Interpretable and Controllable Language Models

Peter Hase

A thesis submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

May 6, 2024

Committee: Mohit Bansal
Ana Marasović
Sameer Singh
Shashank Srivastava
Sridhar Duggirala

To those who have taught me and whom I have learned from, thank you.

Acknowledgments

First, I must thank my advisor and colleagues for their guidance and support in my Ph.D. This includes my advisor Mohit Bansal, co-authors Shiyue Zhang, Swarnadeep Saha, Miles Turpin, Zhuofan Ying, Thomas Hofweber, Harry Xie, Vaidehi Patil, Xiang Zhou, Stephen Casper, Prateek Yadav, Han Guo, and Archiki Prasad (among others), and internship supervisors Asma Ghandeharioun, Been Kim, Sarah Wiegreffe, Peter Clark, and Srinivasan Iyer. I especially thank Mohit Bansal for his mentorship. I am always inspired by the apprenticeship model that academic science is built on. Under Mohit's guidance, I learned how to make small contributions to our collective understanding, and I absorbed countless little insights about how this process works. For my co-authors, I want to say thank you for working with me. It is an enduring pleasure to carry a project from a fuzzy but exciting idea to a concrete problem to be studied. Flying across the world to present research at a conference is reward in excess to that of working alongside the few people in the world who are as interested in a problem as you are. For my internship supervisors, I want to say thank you for these short and fruitful visits to your research groups. I feel lucky to every year work with someone new, someone I've wanted to work with for a while, and someone who I look up to.

Going back a little bit, I would also like to thank Cynthia Rudin for her mentorship during my undergrad at Duke University. It was through research in her lab that I gained confidence that I wanted to pursue further studies in computer science. Many other professors and students there are responsible for me uncovering so much joy in the study of machine learning, artificial intelligence, and human language. Five years later, this document is testament that I found such joy in this path. Scientific research can be unpredictable, taxing, frustrating, and extremely time-consuming. I am so glad to have found something that I truly love doing.

Outside of the lab, I owe a great deal to many friends, roommates, partners, therapists, life coaches, colleagues, and family members. Whether my research has been going well or poorly, it has always helped to talk about it with others, and I have talked about it *a lot* with others. I particularly want to thank Michael, Tyler, Tom, Thomas, Alex, Joe, Sarah, Serge, Archana, David, Deblina, Grace, Miles, Juan, Kevin, Mark, Kaleigh, Justin, George, and my parents, Steve and Ashley Hase.

Lastly, I am thankful that I was paid to do this. My research has been supported by fellowships from the UNC Graduate School and Google, as well as grants from the NSF and DARPA.

Table of Contents

1 Thesis Statement

Over the years, language models have demonstrated steadily increasing performance on a wide variety of tasks. Their internal reasoning processes remain difficult to interpret, however, and individual behaviors are hard to manipulate once a model has been trained. In this thesis, I present research on interpretable and controllable language models. The main goals of this work are to develop and evaluate tools for (1) explaining why language models produce the outputs they do, and (2) exercising fine-grained control of language model behaviors.

2 Abstract

In this thesis, I present research on interpretable and controllable language models. The immediate goals of this work are to develop and evaluate tools for (1) explaining why language models produce the outputs they do, and (2) exercising fine-grained control of language model behaviors. The broader goal of this research is to make AI systems safer for use in society. The main contributions of this thesis are summarized below.

First, I discuss *Human Evaluation of ML Explanations* and work on evaluation protocols for one-size-fits-all tests for model explanation faithfulness. We find that many popular explanation methods do not help humans build a better mental model of how an AI system works.

Second, I cover *Natural Language Explanation Methods*, extending our previous work in evaluating ML explanations to include natural language explanations generated by LMs. Here, we develop faithfulness tests that rely on model-based evaluation of natural language explanations. We observe that explanations generated by language models often do explain a small amount about the reasoning used to arrive at the task output.

Third, I survey approaches for *Adding Explanation Data to Traditional Discriminative Learning*, which is a mirror problem to evaluating model explanations (i.e., humans should learn from LM explanations and LMs should learn from human explanations). I argue that explanation data is best utilized as model inputs rather than as model targets or a prior over model weights in the context of discriminative learning.

Fourth, I introduce new approaches for *Feature Attribution Methods and Evaluation*, a popular class of explanation methods from our prior studies. I suggest that, rather than assessing feature importance for a model via ablated inputs (i.e. partly missing features), it is preferable to explain only models that sometimes see ablated inputs during training. I also introduce a compute-adjustable search method to search specifically for features that would be sufficient or necessary for a model's test-time prediction.

Fifth, I explore *Model Editing and Belief Graphs for LMs.* Linguistic features are a useful unit of analysis for understanding LM behaviors, and this perspective motivates the above work in explainability, but we may also take an intentional stance toward LMs and explain their behavior in terms of beliefs and desires. In this section, I present methods for measuring, manipulating, and visualizing factual "beliefs" in language models. We show that this is a difficult problem, and past work does not always evaluate all of the relevant cases involved in editing model beliefs.

Lastly, I describe work on *Localization and Editing of Knowledge in LMs.* This topic combines interpretability and model editing. There has been great interest in localizing model knowledge to specific model components, then editing those components to change model knowledge. Our conclusions highlight important subtleties regarding (1) localization findings' utility for model editing, and (2) the use of model editing to substantiate the validity of a localization claim.

In summary, the main goals of this work are to develop and evaluate tools for (1) explaining why language models produce the outputs they do, and (2) exercising fine-grained control of language model behaviors.

3 Introduction

The field of machine learning has reached the point where learning systems now accomplish complicated tasks that we cannot ourselves describe algorithmic solutions to. This progress has been achieved primarily by training neural networks on extensive data in a self-supervised fashion, followed by relatively small adjustments to models using only input-output supervision for specific tasks of interest [21]. The unfortunate reality of the situation, however, is that neural networks execute algorithms unbeknownst to us, described perfectly by long series of matrix multiplications and nonlinearities but lacking any adequate description in plain language. This means that it is hard to verify that a system will produce correct outputs in general and that it does so with an acceptable reasoning process [46, 86]. It also means that it is hard to fix undesirable behaviors in an ML system after it has been trained [38, 242].

In this thesis, I present research on interpretable and controllable language models. The immediate goals of this work are to develop and evaluate tools for (1) explain why language models produce the outputs they do, and (2) exercising fine-grained control of language model behaviors. The broader goal of this research is to make AI systems safer for use in society.

The view of neural network interpretability in this thesis is heavily informed by prior theoretical work on the purpose of explanation [46, 131], faithfulness of explanations to causal mechanisms in models [84], human interpretation of explanations [85], the nature of trust in AI [86], and expository work on the intentional stance by Dennett [40]. The goal of AI interpretability is to develop accurate causal models of AI that invoke concepts that we as humans are familiar with, and we want these causal models to inform us about the (long) chains of events that lead to LM outputs [56]. The aim here is primarily to verify AI systems. We want to check that AI makes decisions in a way that we find reasonable, safe, ethical, value-aligned, etc. (above and beyond what we can confirm with observational test sets) [86]. Notably, this requires us explaining neural networks at the right level of abstraction. Giving someone the weights to a neural network is in a limited sense a complete explanation of how the model works, but it does not communicate anything to a person that would be useful for them predicting when in the future they can trust the system to succeed or fail. In order to judge whether AI makes decisions in reasonable, safe, or value-aligned manner, we need to explain AI decisions at a level that is intelligible to people.

If an AI is not making decisions in a way we like, the next step is to improve the model so that it does. This is broadly known as the problem of model *control* (not to be confused with the control problem in dynamical systems). Better model control may follow from improved understanding of how models work. Increasingly, researchers in interpretability have sought out causal interventions that demonstrate the accuracy of their interpretations and enable a researcher to adjust model behavior in a desirable direction, such as updating mistaken factual knowledge in models [36] and improving the truthfulness of model generations [116]. However, given our ability to supervise models end-to-end, it is not always necessary to develop a precise theory of internal model mechanisms before exploring how to better control their behavior [38, 72]. Sometimes optimizing for a new behavior with demonstrations is all that's needed; it is superfluous to know *why* the model made its previous mistake or *how* the model now reaches the correct solution. Largely for this reason, the problem of model control is its own research area that is historically distinct from machine learning interpretability. Yet, the connection between the two areas will likely grow over time as we gain a better low-level understanding

of internal model mechanisms, which hopefully will enable better model control than trying to teach a model a preferred behavior via input-output demonstrations (or RL) alone.

Below, I describe what each chapter covers. The overall flow of topics goes from interpretability, to model editing (controllability), to a problem at the intersection of interpretability and model editing.

3.1 Overview of Chapters

The thesis proceeds in chapters on different research contributions to the areas of interpretability and model control, specifically in the context of language models. I see language models as a good object of study since we lack complete explanations for their behavior, and human language provides a rich means of interaction with models. The research presented includes new evaluation procedures, modeling methods, interpretability tools, theoretical arguments, visualization techniques, and conceptual work. I briefly summarize the chapters below.

1. **Human Evaluation of ML Explanations** [64]. A variety of methods exist for "explaining" blackbox machine learning models, including feature attribution methods that assign importance scores to parts of the model input, counterfactual explanations that highlight what changes to an input lead the model prediction to change, prototype explanations that refer to previous examples of model behavior to explain current model behavior, influence functions for identifying training data that caused a test-time prediction, and natural language explanations that attempt to explain the reasoning for a decision in plain English (or another language), *inter alia*. These diverse methods are often evaluated in different ways, however, making it difficult to compare which is best. Specifically, it is hard to tell which method is the best in terms of equipping humans with the most accurate understanding of the ML model being explained, a property known as faithfulness [84].

   I developed human subject test protocols for assessing the faithfulness of different kinds of model explanations all within the same framework, and using these protocols I showed that a number of popular explanation methods failed to improve understanding of even small neural networks on relatively simple tasks [64]. While these protocols had been proposed before in the community [46], they had not been implemented with proper controls designed to isolate the effect of ML explanations on human understanding of the models. Our negative results echoed a broader trend within the Explainable AI (XAI) literature that many methods made popular by their theoretical backing or impressive visualizations are nonetheless insufficient for explaining model behavior to users (in our case, undergraduates with at least one math or CS class).

2. **Natural Language Explanation Methods** [68]. The rise of large language models and datasets of textual datapoint explanations enables language models (LMs) to produce textual explanations of their outputs for a given input.

   I conducted the first evaluation of such natural language explanations for their faithfulness to the LMs generating them [69]. To circumvent the need for expensive human faithfulness studies [64], I developed a causal inference method that allowed for a model-based evaluation of explanation faithfulness while controlling for explanations' propensity to "leak" their answer (i.e., restate or heavily imply the model's answer to the question while giving the underlying reasoning for the answer). Using this method, experiments showed the surprising conclusion that language models in the 500M parameter range often could output natural language explanations that indicated the reasoning supporting their answer.

Additionally, I showed that by optimizing for this model-based metric in a multi-agent communication game, agents could sometimes improve the faithfulness of their explanations, giving the reasoning behind their answers without trivially restating their answer within the explanation.

3. **Adding Explanation Data to Traditional Discriminative Learning** [66]. My previous study of LMs generating explanations prompted me to consider whether LMs could learn from human natural language explanations. While similar to the common "instruction-following" task of reinforcement learning and robotics, this task had received relatively little attention within NLP.

   I developed a synthetic task for studying the role of explanation data in supervised machine learning, surveyed and formalized existing graphical models for incorporating this data into a discriminative model, and proposed a retrieval-based model for incorporating explanation data into the modeling process in a scalable way [66]. On synthetic tasks, a 500M-parameter-retrieval based model achieved 95%+ task performance, while its counterpart without access to explanations for training data achieved less than 60% on the task. While results with "real" datasets were inconclusive, I argued that explanation data is best utilized as model inputs rather than as model targets or a prior over model weights, a position later substantiated by a follow-up study with 280B parameter models using in-context learning [106].

4. **Feature Attribution Methods and Evaluation** [71]. Feature attribution, the process of assigning importance scores to parts of an input representing their contribution to a model output, is often evaluated with automated metrics that insert or delete parts of an input according to their importance scores in order to assess which of two different methods rank-ordered the features in a more appropriate way. This commonly accepted evaluation approach based on input ablations exhibited two peculiarities: (1) very few works attempted to optimize for metric performance directly when developing new explanation methods, (2) many works reporting these metrics used them despite the well-known problem that they required passing OOD inputs to a model during evaluation [42].

   I developed a new feature attribution method based on local search that directly optimized for these metrics and significantly improved over popular existing methods like LIME, and I presented a novel theoretical argument for why our interpretation of approaches using input ablations would be skewed by these inputs being OOD to the model, which opened the door to a simple empirical fix to this issue [71]. The result suggested a change of practice: rather than using a linear attribution method on an arbitrary blackbox model, one should preferably explain only models that sometimes see ablated inputs during training (i.e. partly missing features), while obtaining explanations with a compute-adjustable search method to search specifically for features that would be sufficient or necessary for a model's test-time prediction.

5. **Model Editing and Belief Graphs for LMs** [70]. As large language models were shown to store significant relational knowledge about real-world entities, an interest developed in editing this relational knowledge to correct individual factual mistakes in a model's world knowledge.

   I proposed a model-editing method that, relative to past work [38], was able to edit multiple relational facts one after the other, while using editing objectives that respected semantic equivalence between facts (like paraphrase), logical consequences of updated facts (i.e. entailed facts), and existing and unrelated knowledge about entities for which some specific

knowledge was changing [70]. Additionally, I used this method to construct exploratory "belief graphs" for models in the 500M range, representing causal dependencies between model beliefs: if a model thought one sentence was true, and editing the model to think that sentence was false led the prediction to flip from true to false for a different sentence, this would be visualized in our belief graph as an edge between the two statements. Since then, model editing has become an increasingly popular problem area, and I believe belief graphs will be a useful tool for understanding counterfactual dependencies in LMs' models of the world going forward.

6. **Localization and Editing of Knowledge in LMs** [72]. As model editing has surged in its popularity, new methods making progress on the problem have grounded their approaches in low-level interpretability results for LMs. For example, the popular ROME method of Meng et al. [128] based its approach on results from a causal localization technique called Causal Tracing that assigns scores to specific weights in an LM indicating how much they contribute to the model's knowledge of a particular fact. Meng et al. [128] used Causal Tracing to conclude that facts are typically "stored" in mid-layer MLP weights, and therefore these are the best weights in a model for editing.

   I showed that, surprisingly, there is no the relationship between localization and knowledge editing models as measured by representation intervention methods like Causal Tracing and layer-level MLP-editing methods like ROME. In fact, Causal Tracing tells you nothing about which layer should be edited in order to adjust a model's factual knowledge. This finding highlights that where a fact is *currently* stored in a model is different from where an edited version of that fact *could be* stored, and it points to a need for increased caution and rigor regarding claims of (1) localization findings' utility for model editing, and (2) the use of model editing to substantiate the validity of a localization claim.

4 Human Evaluation of ML Explanations

This first contribution focuses on evaluation of explanations of individual ML outputs that aim to explain why a model produces output y for input x. We aim to assess the faithfulness of explanations, and we do so with a proxy for faithfulness called simulatability.

4.1 Introduction

Interpretable machine learning is now a widely discussed topic [172, 46, 120, 61]. While survey papers have not converged on definitions of "explainable" or "interpretable," there are some common threads in the discourse. Commentators observe that interpretability is useful for achieving other model desiderata, which may include building user trust, identifying the influence of certain variables, understanding how a model will behave on given inputs, and ensuring that models are fair and unbiased.

In their review, Doshi-Velez and Kim [46] outline an approach to measuring interpretability. They describe two human-subject tasks that test for a particularly useful property: *simulatability*. A model is simulatable when a person can predict its behavior on new inputs. This property is especially useful since it indicates that a person understands *why* a model produces the outputs it does. Thus, simulatability is a proxy for faithfulness [84], as faithful explanations should improve simulatability (but are not guaranteed to, if users cannot leverage information from the explanations for predicting model outputs on future inputs). The first of the two tasks is termed *forward simulation*: given an input and an "explanation," users must predict what a model would output for the given input. The second is *counterfactual simulation*: users are given an input, a model's output for that input, and an "explanation" of that output, and then they must predict what the model will output when given a perturbation of the original input. The explanation itself is algorithmically generated by a method for interpreting or explaining a model. Simulation tests have been carried out before, but no study to date has isolated the effect of explanations on simulatability [167, 25, 141, 9].

We carry out simulation tests that are the first to incorporate all of the following design choices: (1) separating explained instances from test instances, so explanations do not give away the answers, (2) evaluating the effect of explanations against a baseline of *unexplained* examples, (3) balancing data by model correctness, so users cannot succeed by guessing the true label, and (4) forcing user predictions on all inputs, so performance is not biased toward overly specific explanations. We display our study design in Figure 4.1.

We provide results from high-quality human user tests (with over 2100 responses) that include both forward and counterfactual simulation tasks. Through these tests, we measure explanation effectiveness for five methods across text and tabular classification tasks. Our evaluation includes two existing explanation techniques, LIME and Anchor [165, 167], and we translate two other explanation methods from image recognition models to work with our textual and tabular setups. The first of these is a latent space traversal method, which we term the Decision Boundary approach [94, 177], and the second is a case-based reasoning method, which we term the Prototype method [27]. The final method is a novel Composite approach that combines complementary explanations from each method. Lastly, we also collect subjective, numerical user ratings of explanation quality. Our key findings are:

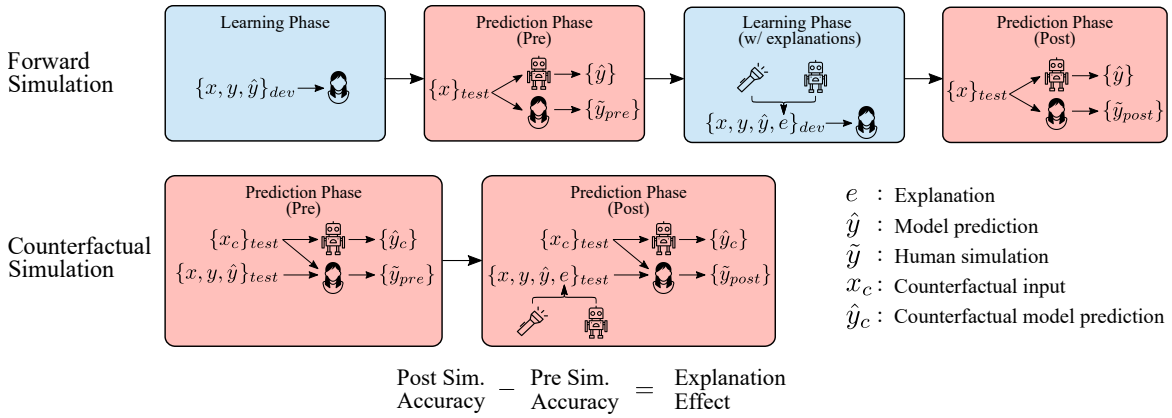


Figure 4.1: Forward and counterfactual simulation test procedures. We measure human users' ability to predict model behavior. We isolate the effect of explanations by first measuring baseline accuracy, then measuring accuracy after users are given access to explanations of model behavior. In the forward test, the explained examples are distinct from the test instances. In the counterfactual test, each test instance is a counterfactual version of a model input, and the explanations pertain to the original inputs.

1. LIME improves forward and counterfactual simulatability in our tabular classification task.
2. Prototype improves counterfactual simulatability across textual and tabular data domains.
3. No method definitively improves forward and counterfactual simulatability together on the text task, though our Prototype and Composite methods perform the best on average.
4. It appears that users' quality ratings of explanations are not predictive of how helpful the explanations are with counterfactual simulation.
5. While users rate Composite explanations as among the best in quality, these combined explanations do not overtly improve simulatability in either data domain.

4.2 Background and Related Work

4.2.1 What Does "Interpretable" Mean?

Survey papers use key terms in varying ways. Rudin [172] draws a distinction between interpretability and explainability, suggesting that a model is interpretable if it performs computations that are directly understandable. Post-hoc explanations, on the other hand, are potentially misleading approximations of the true computations. Gilpin et al. [61] also distinguish between the two concepts, though they define them differently.

