on semantics. However, as shown in §3–§5, a *mix* of label-preserving and label-flipping counterfactuals generated by POLYJUICE is quite useful for training, evaluation, explanation, and analysis. Further, general-purpose counterfactuals may lead to serendipitous discoveries (§5), especially as POLYJUICE is not fine-tuned to the target domain (and thus less liable to merely replicate what is already there). Finally, by allowing control through control codes and `[BLANK]`s, POLYJUICE supports human-generator collaboration, where a person specifies desired changes (*e.g.,* perturb *the sentence subject*). Such collaboration is hard to imagine using automatic generators with no control, or with coarser control through predefined style attributes or labels (Madaan et al., 2020; Malmi et al., 2020). To our knowledge, prior work on controlled generation (Keskar et al., 2019; Dathathri et al., 2020) does not address *counterfactual* generation.

7 Conclusion and Future Work

We propose POLYJUICE, a general-purpose generator that produces fluent and diverse counterfactuals, allowing for control over the kinds and locations of perturbations. With simple, *task-specific* selection heuristics, POLYJUICE supports various downstream tasks on different domains, including counterfactual data augmentation, contrast set generation, counterfactual explanation, and error analysis.

While POLYJUICE is broadly applicable, it is not bias-free: control codes are pre-defined and certainly not exhaustive, and the model is fine-tuned on a collection of paired datasets where certain perturbations are more or less likely (*e.g.,* we observe that words with negative sentiment tend to be slightly more likely than positive ones in some contexts). Collecting naturally occurring counterfactuals is an important area of future research, as is the development of generators that allow for control even without *a-priori* control codes.

Besides improving the generators, further work is needed to improve the value of counterfactuals. For example, while POLYJUICE shows consistent gains across tasks in data augmentation, the improvements on some datasets are not as significant. This aligns with observations in prior work that even manual counterfactuals can be marginally beneficial (Kaushik et al., 2020; Huang et al., 2020), possibly because the original data is already diverse enough, or the perturbed signal in counterfactuals is too subtle to affect the model (*e.g.,* when only a single word is changed in a long sentence.) We hope to perform more thorough experiments on tuning the amount and the distribution of counterfactual augmentation, as well as other ways of incorporating counterfactuals, such as having explicit terms in the loss function for contrasting counterfactuals with original data (Teney et al., 2020), or other forms of contrastive learning.

Although our applications all involved people, the human-POLYJUICE collaboration in labeling and explanations could benefit from richer interaction mechanisms. We believe POLYJUICE motivates future research on more expressive forms of counterfactual training, where users generate counterfactuals together with POLYJUICE, and label counterfactual *patterns* rather than individual instances. Similarly, interactive explanations and analysis are exciting directions, especially as we develop new ways of selecting, presenting, and aggregating counterfactuals for various analysis objectives. Having noted these opportunities, we believe POLYJUICE is already a powerful tool for counterfactual reasoning, in particular for tasks where people are directly involved. POLYJUICE is opensource, and available at `https://github.com/tongshuangwu/polyjuice`.

Acknowledgements

The work was supported by ONR grant N00014-18-1-2193, NSF RAPID grant 2040196, NSF award IIS-1901386, the University of Washington WRF/Cable Professorship, and the Allen Institute for Artificial Intelligence (AI2). We thank Jim Chen, Dianqi Li, Scott Lundberg, Hao Peng, Sameer Singh, Jiao Sun, Victor Zhong, and Sitong Zhou for their helpful comments, as well as our user study participants for their valuable input.

Ethical Considerations

Our work includes labeling counterfactuals on crowdsourcing platforms, as well as conducting user studies with graduate students. As detailed in Appendix B.1 and C.2, we compensated the MTurk workers $2.5 for ≈15 minutes of labeling, and the graduate students $20 for the user study (one hour), above the U.S. federal minimum wage. The studies are with IRB approval.

We only finetune GPT-2 rather than training it from scratch, such that our compute costs are relatively low (around 8 hours for finetuning, Appendix A). All of our finetuning experiments involved finetuning RoBERTa on smaller datasets.

