Counterfactual Sentence Generation with Plug-and-Play Perturbation

***Abstract*—Generating counterfactual test-cases is an important backbone for testing NLP models and making them as robust and reliable as traditional software. In generating the test-cases, a desired property is the ability to control the test-case generation in a flexible manner to test for a large variety of failure cases and to explain and repair them in a targeted manner. In this direction, significant progress has been made in the prior works by manually writing rules for generating controlled counterfactuals. However, this approach requires heavy manual supervision and lacks the flexibility to easily introduce new controls. Motivated by the impressive flexibility of the plug-and-play approach of PPLM, we propose bringing the framework of plug-and-play to counterfactual test case generation task. We introduce CASPer, a plug-and-play counterfactual generation framework to generate test cases that satisfy goal attributes on demand. Our plug-and-play model can steer the test case generation process given any attribute model without requiring attribute-specific training of the model. In experiments, we show that CASPer effectively generates counterfactual text that follow the steering provided by an attribute model while also being fluent, diverse and preserving the original content. We also show that the generated counterfactuals from CASPer can be used for augmenting the training data and thereby fixing and making the test model more robust.**

I. Introduction

Machine learning and deep learning-based decision making has become part of today's software. This creates the need to ensure that machine learning and deep learning-based systems are as trusted as traditional software with increased deployment and wider-use. Traditional software is made dependable by following rigorous practice like static analysis, testing, debugging, verifying, and repairing throughout the development and maintenance life-cycle. Similarly, for testing and repairing NLP systems, we need inputs where models can fail and thereby bringing out issues early on [4], [5]. For this, counterfactual text data [6], [7] can be used. By treating counterfactual text as test cases, we are asking: *Would the model fail if the input text was modified to have different characteristics*? Furthermore, with such counterfactual text, NLP systems can be repaired by augmenting the training samples with these counterfactual test cases and its labels [8]. Hence, enabling model repair by generating counterfactual text is a crucial step in deploying these NLP systems more widely.

An important aspect of model testing and repair is to ensure that we can *control* these counterfactual test cases. The ability to control will allow us to test for specific types of failures that are important for the deployed model. Controlled counterfactuals can also allow us to fix the failures by creating new training samples in a focused manner for augmenting the existing training dataset. Thus we require a model that can generate counterfactuals that can be controlled by providing some goal attributes.

In addition to controlling the test-cases, we would also like to have *flexibility* about which goal attributes to apply and the flexibility to chain together multiple goal attributes in order to test how the deployed model behaves for a wide variety of textual characteristics. Bringing such flexibility requires a model that allows us to plug-and-play new attribute goals as and when required.

In this work, we propose a framework for counterfactual test-case generation also called *Counterfactual Sentence Generation with Plug-and-Play Perturbation* or CASPer that provides both control and flexibility during test case generation. To achieve these, we build on the framework of Plug-and-Play Language Models or PPLMs [9]. PPLMs have shown an impressive ability to flexibly steer pre-trained language models to generate attribute-conditioned text samples. However, PPLMs cannot be directly used for perturbing an input text and generating counterfactual samples. In this work, instead of steering language models, we steer a text-to-text model that is pre-trained to reconstruct its text inputs and we steer this model in a plug-and-play manner. Thus, like PPLM, our model is plug-and-play and is capable of generating counterfactuals for any arbitrary goal attributes provided at sampling time. In CASPer, we take BART [10] as our text-to-text reconstruction model. To generate each counterfactual, we perturb the hidden layer of the BART model similarly to PPLM to sample the counterfactual text. In experiments, we apply our framework to generate counterfactuals by providing named-entity and sentiment-based goal attributes. We show that our simple plug-and-play framework can generate counterfactuals that are fluent and content-preserving while also attaining the goal attributes and being effective as training samples for data-augmentation to improve the performance of the deployed model. We show counterfactual samples generated by CASPer and from existing models in Table I.

The main contribution of the paper can be seen as three folds: 1) We propose, CASPer, the first plug-and-play counterfactual generation model that achieves both control and flexibility. 2) Empirically, we show that our approach can generate fluent counterfactuals that preserve the content and also attain the goal attribute. 3) We also show the effectiveness of our counterfactuals as new training data in making the test models robust.

Inputs	**Token-based** [1]	**Adversarial** [2]	**Polyjuice** [3]	**Ours**
Input Text: Me and a group of friends rent horrible videos to laugh at them, trust me it has lead to some horribly spent money but also some great laughs. *Initial State* : No location named-entity is present in the text. *Steering Goal*: To make the sentence contain at least one **location** named-entity.	Me and a group of friends rent **youtube** videos to laugh at them, trust me it has lead to some horribly spent money but also some great laughs. Me and a group of friends rent **music** videos to laugh at them, trust me it has lead to some horribly spent money but also some great laughs.	Me and a group of friends rent horrible videos to laugh at them , trust me it has lead to some horribly spent money but also some flavorful laughs .	Me and a group of my friends rent horrible videos to laugh at them , trust me it has lead to some horribly spent money but also some **flavorful** laughs. and a group of friends rent videos to laugh at them, trust me it has lead to some horribly spent money but also some **chuckles**.	Me have a group of lads in **Brisbane** and rent horrible videos to get great laughs at. Some extremely expensive videos but some very great.. Me and a group of **Fairfax Bay** friends like to rent videos for laughs. Have spent some money but had great laughs at their terrible videos.