In this chapter, we do not distinguish between interpretability and explainability. Rather, we adopt the conceptual framework of Doshi-Velez and Kim [46], who consider interpretability in terms of downstream desiderata one can assess models with respect to. Our terminology is as follows: we will say that *explanation methods* may improve the *interpretability* of a model, in the sense that an interpretable model is *simulatable*.

4.2.2 Explanation Methods

Several taxonomies have been proposed for categorizing methods for interpretability. We organize methods below into the categories of: feature importance estimation, case-based

reasoning, and latent space traversal.

Feature Importance Estimation. Feature importance estimates provide information about how the model uses certain features. Most prominent among these methods are the gradient-based approaches first introduced for vision by Simonyan et al. [186], which Li et al. [114] show may be translated for use with text data. These approaches have since been demonstrated to sometimes behave in counterintuitive ways [1, 99]. A number of alternative methods have been proposed for quantifying feature importance across data domains [99, 126, 199]. In our study, we choose to evaluate two domain-agnostic approaches, LIME and Anchor [165, 167]. These methods use simple models, i.e. sparse linear models and rule lists, to approximate complex model behavior locally around inputs. They show the estimated effects of directly interpretable features on the model's output. For these methods, what is "local" to an input is defined in a domain-specific manner via a perturbation distribution centered on that input.

Case-based Reasoning. Prototype models classify new instances based on their similarity to other known cases. Two works on prototype models for computer vision introduced neural models that learn prototypes corresponding to parts of images [27, 67]. These prototypes are used to produce classifier features that are intended to be directly interpretable.

Latent Space Traversal. These methods traverse the latent space of a model in order to show how the model behaves as its input changes. In a classification setting, crossing the decision boundary may reveal necessary conditions for a model's prediction for the original input. Several methods exist for vision models [94, 177]. To our knowledge no such approach exists for discriminative models of text and tabular data, so we develop a simple method for these kinds of models (described in Section 4.3.4).

4.2.3 Evaluating Interpretability

Here we discuss works involving automatic and human evaluations of interpretability, as well as how we improve on past simulation test design. While human evaluations are useful for evaluating many aspects of interpretability, we restrict our discussion to works measuring simulatability.

Improving Forward Test Design. Forward simulation tasks have been implemented in many different forms, and there is a serious need for consensus on proper procedure here. Doshi-Velez and Kim [46] originally propose that users predict model behavior, given an input and an explanation. With many explanation methods, this is a trivial task *because the explanations directly reveal the output.* For example, LIME gives a predicted probability that indicates the model behavior with high likelihood. We make a number of experimental design choices that give us more reliable estimates of method effectiveness than past studies. (1) We separate the explained instances from the test instances, to prevent explanations from giving away the answers. In three studies, the same data points were used as both explanation and prediction items [141, 25, 9]. (2) We evaluate the effect of explanations against a baseline where users see the same example data points without explanations. No prior evaluation includes this control. (3) Two choices further distinguish our test from that of Ribeiro et al. [167]. We balance data by model correctness, so users cannot succeed simply by guessing the true label, and we force user predictions on every input, so our metrics do not favor overly niche explanations.

Counterfactual Simulatability. Counterfactual simulatability has, to our knowledge, never been measured for machine learning models. While Doshi-Velez and Kim [46] propose asking users to edit inputs in order to change the model outputs, we instead ask users to

predict model behavior on edited versions of data points, as this approach is more scalable than soliciting creative responses.

Relation to Automatic Tests. Prior works have proposed automatic metrics for feature importance estimates [141, 80, 42]. Typically these operate by checking that model behavior follows reasonable patterns on counterfactual inputs constructed using the explanation, e.g., by masking "important" features and checking that a class score drops. Whereas automatic metrics define appropriate model behavior in advance for counterfactual instances generated by a fixed schema, we present a random counterfactual to a human and elicit their prediction of model behavior for that instance. This allows for human validation of model behavior in a broader range of input scenarios than an automatic procedure, where human expectations are given in response to diverse and concrete examples rather than dictated in advance.

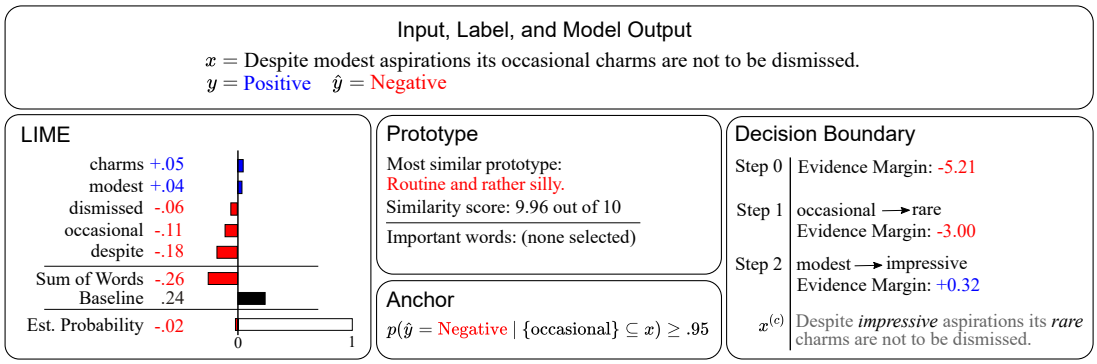


Figure 4.2: Explanation methods applied to an input from the test set of movie reviews.

Subjective Ratings. Hutton et al. [82] measure user judgments of whether word importance measures explain model behavior in a text classification setting. Our rating task is thus similar to theirs; our changes are that we evaluate with a Likert scale rather than forced ranking, using explanation techniques for neural models rather than word importance estimates from a naive Bayes classifier. In another study, users judged image classification explanations on a Likert scale ranging from "no explanation" to "concise explanation" [9]. Whereas this scale focuses on conciseness, we ask users to rate how explanations reveal reasons for model behavior.

4.3 Explanation Methods

In this section, we describe the explanation methods. Example explanations for a test movie review are shown in Figure 4.2. We limit our discussion of LIME and Anchor, since details for these methods can be found in the original papers. Note that LIME, Anchor, and our Decision Boundary method can be used with arbitrary blackbox models. The Prototype method is itself a neural model that also produces an explanation.

4.3.1 LIME

Ribeiro et al. [165] present LIME as a local linear approximation of model behavior. With a user-specified feature space, a linear model is fit to the blackbox outputs on samples from a distribution around an input. We set the number of features to use to 5, and we take class probabilities as our model output. When showing LIME explanations to users, we give them

the selected features with estimated weights, the model intercept, the sum of model weights, and the predicted model output.

4.3.2 Anchor

Ribeiro et al. [167] introduce a method for learning rule lists that predict model behavior with high confidence. With samples from a distribution around an input, they use a PAC learning approach to obtain a rule list. When the rules apply to an input, there is a high probability it will receive the same prediction as the original. The feature space of the rule list is specified by the user. As in the original work, we use individual tokens for our text data, and we use the same learning parameters for each Anchor explanation.

4.3.3 Prototype Model

Prototype models have previously been used for interpretable computer vision [27, 67]. We develop a prototype model for use with text and tabular classification tasks. In our model, a neural network g maps inputs to a latent space, and the score of class c is:

$$f(\mathbf{x}_i)_c = \max_{\mathbf{p_k} \in P_c} a(g(\mathbf{x}_i), \mathbf{p_k})$$

where a is a similarity function for vectors in the latent space, and P_c is the set of protoype vectors for class c. We choose the Gaussian kernel for our similarity function: $a(\mathbf{z}_i, \mathbf{p_k}) = e^{-||\mathbf{z}_i - \mathbf{p_k}||^2}$. The model predicts inputs to belong to the same class as the prototype they're closest to in the latent space. Unlike in Chen et al. [27], we take the max activation to obtain concise explanations.

In lieu of image heatmaps, we provide feature importance scores. What distinguishes these scores from those of standard feature importance estimates is that the scores are prototype-specific, rather than class-specific. We choose a feature omission approach for estimation. With text data, omission is straightforward: for a given token, we take the difference in function output between the original input and the input with that token's embedding zeroed out. In the tabular domain, however, variables can never take on meaningless values. To circumvent this problem, we take the difference between the function value at the original input and the *expected* function value with a particular feature missing. The expectation is computed with a distribution over possible values for a missing feature, which is provided by a multinomial logistic regression conditioned on the remaining covariates.

When presenting prototype explanations, we provide users with the predicted class score, most similar prototype, and top six feature importance scores, provided that score magnitudes meet a small threshold. In the explanation in Figure 4.2, no scores meet this threshold. We set the size of P_c to 40 for our text classification task and 20 for our tabular classification task. For further training and feature importance details, see the Appendix.

4.3.4 Decision Boundary

Joshi et al. [94] and Samangouei et al. [177] introduce techniques for traversing the latent spaces of generative image models. Their methods provide paths that start at input data points and cross a classifier's decision boundary. Such methods may help users see the necessary conditions for the model prediction.

We provide a simple method for traversing the latent space of a discriminative classifier (see example in Figure 4.2). Our algorithm first samples around the original input to get

instances that cross the decision boundary. A counterfactual input is chosen from these by taking the instance with the fewest edited features (tokens or variables), while breaking ties using the Euclidean distance between latent representations. Lastly, we provide a path between inputs by greedily picking the edit from the remaining edits that least changes the model's evidence margin, which is the difference between positive and negative class scores. The explanations we present to users include the input, steps to the counterfactual input, and evidence margin at each step. When the path is longer than four steps, we show only the last four.

4.3.5 Composite Approach

We hypothesize that the above explanations provide complementary information, since they take distinct approaches to explaining model behavior. Hence, we test a Composite method that combines LIME and Anchor with our decision boundary and prototype explanations. We make two adjustments to methods as we combine them. First, we show only the last step of each decision boundary explanation, i.e., the set of changes that flips the prediction. Second, we train our prototype model with its feature extraction layers initialized from the neural task model and thereafter fixed. We do so since we are interested in explaining the task model behavior, and this tactic yields prototypes that reflect characteristics of the task model.

	Text					Tabular				
Method	n	Pre	Change	CI	p	n	Pre	Change	CI	p
User Avg.	1144	62.67	-	7.07	-	1022	70.74	-	6.96	-
LIME	190	-	0.99	9.58	.834	179	-	**11.25**	8.83	.014
Anchor	181	-	1.71	9.43	.704	215	-	5.01	8.58	.234
Prototype	223	-	3.68	9.67	.421	192	-	1.68	10.07	.711
DB	230	-	−1.93	13.25	.756	182	-	5.27	10.08	.271
Composite	320	-	3.80	11.09	.486	254	-	0.33	10.30	.952

Table 4.1: Change in user accuracies after being given explanations of model behavior, relative to the baseline performance (Pre). Data is grouped by domain. CI gives the 95% confidence interval, calculated by bootstrap using n user responses, and we bold results that are significant at a level of $p < .05$. LIME improves simulatability with tabular data. Other methods do not definitively improve simulatability in either domain.

4.4 Experimental Design

In this section, we describe our datasets, task models, user pool, and experimental design.

4.4.1 Data and Task Models

We perform experiments for classification tasks with text and tabular data. The first dataset consists of movie review excerpts [143]. The dataset includes 10,662 reviews with binary sentiment labels, which we split into partitions of 70%, 10%, and 20% for the train, validation, and test sets, respectively. We use the same neural architecture as in Yang et al. [223], limited to use with single sentences. The second dataset is the tabular *Adult* data from the UCI ML repository [48]. This dataset contains records of 15,682 individuals, and the label is whether

	Forward Simulation					Counterfactual Simulation				
Method	n	Pre	Change	CI	p	n	Pre	Change	CI	p
User Avg.	1103	69.71	-	6.16	-	1063	63.13	-	7.87	-
LIME	190	-	5.70	9.05	.197	179	-	5.25	10.59	.309
Anchor	199	-	0.86	10.48	.869	197	-	5.66	7.91	.140
Prototype	223	-	−2.64	9.59	.566	192	-	**9.53**	8.55	.032
DB	205	-	−0.92	11.87	.876	207	-	2.48	11.62	.667
Composite	286	-	−2.07	8.51	.618	288	-	7.36	9.38	.122

Table 4.2: Change in user accuracies after being given explanations of model behavior, relative to the baseline performance (Pre). Data is grouped by simulation test type. CI gives the 95% confidence interval, calculated by bootstrap using n user responses. We bold results that are significant at the $p < .05$ level. Prototype explanations improve counterfactual simulatability, while other methods do not definitively improve simulatability for one test.

their annual income is more than $50,000. We use the same data processing scheme and neural network architecture as Ribeiro et al. [167]. Model accuracies are given in the Appendix.

4.4.2 User Pool

We gathered over 2100 responses via in-person tests with 32 trained undergraduates who had taken at least one course in computer science or statistics.[1] Each user was randomly assigned to one of the ten conditions corresponding to our dataset-method pairs. Once each condition had at least 3 full tests collected, we allocated remaining participants to the Composite method. In order to ensure high quality data, we employed a screening test to check for user understanding of their explanation method and test procedure. Two participants were screened out due to low scores. We also excluded data from a user whose task completion time was extremely low. We paid all users $15 USD per hour. Ten users were tested again with a new dataset and explanation method, giving us a total of 39 user tests. Some users had to exit the experiment before finishing all of the tasks; for data analysis purposes, we consider only task items answered in both Pre and Post test phases.

4.4.3 Simulation Tests

We collect 1103 forward test and 1063 counterfactual test responses in total.

Forward Simulation. This test is represented in Figure 4.1. The test is split into four phases: a learning phase, a Pre prediction phase, a learning phase *with explanations*, and a Post prediction phase. To begin, users are given 16 examples from the validation set with labels and model predictions but no explanations. Then they must predict the model output for either 16 or 32 new inputs, with the number chosen based on user time constraints. Users are not allowed to reference the learning data while in prediction phases. Next, they return to the same learning examples, now with explanations included. Finally, they predict model behavior again on the same instances from the first prediction round. By design, any improvement in user performance in the Post prediction phase is attributable only to the addition of explanations. We show a screenshot of the user testing interface in the Appendix.

[1] We require this advanced background because explanations rely on conditional probabilities, approximations of probabilities, and other quantitative concepts.

	Text Ratings				Tabular Ratings			
Method	n	μ	CI	σ	n	μ	CI	σ
LIME	144	4.78	1.47	1.76	130	5.36	0.63	1.70
Anchor	133	3.86	0.59	1.79	175	4.99	0.71	1.38
Prototype	191	4.45	1.02	2.08	144	4.20	0.82	1.88
DB	224	3.85	0.60	1.81	144	4.61	1.14	1.86
Composite	240	4.47	0.58	1.70	192	5.10	1.04	1.42

Table 4.3: User simulatability ratings by data domain, on a scale of 1 to 7. The mean and standard deviation for ratings are given by μ and σ. The 95% confidence interval for the mean is given by CI, as calculated by bootstrap.

Counterfactual Simulation. Represented in Figure 4.1, this test requires users to predict how a model will behave on a *perturbation* of a given data point. The test consists of Pre and Post prediction rounds, where the only difference between them is the addition of explanations. In both rounds, we provide users with the same 32 inputs from the test dataset (or 16 due to time constraints), their ground truth labels, the model's prediction, and a perturbation of the input. See the Appendix for a description of the perturbation generation algorithm. Users then predict model behavior on the perturbations. In the Post round, users are given the same data, but they are also equipped with explanations of the model predictions for the original inputs. Therefore, any improvement in performance is attributable to the addition of explanations.

Data Balancing. One critical aspect of our experimental design is our data balancing. We aim to prevent users from succeeding on our tests simply by guessing the true label for every instance. To do so, we ensure that true positives, false positives, true negatives, and false negatives are equally represented in the inputs. Likewise, for the counterfactual test, we sample perturbations such that for any instance, there is a 50% chance that the perturbation receives the same prediction as the original input. We confirm user understanding of the data balancing in our screening test.

Data Matching. Within each data domain, all users receive the same data points throughout the experiment. This design controls for any differences in the data across conditions and users, though this does reduce the information added by each test, making our confidence intervals relatively wide given the same sample size. We also match data across prediction rounds in order to control for the influence of particular data points on user accuracy between the Pre and Post phases.

4.4.4 Subjective Simulatability Ratings

Users see explanations in two phases of the tests: the second learning phase in the forward test, and the Post phase of the counterfactual test. In these stages, we ask users to give subjective judgments of the explanations. They rate each method on a 7 point Likert scale, in response to the question, "Does this explanation show me why the system thought what it did?" We explain that users should give higher ratings when the explanation shows the reasons for a model prediction, regardless of whether or not the prediction is correct.

4.5 Results

We report data from a total of 2166 responses from 39 user tests. Each test is for a method and data domain pair, and contains either 16 or 32 task items, with some missingness due to users exiting the study early. In the results to follow, we use the term Change to refer to our estimate of explanation effectiveness: the difference in user accuracy across prediction phases in simulation tests. We perform two-sided hypothesis tests for this quantity by a block bootstrap, resampling both users and unique task items within each condition [50]. In addition, since users complete the first prediction round in either simulation test without access to explanations, we estimate the mean Pre accuracy for each method with a random effects model. This allows us to share information across methods to yield more precise estimates of test performance.

Below, we analyze our experimental results and answer three questions: 1) Do explanations help users? 2) How do users rate explanations? 3) Can users predict explanation effectiveness?

4.5.1 Do explanations help users?

We show simulation test results in Tables 4.1 and 4.2. In Table 4.1, we group results by data domain, and in Table 4.2, we group results by test type.

Our principal findings are as follows:

1. LIME with tabular data is the only setting where there is definitive improvement in forward and counterfactual simulatability. With no other method and data domain do we find a definitive improvement across tests.
2. Even with combined explanations in the Composite method, we do not observe definitive effects on model simulatability.
3. Interestingly, our prototype method does reliably well on counterfactual simulation tests in both data domains, though not forward tests. It may be that the explanations are helpful only when shown side by side with inputs.

These results suggest that: (1) many explanation methods may not noticeably help users understand how models will behave, (2) methods that are successful in one domain might not work equally well in another, (3) combining information from explanations does not result in overt improvements in simulatability. Yet, given our wide confidence intervals, these results should be considered cautiously. It may also be that other methods do in fact improve simulatability, but we have not precisely estimated this. For example, our Prototype and Composite methods do the best on average with text data, though we cannot be confident that they improve simulatability.

Note that estimates of explanation effectiveness could be influenced by users simply regressing to the mean accuracy between prediction rounds. We find that our primary results are not skewed by this phenomenon: the highest estimates of Change in each data domain and test type come from conditions where mean Pre test performance was either above the overall mean or, in one case, within 1.15 percentage points. This potential problem is further mitigated by our random effects model of Pre test performance, which pulls low Pre test means toward the overall mean.

4.5.2 How do users rate explanations?

It seems that, as intended, users rated explanations based on quality rather than model correctness, as we observe no significant difference in ratings grouped by model correctness

(table in Appendix). In Table 4.3, we show user ratings for each method and data domain.

We observe that: 1) ratings are generally higher for tabular data, relative to text data, 2) the Composite and LIME methods receive the highest ratings in both domains, and 3) variance in explanation ratings is quite high, relative to their scale.

4.5.3 Can users predict explanation effectiveness?

We answer this question by measuring how explanation ratings relate to user correctness in the Post phase of the counterfactual simulation test. In this phase, users rate explanations of model predictions for an original input and predict model behavior for a perturbation of that input. If ratings of explanation quality are a good indicator of their effectiveness, we would expect to see that higher ratings are associated with user correctness.

We do not find evidence that explanation ratings are predictive of user correctness. We estimate the relationship via logistic regression with user correctness and ratings. We test models with both absolute ratings and ratings normalized within users, since ratings lack an absolute scale between users. With 640 text data points, we estimate with 95% confidence that moving from a rating of 4 to 5 is associated with between a −2.9 and 5.2 percentage point change in expected user correctness. Using normalized ratings, we find that moving from the mean explanation rating to the first standard deviation is associated with between a −3.9 and 12.2 percentage point change. With 515 tabular data points, we estimate that a change in rating from 4 to 5 is associated with between a −2.6 and 5.3 percentage point change in expected user correctness. Of course, we have not shown that there is no association. Yet it's important to note that if there is no relationship between user ratings and simulatability, then simply querying humans about explanation quality will not provide a good indication of true explanation effectiveness.