More critically, with most of our demonstrated applications using a human-generator hybrid mechanism, we stress that the interaction between the two deserves careful consideration. It has long been reported that algorithms interacting with humans can negatively impact the human.[9] In our case, the concern might be that users can develop an over-reliance on POLYJUICE (Bansal et al., 2021) and hastily accept its generations. Not only can this decrease users' creativity (Green et al., 2014), but it may bias their analysis process: as discussed in §7, POLYJUICE generation is not exhaustive, and may favor some perturbation patterns over others in unpredictable ways. In the short term, we plan to highlight these limitations as part of the model documentation, while future research should identify interaction mechanisms, so as to ensure that POLYJUICE or other counterfactual generators support humans, rather than hindering their performance.

[9] https://www.nytimes.com/interactive/2017/04/02/technology/uber-drivers-psychological-tricks.html?_r=0

References

Nabiha Asghar. 2016. Yelp dataset challenge: Review rating prediction. *arXiv preprint arXiv:1605.05362*.

Gagan Bansal, Tongshuang Wu, Joyce Zhou, Raymond Fok, Besmira Nushi, Ece Kamar, Marco Tulio Ribeiro, and Daniel Weld. 2021. Does the whole exceed its parts? the effect of ai explanations on complementary team performance. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA. Association for Computing Machinery.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Vincent S. Chen, Sen Wu, Alexander J. Ratner, Jen Weng, and Christopher Ré. 2019. Slice-based learning: A programming model for residual learning in critical data slices. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9392–9402.

Noam Chomsky. 2002. *Syntactic structures*. Walter de Gruyter.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Chris Donahue, Mina Lee, and Percy Liang. 2020. Enabling language models to fill in the blanks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2492–2501, Online. Association for Computational Linguistics.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. 2020. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1307–1323, Online. Association for Computational Linguistics.

Sahaj Garg, Vincent Perot, Nicole Limtiaco, Ankur Taly, Ed H Chi, and Alex Beutel. 2019. Counterfactual fairness in text classification through robustness. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 219–226.

Max Glockner, Vered Shwartz, and Yoav Goldberg. 2018. Breaking NLI systems with sentences that require simple lexical inferences. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 650–655, Melbourne, Australia. Association for Computational Linguistics.

Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009.

Spence Green, Sida I. Wang, Jason Chuang, Jeffrey Heer, Sebastian Schuster, and Christopher D. Manning. 2014. Human effort and machine learnability in computer aided translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1225–1236, Doha, Qatar. Association for Computational Linguistics.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 107–112, New Orleans, Louisiana. Association for Computational Linguistics.

Peter Hase and Mohit Bansal. 2020. Evaluating explainable AI: Which algorithmic explanations help users predict model behavior? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5540–5552, Online. Association for Computational Linguistics.

William Huang, Haokun Liu, and Samuel R. Bowman. 2020. Counterfactually-augmented SNLI training data does not yield better generalization than unaugmented data. In *Proceedings of the First Workshop on Insights from Negative Results in NLP*, pages 82–87, Online. Association for Computational Linguistics.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1875–1885, New Orleans, Louisiana. Association for Computational Linguistics.

Daniel Kahneman and Amos Tversky. 1981. The simulation heuristic. Technical report, Stanford Univ CA Dept of Psychology.

Divyansh Kaushik, Eduard H. Hovy, and Zachary Chase Lipton. 2020. Learning the difference that makes A difference with counterfactually-augmented data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Nitish Shirish Keskar, Bryan McCann, Lav Varshney, Caiming Xiong, and Richard Socher. 2019. CTRL - A Conditional Transformer Language Model for Controllable Generation. *arXiv preprint arXiv:1909.05858*.

Daniel Khashabi, Tushar Khot, and Ashish Sabharwal. 2020. More bang for your buck: Natural perturbation for robust question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 163–170, Online. Association for Computational Linguistics.