TABLE I
OVERVIEW OF GENERATIONS FROM THE EXISTING MODELS. WE PROVIDE A TEXT AS INPUT TO THE MODEL WITH A STEERING GOAL OF INTRODUCING A *location named-entity* INTO THE GIVEN TEXT. WE SHOW THE OUTPUTS FROM A TOKEN-BASED SUBSTITUTION MODEL [1], FROM ADVERSARIAL GENERATION [2], FROM POLYJUICE [3] AND FROM OUR PROPOSED MODEL. WE NOTE THAT TOKEN-BASED SUBSTITUTION METHOD, RELYING ON TEMPLATE MATCHING, FAIL TO MATCH A TEMPLATE AND ARE THUS NOT ABLE TO ACHIEVE THE STEERING GOAL. ADVERSARIAL, DUE TO ITS GRADIENT-DESCENT-BASED TOKEN-SUBSTITUTION, FAILS TO GENERATE PLAUSIBLE TEXT. POLYJUICE, DUE TO ITS TEMPLATE MATCHING, CHANGES VERY INSIGNIFICANT PART OF THE TEXT. OUR MODEL, TAKING ADVANTAGE OF BART AUTO-ENCODER, EFFECTIVELY ACHIEVES THE STEERING GOAL.

Notation	Description
$\mathbf{x}$	Input text
$\mathbf{y}$	Perturbed text
$\mathbf{a}$	Steering target
t	Word index
H_t	Hidden state before applying LM head
o_t	Generated logits for word index t
$\mathbf{y}_t$	Generated token at index t
D_{KL}	KL divergence
$\mathbf{e}$	BERT's encoding output

TABLE II
A SUMMARY OF NOTATIONS USED IN THE PAPER.

II. PRELIMINARIES

A. Counterfactual Text Generation for Model Testing and Repair

Taking a text $\mathbf{x}$ from the input distribution and modifying it $\mathbf{x} \rightarrow \mathbf{y}$ is known as the task of counterfactual text generation. However, our goal of counterfactual text generation is to help improve NLP models by using counterfactual text as test-cases in various stages of the model deployment. For such models, we would like to *i)* test for failures, *ii)* explain when those failures occur and *iii)* fix the failures by augmenting the training data with new training samples.

However, from the perspective of model repair, it is not enough to simply obtain uncontrolled and random perturbations $\mathbf{x} \rightarrow \mathbf{y}$ to generate these test-cases. We would like these test-cases to be controlled $\mathbf{x} \xrightarrow{\texttt{control}} \mathbf{y}$ through a given `control` input. By controlling the generated test cases, we would be able to *i)* test for specific types of failures that are important for the deployed model, *ii)* explain which controls lead to high failure and *iii)* create targeted data sets for augmenting the training set and fixing the models. Hence, our work lies at the intersection of counterfactual text generation and controlled text generation. In the following subsection, we provide an overview of controlled text generation.

B. Controlled Text Generation

The goal of controlled text generation is to generate samples $\mathbf{x}$ from a controlled distribution $p(\mathbf{y}|\mathbf{a})$ which is conditioned on a specific attribute or control $\mathbf{a}$. For example, the language model $p(\mathbf{y}|\mathbf{a})$ may be used to generate product reviews conditioned on a specific product category by setting $\mathbf{a} = \texttt{kitchen}$.

1) Plug-and-Play Language Models: Plug-and-Play Language Models (or simply PPLMs) provide an attractive solution to model the class-conditional distribution $p_\theta(\mathbf{y}|\mathbf{a})$. Plug-and-play models take a pre-trained unconditional generative model $p_\theta(\mathbf{y})$ and use the reward signal from the attribute model $p(\mathbf{a}|\mathbf{y})$ to quickly (in $\sim$10 gradient steps) modify the unconditional generative model $p_\theta(\mathbf{y})$ to generate samples from the desired distribution $p_\theta(\mathbf{y}|\mathbf{a})$.

To achieve this, PPLMs [9] take GPT-2 to be the unconditional generative model $p_\theta(\mathbf{y})$. In GPT-2, the text generation is done iteratively word-by-word. In each iteration t, one word is predicted and is fed back to the Transformer to predict the next word. This generation process can be described as follows:

$$\begin{aligned}H_t &= \text{Transformer}(\mathbf{y}_{<t}),\\ \mathbf{o}_t &= \text{PredictionHead}(H_t),\\ \mathbf{y}_t &\sim \text{Categorical}(\mathbf{o}_t).\end{aligned}$$

where t is the word position in the text, H_t is the last hidden layer before the prediction head and $\mathbf{o}_t$ are the log-probabilities of the words in the vocabulary used for sampling the next word $\mathbf{y}_t$. We shall refer to this model as the unmodified language model and denote the distribution that it models for the next word prediction as $p(\mathbf{y}_t|\mathbf{y}_{<t})$.

To generate a text from $p_\theta(\mathbf{y}|\mathbf{a})$ at test time, PPLMs learn a perturbation for the hidden state H_t of the unconditional model $p_\theta(\mathbf{y})$. This is achieved as follows:

$$\begin{aligned}H_t &= \text{Transformer}(\mathbf{y}_{<t}),\\ \mathbf{o}_t &= \text{PredictionHead}(H_t + \Delta H_t),\\ \mathbf{y}_t &\sim \text{Categorical}(\mathbf{o}_t).\end{aligned}$$

where ΔH_t is the learned perturbation. We shall refer to this model as the modified language model and denote the distribution that it models for the next word prediction as $\bar{p}(\mathbf{y}_t|\mathbf{y}_{<t})$. The learning of the perturbation parameters $\{\Delta H_1, \dots, \Delta H_T\}$ is driven by the following objective:

$$\begin{aligned}\mathcal{L}_{\text{PPLM}} = &-\log p(\mathbf{a}|\mathbf{y})\\ &- \sum_{t=1}^{T} D_{\text{KL}}(p(\mathbf{y}_t|\mathbf{y}_{<t})||\bar{p}(\mathbf{y}_t|\mathbf{y}_{<t})),\end{aligned}$$

where the first term provides the learning signal to steer the generation towards the desired class or attribute by trying to maximize the log-probability of the desired attribute. The second term tries to keep the generations close to the unmodified language model to ensure that the text remains fluent and plausible. We note that this learning process is done separately each time we need to generate a new sample. However, the learning of the perturbation parameters $\{\Delta H_1, \dots, \Delta H_T\}$ can be done very quickly and it only takes about 10 gradient steps. This property makes PPLMs flexible during generation.