4.6 Qualitative Analysis

When do explanations succeed at improving user accuracy, and when do they fail at doing so? Below, we present example counterfactual test items, and we analyze how the explanations may have pointed to the reasons for model behavior.

4.6.1 Explanation Success Example

For the example below, 5 of 6 Post test responses for Prototype and LIME were correct that the model output did not change for the counterfactual, up from 3 of 6 in the Pre test.

Original ($\hat{y} = pos$): "Pretty much sucks, but has a funny moment or two."

Counterfactual ($\hat{y}_c = pos$): "*Mostly just bothers*, but *looks* a funny moment or two."

LIME identifies "funny" and "moment" as positive words, with weights adding to 1.04 after including the baseline. The notable negative word is "sucks" ($w = -.23$), which changes to a similar word ("bothers"). All together, LIME suggests the prediction would stay the same since the positive words are unaffected and the only important negative word has a similar substitute.

The **Prototype** model gives the most activated prototype: "*Murders by Numbers* isn't a great movie, but it's a perfectly acceptable widget." It identifies "but" and "funny" as important words for the prototype's activation. The counterfactual is still similar to the prototype in key ways, suggesting the prediction would not change.

4.6.2 Explanation Failure Example

For the item below, only 7 of 13 responses were correct after seeing explanations, with no method improving correctness relative to the Pre test accuracy. Users needed to predict that the model prediction changed to negative for the counterfactual.

Original ($\hat{y} = pos$): "A bittersweet film, simple in form but rich with human events."

Counterfactual ($\hat{y}_c = neg$): "A *teary* film, simple in form but *vibrant* with *devoid* events."

Anchor gives one word as a condition for the original positive prediction: "bittersweet." But what happens when "bittersweet" changes to "teary"? The Anchor explanation does not actually apply to this counterfactual scenario, as its probabilistic description of model behavior is conditioned on the word bittersweet being present.

LIME gives five words, each with small weights ($|w| < .04$), while the baseline is .91. This suggests that LIME has failed to identify features of the input that are necessary to the model output. Among these five words are the three that changed between sentences, but we would not suspect from their weights that the changes made in the counterfactual would flip the model output.

Decision Boundary gives a counterfactual input with a negative prediction: "A *sappy* film, simple in *link* but *unique* with human events." However, it is difficult to tell whether this counterfactual sentence is similar in decision-relevant ways to the proposed counterfactual sentence.

The **Prototype** model gives the activated prototype for the original prediction: "Watstein handily directs and edits around his screenplay's sappier elements...and sustains *Off the Hook*'s buildup with remarkable assuredness for a first-timer." No important words are selected. We are left without a clear sense of why this was the most similar prototype and what circumstances would lead to the model output changing.

These examples reveal areas for improvement in explanations. Better methods will need to distinguish between sufficient and necessary factors in model behavior and clearly point to the ways in which examples share decision-relevant characteristics with new inputs. Further, they must do so in the appropriate feature space for the problem at hand, especially for models of complex data.

4.7 Discussion

Forward Tests Stretch User Memory. We show users 16 examples during learning phases but do not allow them to reference the learning data during prediction phases. Reasonably, some users reported that it was difficult to retain insights from the learning phase during later prediction rounds.

Generating Counterfactual Inputs. It may be difficult to algorithmically construct counterfactual inputs that match the true data distribution, especially when seeking to change the model prediction. Our text counterfactuals are regularly out of the data distribution, in the sense that no real movie review would exhibit the word choice they do. We still consider these inputs to be of interest, for the reason that a model will handle such inputs in *some* manner, and we aim to assess all possible model behaviors in our analysis.

Fair Comparison of Explanation Methods. In our forward simulation treatment phases, we provide users with 16 explained instances and allow them to read at their own pace. We control for the number of data points between methods, but one could instead control for user exposure time or computation time of explanation generation. Further, for LIME and

Anchor, there are approaches for efficiently covering the space of inputs with a limited budget of examples [167]. We opt not to use them since 1) they are not applicable to the Decision Boundary and Prototype methods, which lack a similar notion of coverage, and 2) it is not clear whether these approaches are useful for text data. It may be that when using such approaches, LIME and Anchor perform better on forward simulation tasks.

4.8 Conclusion

Simulatability metrics give a quantitative measure of interpretability, capturing the intuition that explanations should improve a person's understanding of why a model produces its outputs. In this chapter, we evaluated five explanation methods through simulation tests with text and tabular data. These are the first experiments to fully isolate the effect of algorithmic explanations on simulatability. We find clear improvements in simulatability only with LIME for tabular data and our Prototype method in counterfactual tests. It also appears that subjective user ratings of explanation quality are not predictive of explanation effectiveness in simulation tests. These results suggest that we must be careful about the metrics we use to evaluate explanation methods, and that there is significant room for improvement in current methods.

5 Natural Language Explanation Methods

This second contribution extends my previous work on faithfulness evaluation of model explanations to cover natural language explanations that are generated by language models to explain their own answers to problems. Natural language explanations are especially useful since they are a natural medium for communicating with people, and natural language explanations are much more flexible than previous "local" explanation methods (they can communicate more complex reasoning processes). A potential limitation of these explanations is that it may be difficult to communicate certain decision-making processes in natural language (e.g. perception, or decisions relying on concepts that are not intuitive to people). Where explanations are limited, we will need also need to employ other mechanisms for ensuring that models are working as intended, such as testing and redteaming.

5.1 Introduction

Deep neural models have achieved impressive success in many areas. However, their interpretability and explainability have remained broadly limited. To make neural models more interpretable, previous works have proposed methods for explaining model decisions, e.g., through various feature importance estimates [77, 165] or model-generated natural language (NL) [76, 100]. Early work on generating NL explanations focused on providing explanations that were both descriptive of an image and discriminative as labels [76]. Since then, a variety of datasets have been collected with free-form human generated explanations accompanying each data point [22, 100, 230, 210, 163]. Models have been proposed for these datasets with two aims: (1) to teach models how to explain their own decisions in natural language, by offering demonstrations of humans doing this, and (2) to increase model accuracy on the task, by making use of additional information in human explanations.

Past works have proposed varying methods for generating NL explanations, which can be represented by distinct graphical models. In our work, we explore four graphical models, shown in Figure 5.1. Each model generates explanations in either a reasoning (Re) or rationalizing (Ra) mode, where rationalizing models explicitly condition explanations on a label and reasoning models condition only on the input. Approaches further differ by whether they use explanations as inputs to a task model (ST) or as additional supervision in a multi-task framework (MT). Two of these models are drawn from prior works: MT-Ra [22] and ST-Re [163]. We introduce ST-Ra and also test MT-Re as the reasoning counterpart to MT-Ra. To fairly compare the approaches, we implement each graphical model with a state-of-the-art pretrained T5 model [162] (details in Section 5.3).

Generated explanations have typically been evaluated by automatic measures of similarity with human explanations. Most commonly, phrase-matching metrics such as BLEU [144] are used. In a few cases, human evaluations have been employed, also primarily to assess the similarity of explanations to what humans would say. On the basis of these evaluations, past works have suggested their models produce "justifications of its classification decisions" [22] and "explanations to justify its predictions" [163]. While useful starting points, we argue that these evaluations are insufficient, because they do not necessarily indicate anything about a model's true internal reasoning. For example, suppose the ground-truth label is A, while a model predicts B; a higher BLEU score will be observed when the model gives an explanation

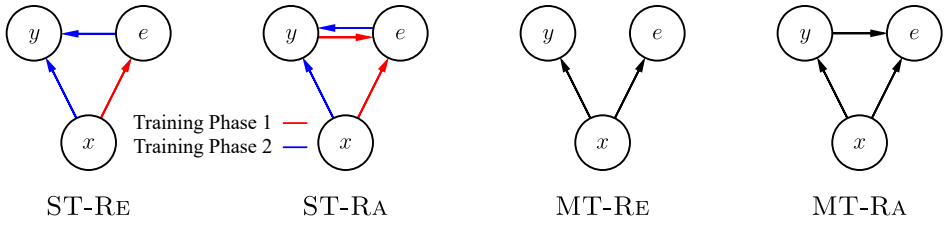


Figure 5.1: Graphical models representing varying roles of explanations, where the task input is denoted by x, task output by y, and explanation by e. We introduce a new rationalizing model, ST-RA, while also testing a reasoning multi-task model, MT-RE, and two other methods from past works [22, 163].

to support human label A, instead of model prediction B. This point is substantiated by Jacovi and Goldberg [84], who advocate for evaluations of explanation *faithfulness* rather than *plausibility*.

To resolve this evaluation problem, we introduce the *leakage-adjusted simulatability* (LAS) metric, which is better suited for identifying when explanations actually support model behavior. LAS scores combine two key mechanisms: they measure *simulatability*, which reflects how well an observer can use model explanations to predict the model's output, while controlling for explanation leakage, which occurs when explanations directly leak the output. This metric is inspired by prior work on model interpretability [46, 65], but to date no simulatability analysis has been carried out for NL explanations. We automate our evaluation by using a pretrained language model as the observer, serving as a proxy for a human. Using LAS scores, we evaluate model-generated as well as human explanations for CommonSenseQA (CQA) [200, 163] and SNLI [20, 22] tasks. We provide two human evaluations to validate our model-based approach. The first is an expert simulatability evaluation, where we manually play the role of the simulator in our LAS metric computation. The second is a subjective ratings task, where we collect data from Mechanical Turkers.

Lastly, since we propose a metric for evaluation, the question naturally arises of whether an objective besides standard language modeling is better suited to improving explanations under this metric. While our formulation of LAS is not differentiable, we present a proxy objective that involves using a simulator during training. This training procedure is neatly interpreted as a multi-agent game. Agents share a common objective, which is for the simulator to predict the task model's output using the explanation it receives, but we penalize agents for pursuing the trivial solution, i.e., restating outputs without giving additional information.

We summarize our key results as follows:

1. We introduce the LAS score, which captures how explanations improve simulatability while controlling for direct label leakage, and we use it to evaluate four generative models.
2. We show that our LAS scores provide a deeper understanding of explanation effectiveness than metrics like BLEU and discuss their relationship with our expert simulation analysis and crowdsourced human quality ratings.
3. We find that our ST-RA approach achieves nearly human-level LAS scores, and that rationalizing models outperform reasoning models.
4. We observe no trade-off between interpretability and accuracy, though this also means that existing methods struggle to learn from human explanations.
5. In a multi-agent game, we show that optimizing explanations for simulatability and

penalizing trivial explanations can improve LAS scores in some settings.

5.2 Related Work

Generating Natural Language Explanations. Early work on this topic proposes to generate explanations for images that are descriptive as captions and discriminative as labels [76]. However, they seek to explain the image's label rather than a classifier's output. Ling et al. [119] introduce induction approaches for solving math problems and generating explanations of solutions. Two works focus on multi-modal problems, explaining visual question answering [147] and self-driving car decisions [100]. A few recent works focus on explanations for language understanding tasks. Camburu et al. [22] introduce e-SNLI, extending the SNLI dataset [20] with free-form human explanations, and they provide an LSTM-based model that jointly predicts labels and generates explanations, shown by MT-Ra in Figure 5.1. Rajani et al. [163] propose the CoS-E dataset, collecting human explanations for CommonSenseQA [200], and they introduce the CAGE model, depicted as ST-Re in Figure 5.1. We build on these works by evaluating both ST-Re and MT-Ra as well as models we introduce, ST-Ra and MT-Re. We implement each graphical model with strong pretrained-T5 models, and for completeness, we also test methods with GPT2 and BERT (results in Appendix B.3) [160, 41].

Evaluating Explanations. There is now a wealth of work on evaluating explanations of machine learning models [165, 46, 80, 84]. For NLP tasks, past works focused on extractive rather than generative explanations [141, 42]. Such methods extract parts of the model input that are important to the output according to some criterion. However, they are not suited to evaluate NL explanations that are not part of the input, which motivates our new simulatability metric.

Measures of similarity between model-generated and human explanations are used to evaluate nearly every method introduced above, with BLEU being the most common [76, 119, 147, 100, 22, 163]. In a few cases, human evaluations are employed for similar purposes [76, 147, 100]. While these evaluations provide a good starting point, they do not support previous claims that explanations show the reasons for model behavior because they evaluate plausibility rather than faithfulness. We introduce a leakage-adjusted simulatability metric (LAS) in response to this issue. As observed by Jacovi and Goldberg [83], faithfulness and simulatability are closely related, but simulatability primarily captures causal attribution of explanations and not necessarily social attribution. Simulatability-based evaluations have been conducted before [166, 65], but we are the first to consider NL explanations and employ model-based controls for label leakage. Two contemporaneous works also explore relevant topics. Narang et al. [139] train a T5 model to generate explanations in a set-up analogous to our MT-Ra setting. They also notice the shortcomings of BLEU and collect binary human ratings of whether explanations "support" model outputs. Kumar and Talukdar [102] introduce label-specific versions of the method in Rajani et al. [163], one of which shares the graphical structure of our ST-Ra model. However, their evaluation focuses on whether humans can recover ground truth labels from generated explanations alone, which they term "explanation accuracy." Given these interesting concurrent works, our contributions are still distinguished by our joint focus on (1) simulatability-based evaluation, (2) controls for explanation label leakage, and (3) comparison of several distinct graphical models.

Multi-Agent Communication. The most relevant work to our multi-agent game concerns discrete communication policies with natural language or artificial protocols grounded in NL. Lazaridou et al. [107] ground a communication protocol in natural language via an auxiliary

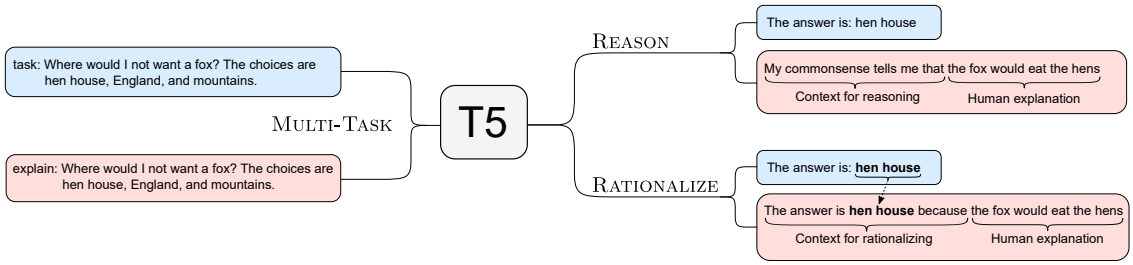


Figure 5.2: Inputs and outputs for the T5 Multi-task framework. In the reasoning mode, explanations are not conditioned on the model's prediction, whereas in the rationalizing mode they are dependent on the model output.

image classification task. In concurrent work, Lazaridou et al. [108] learn NL protocols for an image-based reference game by pretraining with image captions. While our approach shares the premise that language use is goal-oriented, we optimize full explanations of model outputs rather than descriptions of images in reference games. Another contemporaneous work optimizes for simulatability in a multi-agent setting, but they use extractive rather than generative explanations [204].

5.3 Modeling With Explanations

In this section, we delineate our baseline model and the four graphical models we study. The graphical models are depicted in Figure 5.1. We also summarize the key features of each approach in Table 5.1. We show examples of task inputs and outputs along with explanations in Table 5.2. In general, we initialize models from T5-Base, which is a Transformer-based sequence-to-sequence model, pretrained with a large-scale English corpus.

Baseline. The baseline model simply predicts y given x. We adopt the approach of Raffel et al. [162] for fine-tuning to multiple-choice problems, which is to maximize the likelihood of correct answer tokens conditioned on the task inputs. To produce predictions, however, we compute a likelihood for each answer choice and select the most likely choice, rather than sampling text. SNLI also fits into this framework by taking the three relations as answer choices.

ST-Re. Rajani et al. [163] proposed a Commonsense Auto-Generated Explanation (CAGE) framework for CQA, with a two-phase training procedure: first, with human explanations as supervision, a model is trained to generate explanations given task inputs; then generated explanations are supplied with task inputs to a classifier that performs the task. We represent this framework in Figure 5.1, where we term it ST-Re to fit within our data-agnostic model taxonomy. ST stands for serial-task (from the separate training phases) and Re for the reasoning explanation generation. While originally composed of GPT and BERT, we implement this approach with two separate T5 models.

ST-Ra. We extend the ST-Re approach to operate in a rationalizing mode (shown in Figure B.1 in Appendix). Instead of generating one explanation per example, we propose to generate explanations for each possible task output, conditioned on that output. Then, we give each answer choice its own input sequence, which includes the task input and an

Method	Task Set	Conditioning
T5-Base	Single-task	-
ST-RE	Serial-task	$e\|x$
ST-RA	Serial-task	$e\|x, y$
MT-RE	Multi-task	$e\|x$
MT-RA	Multi-task	$e\|x, y$

Table 5.1: The graphical models and baseline we evaluate. MT and ST refer to multi-task and serial-task, while RE and RA refer to reasoning and rationalizing.

	Model		Human	
Input, Output, and Explanation	Leaking?	LAS	Leaking?	LAS
Question: Marathoners feel fatigued after running twenty six miles, but some that have pushed them self too hard might be prone to what? Choices: A. passing out; B. death; **C. exhaustion** STRA explanation: if you are running too hard, you are likely to be exhausted.	Yes	1	Yes	1
Question: When are people buying products more? Choices: **A. economic boom**; B. disagreements; C. being able to use HUMAN explanation: being able to use.	No	-1	No	-1

Table 5.2: Two example data points from CQA with HUMAN or STRA label (**bold** in text) and explanation. We give leakage indicators and example-level LAS scores from both model-based (T5) and human simulators (see Section 5.4). More examples can be found in Table B.1.

explanation supporting that answer choice. Finally, a classifier scores each input and output sequence.

Instead of maximizing the likelihood of correct answer tokens, we find that a new learning objective is necessary for training the task model. We renormalize the decoder likelihoods of each answer choice a_i given the encoder input s_i. With the set of encoder sequences S and answer choices A, we define the probability of each answer choice as:

$$p(a_i|A, S) = \frac{p(a_i|s_i)}{\sum_{a_i \in A, s_i \in S} p(a_i|s_i)}$$

Then we maximize the likelihood of the correct answer choice.

MT-Re. The alternative to using explanations as task model inputs is to use them as supervision in a multi-task framework. As a counterpart to ST-RE, we test a reasoning multi-task model, where explanations are conditioned only on the task input (shown in Figure 5.2). We use a single task-specific word prepended to the input sequence so that the encoder hidden states will be tailored to either the task or explanation generation. For this model, the multi-task learning objective mixes a label prediction loss $\mathcal{L}_{task}$ (for the task itself), and a language modeling loss $\mathcal{L}_{LM}$ (for explanation generation):

$$\mathcal{L}_{MT} = \alpha \mathcal{L}_{task} + (1 - \alpha)\mathcal{L}_{LM},$$

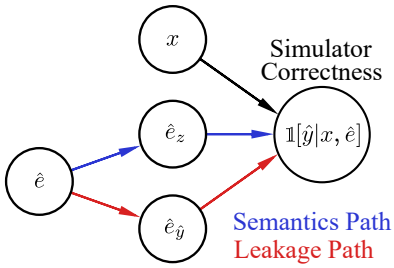


Figure 5.3: Causal diagram of model simulation. The simulator prediction's correctness, $\mathbb{1}[\hat{y}|x,\hat{e}]$, is influenced by three variables: (1) the task model input, (2) the model explanation's semantic content, $\hat{e}_z$, and (3) whether the explanation leaks the model output, $\hat{e}_{\hat{y}}$

where α is the mixing ratio to be tuned on development set. We reach a value of $\alpha = .5$ on both datasets when tuning for task accuracy.

MT-Ra. Represented in Figure 5.2, MT-Ra is a multi-task model where explanations are conditioned on the model output. This approach originates in Camburu et al. [22], where it is introduced as an LSTM-based model. As above, we use a task mixing weight of $\alpha = .5$ for both datasets.

