Fixing Model Bugs with Natural Language Patches

Anonymous ACL submission

Abstract

The de-facto standard for fixing bugs in models post training is to finetune the model on additional annotated data, or patch the model with tenuous if-else rules. In contrast, humans can often use natural language as a tool for providing corrective feedback to each other. In this work, we explore using *natural language* patches from users to fix bugs in NLP models. Our overall approach uses a gating head to softly combine the original model output with a patch-conditioned output from an interpreter head. Both of these heads are trained by inserting a patch finetuning stage between training and deployment, where the training objective is based on synthetically generated inputs and patches. Surprisingly, we show that this synthetic patch training phase is enough to enable patching inputs on *real* data—on two data slices from a sentiment analysis dataset, we show that 1 to 5 language patches can improve performance by ~1-4%. Next, on an adversarial relation extraction diagnostic test set, we improve F1 by over 30% with just 6 patches.

1 Introduction

Despite the prevalence of bugs or undesirable behaviors in NLP models (Ribeiro et al., 2020), the problem of *fixing* such bugs is less well understood. Suppose a user identifies two bugs in a sentiment analysis model: it fails to identify low star ratings as sufficient conditions for a negative prediction, and it doesn't understand colloquial usage of the word 'bomb' (Figure 1). In order to fix such bugs, the user may collect additional data for finetuning, a tedious and computationally demanding process that can lead to shortcuts such as "if the word bomb is in the review, it is positive" or "the number 2 indicates a negative review". Similarly, rule-based patches (Figure 1b) lead to brittleness, as these rely on word matching rather than the right abstractions.

In this work, we propose *natural language patches* as an approach to fixing bugs. Natural language makes it easy for users to express a patch with the right abstraction, without having to specify exactly how the condition is applied, e.g. "If food is described as bomb, then food is good" is applied to "The tacos were bomb" and "Pizza was just the bomb!". Our modeling approach includes a *gating* head to soft-predict whether the patch should be applied (e.g. "food is described as bomb"), and a *conditioning* head to predict a new output when the patch applies, combining the consequent (e.g. "food is good") with the original input. Both heads are trained on synthetic data in a step between training and deployment, such that new patches can be applied at test-time without further training, and can be user-specific (with a shared base model).

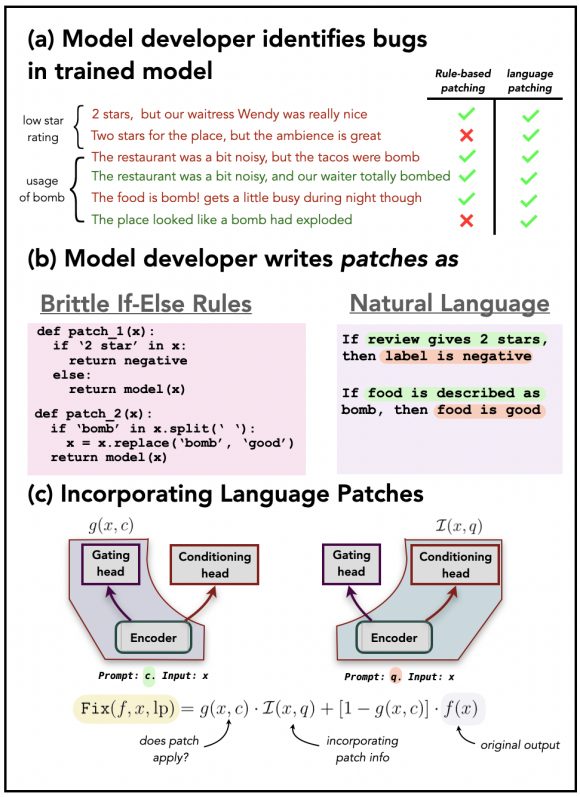

Figure 1: While rule-based patches rely on hard pattern matches, a language patch can use the right abstractions.

Our controlled experiments indicate that natural language patches are applied correctly when the condition is met, and do not change predictions when it does not – even for abstract conditions where rule-base patches would be infeasible or very difficult. This is also the case for test-time patches that are very different than the ones used in the synthetic training data. We show how a small set of simple patches can be used to fix bugs (and thus improve performance) on real benchmarks for two different tasks, despite the synthetic nature of the patch tuning phase. On two data slices from the Yelp reviews dataset, we show that 1 to 5 language patches can improve performance by ~1-4%. Finally, on an adversarial relation extraction diagnostic test set, 6 patches improve F1 by over 30%.