Najoung Kim, Roma Patel, Adam Poliak, Patrick Xia, Alex Wang, Tom McCoy, Ian Tenney, Alexis Ross, Tal Linzen, Benjamin Van Durme, Samuel R. Bowman, and Ellie Pavlick. 2019. Probing what different NLP tasks teach machines about function word comprehension. In *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM 2019)*, pages 235–249, Minneapolis, Minnesota. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

VI Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707.

Dianqi Li, Yizhe Zhang, Hao Peng, Liqun Chen, Chris Brockett, Ming-Ting Sun, and Bill Dolan. 2021. Contextualized perturbation for textual adversarial attack. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5053–5069, Online. Association for Computational Linguistics.

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. 2020. CommonGen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1823–1840, 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*.

Shayne Longpre, Yu Wang, and Chris DuBois. 2020. How effective is task-agnostic data augmentation for pretrained transformers? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4401–4411, Online. Association for Computational Linguistics.

Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4765–4774.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Aman Madaan, Amrith Setlur, Tanmay Parekh, Barnabas Poczos, Graham Neubig, Yiming Yang, Ruslan Salakhutdinov, Alan W Black, and Shrimai Prabhumoye. 2020. Politeness transfer: A tag and generate approach. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1869–1881, Online. Association for Computational Linguistics.

Nishtha Madaan, Inkit Padhi, Naveen Panwar, and Diptikalyan Saha. 2021. Generate your counterfactuals: Towards controlled counterfactual generation for text. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Eric Malmi, Aliaksei Severyn, and Sascha Rothe. 2020. Unsupervised text style transfer with padded masked language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8671–8680, Online. Association for Computational Linguistics.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.

Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1 – 38.

John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. TextAttack: A framework for adversarial attacks, data augmentation, and adversarial training in NLP. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 119–126, Online. Association for Computational Linguistics.

Aakanksha Naik, Abhilasha Ravichander, Norman Sadeh, Carolyn Rose, and Graham Neubig. 2018. Stress test evaluation for natural language inference. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2340–2353, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. 2013. SemEval-2013 task 2: Sentiment analysis in Twitter. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 312–320, Atlanta, Georgia, USA. Association for Computational Linguistics.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. 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 188–197, Hong Kong, China. Association for Computational Linguistics.

Judea Pearl. 2018. Causal and counterfactual inference. *The Handbook of Rationality*, pages 1–41.

Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C. Lipton. 2020. Learning to deceive with attention-based explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4782–4793, Online. Association for Computational Linguistics.

Lianhui Qin, Antoine Bosselut, Ari Holtzman, Chandra Bhagavatula, Elizabeth Clark, and Yejin Choi. 2019. Counterfactual story reasoning and 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 5043–5053, Hong Kong, China. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. 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 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should I trust you?": Explain-

ing the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144. ACM.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically equivalent adversarial rules for debugging NLP models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 856–865, Melbourne, Australia. Association for Computational Linguistics.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online. Association for Computational Linguistics.

Alexis Ross, Ana Marasović, and Matthew E. Peters. 2020. Explaining nlp models via minimal contrastive editing (mice).

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Damien Teney, Ehsan Abbasnedjad, and Anton van den Hengel. 2020. Learning what makes a difference from counterfactual examples and gradient supervision. In *Computer Vision – ECCV 2020*, pages 580–599, Cham. Springer International Publishing.

Vijay V Vazirani. 2013. *Approximation algorithms*. Springer Science & Business Media.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. 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 11–20, Hong Kong, China. Association for Computational Linguistics.

John Wieting and Kevin Gimpel. 2018. ParaNMT-50M: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 451–462, Melbourne, Australia. Association for Computational Linguistics.

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.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2019. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 747–763, Florence, Italy. Association for Computational Linguistics.

Tongshuang Wu, Kanit Wongsuphasawat, Donghao Ren, Kayur Patel, and Chris DuBois. 2020. Tempura: Query analysis with structural templates. In *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, pages 1–12. ACM.

Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. 2019a. Generating fluent adversarial examples for natural languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5564–5569, Florence, Italy. Association for Computational Linguistics.

Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262.

Yuan Zhang, Jason Baldridge, and Luheng He. 2019b. PAWS: Paraphrase adversaries from word scrambling. 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 1298–1308, Minneapolis, Minnesota. Association for Computational Linguistics.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texygen: A benchmarking platform for text generation models. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 1097–1100. ACM.

Dataset	negation	quantifier	lexical	resemantic	insert	delete	restructure	shuffle	*global*
CAD	3,274	292	8,143	2,603	960	952	220	36	3,466
Contrast	336	436	1,607	1,291	589	586	275	149	877
HANS	50	0	0	0	3,926	3,926	494	1,602	2
ParaNMT	2,797	825	10,000	10000	6,442	6,205	5,136	1,417	10,000
PAWS	81	1,815	10,000	10000	3,630	3,403	4,551	10,000	10,000
WinoGrande	3,011	94	10,000	6,927	120	124	453	65	3184
Crawled	0	0	5,000	0	5,000	5,000	0	108	5,000
Total	9,549	3,462	44,750	30,821	20,667	20,167	11,129	13,377	32,529

Table 7: The datasets used for finetuning POLYJUICE, and the control code distributions.

A GPT-2 as Counterfactual Generator

A.1 Training Data and Parameters

We combine several datasets to finetune POLYJUICE.

Contrast set. Authors of 10 existing NLP dataset each manually perturbed 100–1,000 instances to change the gold label, so to inspect a model's local decision boundary (Gardner et al., 2020). The perturbation patterns vary based on the tasks and the annotators, allowing us to learn diverse strategies. To make sure we can use the contrast set to evaluate the *Sentiment* model, we excluded the IMDb movie review from the training.

Counterfactually-augmented data (CAD). Kaushik et al. (2020) crowdsourced counterfactuals for IMDb movie review (1.7k), which we split into paired sentences to match the text length of other datasets. CAD's perturbation patterns also vary based on the task, but can especially contribute to `negation`. As NLI is in our demonstrating applications, we did not use their 6.6k SNLI counterfactuals.[10]

WinoGrande is a large-scale dataset of 44k instances for testing common sense problems (Sakaguchi et al., 2020). It contains sentences that differ only by one trigger word (*e.g.,* one noun), making it most suitable for learning lexical exchanges.

ParaNMT-50M contains 50 million English-English sentential paraphrase pairs, covering various domains and styles of text, as well as different sentence structures (Wieting and Gimpel, 2018).

PAWS (Zhang et al., 2019b) contains pairs with high text overlaps, created through controlled word swapping, best demonstrating `shuffle` and `restructure`. We used its 49k Wikipedia parts.

HANS (McCoy et al., 2019), a challenge set for NLI, contains 10k pairs of premises and hypotheses created based on 10 heavily fallible syntactic templates, and therefore compensates rarer structural changes that may be missed by PAWS.

[10]Similarly, though *QQP* is suitable for training POLYJUICE, we omitted it so *QQP* can be used in our evaluation.

Crawled We additionally crawl naturally occurring sentence pairs from non-paired datasets boost some specific patterns and increase lexical diversity. This include (1) CommonGen (Lin et al., 2020), sentences with common sense concepts; (2) Natural Questions (Kwiatkowski et al., 2019), collections of queries issued to Google Engines (and therefore involve various paraphrases of similar user intents), and (3) SQuAD (Rajpurkar et al., 2016), whose paragraphs involve Wikipedia knowledge. We estimate *close* pairs using edit distance, and broadly accept those with less than 60% editing. To exclude tricky cases (*e.g.,*"how do I not be" can be incorrectly regarded as `negation` for "how do I recover it"), we only augment the most determined patterns: `lexical`, `insert`, `delete`, and `shuffle`.