PPLMs provide some useful properties that are lacking in other conditional generative models that are not plug-and-play. Plug-and-play models are *flexible* during sampling – meaning that new class-conditioning can be easily introduced at test time by simply replacing the attribute model $p(\mathbf{a}|\mathbf{y})$ with a new attribute model for the new class without requiring costly retraining with respect to the new attribute. Furthermore, plug-and-play models can support conditioning on logical clauses by simply composing multiple attribute models together. For instance, to generate product review text conditioned on a logical clause `kitchen + electronics + not electrical`, the attribute model $p(\mathbf{a}|\mathbf{y})$ can be written as the product of individual attribute models $p(\texttt{kitchen}|\mathbf{y}) \cdot p(\texttt{electronics}|\mathbf{y}) \cdot [1 - p(\texttt{electrical}|\mathbf{y})]$ and easily plugged into the PPLM framework [9]. Our model architecture is shown in Figure 1.

III. Method: Plug-and-Play Counterfactual Generation Framework

We now describe our proposed method to generate controlled counterfactual text in a plug-and-play fashion. In particular, given a text $\mathbf{x}$ and a control attribute $\mathbf{a}$, we seek to generate a controlled counterfactual $\mathbf{y}$. That is, we seek to draw samples from a distribution $p(\mathbf{y}|\mathbf{x}, \mathbf{a})$ where the generated sample $\mathbf{y}$ depends both on the input text $\mathbf{x}$ and the given control attribute $\mathbf{a}$. Hence, our task is different and more challenging than the simple controlled text generation task where the generated samples need to depend only on the control attribute $\mathbf{a}$.

Our main idea is as follows: Similar to how PPLM [9] steers a pretrained text generator $p(\mathbf{y}) \to p(\mathbf{y}|\mathbf{a})$ using the control attribute $\mathbf{a}$, we shall steer a pretrained text-to-text generator $p(\mathbf{y}|\mathbf{x}) \to p(\mathbf{y}|\mathbf{x}, \mathbf{a})$. In PPLM [9], the base model $p(\mathbf{y})$ is a pretrained GPT-2, while in our model, the base model $p(\mathbf{y}|\mathbf{x})$ is a pretrained BART model.

A BART model is a text-to-text model that takes as input a text $\mathbf{x}$ and produces a reconstruction $\mathbf{y}$ of the input text. The BART text-to-text framework consists of two modules: A BERT encoder and a GPT-2 decoder. That is, the model takes an input text and the BERT encoder first returns a text representation $\mathbf{e}$. This text representation is then given to the GPT-2 decoder to reconstruct the input text word-by-word. This can be summarized as follows:

$$\begin{aligned}\mathbf{e} &= \text{BERT}(\mathbf{x}),\\ H_t &= \text{Transformer}(\mathbf{y}_{<t}, \mathbf{e}),\\ \mathbf{o}_t &= \text{PredictionHead}(H_t),\\ \mathbf{y}_t &\sim \text{Categorical}(\mathbf{o}_t).\end{aligned}$$

where t is the word position in the text, H_t is the last hidden layer before the prediction head and $\mathbf{o}_t$ are the log-probabilities of the words in the vocabulary used for sampling the next word $\mathbf{y}_t$. We will refer to this as the unmodified BART model having the next word prediction distribution $p(\mathbf{y}_t|\mathbf{y}_{<t}, \mathbf{x})$. To steer the BART model, similarly to PPLM, we add a learnable perturbation ΔH_t to the hidden states H_t of the unmodified BART model. This can be summarized as follows:

$$\begin{aligned}\mathbf{e} &= \text{BERT}(\mathbf{x}),\\ H_t &= \text{Transformer}(\mathbf{y}_{<t}, \mathbf{e}),\\ \mathbf{o}_t &= \text{PredictionHead}(H_t + \Delta H_t),\\ \mathbf{y}_t &\sim \text{Categorical}(\mathbf{o}_t).\end{aligned}$$

We will refer to this as the modified BART model having the next word prediction distribution $\bar{p}(\mathbf{y}_t|\mathbf{y}_{<t}, \mathbf{x})$ Similarly to PPLM, the learning of the perturbation parameters $\{\Delta H_1, \dots, \Delta H_T\}$ is driven by the following objective:

$$\begin{aligned}\mathcal{L}_{\text{CASPer}} = &-\log p(\mathbf{a}|\mathbf{y})\\ &- \sum_{t=1}^{T} D_{\text{KL}}(p(\mathbf{y}_t|\mathbf{y}_{<t}, \mathbf{x})||\bar{p}(\mathbf{y}_t|\mathbf{y}_{<t}, \mathbf{x})).\end{aligned}$$

where the first term provides the learning signal to steer the counterfactuals towards the desired goal attribute by