5.4 LAS: Leakage-Adjusted Simulatability

While many motivations drive humans' explanations for their behavior, we consider one central purpose to be helping others understand one's internal reasoning. This notion is captured by the concept of simulatability [46]. A model is simulatable to the extent that an observer, or simulator, can predict its outputs. The simulator can be either a human or a learned model; we will consider both settings. From this perspective, one might use the simulator's accuracy to measure explanation quality. With task inputs $X = \{x_i\}$, model outputs $\hat{Y} = \{\hat{y}_i\}$, model explanations $\hat{E} = \{\hat{e}_i\}$, simulator correctness as $\mathbb{1}[\hat{y}_i|x_i,\hat{e}_i]$,[1] the accuracy is defined as:

$$\text{Acc}(\hat{y}|x,\hat{e}) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}[\hat{y}_i|x_i,\hat{e}_i]$$

However, this measure fails to distinguish between different ways in which the simulator can successfully predict the task model output, as shown in the causal diagram in Figure 5.3. We suggest that the simulator's success does not reflect explanation quality when (1) the simulator can guess behavior correctly from the input x alone, or (2) the explanation $\hat{e}$ directly restates the task model output, i.e., leaking the label to the simulator. What we are truly looking for in explanations is that they provide semantic content that informs the simulator of the task model's output in the context of its input. Note that we do not think label leakage means an explanation is bad. Explanations will leak more often than not, as human explanations leak about 85% of the time for CoS-E and about 97% of the time for e-SNLI (estimated by T5 simulator). Instead, we think the more important aspect is to evaluate the explanation's semantic content. For examples of leaking and nonleaking explanations, see Table 5.2.

[1]For the remainder of the paper, we use the indicator function in this way to describe the correctness of predictions, which is a slight abuse of notation for the sake of brevity.

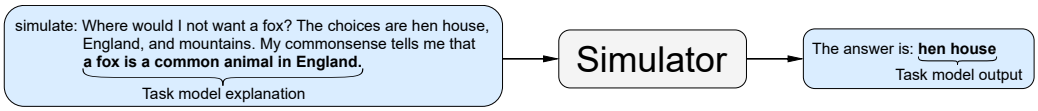


Figure 5.4: A simulator model predicts a task model output, given its input and a model-generated explanation.

To deal with issue (1) above, we introduce an input-only baseline and measure the effect of an explanation on simulatability as $\mathbb{1}[\hat{y}|x,\hat{e}] - \mathbb{1}[\hat{y}|x]$. To resolve the issue (2), we propose to control for a label leaking variable, which has the effect of blocking that causal pathway [150]. We do so by using a proxy variable for label leakage, which is an indicator variable for if the simulator can predict $\hat{y}$ solely from $\hat{e}$. The correctness of this prediction suggests that the explanation gives away the answer directly. With this approach, we can estimate explanations' leakage-controlled effect on simulatability by (1) grouping data by the level of explanation label leakage, (2) computing the average difference $\mathbb{1}[\hat{y}|x,\hat{e}] - \mathbb{1}[\hat{y}|x]$ within each leakage group, and (3) taking the raw average of the effect across groups (to avoid favoring the larger subset). Note that there are only two levels of label leakage, $\mathbb{1}[\hat{y}|\hat{e}] = 1$ (leaking) and $\mathbb{1}[\hat{y}|\hat{e}] = 0$ (nonleaking), and we use model correctness rather than probabilities since T5 probabilities are uncalibrated.

Now with simulator correctness as $\mathbb{1}[\hat{y}_i|x_i,\hat{e}_i]$ or $\mathbb{1}[\hat{y}_i|x_i]$, and our leakage indicator as $k_i = \mathbb{1}[\hat{y}_i|\hat{e}_i]$, we write our Leakage-Adjusted Simulatability (LAS) metric as:

$$\text{LAS}_0 = \frac{1}{n_0} \sum_{i:k_i=0} \left(\mathbb{1}[\hat{y}_i|x_i,\hat{e}_i] - \mathbb{1}[\hat{y}_i|x_i]\right)$$

$$\text{LAS}_1 = \frac{1}{n_1} \sum_{i:k_i=1} \left(\mathbb{1}[\hat{y}_i|x_i,\hat{e}_i] - \mathbb{1}[\hat{y}_i|x_i]\right)$$

$$\text{LAS} = \frac{1}{2}(\text{LAS}_0 + \text{LAS}_1)$$

where n_0 and n_1 are the number of examples in nonleaking and leaking groups respectively. We use a pretrained T5-Base model as a proxy for a human simulator (depicted in Figure 5.4). This approach has the advantage of scaling across large datasets with uniform quality in predictions, and, as described in Section 5.5, it enables directly optimizing explanations for simulatability. We validate this choice of proxy with two human subject experiments (see Section 5.6.2). Simulator models are trained with task model outputs as labels and x and $\hat{e}$ combined into input sequences. In order to make sure the simulator makes good use of both x and $\hat{e}$, we randomly dropout either x or $\hat{e}$ from the input during training. When testing, the simulator's correctness on each example is $\mathbb{1}[\hat{y}_i|x_i,\hat{e}_i]$, and we obtain $\mathbb{1}[\hat{y}_i|x_i]$ and $\mathbb{1}[\hat{y}_i|\hat{e}_i]$ by dropping $\hat{e}_i$ or x_i from the input.

We will compare LAS and $\text{Acc}(\hat{y}|x,\hat{e})$ for explanations from the models introduced above as well as human explanations. We discuss the relationship with human experiments for both metrics in Section 5.6.2. In analysis to follow, we will also refer to example-level LAS scores, which are given as $\mathbb{1}[\hat{y}|x,\hat{e}] - \mathbb{1}[\hat{y}|x]$ and take values -1, 0, or 1 (see Table 5.2 for examples). Lastly, while we use a binary proxy for label leakage, a continuous measure can be obtained from $p(\hat{y}|\hat{e})$. After calibrating the simulator probabilities via Platt scaling [153], we perform a sensitivity analysis of our results for bin counts between 2 and 100: LAS estimates typically vary by less than 1 point across bin counts. For further details, see Appendix B.2.1.

5.5 Multi-Agent Explanation Optimization

In this section, we explore an approach to optimizing explanations for LAS, rather than just relying on a standard language modeling loss to produce explanations. The approach is naturally framed as a multi-agent game. Note that we do not aim to improve model accuracy or explanations' BLEU scores in these experiments.

	SNLI			CQA		
Explanations	LAS Score (CI)	Acc($\hat{y} \mid x, \hat{e}$)	BLEU	LAS Score (CI)	Acc($\hat{y} \mid x, \hat{e}$)	BLEU
Human	4.31 (1.97)	98.36	-	14.73 (3.84)	90.11	-
MT-Re	-15.83 (1.81)	93.72	19.54	-7.07 (3.59)	81.05	6.33
MT-Ra	4.34 (4.12)	99.97	19.41	-1.31 (4.04)	92.31	5.43
ST-Re	0.55 (0.87)	93.87	19.96	3.76 (1.83)	82.21	7.12
ST-Ra	6.74 (4.53)	99.84	20.94	10.32 (3.39)	88.53	7.14
Multi-Agent						
MT-Re (SGD)	-10.08 (1.72)	94.14	16.74	-6.32 (3.27)	76.63	4.44
MT-Ra (SGD)	3.03 (4.72)	99.89	16.61	3.08 (3.79)	87.68	4.43
MT-Re (RL)	-10.80 (1.51)	93.45	15.41	-5.04 (3.55)	84.00	2.15
MT-Ra (RL)	-0.61 (0.45)	93.05	9.83	-9.15 (2.95)	77.47	3.54

Table 5.3: Evaluations of human and model-generated explanations by LAS score, overall simulator accuracy, and BLEU. 95% confidence intervals as calculated by bootstrap are shown in parentheses [50].

In our game, there are two agents. The first is a **task model** that predicts labels and generates explanations jointly. Here, we use MT-Re or MT-Ra. The second agent is a **simulator** model that predicts the task model's output $\hat{y}_i$ given its explanation $\hat{e}_i$ and the model input x_i, matching the previous simulation format shown in Figure 5.4. These two agents are jointly trained during the multi-agent training procedure. The objective of the simulator is the same as discussed in the above section, which is to predict $\hat{y}_i$ given x_i and $\hat{e}_i$, and we randomly dropout x_i or $\hat{e}_i$ to ensure they are both being used. As in Section 5.3, the task model learns to perform the task (minimizing $\mathcal{L}_{task}$) and generate explanations (minimizing $\mathcal{L}_{LM}$) via supervision from ground-truth labels and human explanations. Here, the task model also tries to minimize the simulator's loss through its explanations. The chief computational challenge with this approach is that explanations are sampled by greedy decoding, and thus the loss is not differentiable with respect to the task model. We explore two optimization methods circumventing this issue: Approximate SGD via argmax relaxation [127] and REINFORCE [218]. Our aim is for explanations to better communicate the task model's reasoning process, without adopting the trivial solution, i.e., directly stating its output. Thus while we optimize explanations for simulatability, we also penalize label leakage, which we formalize below. Note that the task model's predictions are not optimized to agree with the simulator; only its explanations are optimized.

Approximate SGD. With a simulator model p_ϕ, the simulatability loss term for explanations is

$$\mathcal{L}_{exp} = -\frac{1}{N}\sum_{i=1}^{N}\big(\alpha \log p_\phi(\hat{y}_i|x_i, \hat{e}_i) - (1-\alpha)\log p_\phi(\hat{y}_i|\hat{e}_i)\big)$$

where α is a mixing weight between terms. To differentiate through the greedy decoding for explanation sampling, we use one half of the Gumbel-Softmax trick [127]. During the forward pass in training, the argmax is used as normal, while during the backward pass, we relax the argmax to a softmax with temperature 1 for purposes of computing gradients.

Reinforce. Our second approach is to use the REINFORCE RL algorithm proposed by Williams [218]. Here we take the simulator's output probabilities as a reward for the task model. Now with the same goals as above, we define the reward for x_i as $r_i = \alpha p_\phi(\hat{y}_i|x_i, \hat{e}_i) - (1-\alpha)p_\phi(\hat{y}_i|\hat{e}_i)$. Then, the $\mathcal{L}_{exp}$ for task model p_θ is defined as:

$$\mathcal{L}_{exp} = \frac{1}{N}\sum_{i=1}^{N} -r_i \log p_\theta(\hat{e}_i|x_i, \hat{y}_i)$$

Finally, with either method, the full learning objective of the task model is $\mathcal{L}_{TaskModel} = \lambda_1\mathcal{L}_{task} + \lambda_2\mathcal{L}_{LM} + \lambda_3\mathcal{L}_{exp}$. The tuning procedure and values for mixing weights are given in Appendix B.1.5.

5.6 Experimental Results

Here, we discuss experiments conducted with each method using two (English) datasets: The first is the CommonSenseQA (CQA) dataset of Talmor et al. [200], with explanations collected by Rajani et al. [163] to make a combined CoS-E dataset (examples in Table 5.2). We use the Version 1.0 of this dataset, since it has higher quality explanations than Version 1.1.[2] CQA has approximately 8k/1k/1k train/dev/test data points, while NLI has roughly 549k/10k/10k train/dev/test points. Note that, in the main paper, we report results using 10% of the SNLI training data, due to computational demands of tuning multi-task models (1 week for convergence with 100% data), and we report CQA dev results since human explanations are not available for test data. See Tables B.6 and B.8 in the Appendix for results for CQA test data and SNLI with full training data, where we confirm the results discussed here. For the model selection procedure and further training details, see Appendix B.1.3, and for robustness checks of LAS scores across seeds and simulator architectures, see Appendix B.2.2.

5.6.1 Automatic Explanation Evaluation

Below we describe key conclusions from our evaluation of leakage-adjusted simulatability (LAS), and we show results alongside overall simulator accuracy $\text{Acc}(\hat{y}|x, \hat{e})$ and BLEU in Table 5.3.

Humans vs. Models. Some models do achieve roughly human-level LAS scores for CQA and NLI. First, we find that human explanations are helpful to models: we estimate that explanations improve humans' simulatability by 4.31 percentage points for SNLI and by 14.73 points for CQA. Our ST-Ra method performs similarly to humans on both datasets. On SNLI, MT-Ra also achieves about human performance. We emphasize that this does not mean these models match human explanations in every respect. Rather, the semantics of the explanations have a similar effect on simulator accuracy as human explanations in our experimental settings. Additionally, we note that scores across datasets are not directly comparable since they depend on the underlying difficulty of the task.

[2] In Version 1.1, about 20% of explanations belong to a small set of duplicates unrelated to the data point. See https://github.com/salesforce/cos-e/issues/2.

Leakage	Human		**LAS**	Human		
Model	0	1	Model	-1	0	1
0	127	87	-1	23	56	6
1	45	341	0	29	278	49
			1	5	104	50

Table 5.4: Correlation between model-based and human variables resulting from the expert simulation analysis. For the leakage variable, Spearman's rank correlation is $\rho = 0.53$ ($p <$ 1e−15). For the example-level LAS, the rank correlation is $\rho = 0.29$ ($p <$ 1e−12).

Re vs. Ra. Rationalizing models outperform their reasoning counterparts on both datasets. For MT-Re, the drop in LAS stems from non-leaking explanations – these explanations tend to mislead the simulator, meaning $p(\hat{y}|x, \hat{e})$ is inaccurate. For ST-Re, explanations tend to leak for examples where it is already easy to guess model behavior from x, i.e. $p(\hat{y}|x)$ sets a high baseline.

BLEU vs. Simulatability. BLEU is not correlated with our LAS metric, which supports our conjecture that BLEU does not reflect the effect of explanations on simulatability. LAS also does not correlate with the simulator accuracy, $\text{Acc}(\hat{y}|x, \hat{e})$, which is expected given how the simulator accuracy is heavily influenced by explanation leakage.

5.6.2 Human Validation of LAS

We validate our model proxy variables with two human evaluations, an expert simulation experiment, and a crowdsourced subjective rating test.

Expert Simulation. We (meaning the first three authors as expert annotators) validate our use of models as simulators of both model-generated and human explanations by manually playing the role of the simulator for 600 data points. With effectively the same design as our automatic metric computation, we simulate humans and our ST-Ra model with both datasets, only with no training period in this case. Each annotator is randomly assigned a role for each data point (whether they see the input, explanation, or both), and points are sampled such that an annotator never sees the same point in different roles. The sample is roughly balanced across the strata of our model's proxy variables. We note that ideally, we would use only expert human simulators instead of proxies, though even annotating less than 1% of the data across conditions required 1800 individual responses.

The correlations between proxy variables and our own are shown in Table 5.4. We group the data across subsets (e.g., explanation source and dataset) since the trends were similar between them. We find a strong correlation between the leakage proxy variable and the human leakage variable, with a Spearman rank correlation of $\rho = 0.53$ ($p <$ 1e−15), and we observe a moderate correlation between the model-based and human example-level LAS, $\rho = 0.29$ ($p <$ 1e−12) [33].

The disagreements are concentrated in false negatives for leakage, where we identify leaking explanations when the model does not. With LAS, model scores of -1 and 1 often end up as a human 0, meaning that an explanation confuses the model but not the human rater (for -1), or the human can predict based on the input alone when the model cannot (for 1). Because of this tendency toward 0, human LAS will shrink slightly toward 0 in expectation, relative to the model LAS (see row-normalized Table B.7 in Appendix). We also observe a degree of *pragmatic drift* between models and humans. Lazaridou et al. [108] operationalize this as the

	Example-Level LAS Score		
Data & Leakage	-1	0	1
CQA: Leaking	2.39 (.36)	2.65 (.08)	2.58 (.15)
Non-leaking	2.31 (.21)	2.40 (.10)	2.28 (.34)
SNLI: Leaking	2.96 (.45)	3.25 (.06)	3.18 (.15)
Non-leaking	2.78 (.31)	2.94 (.12)	2.61 (.46)

Table 5.5: Human explanation ratings grouped by dataset, label leakage. 95% confidence intervals in parentheses.

difference in performance between human and model listeners in a reference game. Similarly, we can use simulator accuracy given the input and explanations. We find that humans are better simulators of humans, and models are better at predicting model outputs. Across datasets and simulators, the difference in accuracies is 12.83 percentage points on average.

Lastly, one may notice from Table 5.4 that our predictions of the human label are sometimes wrong. In fact, our own task accuracy is 70% (±7.33) for SNLI and 72% for CQA (±7.19). These accuracies are similar to those obtained by Pavlick and Kwiatkowski [149] when re-annotating the SNLI dataset. Interestingly, they find that tasks such as these can have distributions over labels under human annotation, rather than consensus.

Human Subjective Quality Ratings. We collect human ratings from Mechanical Turkers for 200 test examples for both CQA and SNLI. Each example includes shuffled, unlabeled explanations (one from each model, plus humans, for a total of five), which we ask workers to separately rate on a 5-point Likert scale. After collecting 3 responses per item, we apply a worker quality filter, obtaining 902 ratings total. Further collection details are provided in Appendix B.4.

We investigate whether LAS and simulator accuracy are correlated with human explanation ratings. For each example, we obtain human ratings, the example's LAS score $\mathbb{1}[\hat{y}|x,\hat{e}]-\mathbb{1}[\hat{y}_i|x_i]$ (taking values -1,0,1), and simulator prediction accuracies, $\mathbb{1}[\hat{y}|x,\hat{e}]$, $\mathbb{1}[\hat{y}|x]$, and $\mathbb{1}[\hat{y}|\hat{e}]$ (taking values 0 or 1).

Human rating trends across example-level LAS scores are shown in Tables 5.5. A first observation is that LAS scores do not correlate well with human ratings. Curiously, though, simulator accuracies correlate with human ratings. We show these trends in Table 5.6, along with regression coefficients for predicting ratings from simulator accuracies. For both datasets, $\mathbb{1}[\hat{y}|\hat{e}]$ best correlates with human ratings and the association with $\mathbb{1}[\hat{y}|x,\hat{e}]$ is only significant for SNLI. Since good explanations tend to leak the label, it is not surprising that ratings correlate with label leakage. However, it is surprising that this association is stronger than the relationship with overall accuracy, $\mathbb{1}[\hat{y}|x,\hat{e}]$. Together, these results help explain why models may struggle to learn from human explanations, since models may focus on label leakage in human explanations at the expense of other information. They may also suggest that to collect human ratings that do not correlate with label leakage, a highly controlled environment for human ratings may be required.

5.6.3 Accuracy-Interpretability Trade-off

Past works on model interpretability have observed trade-offs between accuracy and model constraints for interpretation purposes [12, 88]. Yet, Rudin [172] and Jacovi and Goldberg [83] argue that we need not always face such a trade-off. Our findings provide quantitative evidence supporting these prior qualitative arguments. We observe consistently small changes

	Simulator Correctness		Regression Coef.	
Prediction	0	1	β	p
CQA: $\hat{y}\|x, \hat{e}$	2.34 (.11)	2.60 (.06)	.14	.07
$\hat{y}\|x$	2.38 (.09)	2.63 (.07)	.09	.20
$\hat{y}\|e$	2.44 (.10)	2.58 (.07)	.21	<.001
SNLI: $\hat{y}\|x, \hat{e}$	2.85 (.14)	3.22 (.05)	.20	.03
$\hat{y}\|x$	2.90 (.11)	3.24 (.06)	.10	.15
$\hat{y}\|e$	3.02 (.11)	3.21 (.08)	.27	<.001

Table 5.6: Human ratings broken down by dataset and simulator prediction, shown alongside regression results. 95% confidence intervals in parentheses.

in accuracy for our four models, and the largest changes, -.47 ($p = .3124$) for SNLI and -2.10 for CQA ($p = .3272$), are not statistically significant. We also test methods using human explanations purely for improving accuracy, e.g., through Masked Language Modeling objectives that have been successful for pretraining models. We find that this objective does not lead to statistically significant accuracy improvements, suggesting models still struggle to truly learn from human explanations (results are shown in Table B.8).

5.6.4 Multi-Agent Game

Multi-agent game results appear in Table 5.3, though we note that RL results should be cautiously interpreted as we observe unstable training behavior from this method. We find that optimization with SGD can reduce label leakage (from, e.g., 85.58% to 75.21% for CQA MT-RA) while slightly improving LAS scores, but only one of four changes in LAS scores is statistically significant, for MT-RE on SNLI. This approach does pull BLEU scores down. No statistically significant differences in accuracy are found; the largest change, a 3.37 point drop on CQA, has a p-value of .1287. We note that this kind of optimization may have the effect of increasing pragmatic drift, as is found for jointly optimized agents in [108].