To balance the distribution (Table 7), for each dataset, we extract control codes from all the $(x, \hat{x})$, and randomly sample up to 10,000 instances per codes. Still, `quantifier` and `negation` have less training data compared to other codes. Fortunately, these codes tend to be limited to more specific patterns ("more than", "not", "never") when compared to "broad" codes like `lexical`, and thus even a small sample is enough to learn them. We finetuned an off-the-shelf GPT-2 model from Wolf et al. (2020) for 10 epochs with an initial learning rate 5e-5, a batch size of 8, and a sequence length of 120 (but any LM can potentially be used). We select the best epoch based on the evaluation loss on a holdout set of size 5,000. The training took around 8 hours on two Titan RTXs.

A.2 Intrinsic Evaluation Details

A.2.1 Closeness and Diversity

Similar to Madaan et al. (2021), we compare the *diversity* and *closeness* of POLYJUICE with alternative generators, *i.e.,* RoBERTa and T5, representing masked language models that prioritize word and span substitution, and original *GPT-2*, representing the standard generative model not conditioned on x. For a given x and its counterfactuals $\hat{\mathbf{X}}$, we approx-

imate *diversity* using self-BLEU (Zhu et al., 2018) within $\hat{\mathbf{X}}$. Meanwhile, *closeness* is the average distance between x and every $\hat{x} \in \hat{\mathbf{X}}$, both with the normalized word level Levenshtein edit distance ((Levenshtein, 1966), used in MiCE (Ross et al., 2020)), and syntactic tree edit distance ((Zhang and Shasha, 1989) in GYC (Madaan et al., 2021)).

We run the three generators on 300 sentences in total. In GPT-2, we take the first two words of an x as the input context (prompt), limit the length of the generation to be similar to x, and collect 10 counterfactuals. As for RoBERTa and T5, we repeatedly perturb x for three times, each time randomly placing up to three `[MASK]` tokens, and ask the generator to generate 5 counterfactuals through beam search, following Ribeiro et al. (2020). POLYJUICE uses the same blank (mask) placement as in RoBERTa and T5, but we additionally enumerate through all control codes. For each x, we randomly sample 5 counterfactuals to form $\hat{\mathbf{X}}$ per generator.

As shown in Table 2, POLYJUICE achieves a balance between diversity and closeness. Ideally, we would also like to compare POLYJUICE with concurrent work (Madaan et al., 2021; Ross et al., 2020), but these are yet to be open-sourced and require extensive implementation or finetuning.

A.2.2 Controllability

To evaluate controllability, we compare POLYJUICE with T5, and GPT-2 finetuned on prompts *without* codes (called POLYJUICE *-a*), such that both baselines consider sufficient context. For each control code, we compare the *control success rate* of POLYJUICE and POLYJUICE-a on 300 prompts. For each prompt, we generate counterfactuals through beam search (beam = 5), and recompute the codes on the top three generated $\hat{x}$. We deem the control successful if at least one of the three recomputed codes matches the desired control code (though in POLYJUICE-a, we only measure whether the code naturally occurs in the uncontrolled generation.) The success rate increases by 26% ± 13% across all control codes, ranging from `quantifier` (increasing 6%, from 50% to 56%) to `negation` (42%, from 5% to 47%). Non-finetuned T5 also achieves less control (success rate decreases by 33% on average.)

Common failure cases include (1) The control codes conflict with the blanks, *e.g.,*"a dog is embraced by a `[BLANK]`" would not respond to `negation`. (2) x does not have a corresponding pattern, *e.g.,* `shuffle` is not applicable to "the movie is good." (3) certain salient patterns dominate the generation probability, *e.g.,* the model tends to perturb the quantifier "two" in "two dogs are running," regardless of the code.