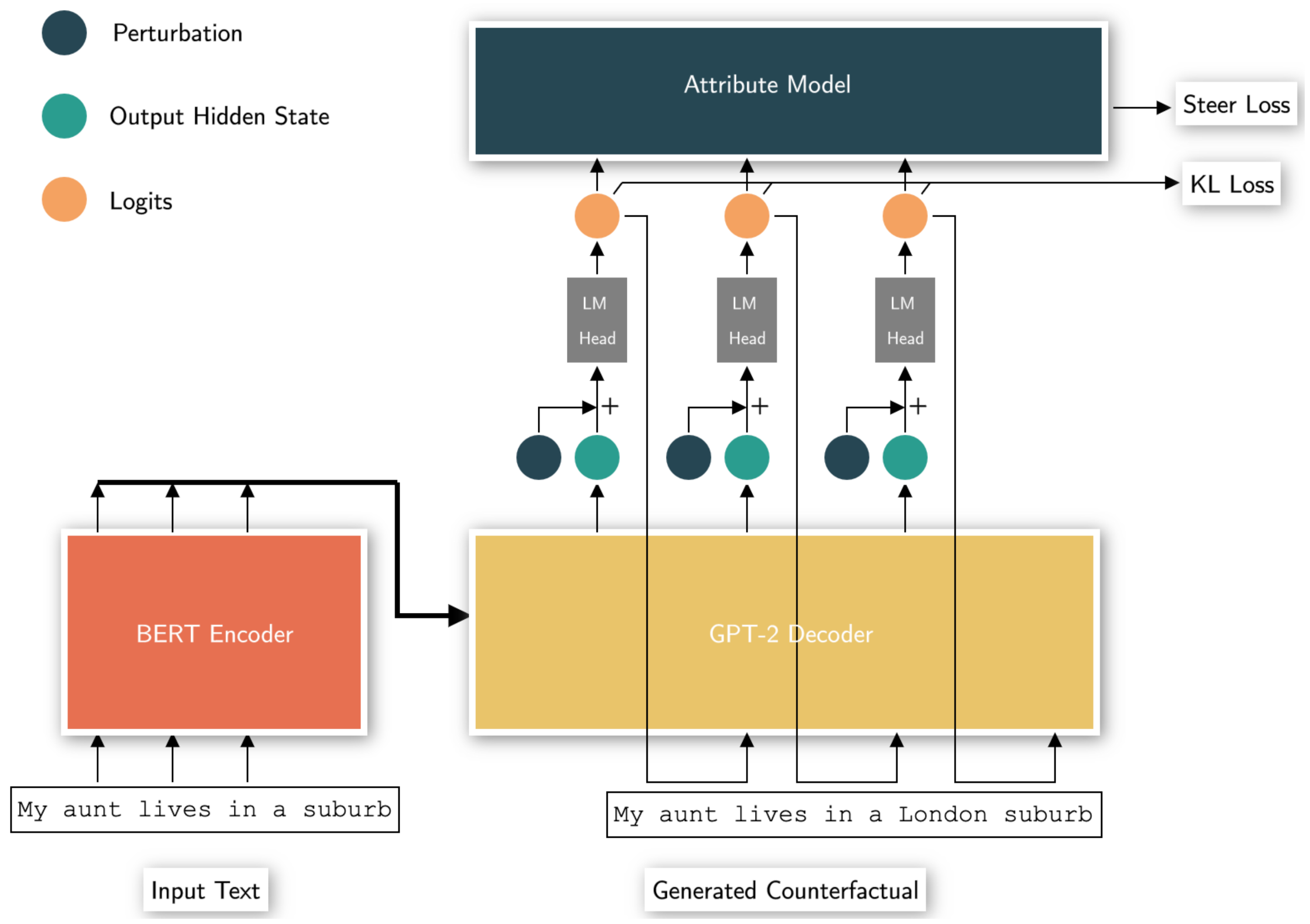

Fig. 1. Model Architecture of CASPer. The BERT encoder (shown in orange) takes the input text and returns a representation of the input text as a set of vectors. These are provided to the GPT-2 decoder for cross-attention. At each step of decoding, the decoder returns an output hidden state (shown as green circle). To this, we add a perturbation matrix (shown as blue circle). The perturbed hidden state is then provided to the language model head to obtain the logits over the vocabulary for sampling the next token. These logits are also provided to the attribute model for computing the loss for steering the generation. The KL divergence between the perturbed logits and the unperturbed logits is also minimized to keep the semantic content of the generated text close to the original input text.

trying to maximize the log-probability of the desired attribute. The second term tries to keep the generations close to the unmodified BART to ensure that the text remains similar in content to the original input text and also remains fluent and plausible. We note that this learning process is done separately each time we need to generate a new sample. However, the learning of the perturbation parameters $\{\Delta H_1, \dots, \Delta H_T\}$ can be done very quickly and it only takes about 100 gradient steps. This property makes CASPer a flexible way to generate counterfactuals of a given text.

Discussion. Note that if we simply obtain samples from a pre-trained BART model $p(\mathbf{y}|\mathbf{x})$, a sample $\mathbf{y}$ can be considered as a counterfactual of the input text $\mathbf{x}$. However, this sample would be an almost exact reconstruction of the input text. Hence from the perspective of model testing, this type of counterfactual would not be much useful. However, by applying steering to $p(\mathbf{y}|\mathbf{x})$ using a control attribute $\mathbf{a}$, we are able to control in what way we want to modify the input text to generate the counterfactual. Hence, from the perspective of model testing, this type of counterfactual would be useful because we can test *how a deployed model will behave if the distribution of its inputs are perturbed in a certain way*.

IV. Related Work

The task of controlled text generation is well studied in literature. [11] propose a model aims to generate plausible sentences conditioned on representation vectors with semantic structure. Another work [12] focuses on controlled text generation, however, unlike the previous work [11], the conditioning need not be simply a class label. The conditioning can be a data structure such as a table. The model is trained end-to-end similarly to the objective of [11]. PPLM [9] combine a pre-trained language model, similarly to [13] with an attribute classifier to perform controlled language generation and use the attribute classifier to steer the text generation process without further training of any of the two models. [14], adopting a similar direction, deal with story completion with a desired sentiment. [15] is a model that controls text generation via 50 rigid control codes predetermined at training time. However all these works, cannot be used for counterfactual text generation as these are purely class-conditional generative models and do not allow generation conditioned on a given input text. Some earlier works, including and not limited to, [9], [16]–[20] propose the idea of steering Language Models but these also can not be directly used for counterfactual generation task.