5.7 Conclusion

We introduce a leakage-adjusted simulatability metric to evaluate the influence of natural language explanations on model simulatability while controlling for explanations leaking the model outputs. We validate our metric with two human subject experiments, and find that: (1) our ST-RA model attains similar LAS scores to human explanations, (2) rationalizing methods do better than reasoning methods, (3) no statistically significant relationship emerges between simulatability and accuracy, (4) our automatic metric correlates with expert simulation results, (5) the strongest predictor of crowdsourced explanation ratings is whether explanations leak the answer choice, and (6) optimizing explanations for simulatability can improve LAS scores.

6 Adding Explanation Data to Discriminative Learning

This third contribution addresses the mirror problem of evaluating LM-generated explanations (Chapter 5): we also want LMs to learn from human explanations.

6.1 Introduction

A long line of past work has sought to use free-text explanations, rationales, and other similar data to improve machine learning models. Proposed methods use explanations to constrain or regularize the learned model [228, 188, 7, 232, 192, 117], to automatically label data for data augmentation [63, 210, 6], as additional supervision [139, 68, 157] or intermediate structured variables [22, 163, 217], and simply as model inputs [173, 32, 238].

However, there are many tasks in NLP where improvements in performance prove elusive even when using thousands of explanations as additional data [139, 68]. A few observations could explain this situation: (1) the modeling space has not been fully explored for these tasks, but improvements are possible; (2) pretrained language models already store the knowledge that the explanations would have provided, so they do not need them; (3) the language models do not need any information that is not already learnable from the task's input-output pairs. We do not yet know which explanation is best, and therefore it would be helpful to more deeply understand the motivations behind existing modeling approaches.

In this chapter, we (1) present a formal framework for characterizing approaches to learning from explanation data, and (2) we propose a synthetic task for studying how models learn from natural language data. Specifically, we first present graphical models for various approaches where explanation data is used either as model *inputs*, *targets*, or *priors*, and we characterize existing methods according to these graphical models. Then, based on past results, we suggest which models might be most appropriate for explanation data. Next, we present a synthetic task which shares important properties with NLP tasks involving explanation data. Constructing this task helps us carefully specify the manner in which we expect explanations to be useful to models. We provide simple experimental verification that the task is solvable by existing Transformer models when using explanations as additional data but very difficult to solve without them. Our aim is to outline promising approaches in the area and contribute a concrete test bed to assist others in developing new models for learning from natural language explanations.

6.2 Formalizing the Roles of Explanations

In what follows, we discuss our framework for modeling with explanations and relevant work (Sec. 6.2.1), as well as promising approaches for learning from explanations (Sec. 6.2.2).

What Is an Explanation? We use the term "explanation" to refer to the data one might collect if asking a person to answer the question, "Why does data point x have label y?" This is a formulation of the explanation as an answer to a *why-question* of the kind discussed in Miller [131]. Rather than try to give a formal definition of the kind of data generated from this question, we proceed with some illustrative examples, shown in Fig. 6.1.

Illustrative Example #1

τ: When asked for travel times, give them in terms of travel by car.

x: How many hours does it take to travel from Addis Ababa to Dessie?

y: About 8 hours.

e: Addis Ababa and Dessie are 400km apart by road, and assuming you could average 50kph in a car, the travel time would be about 8 hours.

Illustrative Example #2

τ: What are the names of people in the text?

x: She was in particular interested in Babbage's work on the Analytical Engine. Lovelace first met him in June 1833, through their mutual friend, and her private tutor, Mary Somerville.

y: Babbage, Lovelace, Mary Somerville.

e: Names will refer to people, who can work on things, meet others, and be tutors. Not all capitalized things are names. Engines are not people, and here June is a date.

Figure 6.1: Hypothetical data and explanations. Here, x is an input that one might expect a model to produce the correct output for after fitting to (x, y) pairs. For some models, x may be sufficient, while others may benefit from additional information provided by e.

6.2.1 Formal Framework and Relevant Work

In this section, we lay out our theory of how explanations may be used in modeling a task, in a standard supervised learning setup for obtaining a MAP estimate of model parameters:

$$\hat{\theta} = \arg\max_{\theta} p(\theta|X, Y)$$
$$p(\theta|X, Y) \propto p(Y|X, \theta)p(\theta)$$

where Y is a set of labels for inputs X. We refer to the role of Y in this probabilistic model as the *target*, X as an *input*, and $p(\theta)$ as a *prior*. Below we describe existing approaches to adding explanations into this framework. An overview of the corresponding graphical models is shown in Fig. 6.2.

Using Explanations as Targets. Explanations are often used as additional supervision (shown as Multi-Task in Fig. 6.2). For instance, Pruthi et al. [157] consider using attention weight explanations (from a model) as targets in a multi-task framework, and they observe accuracy improvements in what is essentially model distillation. Meanwhile, natural language explanations appear as targets in a multi-task framework, using datasets with explanations for each data point [22, 139, 68, 217]. None of these works find improvements in task performance from incorporating explanations. It is perhaps even concerning that a model could learn to generate coherent "explanations" without the learning of this ability influencing the models that are found for the task.

Using Explanations as Inputs. Additional inputs may be valuable for solving some tasks. One family of approaches uses explanations as model inputs for each data point (Per Data Point Input in Fig. 6.2). Talmor et al. [202] systematically study RoBERTa's ability to combine pieces of knowledge for a task by including relevant factoids in the text input. Co-Reyes et al. [32] provide online natural language feedback to RL agents, and Rupprecht et al. [173] take a similar approach to interactive image segmentation with language feedback.

More commonly, approaches do not use human explanations at test time. In ExpBERT [138], a model conditions on vector representations of an input x and a single "global" set of

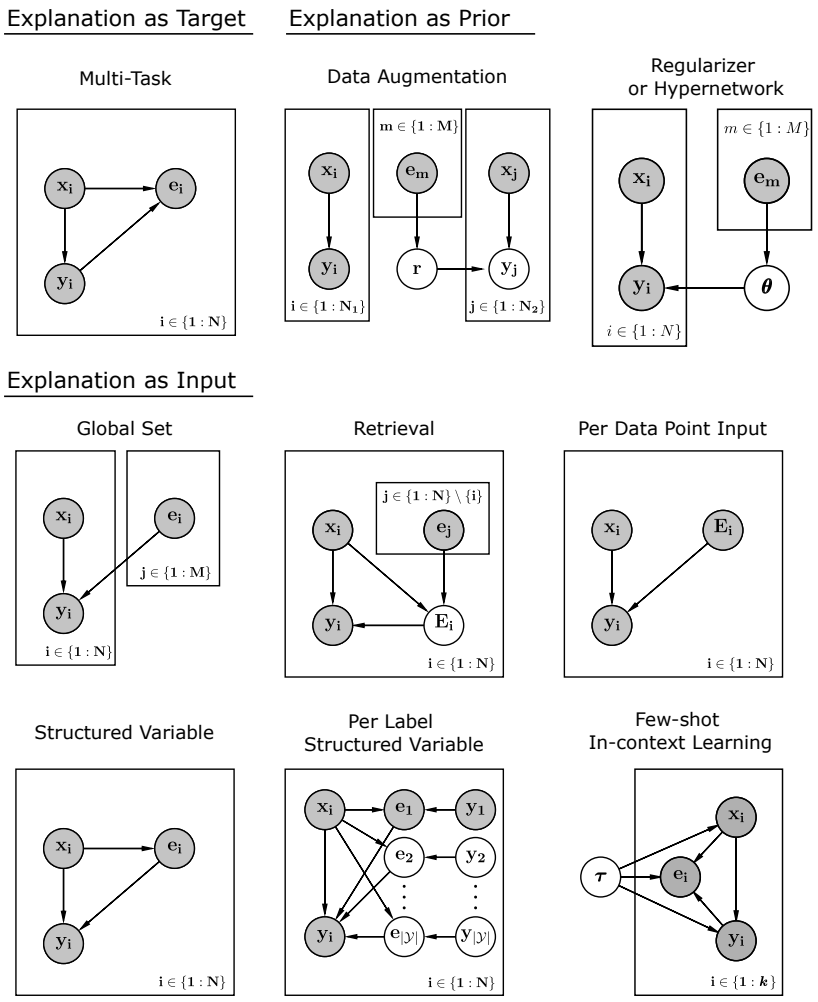


Figure 6.2: Graphical models for several approaches to using explanations as *targets*, as *inputs*, and as *priors*. Typically past works do not condition on human-given explanations at test time, unless they are designed to not leak the data point label.

explanations in order to make each prediction (Global Set in Fig. 6.2). This approach may not scale well to large numbers of explanations, however. Zhou et al. [238] treat explanations as latent variables, and at inference time they retrieve explanations from the training data (Retrieval in Fig. 6.2). A number of works condition on explanations generated at test time using generative models learned with human explanations as supervision, which are represented as Structured Variable and Per-Label Structured Variable in Fig. 6.2 [22, 163, 102, 68, 217, 234]. While such structured variables could be useful in principle, these methods have not produced sustained improvements in model accuracy.

Lastly, large language models have recently opened the door for using explanations in few-shot in-context learning [21]. We represent this approach as Few-shot In-context Learning in Fig. 6.2. We do not draw the dependencies between distinct data points in the context that would be implied by the attention graph of Transformers, but instead represent the dependence of each data point on the unknown task τ, which models evidently do inference over at test time. Initial work in this direction suggests that models of a sufficiently large size (280B parameters) can learn from explanations provided in a few-shot in-context learning

setting [106].

Using Explanations as Priors. We group together approaches to defining a distribution over model parameters, including those conditioning on data, $p(\theta|data)$. This is a prior over model weights not in the sense that the distribution is independent of data (which it is not), but rather that the posterior parameters are conditioned on the prior. Explanations have been used to constrain the learned model [192, 193] or to place priors over how features are weighted or extracted [228, 188, 232, 171, 10, 183, 117, 195, 157, 194]. Other works map directly from text to model parameters [7, 3]. These methods are all effectively described by Regularizer or Hypernetwork in Fig. 6.2. Lastly, a few approaches learn to use explanations for automatically labeling data for data augmentation purposes [63, 214, 6], which is effectively fitting to data from a prior distribution given by the labeling mechanism (Data Augmentation in Fig. 6.2).

6.2.2 Promising Models

Based on our review of existing approaches, we make a few key observations that we believe will assist in the design of future techniques:

1. Using free-text explanations as structured variables and as targets do not appear to be promising approaches at the moment [68, 139].
2. Free-text explanations may be useful as priors in computer vision [117], but we know of no successful use case for tasks besides Stacey et al. [194], which effectively reduces free-text explanations to a bag of words.
3. The only cases we know of where free-text explanations improve model performance on NLP tasks is when they are used as model inputs via the Global Set model, [138] a Retrieval model [238], and an In-Context Learning model using 280B parameters [106].

The upshot of these results is that the most promising approaches for learning from explanation data are likely those treating explanations as inputs (in a manner that does not require new explanations at test time). However, we recommend that other graphical models not be ruled out completely, in case there are promising methods in those families that have yet to be explored.

6.3 Synthetic Task

Following recent work using synthetic data to investigate sequence modeling questions [121, 125], we design a synthetic dataset so that we can carefully control several important data properties. In Fig. 6.3, we show an example data point and description of how it gets its label. The premise of our task is to classify sequences by counting different integers in them.

Core Idea Behind Data. We wish to design a task where, for a data point (x, y), an explanation e communicates information about *why* input x receives label y. The premise of the task is that a binary label for a sequence of integers x is determined by whether there are more of an integer a in the sequence than there are of an integer b. We refer to integers (a, b) that need to be counted as the *label reason*. This *label reason* forms the basis of the explanation for each data point, and it is always exactly specified by the first two integers in x, which we term the *index* and *indicator*. For every data point x, there is an *explanation* $e = (index, m, n, r, d)$ where the *label reason* is given by either (m, n) or

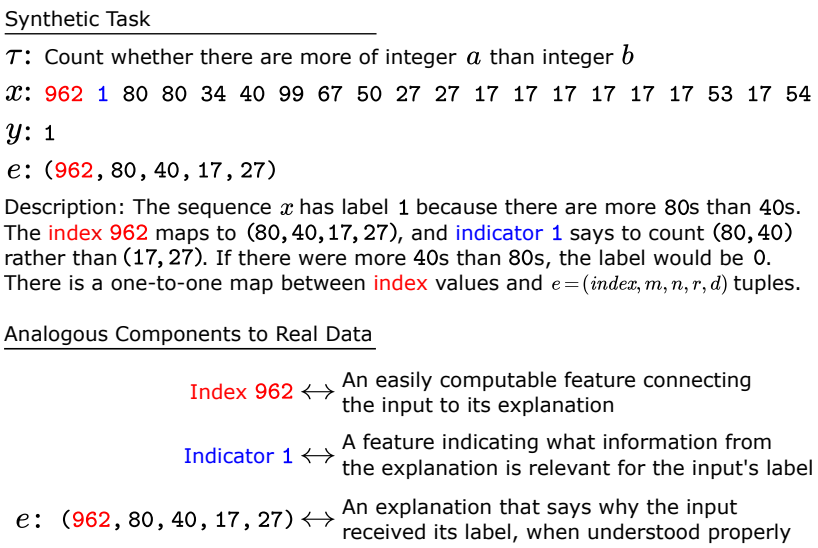


Figure 6.3: An example of our synthetic task.

(r, d). Whether the *label reason* is the (m, n) integer pair or the (r, d) pair is dictated by the *indicator*. As represented in Fig. 6.3, $(a, b) = (m, n)$ if the *indicator* is 1 and $(a, b) = (r, d)$ if the *indicator* is 2. We call the data e an explanation because it is a direct encoding of a natural language explanation for the data (x, y). For the data point in Fig. 6.3, this natural language explanation is "input x receives label 1 because it contains more 80's than 40's, and we do not need to count 17's or 27's for this sequence."

Proposed Dataset. We describe the proposed dataset using some default data parameters for preliminary experiments, but any specific numbers appearing below are easily adjusted. See Supplement C.4 for the full generative process.

1. Train set: 5000 sequences of 20 integers (including *index* and *indicator*), each accompanied by an explanation. There are 500 unique values of *index* in the dataset drawn from $unif(1, 10000)$, so there are 10 points for each *index*, whose values of m, n, r, and d are drawn from $unif(1, 100)$ while requiring that $m{\neq}n{\neq}r{\neq}d$. The corresponding 10 values of *indicator* are split between 1 and 2. Half of the points have label $y{=}1$, i.e. either $\#m{>}\#n$ or $\#r{>}\#d$, depending on which feature is causal. In each x_i, after m, n, r, and d have been randomly placed into the sequence, unfilled slots are filled with samples from $unif(1, 100)$.
2. Dev set: 10,000 points, none appearing in Train, with the same 500 *index* values, and twice the number of points per *index* as Train.
3. Test set: 50,000 points of similar construction to the Dev set, but with five times the points per *index* as Train.

Analogous Properties to Human-Curated Data. We claim that aspects of our synthetic task are analogous to properties that natural language data might take on, which we represent in Fig. 6.3. First, e is an explanation in the sense that, when understood properly, it is a plausible answer to the question: "why does point x have label y?" The explanation describes the feature that causes the label, i.e. the integers that should be counted. We suggest that the *index* in a sequence is analogous to the topic of some text or the things it refers to: it is an easily computable feature that connects the input to the appropriate explanation. Meanwhile, the *indicator* is a feature that tells how information from an explanation is relevant to deciding the label. Similarly, an explanation might only be understood in the context of the input it

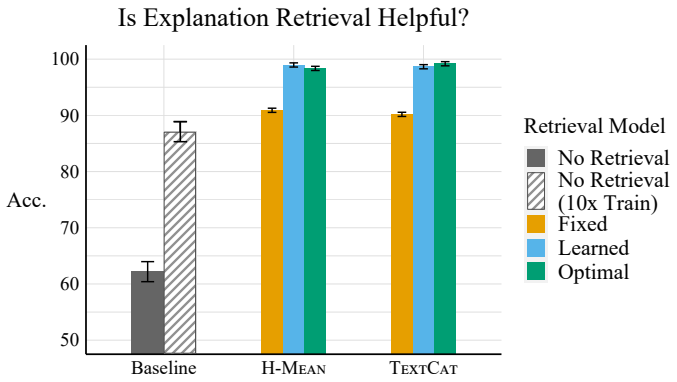


Figure 6.4: Synthetic task accuracy for our baseline and retrieval model with two conditioning mechanisms, H-Mean and TextCat.

explains.

6.4 Initial Experiments

We include experiments below that (1) show explanation data is helpful for solving our task and (2) demonstrate why the task is hard without explanation data. We make use of a retrieval-based model similar to Zhou et al. [238], which learns to retrieve explanations from the training dataset to help with prediction at test time (details in Appendix C.2 and C.3). This model is composed of a RoBERTa-base classifier [123] and a SentenceRoBERTa model used for retrieval [164]. The baseline in our experiments is the RoBERTa classifer on its own.

6.4.1 Explanation Retrieval Enables a Model to Solve Our Task

Design. Using our default dataset containing one explanation per training point, we measure model accuracy with retrieval in a 3×2 design. There are three conditions for the retrieval model: (1) *fixed*, where the Sentence-RoBERTa retriever is fixed and only the classifier is trained, (2) *learned*, where both classifier and retriever are trained end-to-end, and (3) *optimal* where the optimal retrieval model is used and the classifier is trained. We know the optimal retrieval model retrieves explanations with an *index* matching the query point's *index*. The two conditioning mechanisms, H-Mean and TextCat, differ in how they combine information across multiple retrieved explanations to produce a final prediction (see Appendix C.2.1).

Results. The results in Fig. 6.4 show that explanation retrieval can reach accuracies above 98%, improving accuracy by around 37 points over a no-explanation baseline. We also find that the learned retrieval model does as well as the optimal retrieval model, improving over the *fixed* condition by about 7 points. Thus, access to explanations allows the model to perform much better than a no-explanation baseline. In fact, the explanation retrieval model outperforms a no-explanation baseline with as many as 50,000 training data points (a 10x increase), which obtains 87.11% accuracy.

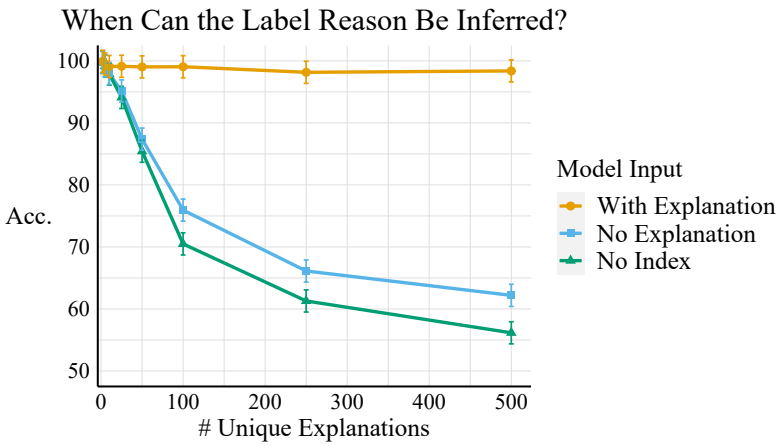


Figure 6.5: Synthetic task accuracy as a function of the number of unique explanations for data point labels.

6.4.2 Why Is The Task Hard Without Explanations?

Design. We measure test accuracy as a function of how many unique explanations (and therefore *label reasons*) there are in the data. While keeping the train set size fixed at 5000 points, we vary how many points share the same explanation $(index, m, n, r, d)$. By default there are 10 points per *index*, and with 5000 points this means that there are 500 unique explanations in the data. We use many as 2500 points per *index*, meaning using two unique explanations. The experiment conditions also vary in how task information is available in the input: (1) for With Explanation, each 20-integer sequence x_i has its explanation appended to it; (2) for No Explanation, only x_i is given, which requires the model to learn the map $index \rightarrow (m, n, r, d)$; (3) for No Index, the *index* is omitted from the input, so the model must infer the *label reason* from the sequence's contents alone.