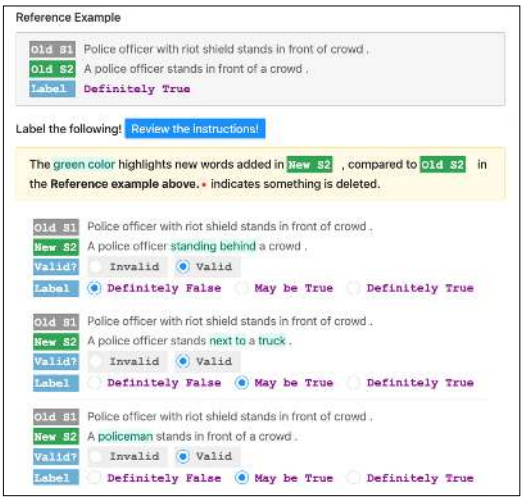

Figure 7: A sample labeling task: The crowdworkers annotate three counterfactuals based on their validity and class label, with respect to the original instance.

B Additional Train & Eval Details, §3

B.1 MTurk Labeling Details

Procedure The study started with an introduction that explained the context and tasks. To familiarize crowdworkers with the task, we asked them to complete 1-2 training rounds, and explained the expected labels. Each annotator then completed 22 tasks, labeling 3 counterfactuals of a single example in each round, as in Figure 7. The 22 rounds consisted of 20 actual labeling tasks and 2 extra "gold rounds" with known correct labels. The gold cases later served to filter low-quality crowdworkers. The median annotation time was around 15 minutes, and participants received $2.5.

Participants. We recruited participants from MTurk, limiting the pool to subjects from within the US with a prior task approval rating of at least 97% and a minimum of 1,000 approved tasks.

Data quality. We applied two filtering strategies: (1) *High-quality worker.* We only kept data from participants whose median labeling time per round was more than 18 seconds and correctly labeled at least 4 gold counterfactuals (out of 6), or who correctly labeled all gold ones. (2) *Majority vote labeling.* We collected two annotations per counterfactual, and only kept those that at least one annotator deemed valid, and both annotators agreed on a particular class label. One of the authors la-

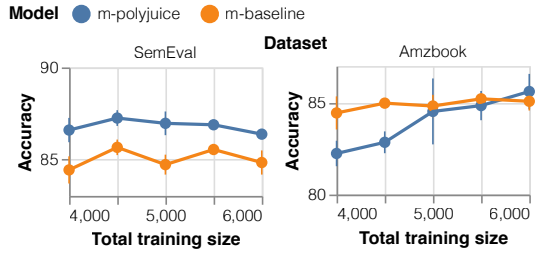

Figure 8: The accuracy trend on two *Sentiment* datasets, as the total training datasize (m+n) varies. The blue line shows an augmentation of $m = 2k$ counterfactuals, and the blue one represents the corresponding `m-baseline`. Though the counterfactuals remains useful on datasets like SemEval across all m+n, it appears too many counterfactuals may be harmful (Amzbook).

beled a subset of 100 $\hat{x}$ on 100 x in *Sentiment*, and reached high agreement with the majority-voted results ($\kappa = 0.77$, raw labeling agreement 88%).

B.2 Training Details & m/n Ratios, for §3.2

For each (m, n), we created three samples of training data. Each sample was further averaged over four random seeds. For each run, we heuristically picked the initial learning rates 1e-5, 2e-5, 2e-5 for *Sentiment*, *NLI* and *QQP*, and trained 20 epochs with a dropout rate of 0.1 and a batch size of 16. We selected the epoch that had the highest accuracy on the corresponding validation set, which takes 1/5 of the training data size, with the same ratio of m/n counterfactual and original examples.

We further explore ratios of added counterfactuals. Take *Sentiment* as an example: while the counterfactual remains effective on most datasets, it hurts the model performance on Amzbook when the counterfactual takes a large proportion (Figure 8, Yelp followed a similar but more mild trend). We suspect that flipping out too much original data affects the data diversity, and in turn decreases the model performance. Similarly, Huang et al. (2020) asserted that augmenting $n = 1.7k$ NLI data with $m = 6.6k$ counterfactuals did not improve model generalization accuracy.

C Additional Explanation Details §4

C.1 Selection Methods

Because SHAP weights reflect the average effect of masking a token t, we also focus on *word features that are abnormal on average*.