To tackle text-to-text generation tasks dealing with transfer of style or content, models such as [21]–[23] have been proposed. However, these works are not plug-and-play and lack the use of attribute model that can plugged flexibly at sampling time. This task of generating controlled counterfactuals has been attempted in prior works by relying on template-matching and token-based substitutions to generate the test-cases [1], [3]. However, this can require significant human-involvement to curate the templates and the dictionaries. Hence, it cannot scale well when template and dictionaries need to be updated frequently. The work [1] employs a tool Checklist which is one of the attempts to come up with generalized perturbations. For generation, Checklist uses a set of pre-defined templates, lexicons, general-purpose perturbations, and context-aware suggestions. To better evaluate the deployed models, some prior works have relied on human designed test examples or either using templates [1], [24]–[29]. Polyjuice [3], while seeking to automate the process, still requires paired dataset in the form of text and their perturbed versions for different control codes. Therefore the mapping between text and perturbed version is learned through supervision. Another parallel work Tailor [30] generates perturbations designed for different control codes by making use of a combination of semantic roles and content keywords. And thereby require supervision for different controls. In contrast, CASPer does not require any task-specific or control-code specific training and can be used to work with different control code models given input text. One work related to ours has been tackled in [31] which generates text samples given a text with a controlling that specify the scope of the generated text. LEWIS attempts to generate text perturbations by introducing blanks via template matching and filling in using pretrained language models [32]. However, this relies on rule-based template matching and human supervision to develop such templates. CATGen [33] tries to generate attribute-specific text but it requires training of sequence to sequence model with pre-determined control codes for perturbation. Hence, it lacks the flexibility of a plug-and-play approach like ours. MiCE [34] proposes a technique to generate counterfactual explanations which are human interpretable and user-centric. It fine-tunes a T5 model to generate counterfactual text and use them as explanations for the behavior of the deployed models but lack feature-attributions. Another work [30] tries to generate perturbation with semantic controls but rely on specific templates derived using semantic roles and other labeling heuristics. A work close to ours, GYC, the inference of latent representation of the input text with respect to a GPT-2 decoder is done directly via optimization. This approach fails to achieve good inference for long and complex text [31]. To target model failure, thus implicitly acting as a form of model testing, prior works have attempted the use of adversarial approaches [35]–[37] stemming from the need to build robust models via adversarial testing [2], [38]–[40]. However, these are still limited to specific domains and generations are likely to be not plausible to be seen in the input text [41] or may require additional human effort [42]. Some works have attempted to change style attributes automatically either with no control or with predefined style templates [43], [44]. The notion of counterfactuals [6], [45] and their use in model testing for has also been applied towards testing in models that consume structured inputs [46]–[48]

V. Experiments

The goal of the experiments is to: *1)* show that a flexible plug-and-play framework can effectively achieve controlled counterfactual text generation and the generated text is fluent, plausible, diverse and follows the steering provided by the attribute model. *2)* We evaluate how well the generated perturbations can act as data-augmentation samples in order to make a downstream classification task performance more robust.

A. Datasets

We evaluate the models on the following data sets text.

1) **YELP Sentiment Dataset.** To evaluate how our model is able to change the sentiment of the original text and achieve the target sentiment, we use the YELP sentiment dataset [49]. This dataset is also characterized by informal text which can be seen in realistic user inputs.
2) **IMDB Sentiment Dataset.** To further evaluate how our model is able to change the sentiment of the original input text, we test on IMDB Sentiment Dataset [50]. This dataset is also characterized by long and complex text and is thus a challenging dataset.

B. Controlled Text Generation with Attribute Steering

We first evaluate the quality of our generated text with respect to the steering signal. We expect our generated text to preserve the semantic content and syntactic structure of the input text while being fluent and diverse as we steer the text towards the target attribute. The steering signal we evaluate in this work is to make the sentiment of the target text from negative to positive.

1) Baselines: To compare with state-of-the-art template-based methods relying on token substitutions via dictionaries we compare with Checklist [1]. We specifically consider the perturbation helper that relies on RoBERTa [51] to fill-in-the-blank. In comparison to this baseline, we expect ours to generate more fluent and diverse text samples that is free from the restrictions of the pre-specified templates. We also compare against Masked-LM [52], which is a dictionary-free approach but still relies on masking a specific token in the input text and and letting the model fill-in the masked token.

Metrics	**Dataset**	**RoBERTa**	**Token-based**	**GPT−2**	**Gradient-based**	**Finetuning**	**Ours**
		Mask−LM	Checklist	PPLM	Hotflip	Polyjuice	CASPer
CP ↑	YELP	0.30	0.321	0.064	0.365	0.212	0.202
	IMDB	0.29	0.30	0.048	0.291	0.317	0.231
Perplexity ↓	YELP	3.82	3.79	3.544	3.95	3.64	3.44
	IMDB	3.05	3.12	3.35	3.69	3.331	2.80
BLEU−4 ↓	YELP	0.903	0.530	0.064	NA	0.521	0.309
	IMDB	0.9027	0.909	0.042	NA	0.861	0.231

TABLE III
COMPARISON BETWEEN MODELS ON THE YELP AND IMDB DATASET. THE MODEL USED FOR STEERING IS A PRE-TRAINED SENTIMENT CLASSIFICATION MODEL.

The randomness in this filling-in process leads to generation of counterfactual samples. Hence this model generates only token-level substitutions and does not generate fluent sentence-level text. We expect CASPer to address this limitation of purely token-level substitutions.

To compare with state-of-the-art adversarial methods we compare with Hotflip [39] [2]. In comparison to this baseline, we expect ours to generate more fluent samples. We expect to see that content preservation of such approaches is high as these methods rely on changing the highest gradient word with another word that would flip the label.

To compare with state-of-the-art text generation methods we compare with PPLM [9] based on GPT-2 [53] and Polyjuice [3] based on finetuned GPT-2 [53]. In comparison to this baseline, we expect ours to generate more fluent and diverse samples. While for PPLM, since it simply takes a prompt text and completes the text, it has no incentive to generate text that preserves its content. Thus, we expect ours to preserve content better. In diversity of the samples, PPLM, because it is not tasked with preserving content, can generate arbitrary and overly diverse samples. Thus we highlight that while we perform a comparison of sample diversity between PPLM and our model, still the performances are not directly comparable because the expectations are different from both models. For Polyjuice, we expect ours to generate more fluent and diverse samples because, unlike Polyjuice, ours is free from specific template-based generation.