Results. The results are shown in Fig. 6.5. We see that, when the number of unique explanations (and therefore possible *label reasons*) is small, the No Explanation model can achieve an accuracy as high as if it had been directly given the label reason, i.e. as high as the With Explanation condition. Yet, No Explanation model accuracy falls off quickly with the number of unique explanations, reaching accuracies as low as 62.2% with 500 explanations. Evidently, with this many unique explanations, it is too difficult to learn the map between the *index* and the latent *label reason*. Without the *index* in the input (No Index condition), it is even harder to infer the label reason. While accuracy does rise significantly with the size of the training data (see Fig. 6.4), even using 10x as much train data does not close the gap with the explanation retrieval model.

6.5 Discussion & Conclusion

We present a synthetic dataset with key similarities to natural language explanation data, and we show that our explanations are highly useful for model learning. However, we emphasize that if a model already "knew" the information in some explanations, it might not need them. This may plausibly occur with sufficiently large pretrained models that store a great deal of factual knowledge [151]. Similarly, the necessary information might be learnable from

(X, Y) data alone. Future work on modeling approaches we outline in this chapter (Fig. 6.2) will benefit from testing their methods on controlled synthetic tasks as a test of their ability to learn from explanation data. Then, further analysis will be helpful for understanding how explanations contain novel information that is not learned elsewhere in pretraining or finetuning.

7 Feature Attribution Methods and Evaluation

In this chapter, I introduce new methods for and address conceptual problems with feature attribution, a popular approach to model explanation that appears prominently in our initial work on evaluating ML explanations (Chapter 4).

7.1 Introduction

Estimating feature importance (FI) is a common approach to explaining how learned models make predictions for individual data points [185, 165, 115, 198, 126, 37]. FI methods assign a scalar to each feature of an input representing its "importance" to the model's output, where a feature may be an individual component of an input (such as a pixel or a word) or some combination of components. Alongside these methods, many approaches have been proposed for evaluating FI estimates (also known as attributions) [141, 5, 42, 80, 65, 239]. Many of these approaches use test-time input ablations, where features marked as important are removed from the input, with the expectation that the model's confidence in its original prediction will decline if the selected features were truly important.

For instance, according to the *Sufficiency* metric [42], the best FI explanation is the set of features which, if kept, would result in the highest model confidence in its original prediction. Typically the top-k features would be selected according to their FI estimates, for some specified *sparsity* level k. Hence, the final explanation e is a k-sparse binary vector in $\{0,1\}^d$, where d is the dimensionality of the chosen feature space. For an explanation e and a model f that outputs a distribution over classes $p(y|x) = f(x)$, Sufficiency can be given as:

$$\text{Suff}(f, x, e) = f(x)_{\hat{y}} - f(\texttt{Replace}(x, e))_{\hat{y}}$$

where $\hat{y} = \arg\max_y f(x)_y$ is the predicted class for x and the `Replace` function replaces features in x with some uninformative feature at locations corresponding to 0s in the explanation e.

The `Replace` function plays a key role in the definition of such metrics because it defines the *counterfactual* input that we are comparing the original input with. Though FI explanations are often presented without mention of counterfactuals, all explanations make use of counterfactual situations [131], and FI explanations are no exception. The only way we can understand what makes some features "important" to a particular model prediction is by reference to a counterfactual input which has its important features replaced with a user-specified (uninformative) feature.

In this chapter, we study several under-explored dimensions of the problem of finding good explanations according to test-time ablation metrics including Sufficiency and a related metric, Comprehensiveness, with a focus on natural language processing tasks. We describe three primary contributions below.

First, we argue that standard FI explanations are heavily influenced by the out-of-distribution (OOD) nature of counterfactual model inputs, which results in *socially misaligned* explanations. We use this term, first introduced by Jacovi and Goldberg [85], to describe a situation where an explanation communicates a different kind of information than the kind that people expect it to communicate. Here, we do not expect the model prior or random weight initialization to influence FI estimates. This is a problem insofar as FI explanations are not telling us what we think they are telling us. We propose a training algorithm to resolve

the social misalignment, which is to expose models to counterfactual inputs during training, so that counterfactuals are not out-of-distribution at test time.

Second, we systematically compare `Replace` functions, since this function plays an important role in evaluating explanations. To do so, we remove tokens from inputs using several `Replace` functions, then measure how OOD these ablated inputs are to the model. We compare methods that remove tokens entirely from sequences of text [141, 42], replace token embeddings with the zero embedding or a special token [115, 198, 5, 228, 198], marginalize predictions over possible counterfactuals [241, 101, 224], and edit the input attention mask rather than the input text. Following our argument regarding the OOD problem (Sec. 7.4), we recommend the use of some `Replace` functions over others.

Third, we provide several novel search-based methods for identifying FI explanations. While finding the optimal solution to $\arg\max_e \text{Suff}(f, x, e)$ is a natural example of binary optimization, a problem for which search algorithms are a common solution [152, 180, 11], we are aware of only a few prior works that search for good explanations [54, 167, 47]. We introduce our novel search algorithms for finding good explanations by making use of general search principles [152]. Based on experiments with two Transformer models and six text classification datasets (including FEVER, SNLI, and others), we summarize our core findings as follows:

1. We propose to train models on explanation counterfactuals and find that this leads to greater model robustness against counterfactuals and yields drastic differences in explanation metrics.

2. We find that some `Replace` functions are better than others at reducing counterfactual OOD-ness, although ultimately our solution to the OOD problem is much more effective.

3. We introduce four novel search-based methods for identifying explanations. Out of all the methods we consider (including popular existing methods), the only one that consistently outperforms random search is the Parallel Local Search (PLS) that we introduce, often by large margins of up to 20.8 points. Importantly, we control for the compute budget used by each method.

7.2 Related Work

Feature Importance Methods. A great number of methods have been introduced for FI estimation, drawing upon local approximation models [165, 167, 103], attention weights [87, 216, 235], model gradients [185, 184, 198, 189], and model-based feature selection [13, 9, 219, 145, 28, 37]. While search approaches are regularly used to solve combinatorial optimization problems in machine learning [180, 16, 11, 49, 136], we know of only a few FI methods based on search [54, 167, 47]. Fong and Vedaldi [54] perform gradient descent in explanation space, while Ribeiro et al. [167] search for probably-sufficient subsets of the input (under a perturbation distribution). In concurrent work, Du and Xu [47] propose a genetic search algorithm for identifying FI explanations. We introduce several novel search algorithms for finding good explanations, including (1) a gradient search similar to Fong and Vedaldi [54], a (2) local heuristic search inspired by an adversarial attack method [49], (3) a global heuristic search, and (4) a parallel local search (PLS) making use of general search principles [152].

Choice of `Replace` Function. Past evaluations of explanation methods typically remove tokens or words from the text entirely [141, 42] or replace them with fixed feature values [80, 225]. Methods for creating explanations also use several distinct approaches, including

(1) replacing token embeddings with the zero vector [115, 198, 5], (2) using a special token [228, 198], (3) marginalizing predictions over a random variable representing an unobserved token [241, 101, 224, 90], and (4) adversarially selecting counterfactual features [81]. Sturmfels et al. [196] carry out a case study involving a few `Replace` functions for a vision model, which they compare via test-time ablations with image blurring techniques, though the case study is not intended to be a full comparison of methods. Haug et al. [74] assess a number of `Replace` functions for explanation methods used with tabular data, but they compare between functions to use when generating explanations, rather than when evaluating explanations, for which they offer no recommendation. In addition to evaluating `Replace` functions from the above works, we also consider setting attention mask values for individual tokens to 0.

The Out-of-distribution Problem of Explanations. Many papers have expressed concerns over how removing features from an input may result in counterfactuals that are out-of-distribution to a trained model [228, 198, 54, 26, 80, 101, 81, 224, 197, 90, 158, 74, 179, 91, 205]. In response to the problem, some have proposed to marginalize model predictions over possible counterfactual inputs [241, 101, 224, 90], use counterfactuals close to real inputs [26, 179], weight their importance by their closeness to real inputs [158], or to adversarially select counterfactual features rather than use any user-specified features [81]. Others reject the whole notion of test-time ablations, preferring metrics based on train-time ablations [80]. Jethani et al. [91] propose a specialized model for evaluating explanations that is trained on counterfactual inputs in order to make them in-distribution, but since the evaluation model is distinct from the model used to solve the task, explanation metrics calculated using this evaluation model may not reflect the faithfulness of explanations to the task model. In concurrent work, Vafa et al. [205] independently adopt a solution equivalent to our Counterfactual Training with an Attention Mask `Replace` function, an approach which we empirically justify in Sec. 7.5. In general, prior works make arguments for their approach based on intuition or basic machine learning principles, such as avoiding distribution shift. In Sec. 7.4, we give a more detailed argument for preferring in-distribution counterfactuals on the basis of *social alignment*, a concept introduced by Jacovi and Goldberg [85], and we propose a solution to the OOD problem. Our solution allows for test-time evaluation of explanations of a particular model's decisions for individual data points, unlike similar proposals which either evaluate large sets of explanations all at once [80] or use a separate model trained specifically for evaluation rather than the blackbox model [91].

7.3 Problem Statement

Feature Importance Metrics. The problem we are investigating is to find good feature importance explanations for single data points, where explanation are evaluated under metrics using test-time ablations of the input. In this context, an explanation for an input in a d dimensional feature space is a binary vector $e \in \{0,1\}^d$, which may be derived from discretizing an FI estimate $v \in \mathbb{R}^d$. We consider two primary metrics, *Sufficiency* and *Comprehensiveness* [42]. Sufficiency measures whether explanations identify a subset of features which, when kept, lead the model to remain confident in its original prediction for a data point. Comprehensiveness, meanwhile, measures whether an explanation identifies all of the features that contribute to a model's confidence in its prediction, such that removing these features from the input lowers the model's confidence.

The Sufficiency metric for an explanation $e \in \{0,1\}^d$ and model $p(y|x) = f_\theta(x)$ is given as

$$\text{Suff}(f_\theta, x, e, s) = f_\theta(x)_{\hat{y}} - f_\theta(\texttt{Replace}_s(x, e))_{\hat{y}} \tag{7.1}$$

where $\hat{y} = \arg\max_y f(x)_y$ is the predicted class for x, and the $\texttt{Replace}_s$ function retains a proportion s of the input features (indicated by e) while replacing the remaining features with some user-specified feature. In order to control for the explanation sparsity, i.e. the proportion s of tokens in the input that may be retained, we average Sufficiency scores across sparsity levels in {.05, .10, .20, .50}, meaning between 5% and 50% of tokens in the input are retained [42]. A **lower** Sufficiency value is better, as it indicates that more of the model's confidence is explained by just the retained features (increasing $f_\theta(\texttt{Replace}(x, e))_{\hat{y}}$).

Similarly, Comprehensiveness is given as $\text{Comp}(f_\theta, x, e, s) = f_\theta(x)_{\hat{y}} - f_\theta(\texttt{Replace}_s(x, e))_{\hat{y}}$ but with sparsity values in {.95, .90, .80, .50}, as we are looking to remove features that are important to the prediction (while keeping most features). A **higher** Comprehensiveness value is better, as it indicates that the explanation selects more of the evidence that contributes to the model's confidence in its prediction (resulting in a lower $f_\theta(\texttt{Replace}(x, e))_{\hat{y}}$).

Overall Objective. Finally, our overall Sufficiency and Comprehensiveness objectives are given by averaging Suff (or Comp) scores across several sparsity levels. With a model $p(y|x) = f(x)$, a single data point x with d features, and a set of sparsity levels S, the Sufficiency objective is optimized by obtaining a set $E = \{e_i\}_{i=1}^{|S|}$ with one explanation per sparsity level as

$$\arg\max_E \frac{1}{|S|} \sum_{i=1}^{|S|} \text{Suff}(f, x, e_i, s_i) \quad \text{s.t. } e_i \in \{0,1\}^d \text{ and } \sum_d e_i^{(d)} \leq \text{ceiling}(s_i \cdot d)$$

where the ceiling function rounds up the number $s_i \cdot d$ of tokens to keep. When optimizing for Comprehensiveness, we use the Comp and arg min functions, and the inequality is flipped. In general, we will optimize this objective using a limited compute budget, further described in Sec. 7.6.2.

7.4 The Out-of-Distribution Problem in Explanations

In this section, we first give a full argument for why it is problematic for explanations to be created *or* evaluated using OOD counterfactuals. Then, we propose a solution to the OOD problem. We rely on this argument in our comparison of `Replace` functions in Sec. 7.5. We also assess our proposed solution to the OOD problem in Sec. 7.5 and later make use of this solution in Sec. 7.6.

The OOD problem for explanations occurs when a counterfactual input used to create or evaluate an explanation is out-of-distribution (OOD) to a model. Here, we take OOD to mean the input is not drawn from the data distribution used in training (or for finetuning, when a model is finetuned) [135]. In general, counterfactual data points used to produce FI explanations will be OOD because they contain feature values not seen in training, like a MASK token for a language model. A long line of past work raises concerns with this fact [228, 198, 54, 26, 80, 101, 81, 224, 197, 90, 158, 74, 179, 91, 205]. The most concrete argument on the topic originates from Hooker et al. [80], who observe that using OOD counterfactuals makes it difficult to determine whether model performance degradation is caused by "distribution shift" or by the removal of information. It is true that, for a given counterfactual, a model might have produced a different prediction if that counterfactual was in-distribution rather than out-of-distribution. But this is a question we cannot ask about a single, trained model, where there is no ambiguity about what causes a drop in model confidence when replacing features: the features in the input were replaced, and this changes

that model's prediction. If the counterfactual was in-distribution, we would be talking about a different model, with a different training distribution. Hence, we believe we need a stronger argument for why we should not use OOD counterfactuals when explaining a trained model's behavior.

Our principal claim is that feature importance explanations for a standardly trained neural model are *socially misaligned*, which is undesirable. Jacovi and Goldberg [85] originally introduce this term as they describe shortcomings of explanations from a pipeline (select-predict) model, which is a kind of classifier designed to be interpretable. Explanations are socially misaligned when people expect them to communicate one kind of information, and instead they communicate a different kind of information. For example, if we expected an explanation to be the information that a model relied on in order to reach a decision, but the explanation was actually information selected after a decision was already made, then we would say that the explanations are socially misaligned. Our argument now proceeds in two steps: first, we outline what the social expectations are for feature importance explanations, and then we argue that the social expectations are violated due to the fact that counterfactuals are OOD.

We suggest that, for a particular trained model and a particular data point, **people expect a feature importance explanation to reflect how the model *has learned* to interpret features as evidence for or against a particular decision.**[1] This social expectation is upheld if FI explanations are influenced only by the interaction of an untrained model with a training algorithm and data. But our expectations are violated to the extent that FI explanations are influenced by factors such as the choice of model prior and random seed (which we do not intend to influence FI explanations). We depict these possible upstream causes of individual FI explanations in Fig. 7.1. In fact, the model prior and random seed are influential to FI explanations when the counterfactuals employed in these explanations are OOD to the model. A simple example clearly illustrates the potential influence of model priors: Suppose one trained a BERT model to classify the sentiment of individual words using training data from a sentiment dictionary, then obtained feature importance explanations with the MASK token `Replace` function. In this situation, model predictions on counterfactual data are always equal to the prediction for a single MASK token, $f_\theta(\text{MASK})$. So, by construction, the MASK token never appears in the training data, but FI explanations for this model make use of the quantity $f_\theta(\text{MASK})$. Since a model could not have learned its prediction $f_\theta(\text{MASK})$ from the data, this quantity will be largely determined by the model prior and other training hyperparameters, and therefore explanations based on this prediction are socially misaligned. Now, in general, we know that neural models are sensitive to random parameter initialization, data ordering (determined by the random seed) [44], and hyperparameters (including regularization coefficients) [30, 156, 206], even as evaluated on in-distribution data. For OOD data, then, a neural model will still be influenced by these factors, but the model has no data to learn from in this domain. As a result, FI

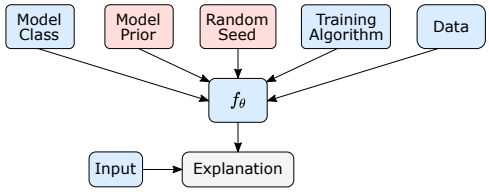


Figure 7.1: Causal diagram of a feature importance explanation for a trained model and an input.

[1] We mean "people" to refer to the typical person who has heard the standard description of these explanations, i.e. that they identify features that are "important" to a model decision. Of course, there will be some diversity in how different populations interpret feature importance explanations [51].

explanations are socially misaligned to the extent that these unexpected factors influence the explanations (while the expected factors like data are not as influential). In other words, we do not expect explanations to influenced by random factors, the priors of the model designer, or uninterpretable hyperparameters, but we do expect them to be influenced by what the model learns from data.

The argument applies equally to explanation metrics. When metrics are computed using OOD counterfactuals, the scores are influenced by unexpected factors like the model prior and random seed, rather than the removal of features that a model has learned to rely on. As a result, the metrics are socially misaligned. They do not represent explanation quality in the way we expect them to.

The solution to the OOD problem is to **align the train and test distributions, which we do by exposing the model to counterfactual inputs during training**, a method we term Counterfactual Training. Since common explanation methods can require hundreds or thousands of model forward passes when explaining predictions [165, 198], explanations from these methods would be prohibitively expensive to obtain during training. We therefore propose to train with random explanations that remove most of the input tokens, which provides a good objective in theory for models to learn the counterfactual distribution that will be seen at test time [91]. Specifically, we double the inputs in each training batch, adding a `Replace`(x, e) version of each input (with the same label) according to a random explanation e with sparsity randomly selected from {.05, .1, .2, .5}. The resulting Counterfactual-Trained (CT) models make in-distribution predictions for both observed and counterfactual data points. While we cannot guarantee that this approach fully eliminates the influence of the model prior and random seed on FI explanations, the fact that explanations are influenced by what the model learns from data will resolve social misalignment to a great extent. We find that these models suffer only slight drops in test set accuracy, by 0.7 points on average across six datasets (see Table D.2 in Supplement). But we observe that this approach greatly improves model robustness to counterfactual inputs, meaning the counterfactuals are now much more in-distribution to models (described further in Sec. 7.5). Similar to the goals of ROAR [80] and EVAL-X [91], our proposed solution also aims to align the train and test-time distributions. However, our approach allows for test-time evaluation of individual explanations for a particular trained model, while ROAR only processes large sets of explanations all at once and EVAL-X introduces a specialized model for evaluation, which may not reflect the faithfulness of explanations to the task model.

7.5 Analysis of Counterfactual Input OOD-ness

Here, we assess how out-of-distribution the counterfactual inputs given by `Replace` functions are to models, and we measure the effectiveness of Counterfactual Training. We do this before designing or evaluating explanation methods because, given our argument in Sec. 7.4, it is important to first identify which `Replace` function and training methods are most appropriate to use for these purposes.

Experiment Design. We compare between `Replace` functions according to how robust models are to test-time input ablations using each function, where the set of input features to be removed is fixed across the functions. We measure robustness by model accuracy, which serves as a proxy for how in-distribution or out-of-distribution the ablated inputs are. If we observe differences in model accuracies between `Replace` functions for a given level of feature sparsity, we can attribute the change in the input OOD-ness to the `Replace` function itself. In

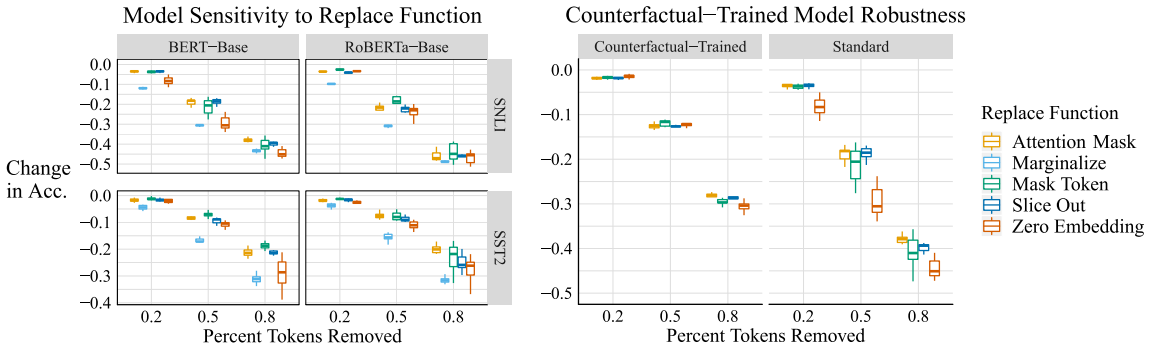


Figure 7.2: Model sensitivity to input ablations for several choices of `Replace` function and training algorithm. On the left we show the sensitivity of standardly trained models. On the right we show the effect of using Counterfactual-Trained models.

the same manner, we compare between Counterfactual-Trained (CT) models and standardly trained models (termed as Standard).