More concretely, we define the expected change-in-prediction for perturbing a token t to be the SHAP importance on it, $\mathbf{H}[\mathrm{D_f}(t, x)] = s(t)$. In Figure 3, $s(t$=depression$) = 0.276$. The actual prediction change $\mathrm{D_f}(t, x)$ is the weighted average of $|\mathrm{f_p}(x) - \mathrm{f_p}(\hat{x})|$ for all the $\hat{x}$ that affect t (depression → trouble, depression → a mood), where $\mathrm{f_p}(x)$ is the prediction probability of f on x. The weight corresponds to the number of words modified in $\hat{x}$: If $e(\hat{x})$ denotes the set of edited words in x, then $w(\hat{x}) = 1/|e(\hat{x})|$. Intuitively, the more words changed in $\hat{x}$, the less impact each word has; In Figure 3D, we regard "depression" to be responsible for half of the impact in in depression → suicidal. We group $\hat{x}$ based on their affected words $G_t = \{\hat{x} \mid t \in e(\hat{x})\}$. $\mathrm{D_f}(t, x)$ then becomes:

$$\mathrm{D_f}(t, x) = \frac{1}{|G_t| + 1}\left(s(t) + \sum_{\hat{x} \in G_t} w(t) \cdot |\mathrm{f_p}(x) - \mathrm{f_p}(\hat{x})|\right)$$

The additional SHAP weight $s(t)$ acts as a smoothing factor to penalize outliers. Then the gap between the expectation and reality is:

$$\Delta\,\mathrm{D_f}(t, x) = \mathrm{D_f}(t, x) - \mathbf{H}[\mathrm{D_f}(t, x)]$$

We first find the abnormal tokens: (1) t with small SHAP weight, but $\hat{x}$ that change t experience large prediction change on average: $t_L = \arg\max_{t \in x} \Delta\,\mathrm{D_f}(t, x)$, and (2) t with large SHAP weight, but $\hat{x}$ with t changed usually have intact prediction: $t_U = \arg\max_{t \in x} -\Delta\,\mathrm{D_f}(t, x)$.

Then, we use the most extreme cases within the groups of G_{t_L} and G_{t_U} as the concrete counterfactual explanations, based on their prediction change $|\mathrm{f_p}(x) - \mathrm{f_p}(\hat{x})|$, and the aggregated SHAP weights of all the changed tokens:

$$\hat{x}_L = \arg\max_{\hat{x} \in G_{t_L}}\left(|f_p(x) - f_p(\hat{x})| - \sum_{u \in r(\hat{x})} s(u)\right)$$

C.2 User Study Details

Figure 9 shows the sample interface. Participants started by just seeing the reference example and the model query box on the left hand side. When they chose to start the task or after they had exhausted their ten query chances, the query box was disabled, the tasks on the right were displayed, and the participants completed the tasks. We compensated participants $20 for the one hour study.

D Additional Err. Analysis Details §5

D.1 Additional Case Study: Quantifiers

As a follow-up to Figure 6, we slice the data to find *entailment* instances that have numbers in the hypothesis sentence, and perturb their `quantifiers`.