2) Metrics: To assess the quality of generated counterfactual text we focus on evaluating content preservation, fluency, diversity and syntactic similarity. We use the following metrics to measure the above characteristics.

1) **Content Preservation.** By measuring content preservation, we assess the similarity between input text and the counterfactual text samples. For this, we use the transformer model proposed in [54]. While higher content preservation is desirable in general, this metric alone does not provide the complete evaluation. Therefore for proper evaluation, we will introduce a second metric that measures sample diversity.
2) **Diversity.** This metric evaluates how different are the generated samples from each other. We find the BLEU-4 score between the input text and the generated text. Hence, if this score is lower, then the generated counterfactual samples have a high diversity at the token level.
3) **Fluency.** Fluency of the generated samples is important to evaluate because the samples must come from a distribution that the test model is likely to see when it is deployed. This is computed by finding the perplexity score of the generated output. We take a GPT-2 model for computing the perplexity. Lower perplexity implies that the generated text is more fluent.

3) Quantitative Results: In Tab. III, our results on perplexity show that CASPer outperforms the baselines significantly and achieves a lower perplexity score. This shows that the samples generated by CASPer are fluent and plausible. We also show that CASPer is able effectively preserve the content in its samples. We note that our model is competitive with rule-based token substitution methods like Checklist. Lastly, in terms of BLEU-4 score, we note that our model again outperforms the baselines with our generated samples achieving the lowest values of BLEU-4 score with the original input text only exception being PPLM. This shows that our model indeed generates diverse samples with that have low token-level match with the input text. In diversity of the samples, the baseline model PPLM, because it is not tasked with preserving content, can generate arbitrary and overly diverse samples. Thus we highlight that while we perform a comparison of sample diversity between PPLM and our model, still the performances are not directly comparable because the expectations are different from both models. Considering the performance on all the metrics together shows that CASPer is able to effectively generate samples that preserve the original content, are fluent and diverse in comparison to the baselines.

4) Qualitative Analysis: In Tab. IV, we show samples of text generated by CASPer. We show two experiments. In the first experiment, our steering goal was to take an input text and perturb it so that the probability of its sentiment becomes large. The probability of the sentiment is estimated using a pre-trained sentiment classification model. The initial label of the text was negative. On the right we note that CASPer has successfully perturbed the text to change its sentiment label to positive. Furthermore, note that the generated sample have good content preservation as all the samples talk about movies and actors. Furthermore, the model makes some important changes to the content that result in a change in the sentiment of the text. We also note that each sample is different from the other thus producing diverse samples. Lastly, we note that

Inputs	**Generated Controlled Perturbations**
Steering Goal : To maximize the probability of the positive class label with respect to a pre-trained sentiment classifier. **Input Text**: `We know from other movies that the writers can't solve problems, like this one. The actors are also good.` **Initial Label** : Negative	We know from other movies that the actors are good but they cannot help the movie. The premise was not too bad and is enjoyable. **New Label** : Positive We know from other documentaries that the acting was quite strong. We can't say the premise was good but the execution was good. **New Label** : Positive We know from other movies that the actors are good but they must win over the fans. The premise was not too bad. **New Label** : Positive We know from other movies that the actors are good. They can not help the movie. The premise was not too bad. **New Label** : Positive
Steering Goal: To make the sentence contain at least one location named-entity. **Input Text**: `A wonderful little production.The filming technique is very unassuming- very old-time-BBC fashion.` **Initial State** : No location named-entity is present in the text	A wonderful first look at this film from the UK. The filming is unassertive. ¡/s¿io is also one of the best productions. **Location Named-Entity**: UK A wonderful little Theatre is an old-time BBC production, set in an old London apartment block. The film is set in London landmarks **Location Named-Entity**: London A Wonderful Little Edition by BBC Films - set, out in UK now via iPlayer. A wonderful little production with some nice twists. **Location Named-Entity**: UK A wonderful little series about a BBC crew in Africa. Well-executed by the team who are experienced in film and rigging procedures. **Location Named-Entity**: Africa A wonderful little show about BBC's wartime bombing of Libya contributed by Peter Bradshaw in 1959. **Location Named-Entity**: Libya
Steering Goal : To maximize the probability of the positive class label with respect to a pre-trained sentiment classifier. **Input Text**: `I had the terrible misfortune of having to view this "b-movie" in it's entirety. All I have to say is--- save your time and money!!` **Initial Label** : Negative	I have the wonderful misfortune of having to view this New Girl in it's entirety. To view it positively, I must open up and say. **New Label** : Positive I had the terrible misfortune of having to view this ẗransform into... in it's entirety. It's truly amazing how some programs remain.. **New Label** : Positive I had the terrible misfortune of having to view this movie in it endearing ways. It's a wonderful salute wives lenders acknowledge.. **New Label** : Positive I had the terrible misfortune of having to view this in it's full glory. All I have to say is wow. everyone agrees. **New Label** : Positive
Steering Goal : To maximize the probability of the negative class label with respect to a pre-trained sentiment classifier. **Input Text**: `For those who'd like to see this movie? I'd say: go! Without the narration it might be a very good movie/documentary.` **Initial Label** : Positive	For those who'd like to see it I suggest that they see this movie without narration. The movie isn't a very good movie. **New Label** : Negative For those want to get to know the movie through its narration? I'd say: go! Without the narration it might be a very bad movie. **New Label** : Negative

TABLE IV
GENERATED CONTROLLED PERTURBATIONS FROM THE PROPOSED MODEL CASPER.