Specifically, we train 10 BERT-Base [41] and RoBERTa-Base [123] on two benchmark text classification datasets, SNLI [20] and SST-2 [191]. These are all Standard models, without the counterfactual training we propose. We use ten random seeds for training these models. Then, we evaluate how robust the models are to input ablations, where we remove a proportion of random tokens using one of five `Replace` functions (i.e. we `Replace` according to a random explanation). We evaluate across proportions in $\{0.2, 0.5, 0.8\}$. The five `Replace` functions we test are:

1. **Attention Mask.** We introduce this `Replace` function, which sets a Transformer's attention mask values for removed tokens to 0, meaning their hidden representations are never attended to.
2. **Marginalize.** This method marginalizes model predictions over counterfactuals drawn from a generative model $p_\phi(x)$ of the data, i.e. as $\arg\max_y \ln \sum_{\tilde{x} \sim p_\phi(\tilde{x}|x,e)} p_\theta(y|\tilde{x}) p_\phi(\tilde{x}|x,e)$, where $p_\phi(\tilde{x}|x,e)$ is the distribution over tokens to be replaced given by e.g. BERT [241, 101, 26, 224].
3. **MASK Token**. In this method, we simply replace tokens with the MASK token.
4. **Slice Out**. This approach removes selected tokens from the input sequence itself, such that the resulting sequence has a lower number of tokens.
5. **Zero Vector**. Here, we set the token embeddings for removed tokens to the zero vector.

We train additional CT models for BERT-Base on SNLI, with ten random seeds per model, for all `Replace` functions except Marginalize, since this function is exceedingly expensive to use during Counterfactual Training. For further details, see the Supplement.

Results for `Replace` functions. We show the results of this experiment in Fig. 7.2, via boxplots of the drops in accuracy for each of the 10 models per condition. First, we describe differences in `Replace` functions for Standard models, then we discuss the effect of Counterfactual Training. On the left in Fig. 7.2, we see that Standard models are much more sensitive to some `Replace` functions than others. The Attention Mask and Mask Token functions are the two best methods. The best of these two methods outperforms the third best

method by up to 1.61 points with BERT and SNLI ($p = .0005$),[2] 5.48 points with RoBERTa and SNLI ($p < 1e{-}4$), 2.42 points with BERT and SST-2 ($p = 0.0008$), and 4.72 points with RoBERTa and SST-2 ($p < 1e{-}4$). The other methods often far underperform the best method. For instance, with BERT on SST-2, Zero Embedding is up to 10.45 points worse than Mask Token ($p < 1e{-}4$), and with RoBERTa on SST-2, Slice Out underperforms Attention Mask by up to 4.72 points ($p < 1e{-}4$). Marginalize is regularly more than 10 points worse than the best method. Overall, we recommend that, *when not using Counterfactual Training,* researchers use either the Attention Mask or Mask Token `Replace` functions.

Counterfactual Training vs. Standard Training. On the right side of Fig. 7.2, we see the effect of Counterfactual Training on model robustness for several `Replace` functions. We find that counterfactual inputs are much less OOD for Counterfactual-Trained models than for Standard models, regardless of the `Replace` function used. The improvement in robustness is up to **22.9** points. Moreover, the difference between `Replace` functions is almost entirely erased, though we do observe a statistically significant difference between Attention Mask and Zero Embedding with 80% of tokens removed (by 2.23 points, $p < 1e{-}4$). Given these results, and following Sec. 7.4, **we ultimately recommend that researchers use Counterfactual Training with the Attention Mask, Mask Token, or Slice Out `Replace` function whenever they intend to create FI explanations.**

7.6 Explanation Methods and Experiments

7.6.1 Explanation Methods

We describe explanation methods we consider below, with implementation details in the Supplement.

Salience Methods. One family of approaches we consider assigns a scalar *salience* to each feature of an input. The key property of these scores is that they allow one to rank-order features. We obtain binarized explanations through selecting the top-k features, or up to the top-k features when some scores are negative (suggesting they should not be kept). We list the methods below:[3]

1. LIME. LIME estimates FI by learning a linear model of a model's predicted probabilities with samples drawn from a local region around an input [165]. Though it is common to use the Slice Out `Replace` function with LIME, we use the Attention Mask `Replace` function (following Sec. 7.5), meaning we sample local attention masks rather than local input sequences.

2. Vanilla Gradients. We obtain model gradients w.r.t. the model input as salience scores, one early method for explanation [113]. We compute the gradient of the predicted probability w.r.t. input token embeddings, and we obtain a single value per token by summing along the embedding dimension.

3. Integrated Gradients. We evaluate the Integrated Gradients (IG) method of Sundararajan et al. [198]. This method numerically integrates the gradients of a model output w.r.t. its input along a path between the observed input and a user-selected baseline. Given our results in Sec. 7.5, we use a repeated MASK token embedding for our baseline $\tilde{x}$ rather than the all-zero input suggested by Sundararajan et al. [198] for text models. We use the model's

[2] p-values for two-sided difference in means tests are calculated by block bootstrap with all data points and model seeds being resampled 100k times [50].

[3] In early experiments, we found that a parametric model (similar to [13, 9, 145]) performed far worse than other salience methods, and hence we leave out parametric models from further consideration.

predicted probability as the output, and to obtain token-level salience scores, we sum the output of IG along the embedding dimension.

Search Methods. An alternative class of methods searches through the space of possible explanations. Search methods are regularly used to solve combinatorial optimization problems in machine learning [180, 16, 11, 49, 136]. All search methods use the Attention Mask `Replace` function, and the search space is restricted to explanations of the maximum allowable sparsity (or minimum, with Comprehensiveness), except for Anchors which takes a maximum number of features as a parameter.

1. Random Search. For each maximum explanation sparsity k (or minimum, for Comprehensiveness), we randomly sample a set of k-sparse explanations, compute the current objective for each of them, and choose the best explanation under the objective.

2. Anchors. Ribeiro et al. [167] introduce a method for finding a feature subset that almost always yields the same model prediction as its original prediction for some data point, as measured among data points sampled from a distribution centered on the original data point. Explanations are also preferred to have high coverage, meaning the feature subset is likely to be contained in a local sample. The solution is identified via a Multi-Armed Bandit method combined with a beam search.

3. Exhaustive Search. Exhaustive search returns the optimal solution after checking the entire solution space. This is prohibitively expensive with large inputs, as there is a combinatorial explosion in the number of possible explanations. However, we are able to exactly identify optimal explanations for short sequences, typically with 10 or fewer tokens.

4. Gradient Search. Fong and Vedaldi [54] propose to search through a continuous explanation space by gradient descent. We introduce a Gradient Search that uses discrete explanations, because continuous explanations do not reflect test-time conditions where discrete explanations must be used. For an input of length L, this method sequentially updates a continuous state vector $s \in \mathbb{R}^L$ via gradient descent to minimize a regularized cross-entropy loss between the model prediction on the input x and the model prediction for $\texttt{Replace}(x, e_t)$, where e_t is a discrete explanation sampled from a distribution $p(e_t|s_t)$ using the Gumbel-Softmax estimator for differentiability [127, 89]. The regularizer maintains sparsity of the solution. The final explanation is obtained from the last state s_t.

5. Taylor Search. Inspired by HotFlip [49], this method explores the solution space by forecasting the change in the objective using a first-order Taylor approximation. Specifically, this is a local search method with a heuristic function computed as follows: We first calculate the gradient $g \in \mathbb{R}^L$ of the cross-entropy loss (same loss as Gradient Search) with respect to the explanation e_t. Then we find the two indices i and j as the solution to $\arg\max_{i,j}\ g^{(i)} - g^{(j)}$. The next state is obtained by setting $e_t^{(i)} = 1$ and $e_t^{(j)} = 0$. This is a first-order approximation to the change in loss between the new and old state, based on the Taylor expansion of the loss [49]. Note when optimizing for Comprehensiveness, we use the $\arg\min$. Following Ebrahimi et al. [49], we ultimately use this search heuristic within a beam search, starting from a random explanation, with width $w = 3$.

6. Ordered Search. Next, we introduce a global search algorithm, Ordered Search. This method searches through all explanations in an order given by a scoring function $f : e \to \mathbb{R}$. We only require that f is linear in e, as this allows for efficient ordering of the search space using a priority queue [11]. The algorithm proceeds by first estimating parameters for f_θ, then searching through explanations in order of their score, $\theta^T e$. For the first stage, we obtain parameters for f_θ from the per-token salience scores given by LIME, which is the best salience method evaluated in Sec. 7.6. In the second stage, we enumerate the search space in order of

the score given by f_θ. We allocate 25% of the compute budget to the first stage and 75% to the second (measured in terms of forward passes).

7. Parallel Local Search (PLS). Lastly, we again consider the class of local search algorithms, which have a long history of success with constrained optimization problems [152, 11]. We propose a parallelized local search algorithm (PLS) tailored to the problem at hand. Given a number r of parallel runs to perform, a proposal function `Propose`, and compute budget b per run, an individual search proceeds as follows: (1) Sample a random initial explanation e_1 and compute the objective function for that explanation. (2) For the remaining budget of $b-1$ steps: sample a not-already-seen explanation e_t according to `Propose` and adopt e_t as the new state only if the objective is lower at e_t than at the current state. The `Propose` function samples not-already-seen explanations from neighboring states to the current state. We use $r = 10$ parallel runs following tuning results.

7.6.2 Experimental Setup

Data. We compare the above explanation methods on six benchmark text classification datasets: SNLI [20], BoolQ [31], Evidence Inference [110], FEVER [203], MultiRC [97], and SST-2 [190]. One important distinction among these datasets is that BoolQ, FEVER, MultiRC, and Evidence Inference data points include both a *query* and an accompanying *document*. The query is typically critical information indicating how the model should process the document, and therefore we never replace query tokens. We use 500 test points from each dataset to compare methods. See Table D.1 in the Supplement for dataset statistics, including average sequence length.

Models. We train ten seeds of BERT-Base models on each dataset [41], which we term Standard models. For each dataset we train another ten Counterfactual-Trained (CT) models using the Attention Mask `Replace` function, following the approach outlined in Sec. 7.4 (further details in Supplement).

Controlling for Compute Budget. We wish to control for the available compute budget in order to fairly compare explanation methods. Some explanations require a single forward and backward pass [185, 113], while others can require hundreds of forward and backward passes [198] or thousands of forward passes [165]. Since this is expensive to perform, we limit each method to a fixed number of forward and backward passes (counted together), typically 1000 in total, to obtain a single explanation.

7.6.3 Main Results

In Table 7.1, we show Suff and Comp scores across methods and datasets, for both the Standard and Counterfactual-Trained (CT) models. 95% confidence intervals are shown in parentheses, obtained by bootstrap with data and model seeds resampled 100k times. Bolded numbers are the best in their group at a statistical significance threshold of $p < .05$. We give results for SST-2 in the Supplement, including for Exhaustive Search since we use short sequences there, as well as for Vanilla Gradient as it performs much worse than other methods. We summarize our key observations as follows:

1. PLS performs the best out of all of the explanation methods, and it is the only method to consistently outperform Random Search. Improvements in Sufficiency are statistically significant for every dataset using both Standard and CT models, with differences of up to 12.9 points over LIME and 7.6 points over Random Search. For Comprehensiveness, PLS is the best method in 9 of 10 comparisons, 7 of which are statistically

Table 7.1: Explanation metrics across methods and datasets

Dataset	Method	Sufficiency ↓		Comprehensiveness ↑	
		Standard Model	CT Model	Standard Model	CT Model
SNLI	LIME	20.00 (2.02)	27.08 (1.68)	82.18 (2.82)	75.34 (1.93)
	Int-Grad	43.76 (3.27)	32.91 (2.36)	34.01 (2.55)	43.22 (2.28)
	Anchors	11.93 (1.53)	30.96 (1.87)	55.72 (2.60)	48.86 (2.38)
	Gradient Search	17.55 (1.47)	33.98 (1.43)	53.15 (2.53)	49.36 (1.95)
	Taylor Search	6.91 (1.10)	28.00 (1.46)	73.20 (2.57)	66.76 (2.12)
	Ordered Search	-1.45 (0.93)	15.06 (1.37)	87.78 (2.41)	84.67 (1.61)
	Random Search	-1.54 (0.96)	15.38 (1.39)	87.36 (2.47)	84.63 (1.68)
	PLS	**-1.65** (1.07)	**14.16** (1.38)	**87.95** (2.55)	**86.18** (1.45)
BoolQ	LIME	2.15 (1.75)	-1.56 (0.63)	52.02 (3.69)	36.25 (3.45)
	Int-Grad	20.78 (3.57)	9.05 (1.53)	16.80 (1.57)	12.20 (1.68)
	Anchors	11.98 (2.62)	6.07 (1.06)	29.87 (4.17)	15.46 (1.97)
	Gradient Search	5.12 (1.41)	1.65 (0.81)	30.04 (2.58)	17.65 (1.85)
	Taylor Search	6.01 (1.33)	2.28 (0.87)	46.32 (3.89)	26.65 (2.68)
	Ordered Search	0.09 (0.84)	-2.58 (0.70)	51.59 (3.52)	34.36 (3.34)
	Random Search	-0.58 (0.63)	-2.51 (0.70)	55.78 (3.71)	31.62 (3.06)
	PLS	**-1.17** (0.47)	**-3.52** (0.88)	**72.78** (4.06)	**47.80** (3.57)
Evidence Inference	LIME	-16.07 (2.84)	-14.92 (1.38)	47.60 (5.66)	33.97 (4.22)
	Int-Grad	1.22 (4.42)	-2.98 (1.68)	26.51 (2.68)	20.87 (2.57)
	Anchors	7.08 (4.70)	3.04 (0.99)	25.01 (6.52)	13.89 (1.55)
	Gradient Search	-10.57 (2.58)	-7.56 (1.46)	31.73 (4.43)	18.07 (2.13)
	Taylor Search	-4.55 (2.66)	-3.33 (1.27)	41.95 (5.63)	26.70 (3.00)
	Ordered Search	-16.80 (2.75)	-14.26 (1.36)	45.37 (5.53)	31.14 (3.73)
	Random Search	-17.05 (2.83)	-12.69 (1.30)	42.81 (6.00)	26.48 (3.15)
	PLS	**-20.76** (3.77)	**-20.33** (2.65)	**56.31** (9.81)	38.71 (3.91)
FEVER	LIME	-0.24 (0.50)	0.39 (0.96)	33.86 (3.43)	22.06 (2.36)
	Int-Grad	9.72 (1.80)	4.99 (1.40)	17.81 (2.47)	13.69 (1.71)
	Anchors	6.19 (1.22)	6.36 (1.10)	20.82 (2.58)	11.94 (1.84)
	Gradient Search	0.66 (0.68)	2.63 (1.12)	19.26 (2.68)	11.44 (1.65)
	Taylor Search	4.17 (0.96)	4.20 (1.20)	24.51 (2.78)	15.62 (1.85)
	Ordered Search	-1.26 (0.41)	-0.01 (0.90)	31.79 (3.28)	18.90 (2.46)
	Random Search	-1.51 (0.51)	-1.24 (2.33)	32.47 (3.33)	18.84 (2.11)
	PLS	**-2.04** (0.62)	**-3.66** (0.82)	**37.72** (3.28)	**24.07** (2.46)
MultiRC	LIME	-5.20 (1.18)	-5.90 (1.19)	39.75 (4.84)	**28.57** (2.18)
	Int-Grad	13.19 (3.14)	4.66 (1.71)	15.53 (3.39)	11.84 (1.31)
	Anchors	5.40 (3.34)	3.33 (1.27)	24.53 (8.77)	14.55 (1.66)
	Gradient Search	-0.09 (1.33)	-0.73 (1.18)	20.16 (2.92)	11.41 (1.13)
	Taylor Search	7.54 (2.53)	1.43 (1.47)	30.76 (4.04)	20.15 (1.83)
	Ordered Search	-6.43 (0.98)	-5.49 (1.13)	35.70 (4.40)	24.38 (2.03)
	Random Search	-7.42 (1.08)	-5.97 (1.22)	35.29 (4.59)	22.19 (1.81)
	PLS	**-10.17** (1.43)	**-9.77** (1.49)	39.95 (5.44)	26.96 (2.19)

significant, and improvements are as large as 20.8 points over LIME and 17 points over Random Search.

2. LIME is the best salience method on both Suff and Comp metrics, but it is still outperformed by Random Search on Sufficiency in 9 of 10 comparisons,

by up to 21.5 points. LIME does appear to perform better than Random Search on Comprehensiveness with three of five datasets for Standard models and four of five with CT models, where the largest improvement over Random Search is 7.49 points.

3. Suff and Comp scores are often much worse for CT models than for Standard models. With Random Search, for instance, Comp scores are worse for all datasets (by up to 24.16 points), and Suff scores are worse by 16.92 points for SNLI, though there are not large changes in Suff for other datasets. The differences here show that the OOD nature of counterfactual inputs can heavily influence metric scores, and they lend support to our argument about the OOD problem in Sec. 7.4. In particular, these metrics are more easily optimized when counterfactuals are OOD because it is easier to identify feature subsets that send the model confidence to 1 or 0.

Given the results above, we recommend that explanations be obtained using PLS for models trained with Counterfactual Training. Though explanation metrics are often worse for CT models, the only reason for choosing between Standard and CT models is that CT models' explanations are socially aligned, while Standard models' explanations are socially misaligned. It would be a mistake to prefer standardly trained models on the grounds that they are "more easily explained" when this difference is due to the way we unintentionally influence model behavior for OOD data points. When using CT models, however, we should be comfortable optimizing for Sufficiency and Comprehensiveness scores, and PLS produces the best explanations under these metrics.

We give additional results for RoBERTa and reduced search budgets in the Supplement.

7.7 Conclusion

In this chapter, we provide a new argument for why it is problematic to use out-of-distribution counterfactual inputs when creating and evaluating feature importance explanations. We present our Counterfactual Training solution to this problem, and we recommend certain `Replace` functions over others. Lastly, we introduce a Parallel Local Search method (PLS) for finding explanations, which is the only method we evaluate that consistently outperforms random search.

8 Model Editing and Belief Graphs for LMs

Now, we transition from work on interpretability to work on controllability, though, broadly speaking, treating LMs with the intentional stance is still a mode of interpreting how LMs work (behavior is determined by desires and *beliefs*). Here, I discuss what it would mean for LMs to have "beliefs" and how to edit them.

8.1 Introduction

Language models (LMs) may not have beliefs in the same sense that people do, but there are a few reasons to analyze LMs in terms of the beliefs they may possess. The first is that this is a useful way to understand and speak about how LMs behave. When discussing whether animals have beliefs (raccoons, in particular), philosopher Daniel Dennett [1995] writes:

> *You might as well call the state of the raccoon a belief, since if you call it a "registration" or a "data-structure" in the "environmental information store" or some other technical term, the logic you use to draw inferences about the animal's behavior, given its internal states, will be the standard, "intentionalistic" logic of belief.*

Dennett bases this conclusion in the fact that we can and do draw accurate inferences about animal behavior by first understanding their beliefs. We are drawn to speak about the beliefs of LMs in the same "maximally bland (but maximally useful!)" sense. To the extent that these neural networks act intelligently in response to stimuli, we may form more accurate theories of how they work by understanding their beliefs.

The second reason for ascribing beliefs to language models is that many of the stricter definitions of belief incidentally exclude many real beliefs held by real people. Following Dennett [40], Newen and Starzak [140] define a belief as *an informational state decoupled from any motivational state*, and they outline a few additional properties of beliefs, namely that they should (1) be recombinable with motivational states and other informational states and (2) have some minimal structural organization. Further, an entity with beliefs should (1) be sensitive to new information, (2) categorize new beliefs as they develop, and (3) display some kind of logical consistency. These are all properties that come in degrees, and setting the bar too high will exclude many of the statements that people earnestly express to others in their everyday lives. Meanwhile, animals and neural networks alike store information in accordance with these properties to at least some extent.