2 Setup

We are given a text classifier (the model) f that maps an input text x to a probability distribution over its output space, $f(x) = \Pr(y \mid x)$. In our setting, this model is riddled with *bugs*—defined as any behavior inconsistent with the user's preference or the "ground truth". We use a library of user provided patches $P = \{lp_1, lp_2, \ldots, lp_t\}$ to fix these bugs. In this work, each lp_j is of the from "If c_j, then q_j", where c_j denotes the condition under which the patch is to be applied, and q_j refers to the consequence of applying it. Given the original output $f(x)$, the *updated* output is $\texttt{Fix}(f, \text{x}, P)$.

3 Our Approach

We start with a model f consisting of a classification head on top of an encoder. Our approach consists of two finetuning stages. In the **Task Finetuning** stage, we are given a dataset $\{x_i, y_i\}$ of labeled examples from some task which we use to train f through standard supervised learning. Next, in the **Patch Finetuning** stage, we use the learnt encoder and learn a gating head g (initialized randomly) and an interpreter head $\mathcal{I}$ initialized with the classification head.

For the latter stage, we write a small set of patch templates covering the kinds of patches users might want to write for their own application (see Table 5 in Section A for templates used for our sentiment analysis results). Based on these templates, we instantiate a small number of patches along with synthetic labeled examples. This gives us a dataset $\{x_i, y_i, lp_i\}$, where each lp_i consists of a condition c_i as well as a consequence q_i.

Intuitively, g computes the probability that the condition specified by $lp = (c, q)$ is true for a given input x as $g(x, c)$. The interpreter model $\mathcal{I}$ computes a new distribution over the label space, that conditions on x *and* the consequence q. This is then combined with the original model output $f(x)$ using the above gating probability.

The interpreter head $\mathcal{I}$ is trained to model $\Pr(y_i \mid x_i, q_i)$ through standard log-likelihood maximization. The gating head g is trained via noise contrastive estimation to maximize

$$\log g(x_i, c_i) - \sum_{c_j \in \text{NEG}(x_i)} \log g(x_i, c_j),$$

where $\text{NEG}(x_i)$ is a randomly sampled set of negative conditions for x_i.

Applying Patches. Finally, a single patch $lp = (c, q)$, can be applied to any input x as

$$\begin{aligned}\texttt{Fix}(f, \text{x}, lp) = {} & g(x, c) \cdot \mathcal{I}(x, q) \\ & + [1 - g(x, c)] \cdot f(x)\end{aligned} \tag{1}$$

To extend the above to leverage a library of patches $P = \{lp_1, lp_2, \ldots, lp_t\}$, we find the patch that is "most relevant" for the given input and use that to update the model's output,

$$lp^* = \arg\max_{lp_i \in P} g(x, c_i) \tag{2}$$

$$\texttt{Fix}(f, \text{x}, P) = \texttt{Fix}(f, \text{x}, lp^*) \tag{3}$$

4 Experiments

Applications. We apply our method to binary sentiment analysis and relation extraction. For relation extraction, we consider the Spouse dataset from Hancock et al. (2018), where the task is to determine whether two entities are married or not given a textual context about them.

Patch Types. We consider two categories of patches (see Figure 2 and Table 5 for examples). **Override** patches are of the form "If cond, then label is l" i.e. they *override* the model's prediction on an input if the corresponding condition is true. For these patches, we do not learn the interpreter head (since $\mathcal{I}(x, q)$ is given as l). Next, **Feature-based** patches are of the form "If cond, then feature", i.e. they provide the model with a contextual feature "hint" in natural language, e.g. in Figure 2 the feature is "food is good". For these patches, the model needs to learn how to integrate the hints with the original data, and thus we train both the gating head and the interpreter head.

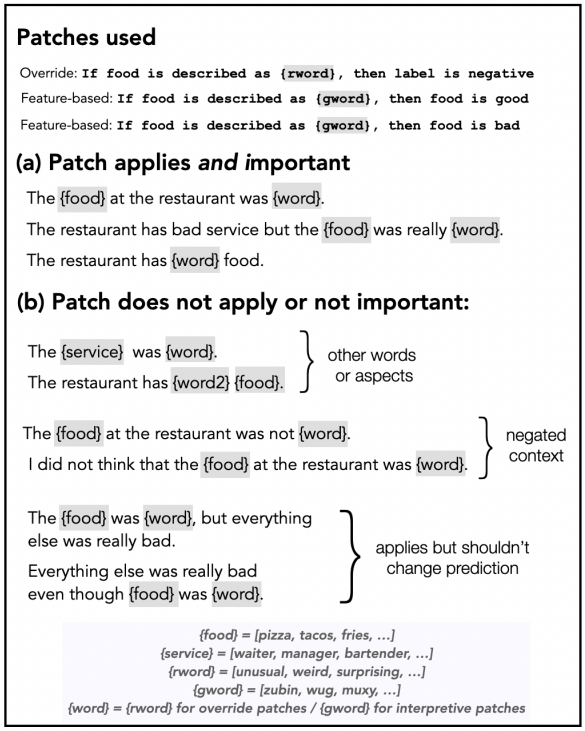

Figure 2: (a) We construct datasets based on checklists to study how well our approach can leverage abstractions in language patches. (b) To control for spurious shortcuts such as copying words from the patch or performing simple string lookups, we also construct corresponding control datasets.