Model	Dataset	Mask-LM	Checklist	PPLM	Hotflip	Polyjuice	CASPer
Accuracy - No Aug	YELP	89.90	89.90	89.90	89.90	89.90	89.90
	IMDB	90.10	90.10	90.10	90.10	90.10	90.10
Accuracy - With Aug	YELP	90.50	90.53	90.82	90.15	90.98	**92.00**
	IMDB	90.11	90.11	90.18	90.10	90.85	**91.20**

TABLE V

COMPARISON OF ACCURACY BETWEEN MODELS ON THE YELP AND IMDB DATASET. THE GENERATED DATA FOR NER TASK ON STEERING IS USED FOR ROBUSTIFYING AN N-GRAM BASED SENTIMENT MODEL.

the samples are fluent and plausible text samples.

In the second experiment, our steering goal was to take a sentence that does not contain a location named-entity and perturb it so that contains a location named-entity. We see that CASPer produces samples that contain a location named-entity tag. We also note that the named entity that the model introduces are diverse and are used in a variety of contexts in the generated text. As before, the text samples are fluent and preserve the content of the original input text. For this task, note that these samples clearly retain the sentiment of the text and only introduce some location entities. Because we expect the actual location (i.e. UK or Libya) should not be a causal term in prediction of the sentiment of the text, these samples can act as effective samples for augmenting the training data when we train a downstream sentiment model. While a model that is biased may predict different labels based on the actual location token used, this kind of data augmentation will regularize the model to be more robust to such changes which should ideally not affect the predicted label of the test model.

C. Controlled Text for Model Robustification

In this section, we evaluate how well our generated samples can improve robustness of the test classifier. For this, we generated text samples to introduce a location named-entity in the input text. We assume that simply introducing a location named-entity should not change the class label of the text with respect to the test model. Hence, after generating the controlled perturbations, we take the original label of the input text from the training set and assign the same label to the generated samples. These new examples are added to the training set and producing data-augmented training set. Using this augmented training set, we then train the test model.

1) Baselines and Metrics: We generate samples using CASPer. We augment the generated samples to the training set and train the test model. We compared the accuracy of the test model trained without data augmentation and then trained with data augmentation via our counterfactual generation method.

2) Quantitative Results: In Table V we show a comparison between the models. We note that the samples generated by CASPer using NER model [1] are effective in robustifying the test model and produces significant improvement in the accuracy as compared to when training with original samples.

VI. CONCLUSION

In this paper, we introduced CASPer, a plug-and-play counterfactual text generation framework. We showed that our generated controlled perturbations preserve the content of the original text while also being fluent, diverse and effective in terms of the provided steering signal flexibly. We showed that samples generated by CASPer can act as effective candidates for training data augmentation and improve the robustness of the target model and preventing the target model from modeling spurious correlations between the target label and non-causal aspects of the input text.

[1]https://github.com/kamalkraj/BERT-NER

REFERENCES

[1] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of nlp models with checklist," *arXiv preprint arXiv:2005.04118*, 2020.

[2] P. Michel, X. Li, G. Neubig, and J. M. Pino, "On evaluation of adversarial perturbations for sequence-to-sequence models," *arXiv preprint arXiv:1903.06620*, 2019.

[3] T. Wu, M. T. Ribeiro, J. Heer, and D. S. Weld, "Polyjuice: Automated, general-purpose counterfactual generation," *arXiv preprint arXiv:2101.00288*, 2021.

[4] P. Ma, S. Wang, and J. Liu, "Metamorphic testing and certified mitigation of fairness violations in nlp models," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2020, pp. 458–465, main track.

[5] K. Holstein, J. Wortman Vaughan, H. Daumé III, M. Dudik, and H. Wallach, "Improving fairness in machine learning systems: What do industry practitioners need?" in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–16.

[6] S. Wachter, B. Mittelstadt, and C. Russell, "Counterfactual explanations without opening the black box: Automated decisions and the gdpr," *Harv. JL & Tech.*, vol. 31, p. 841, 2017.

[7] J. Pearl *et al.*, "Models, reasoning and inference," *Cambridge, UK: CambridgeUniversityPress*, 2000.

[8] S. Garg, V. Perot, N. Limtiaco, A. Taly, E. H. Chi, and A. Beutel, "Counterfactual fairness in text classification through robustness," in *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 2019, pp. 219–226.

[9] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu, "Plug and play language models: a simple approach to controlled text generation," *arXiv preprint arXiv:1912.02164*, 2019.

[10] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[11] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing, "Toward controlled generation of text," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1587–1596.

[12] R. Ye, W. Shi, H. Zhou, Z. Wei, and L. Li, "Variational template machine for data-to-text generation," *arXiv preprint arXiv:2002.01127*, 2020.

[13] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski, "Plug & play generative networks: Conditional iterative generation of images in latent space," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4467–4477.

[14] F. Luo, D. Dai, P. Yang, T. Liu, B. Chang, Z. Sui, and X. Sun, "Learning to control the fine-grained sentiment for story ending generation," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 6020–6026. [Online]. Available: https://aclanthology.org/P19-1603

[15] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher, "Ctrl: A conditional transformer language model for controllable generation," *arXiv preprint arXiv:1909.05858*, 2019.

[16] J. Gu, G. Neubig, K. Cho, and V. O. Li, "Learning to translate in real-time with neural machine translation," *arXiv preprint arXiv:1610.00388*, 2016.

[17] J. Gu, K. Cho, and V. O. Li, "Trainable greedy decoding for neural machine translation," *arXiv preprint arXiv:1702.02429*, 2017.

[18] Y. Chen, V. O. Li, K. Cho, and S. R. Bowman, "A stable and effective learning strategy for trainable greedy decoding," *arXiv preprint arXiv:1804.07915*, 2018.

[19] N. Subramani, S. R. Bowman, and K. Cho, "Can unconditional language models recover arbitrary sentences?" *arXiv preprint arXiv:1907.04944*, 2019.

[20] B. Krause, A. D. Gotmare, B. McCann, N. S. Keskar, S. Joty, R. Socher, and N. F. Rajani, "Gedi: Generative discriminator guided sequence generation," *arXiv preprint arXiv:2009.06367*, 2020.