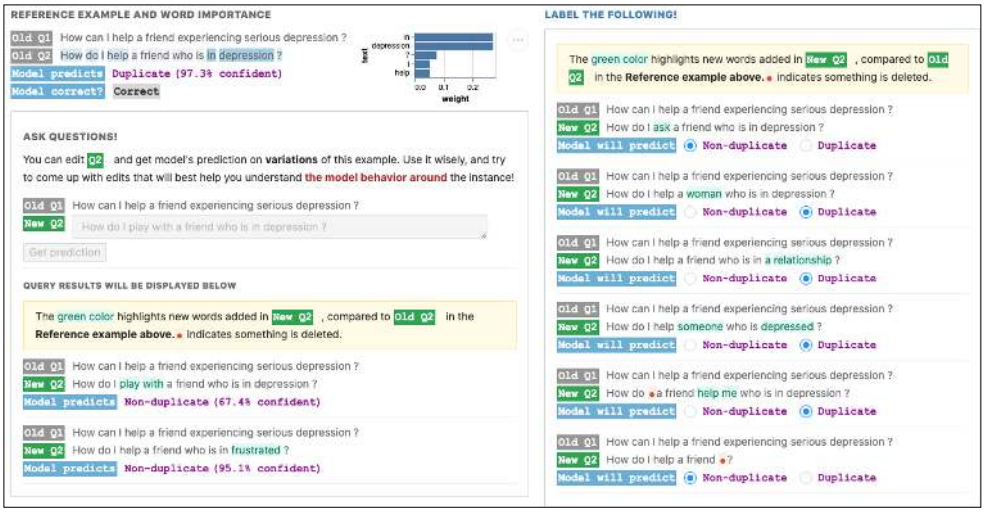

Figure 9: A sample explanation task for §4

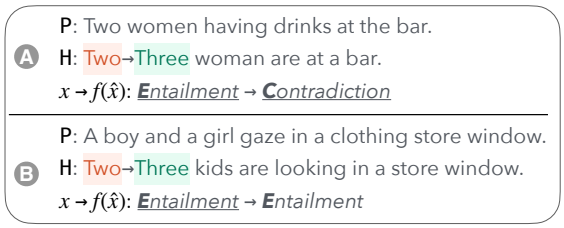

Figure 10: The *NLI* model cannot perform the actual counting when the exact number is missing from P.

The extracted templates show that the model does not perform actual counting. When changing one number to another (NUM → NUM), the model only flips the label in 64.7% cases, while we would expect all cases to be like in Figure 10A. An inspection of instances indicates the model gets confused when the premise does not contain the same number explicitly. Indeed, when we filter for such instances (e.g. Figure 10B), the label flip rate of NUM → NUM is lowered to 30.2%.

Further, the model only reacts to *some* quantifier phrase modifiers. +at least ("at least two women are at a bar") will always still result in *entailment*, prediction, +only and +exactly flip the predicted label to *neutral* 90% of the time ("exactly two women are at a bar"), but the model only changes the prediction 52.6% of the time when we add +more than ("more than two women are at a bar").

D.2 Representative Perturbation Templates

Similar to Wu et al. (2020), the process of finding representative perturbation patterns takes two steps:

Extract template. For each $\hat{x}$, we compare it with its x, and translate the perturbed spans into templates using different combinations of texts, lemmas, sparse and fine-grained part-of-speech tags. We optionally include surrounding contexts determined by the dependency tree structure (tokens that share the same parents as the perturbed span). For example, "is not reading" can result in templates t as fine-grained as is reading → is not reading, or as sparse as +PART. Meanwhile, "are not playing" also translates to +PART or +not, but not is reading → is not reading. As such, the $\hat{x}$ and templates form a many-to-many relationship: each $\hat{x}$ generates multiple templates, and each template covers a different group of $\hat{x}$.

Select Representative Templates. To find representative changes, we prefer (1) templates that cover a large number of $\hat{x}$. Meanwhile, to avoid overfitting to one instance (*e.g.,* extracting a template red → ADJ only because "red" is repeatedly perturbed in one x), we prefer (2) templates that perturb various unique x. We also prefer (3) finer-grained templates, to avoid being unnecessarily abstract (*e.g.,* to avoid abstracting "not" when it is the only PART changed.)

With these intuitions, we form the template selection as a weighted set coverage problem. We see the union of counterfactuals for each x, $\hat{\mathbf{X}}$, as the entire set of elements. Then, each template $t \in T = t_1, ..., t_m$ represents a subset of $\hat{\mathbf{X}}$ that contains a number of counterfactuals $|t|$. We define the weight as $w(t) = g(t)/|t|_x$, where $|t|_x$ quantifies the unique original x covered by t, and $g(t)$ represents the sparsity of t (heuristically decreasing from text to POS). This way, templates that are too abstract or too focused on a certain x are penalized by having a high weight. We use a classic greedy algorithm (Vazirani, 2013) to select a subset of $T^* \subset T$, such that the aggregated coverage is maximized, and the weight is minimized.