Dataset	ORIGINAL	ORIGINAL+PF	PATCHED
I-Food-Abstraction	59.1	59.9	**100.0**
I-Food-Abstraction (Control)	71.06	56.8	56.8
O-Food-Abstraction	50	50	**85.4**
O-Food-Abstraction (Control)	37.5	61.3	61.3

Table 1: We see significant improvements when the patches apply, and no changes when they do not apply or are unimportant. These improvements do not accrue if we simply finetune on synthetic data (Original + PF)

4.1 Sentiment Analysis

We fix bugs in a T5-large (Raffel et al., 2019) model trained on the Stanford Sentiment Treebank dataset (Socher et al., 2013). We report accuracies of the original model with only task finetuning (ORIGINAL) as well as the model obtained after patch finetuning (ORIGINAL+PF), to isolate the gains of natural language patches from those induced by training on additional synthetic data.

4.1.1 Patching with abstract conditions

To study the ability of our approach to patch with abstract conditions, we construct synthetic datasets (Fig. 2) based on CheckList (Ribeiro et al., 2020). We construct analogous datasets for both override (O-Food-Abstraction) as well as feature-based patches (I-Food-Abstraction). For O-Food-Abstraction, we generate inputs with negative adjectives that the base model makes errors on. For I-Food-Abstraction, we use gibberish adjectives to get a large collection of words for which the model doesn't know the sentiment. To control for spurious behaviors such as copying label words from the patch, performing string lookups or ignoring negated contexts, we construct a control dataset O-Food-Abstraction (Control). Additionally for feature-based patches, we want to ensure that the patch doesn't change behavior when it's not important e.g. an input of the form "The food was word, but everything else was really bad" can be classifed as negative without knowing the meaning of word. Thus, in I-Food-Abstraction (Control), we also include inputs where the patch might apply but is not important. Finally, we write a set of patches to use on both of these datasets (Fig. 2).

Results. From Table 1, we first note that on both I-Food-Abstraction and O-Food-Abstraction, ORIGINAL+PF is either worse or only marginally better than ORIGINAL. This is by design, since we ensure that the patch finetuning data is sufficiently different from the dataset we evaluate on. Next, we observe that our patches with abstract conditions, are able to boost model performance by 35-41 accuracy points – i.e., the model recognizes that "pizza", "tacos", etc are "food". At the same time, comparing ORIGINAL+PF and the patched model, we see that these patches do not affect model predictions on *any* of the control datasets (i.e. "waiter" is not "food", negations are parsed appropriately, etc). These results indicate that our patching mechanism is able to handle abstract conditions accurately, and that patching does not work via simple shortcuts.

4.1.2 Patching on real data

Override Patches. We use the following three benchmarks derived from real world datasets. For Yelp-Aspect, we use patches of the form "if aspect is good / bad, then label is positive / negative" where aspect is food or service. To obtain labels, we manually annotate a subset of 500 examples with food and service specific sentiment e.g "The food was good, service not so much" would be labeled as service: 0, food: 1. These aspect specific labels are then used to obtain the ground truth for each patch, such that the ground truth label of

Setting	ORIGINAL	ORIGINAL+PF	PATCHED
Yelp-Aspect	94.7	93.6	**95.7**
Yelp-Stars	93.2	93.6	**94.5**
ClothingReviews	89.6	88.9	**90.1**

Table 2: Using Override Patches on real data.