[21] T. Shen, T. Lei, R. Barzilay, and T. Jaakkola, "Style transfer from non-parallel text by cross-alignment," *arXiv preprint arXiv:1705.09655*, 2017.

[22] J. Li, R. Jia, H. He, and P. Liang, "Delete, retrieve, generate: A simple approach to sentiment and style transfer," *arXiv preprint arXiv:1804.06437*, 2018.

[23] G. Lample, S. Subramanian, E. Smith, L. Denoyer, M. Ranzato, and Y.-L. Boureau, "Multiple-attribute text rewriting," in *International Conference on Learning Representations*, 2018.

[24] M. Gardner, Y. Artzi, V. Basmov, J. Berant, B. Bogin, S. Chen, P. Dasigi, D. Dua, Y. Elazar, A. Gottumukkala, N. Gupta, H. Hajishirzi, G. Ilharco, D. Khashabi, K. Lin, J. Liu, N. F. Liu, P. Mulcaire, Q. Ning, S. Singh, N. A. Smith, S. Subramanian, R. Tsarfaty, E. Wallace, A. Zhang, and B. Zhou, "Evaluating models' local decision boundaries via contrast sets," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, 2020, pp. 1307–1323. [Online]. Available: https://www.aclweb.org/anthology/2020.findings-emnlp.117

[25] D. Teney, E. Abbasnedjad, and A. van den Hengel, "Learning what makes a difference from counterfactual examples and gradient supervision," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 580–599.

[26] D. Kaushik, A. Setlur, E. Hovy, and Z. C. Lipton, "Explaining the efficacy of counterfactually-augmented data," *arXiv preprint arXiv:2010.02114*, 2020.

[27] J. Andreas, "Good-enough compositional data augmentation," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020, pp. 7556–7566. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.676

[28] T. Wu, M. T. Ribeiro, J. Heer, and D. Weld, "Errudite: Scalable, reproducible, and testable error analysis," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 747–763. [Online]. Available: https://www.aclweb.org/anthology/P19-1073

[29] C. Li, L. Shengshuo, Z. Liu, X. Wu, X. Zhou, and S. Steinert-Threlkeld, "Linguistically-informed transformations (LIT): A method for automatically generating contrast sets," in *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*. Online: Association for Computational Linguistics, 2020, pp. 126–135. [Online]. Available: https://www.aclweb.org/anthology/2020.blackboxnlp-1.12

[30] A. Ross, T. Wu, H. Peng, M. E. Peters, and M. Gardner, "Tailor: Generating and perturbing text with semantic controls," *arXiv preprint arXiv:2107.07150*, 2021.

[31] N. Madaan, I. Padhi, N. Panwar, and D. Saha, "Generate your counterfactuals: Towards controlled counterfactual generation for text," in *Proceedings of the AAAI Conference on Artificial Intelligence*, no. 15, 2021, pp. 13 516–13 524.

[32] M. Reid and V. Zhong, "Lewis: Levenshtein editing for unsupervised text style transfer," *arXiv preprint arXiv:2105.08206*, 2021.

[33] T. Wang, X. Wang, Y. Qin, B. Packer, K. Li, J. Chen, A. Beutel, and E. Chi, "Cat-gen: Improving robustness in nlp models via controlled adversarial text generation," *arXiv preprint arXiv:2010.02338*, 2020.

[34] A. Ross, A. Marasović, and M. E. Peters, "Explaining nlp models via minimal contrastive editing (mice)," *arXiv preprint arXiv:2012.13985*, 2020.

[35] M. Iyyer, J. Wieting, K. Gimpel, and L. Zettlemoyer, "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)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1875–1885. [Online]. Available: https://www.aclweb.org/anthology/N18-1170

[36] M. T. Ribeiro, S. Singh, and C. Guestrin, "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)*. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 856–865. [Online]. Available: https://www.aclweb.org/anthology/P18-1079

[37] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu, "BERT-ATTACK: Adversarial attack against BERT using BERT," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, 2020, pp. 6193–6202. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-main.500

[38] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[39] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "Hotflip: White-box adversarial examples for text classification," *arXiv preprint arXiv:1712.06751*, 2017.

[40] Z. Zhao, D. Dua, and S. Singh, "Generating natural adversarial examples," *arXiv preprint arXiv:1710.11342*, 2017.

[41] J. Li, W. Monroe, and D. Jurafsky, "Understanding neural networks through representation erasure," *arXiv preprint arXiv:1612.08220*, 2016.

[42] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," *arXiv preprint arXiv:1707.07328*, 2017.

[43] A. Madaan, A. Setlur, T. Parekh, B. Poczos, G. Neubig, Y. Yang, R. Salakhutdinov, A. W. Black, and S. Prabhumoye, "Politeness transfer: A tag and generate approach," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020, pp. 1869–1881. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.169

[44] E. Malmi, A. Severyn, and S. Rothe, "Unsupervised text style transfer with padded masked language models," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, 2020, pp. 8671–8680. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-main.699

[45] R. K. Mothilal, A. Sharma, and C. Tan, "Explaining machine learning classifiers through diverse counterfactual explanations," in *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 2020, pp. 607–617.

[46] S. Udeshi, P. Arora, and S. Chattopadhyay, "Automated directed fairness testing," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 98–108.

[47] P. G. John, D. Vijaykeerthy, and D. Saha, "Verifying individual fairness in machine learning models," in *Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 749–758.

[48] S. Galhotra, Y. Brun, and A. Meliou, "Fairness testing: testing software for discrimination," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 498–510.

[49] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[50] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, 2011, pp. 142–150. [Online]. Available: https://www.aclweb.org/anthology/P11-1015

[51] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[52] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[53] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *arXiv*, 2019.

[54] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: http://arxiv.org/abs/1908.10084