Setting	ORIGINAL	ORIGINAL+PF	PATCHED
Yelp-bomb	88.2	88.2	**93.5**
Yelp-clothes-dope	90.5	90.1	**95.2**
Yelp-the-shit	84.4	84.4	**90.6**
Yelp-omg	92.8	91.1	**94.6**
Yelp-wtf	**100.0**	92.9	96.4
Yelp-Colloquial (Overall)	89.1	88.6	**93.4**

Table 3: Using Feature-based Patches on real data

	Model	F1
	ORIGINAL	59.2
	ORIGINAL+PF	63.7
Using single patch	If p_1 is the cousin of p_2, then label is negative	65.1
	If p_1 and p_2 went on a honeymoon, then label is positive	82.6
	If p_1 has kids with p_2, then label is positive	75.4
	If p_1 divorced p_2, then label is negative	66.8
	If p_1 has tied the knot with p_2, then label is positive	81.8
	If p_1 and p_2 are coworkers, then label is negative	63.7
	P_{spouse} (Using all patches)	**91.6**

Table 4: Using Override Patches on the diagnostic test set. Over-dependence on sex of the two entities causes the base model to struggle. A small number of language patches, however, are able to fix this.

an input is the original label if the patch condition is false, and the override label associated with the patch otherwise.We report the *average* performance from patching with each of these 4 patches (since these patches are not mutually exclusive)

Yelp-stars consists of all examples in Yelp with the word 'star' present. For this subset, we use a single patch "If review gives 0, 1, 2 stars, then label is negative". Finally, we use the "Women's E-commerce Clothing Reviews" dataset from Zhong et al. (2021) and add two patches: "If review mentions phrases like needs to be returned, then label is negative", "If fit is boxy, then label is negative". For these last two benchmarks, we use the original labels in the dataset. Language patches improve performance by 0.5-1.3 accuracy points (Table 2).

Feature-based Patches. We focus on supplying contextual meanings of colloquial terms in Yelp, for which our model has somewhat low accuracy. To do so, we construct Yelp-Colloquial, consisting of all examples that have the terms {dope, wtf, omg, the shit, bomb}. We then write a patch corresponding to each of these terms (details in Section A). From Table 3, we see an overall improvement of +4.3 accuracy points on Yelp-Colloquial.

4.2 Spouse Relation Extraction

We observe that the Spouse training data is such that the majority of positive examples correspond to opposite-sex couples. We expect this to cause the model to exhibit a strong over-dependence on whether the two entities have different sex.

To test this, we construct a dataset using CheckLists (Ribeiro et al., 2020) in which the sex of the two entities provides no signal for the ground truth label. On this dataset, we observe a large deterioration in model performance compared to the Spouse validation set. To mitigate this, we write a set of 6 override patches P_{spouse}. Importantly, in the patch training phase (details in Table 6 of Appendix A), we maintain the correlation between sex and the label (thus we expect ORIGINAL+PF to not be significantly better than ORIGINAL). From results in Table 4, we observe that patching is able to improve performance by +32.4 accuracy points.

5 Related Work and Discussion

High level language feedback has been used for training fewshot image classifiers (Mu et al., 2020; Andreas et al., 2018) and text classifiers (Zaidan and Eisner, 2008; Srivastava et al., 2018; Camburu et al., 2018; Hancock et al., 2018; Murty et al., 2020).These works are concerned with *reducing* labeled data requirements with language supervision or providing additional knowledge, while our setting involves using language as a *corrective* tool. In a setting more similar to ours, Talmor et al. (2020) introduce a system that can incorporate factual knowledge at test time on synthetic tasks. Cao et al. (2021); Mitchell et al. (2021); Meng et al. (2022) also consider editing factual knowledge in models at inference time. Instead of focusing on factual knowledge, we show that *free-form language* patches can be used to fix bugs on *real data*, to better align model behavior with the end user.

Training models on biased data often leads to various *bugs* that are surfaced when models are tested on challenging, out-of-distribution examples. This work suggests a human-in-the-loop model debugging paradigm that is powerful and expressive (owing to the use of natural language) while requiring no fine-tuning on carefully curated additional data or writing brittle if-else rules.

References

Jacob Andreas, Dan Klein, and Sergey Levine. 2018. Learning with latent language. volume 1.

Oana Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. 2018. E-snli: Natural language inference with natural language explanations. volume 2018-December.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models.

Braden Hancock, Martin Bringmann, Paroma Varma, Percy Liang, Stephanie Wang, and Christopher Ré. 2018. Training classifiers with natural language explanations. volume 1.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual knowledge in gpt. *ArXiv*, abs/2202.05262.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. 2021. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*.

Jesse Mu, Percy Liang, and Noah Goodman. 2020. Shaping visual representations with language for few-shot classification.

Shikhar Murty, Pang Wei Koh, and Percy Liang. 2020. Expbert: Representation engineering with natural language explanations.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

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.

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.

Shashank Srivastava, Igor Labutov, and Tom Mitchell. 2018. Zero-shot learning of classifiers from natural language quantification. volume 1.

Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. 2020. Leap-of-thought: Teaching pre-trained models to systematically reason over implicit knowledge. volume 2020-December.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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 M. 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.

Omar F. Zaidan and Jason Eisner. 2008. Modeling annotators: A generative approach to learning from annotator rationales.

Ruiqi Zhong, Kristy Lee, Zheng Zhang, and Dan Klein. 2021. Meta-tuning language models to answer prompts better. *CoRR*, abs/2104.04670.

A Appendix

A.1 Low-level Implementation Details

We use T5-large (Raffel et al., 2019) as implemented in the transformers library (Wolf et al., 2020) for all our experiments. Both the gating and interpreter heads are separate decoders learnt on top of a shared encoder and all of these components are initialized with the corresponding T5-large weights. An input x is presented as: "Input: x", and any additional context i (c or q) is presented as: "Prompt: i. Input: x". The label probabilities are computed as the conditional probability of the word "positive" and "negative" respectively. We use standard hyperparameters i.e use a learning rate of 1e-4 and use a linear warmup scheduler which ramps up the learning rate from 0 to 1e-4 over 100 steps. To prevent catastrophic forgetting on the original task during patch finetuning, we multi-task learn the patch finetuning loss along with the original task loss.

Patches used for Yelp-Colloquial. We used the following patches for fixing bugs on Yelp-Colloquial:

- "If clothes are described as dope, then clothes are good."
- "If food is described as the shit, then food is good."
- "If food is described as bomb, then food is good."
- "If something is described as wtf, then something is bad."
- "If something is described as omg, then something is good."
- "If food is described as shitty, then food is bad."
- "If something is described as bomb, then something is good."

	Template	Examples
Patches	***Override:*** If aspect is good, then label is positive	e_0: If service is good, then label is positive e_1: If food is good, then label is positive
	Override: If aspect is bad, then label is negative	e_2: If service is bad, then label is negative e_3: If ambience is bad then label is negative
	Override: If review contains words like word, then label is positive	e_4: If review contains words like zubin, then label is positive e_5: If review contains words like excellent, then label is positive
	Override: If review contains words like word, then label is negative	e_6: If review contains words like wug, then label is negative e_7: If review contains words like really bad, then label is negative
	Feature-based: If aspect is described as word, then aspect is good / bad	e_8: If food is described as above average, then food is good e_9: If food is described as wug, then food is bad e_{10}: If food is described as zubin, then service is good e_{11}: If service is described as not great, then service is bad
Inputs	The aspect at the restaurant was adj	The service at the restaurant was really good. e_0, e_3 The food at the restaurant was wug. e_6, e_9
	The restaurant had adj aspect	The restaurant had really bad service. e_7, e_2, e_{11} The restaurant had zubin ambience. e_4, e_{10}
	The aspect1 was adj1, the aspect2 was adj2	The food was good, the ambience was bad. e_1, e_3, e_1 The service was good, the food was not good. e_0, e_1
	The aspect1 was adj1 but the aspect2 was really adj2	The food was good, but the service was really bad. e_7, e_1, e_0 The ambience was bad, but the food was really not wug. e_3, e_9
	The aspect1 was really adj1 even though aspect2 was adj2	The food was really bad even though the ambience was excellent. e_5, e_7, e_8 The food was really zubin, even though the service was bad e_4, e_{10}, e_0

Table 5: Patch and Input templates used for the Patch Finetuning stage for the sentiment analysis task. We divide our patches into 2 categories: *Override* and *Feature-based* (see Section 4 for more details). For each input, we provide examples of some positive and negative patches. The simplistic nature of these templates makes them easy to write without access to additional other data sources or lexicons.

	Examples
Patches	e_0: Person1 divorced Person2 e_1: Person1 has kids with Person2 e_2: Person1 is the parent of Person2 e_3: Person1 and Person2 are engaged e_4: Person1 and Person2 are just friends or coworkers e_5: Person1 or Person2 is not human
Inputs	Person1 and Person2 have a kid named Person3. e_1, e_2 Person1 and Person2 have a kid named Person3. e_2, e_1
	Person1 proposed to Person2. The event was witnessed by Person1's best friend Person3. e_3, e_4 Person1 proposed to Person2. The event was witnessed by Person1's best friend Person3. e_4, e_0
	Person1 has decided to divorce Person2. They have a child named Person3. e_0, e_3 Person1 has decided to divorce Person2. They have a child named Person3. e_2, e_0
	Person1 works at location. e_5, e_0

Table 6: Patches along with a subset of inputs used for the Patch Finetuning stage for the Spouse relation extraction task. For each input, we highlight the two entities and provide examples of some positive and negative patches.