We also note that we use the term *belief* rather than *knowledge* as in related work [240, 38] because we want to analyze beliefs *of* language models rather than knowledge *in* them. LMs may contain a great deal of knowledge *to us*, but in a traditional view of knowledge as Justified True Belief, it is relatively more difficult to say that an LM knows something rather than that it believes it [182].

In the remainder of this paper, we turn our attention to three practical endeavors: *detecting*, *updating*, and *visualizing* beliefs in LMs. We build on work on editing models after training, which is an exciting recent direction of research with many potentially valuable use cases [187, 240, 38, 132]. For LMs, uses include correcting factually inaccurate outputs and preventing otherwise unwanted outputs from models (e.g. toxic generated text) without expensive data curation and retraining efforts. These are important applications given that

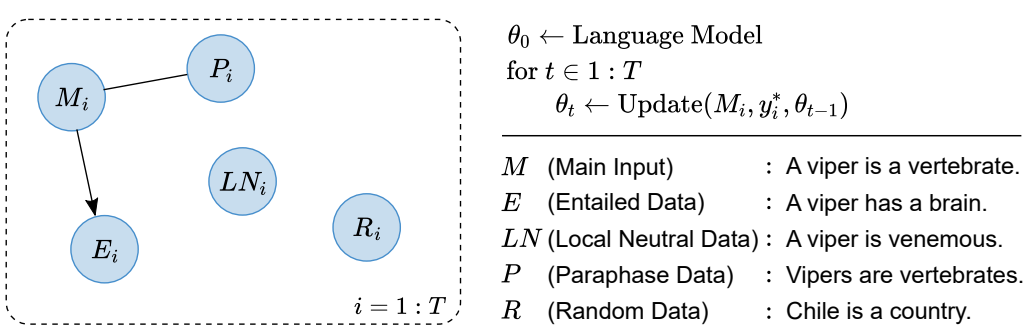


Figure 8.1: Relying only on a Main Input M_i, we want to make a targeted update to a language model that (1) changes the output for input M_i to a desired label y_i^* (e.g. True/False, or an answer to a question), (2) changes the output for equivalent paraphrases of M_i, (3) appropriately changes outputs for data E_i entailed by the tuple (M_i, y_i^*), and (4) does *not* change outputs for other logically neutral data LN_i, even if it is similar (local) to M_i.

LMs (1) struggle with future data when trained on data from the past [109, 43], (2) generate morally undesirable text in many situations [55, 18], and (3) simply give inaccurate outputs for tasks like question answering [118]. Notably, there is good evidence that scaling models to larger sizes will not fix these particular problems or may even exacerbate them in cases like imitative falsehoods in QA, so we will likely need an alternative solution [109, 55, 118]. We next outline a few key contributions of the paper. Figure 8.1 represents the core ideas behind these contributions.

Detecting beliefs. We measure the degree to which LMs exhibit several properties of belief-possessing systems, using models finetuned on knowledge-intensive tasks including fact verification and question answering tasks. Beyond simply checking individual model responses, we want to assess the structural properties of model outputs: Are they consistent under paraphrase? Are they logically consistent? Does changing one belief correctly change other entailed beliefs? Does it erroneously change other unrelated beliefs? Past work has focused primarily on consistency under paraphrase [52, 38, 132]. Here, we adapt data from Talmor et al. [201] to measure consistency under entailment (including for contrapositives), and we use the Wikidata5m dataset [212] to construct logically neutral belief pairs for checking that models do treat these beliefs as independent.

Updating beliefs. We propose a Sequential, Local, and Generalizing belief update objective (SLAG) that substantially improves the performance of the KnowledgeEditor method from De Cao et al. [38]. KnowledgeEditor is a learned optimizer that edits a model's weights in order to change its prediction on an input while satisfying other desiderata, like consistency under paraphrase. Principally, we use more difficult training data for the learned optimizer, and we also train the network to apply multiple small edits rather than just one edit. These changes markedly improve the overall update success rate and lower the rate at which other beliefs are corrupted. Moreover, we find that KnowledgeEditor almost totally fails when updating multiple beliefs in a row as opposed to a changing a single belief. In this setting, off-the-shelf optimizers are far preferable methods. However, by explicitly training the optimizer to update multiple beliefs sequentially, we are able to once again outperform off-the-shelf optimizers. Lastly, we advocate that these methods be evaluated for their ability to correct false or morally undesirable model beliefs, rather than to arbitrarily adjust model

beliefs to plausible alternatives as in past work [240, 38, 132].

Visualizing belief graphs. We explore a new form of interface with LMs, the *belief graph*. Given a set of beliefs, we construct belief graphs by changing each model belief and checking what other beliefs are sensitive to those changes. Each belief becomes a node, and directed edges between nodes show that updating one belief changes the other. We discuss graph metrics that help summarize the dependencies between model beliefs.

We summarize our main conclusions as follows:

1. ∼100M parameter models exhibit limited belief-like qualities, as paraphrase consistency scores are under 70%, and models show mixed levels of consistency under entailment (Sec. 8.5.1).
2. Off-the-shelf optimizers are surprisingly effective baselines for updating model beliefs, and they generally outperform learned optimizers when updating a single belief (Sec. 8.5.2).
3. When updating multiple beliefs in a row, method performance greatly declines (especially for learned optimizers). By using SLAG, we can improve learned optimizers' performance beyond what baselines can reach (Sec. 8.5.2).
4. Belief graphs reveal many nonsensical dependencies between model beliefs. We find that (1) updates are mostly likely to change already incorrect model beliefs and (2) there are highly connected beliefs which influence a large fraction of all model beliefs (Sec. 8.6.3).

8.2 Related Work

Detecting beliefs in language models. Much past work has explored how information is stored and represented in pretrained language models [170], though few discuss what qualifies information as a model belief. Petroni et al. [151] provide evidence that LMs store relational information between entities, and Roberts et al. [168] show that LMs can answer open-ended questions. Subsequent work has explored how much knowledge is stored in LMs [75], approaches to querying models for knowledge [79, 92, 208, 215], and methods for learning more knowledge during pretraining [212, 211]. Most relevant to our work are studies from Talmor et al. [201] and Elazar et al. [52]. Talmor et al. [201] train LMs to perform True/False classification of factual claims, and they measure how a model's belief in one fact correlates with its belief in an entailed fact. We use their LeapOfThought dataset to measure model consistency under entailment before and after updating the up-stream beliefs in models. Meanwhile, Elazar et al. [52] measure the consistency of model predictions for sets of paraphrased inputs. We adopt their metric for paraphrase consistency as a measure of belief. In concurrent work, Kassner et al. [96] discuss consistency under entailment and paraphrase as conditions for belief, and they measure consistency under entailment with a new dataset, BeliefBank.

Updating beliefs in language models. Approaches to making targeted updates to model beliefs vary along a few dimensions. First is whether the methods alter model training or operate in a post-training setting. Sinitsin et al. [187] use a meta-learning objective during training to encourage ease of editing afterwards, though the memory requirements of their approach limit its scalability beyond 100M parameter models. A larger family of methods make post-training updates to models, differing in how they formalize the update problem: Dai et al. [36] propose a hand-crafted algorithm for updating model weights, while Zhu et al. [240] use projected gradient descent for batches of points. De Cao et al. [38] and Mitchell et al. [132] frame the problem as a machine learning problem and train hypernetworks (learned

Dataset	Data Type	Input	Label(s)
zsRE	Main Input	Player Ali Kanaan plays for what team?	{Sporting Al Riyadi Beirut}
	Paraphrase	What team is Ali Kanaan associated with?	
Wikidata5m	Main Input	Mary Good has relation 'award received' to	{Garvan-Olin Medal; Arkansas Women's Hall of Fame; etc.}
	Paraphrase	Mary Lowe Good has relation 'winner of' to	
	Local Neutral	Mary Good has relation 'educated at' to	{The University of Arkansas; U Arkansas; etc.}
FEVER	Main Input	Tardigrades are also known as space bears.	True
	Main Input	The Lion belongs to the genus Vulpes.	False
LeapOfThought	Main Input	A viper is a vertebrate.	True
	Entailed Data	A viper has a brain.	True

Table 8.1: Example datapoint from each dataset, and auxiliary data that accompanies the Main Input. We catalogue examples of noise and other shortcomings for each dataset in Appendix E.3.

optimizers) that process model gradients in order to produce a new model that (1) gives the desired output for the edited point, while (2) incorporating other objectives like minimizing the changes in predictions for other data. Here, we build directly upon the method from De Cao et al. [38], showing where it fails and providing an improved training objective (SLAG). In particular, we find that the method struggles with updating multiple beliefs sequentially. This setting bears some commonality to the continual learning problem, though continual learning methods generally aim to learn new tasks or datasets rather than make targeted updates to specific model beliefs [146].

Not all methods edit model weights. Kassner et al. [96] update model beliefs by adding in relevant information to the input at test time (to improve consistency under entailment). But as with retrieval-based methods, this approach does not change the model weights and hence does not influence model outputs on all other potentially relevant inputs [112, 66].

8.3 Updating Beliefs in Language Models

Following De Cao et al. [38], we approach the problem of updating model beliefs as a machine learning problem and train a learned optimizer to perform desired model updates. We discuss metrics for detecting beliefs in Sec. 8.5.1 and our approach to visualizing belief graphs in Sec. 8.6.3. The core ideas of our approach are outlined in Fig. 8.1.

Problem statement and metrics. We suppose we have a model $f_\theta = p_\theta(y|x)$ parametrized by θ. For an input x_i that has some undesired model output $\hat{y}_i = \arg\max_y p_\theta(y|x)$, we wish to obtain a new model θ^* that produces a desired output y_i^* for x_i. This new model θ^* should also fulfill a few other desiderata. As in past work [38, 132], we operationalize these desiderata in the following metrics:

1. Update Success Rate (*Main Input*): The rate at which the updated model gives the desired output y_i^* for the Main Input x_i.

2. Update Success Rate (*Paraphrase*): The rate at which the updated model gives the same new prediction for x_i as it does for paraphrases of x_i, which are inputs with the same meaning but different surface form.

3. Retain Rate (*All Data*): The rate at which the updated model's predictions are unchanged for all other data besides the Main Input.

4. Δ-Acc (*All Data*): The change in accuracy for the updated model on all other data besides the Main Input.

In practice, Retain Rate (*All Data*) and Δ-Acc are computed with random subsets of a dataset, since these must be computed after every belief update. We add two metrics to those used in past work:

5. Update Success Rate (*Entailed Data*): The rate at which the updated model makes predictions that are logically entailed by the model's prediction for the Main Input.

6. Retain Rate (*Local Neutral*): The rate at which the updated model's predictions are unchanged for data that is similar to the Main Input but still logically neutral.

We use Update Success Rate (*Entailed Data*) to measure logical consistency for an updated model, since changing one belief will entail changes in logically entailed beliefs. We also split "retain accuracy" into two cases, one for randomly sampled data as in past work (*All Data*) and the other for specially constructed *Local Neutral* data. Unlike randomly sampled data, Local Neutral data is guaranteed to be logically independent of the Main Input, while still being similar (local) to it. Together, these six metrics better cover the criteria for belief outlined by Newen and Starzak [140]. We compute the metrics using data of the kind shown in Table 8.1. For a glossary of terms used for these metrics across papers, see Appendix Table E.5.

Evaluation procedure. To date, methods have been evaluated on the basis of their ability to change model predictions for all data points, including correctly and incorrectly predicted points. Moreover, the desired labels $\{y_i^*\}_{i=1}^n$ on sequence prediction tasks have each been selected from the beam search which produced the original model prediction [38, 132]. We propose for method evaluation to focus on a more valuable use case: changing the predictions on incorrect points to be correct. In Sec. 8.5, we show that this is a harder task than simply changing predictions to other similar outputs, so the effectiveness of past methods has been overestimated.

Sequential updates. The default evaluation procedure in past work on learned optimizers is to update a single model belief, evaluate the new model, then rollback the update before repeating the process for each test point. In Sec. 8.5, we show that it is much harder to update multiple beliefs in a row before evaluating the new model. This is notable because in practice, it is likely that model developers will want to update many beliefs of a trained model, possibly over long timescales, meaning sequential updating is a more realistic application of update methods. We obtain sequential versions of all our metrics by applying r model updates in a row before checking the metrics, meaning there are $\text{floor}(n/r)$ measurements for a test set of n points.

Belief updating method. We use the KnowledgeEditor architecture from De Cao et al. [38] with our training objective, SLAG. For the details of this architecture, we refer readers to Appendix E.1. Let it suffice for now to observe that a new model is given as a differentiable function

$$\theta^* = \theta + g_\phi(x_i, \hat{y}_i, y_i^*, \theta)$$

using the learned optimizer g_ϕ, current LM weights θ, Main Input x_i, current prediction $\hat{y}_i$, and desired model output y_i^*. In this chapter, we generalize the update step to occur in a loop. If we package the above update as $\theta^{(k+1)} = \theta^{(k)} + g_\phi(x_i, \hat{y}_i, y_i^*, \theta^{(k)})$, then we can obtain new model parameters as

$$\begin{aligned}\theta^* &= \theta^{(k)} + \sum_{j=0}^{K-1} g_\phi(x_i, \hat{y}_i, y_i^*, \theta^{(k+j)}) \\ &= \text{Update}(x_i, \hat{y}_i, y_i^*, \theta^{(k)}; \phi, K)\end{aligned}$$

for a number of steps K from the initial parameters $\theta^{(k)}$. In fact, De Cao et al. [38] use such a loop at test time; we incorporate the loop into training to align the train and test-time distributions.

Learned optimizer training. The training objective for KnowledgeEditor includes differentiable terms corresponding to Update Success for the Main Input and paraphrases, as well as Retain Rate for all other data. We also include terms for Update Success on entailed data and the Local Neutral Retain Rate, when this is possible given available data. The overall objective requires several kinds of additional data for each point, which we denote by $\mathcal{D}_R$ for other random data, $\mathcal{D}_{LN}$ for local neutral data, $\mathcal{D}_E$ for entailed data, and $\mathcal{D}_P$ for paraphrases of x_i. For a data point x_i with desired prediction y_i^*, the full objective is then:

$$\begin{aligned}\mathcal{L}(\phi; x_i, \hat{y}_i, y_i^*, \theta) &= \lambda_1 \mathcal{L}_{\text{Task}}(f_{\theta^*}(x_i), y_i^*) \\ &+ \lambda_2 \frac{1}{|\mathcal{D}_P|} \sum_{x_P \in \mathcal{D}_P} \mathcal{L}_{\text{Task}}(f_{\theta^*}(x_P), y_i^*) \\ &+ \lambda_3 \frac{1}{|\mathcal{D}_E|} \sum_{x_E, y_E \in \mathcal{D}_E} \mathcal{L}_{\text{Task}}(f_{\theta^*}(x_E), y_E) \\ &+ \lambda_4 \frac{1}{|\mathcal{D}_{LN}|} \sum_{x_{LN} \in \mathcal{D}_{LN}} \text{KL}(f_{\theta^*}(x_{LN}) || f_\theta(x_{LN})) \\ &+ \lambda_5 \frac{1}{|\mathcal{D}_R|} \sum_{x_R \in \mathcal{D}_R} \text{KL}(f_{\theta^*}(x_R) || f_\theta(x_R))\end{aligned} \tag{8.1}$$

where $\mathcal{L}_{\text{Task}}$ is the loss used to get gradients for f_θ. We use the Cross Entropy loss for binary classification and sequence-to-sequence tasks.

We optimize this objective w.r.t. ϕ using AdamW [124]. To obtain update labels $\{y_i^*\}_{i=1}^n$, we always use the opposite class in binary classification. For sequence-to-sequence tasks, we use the correct label when $\hat{y}_i$ is incorrect, and when $\hat{y}_i$ is correct, we randomly select another label from the training data. This choice is in contrast to De Cao et al. [38] and Mitchell et al. [132], who use samples from the model beam search as update labels for all points.

SLAG objective. To better prepare the update method for evaluation in a sequential-update setting, we consider training g_ϕ to update multiple datapoints in a row. Using the

per-datapoint loss in Eq. 8.1, we obtain our Sequential, Local, and Generalizing (SLAG) loss for a set of r Main Inputs $\mathcal{D} = \{x_i, \hat{y}_i, y_i^*\}_{i=1}^r$ as

$$\mathcal{L}_{\text{Sequential}}(\phi; \mathcal{D}, \theta_t) = \sum_{i=1}^{r} \mathcal{L}(\phi; x_i, \hat{y}_i, y_i^*, \theta_{t+i}) \tag{8.2}$$

where $\theta_{t+i} = \text{Update}(x_i, \hat{y}_i, y_i^*, \theta_{t+i-1}; \phi, K)$ are the model parameters obtained from updating on the first i points in $\mathcal{D}$ (starting from θ_t). This objective allows us to train g_ϕ to update multiple beliefs in a row. To ensure training with this objective is still efficient, we limit how far back through the LM history we backpropagate when computing the gradient w.r.t. ϕ for each term in the RHS sum of Eq. 8.2. Each parameter vector θ_t depends on ϕ and θ_{t-1}. We always apply the stop-gradient function to the most recent vector θ_{t-1} to prevent backpropagating through it (visualized in Appendix Fig. E.1). This choice allows our memory use to remain constant in r (see Appendix Fig. E.2).

8.4 Experiment Setup

8.4.1 Datasets

We run experiments with four datasets (example data shown in Appendix Table E.7). (1) FEVER includes 115,409 True/False factual claims [203]. We use the original test set of 10,444 points, and we randomly split the training data into 94,469 train points and 10,496 dev points. (2) zsRE includes 151,631 questions based on relational knowledge from Wikipedia, which we randomly shuffle into train/dev/test splits with 80/10/10% of the data [111]. 32.8% of zsRE questions in each split include paraphrases, and we measure Update Success Rate (*Paraphrase*) for only these points. Talmor et al. [201] introduce (3) the LeapOfThought dataset, consisting of 33,484 factual claims that are entailed to be true or false depending on a fact and distractor statements provided as context. We drop the distractors from each input and filter the data so that the facts are unique, then shuffle the resulting 14,939 points into train/dev/test splits with 60/10/30% of the data.

We also construct (4) a sequence prediction task using data from Wikidata5m, which is a relational knowledge base with over 20 million triplets [212]. We build this dataset in order to get Local Neutral data. Each input consists of an entity e_1 and relation r, and the label is another entity e_2 that completes the triplet. All inputs come in pairs that share the same entity e_1 but use different relations with different labels. The relations are always one of ten relations that apply to people (see Appendix Table E.3). In general, the completion e_2 to the Main Input triplet (e_1, r_1, e_2) has no logical consequences for its paired input, (e_1, r_2, ?). This means that changing the model belief for the Main Input should not change its belief for its neutral paired input. The paired points are also local to the Main Input, i.e. they pertain to the same entity e_1 as the Main Input. We obtain four paraphrases for each Main Input using different aliases for the entity and synonyms of the relation. We construct a train set of 150k points and dev and test sets of 10k points each. See Appendix E.2 for further details.

8.4.2 Methods Evaluated

Models. We train five models with different random seeds for each dataset, using RoBERTa-base for binary tasks and BART-base for sequence-to-sequence tasks (accuracies in Appendix Table E.6). For each of the five models, we train one learned optimizer using SLAG and

	Belief Consistency ↑		
Dataset	Paraphrase	Entailed	Contrapos.
LeapOfThought	-	85.6 (1.1)	16.5 (2.7)
zsRE	69.5 (1.1)	-	-
Wikidata5m	25.8 (0.5)	-	-

Table 8.2: Belief metric results across datasets.

	Paraphrase Consistency ↑	
Dataset	Model Incorrect	Model Correct
zsRE	61.39 (1.33)	91.82 (1.17)
Wikidata5m	24.55 (0.48)	37.20 (2.06)

Table 8.3: Paraphrase consistency by the correctness of the model prediction on the Main Input.

one with the objective from De Cao et al. [38], which we list as KE in tables below. Our model selection criterion is the mean of: the average Update Success Rate (across data types), Retain Rate (only for Local Neutral data), and Δ-Acc for All Data. We tune the choice of SLAG objective terms for each task separately (see Appendix Table E.2 for final selections; results discussed in Sec. 8.5.3). Other hyperparameters are given in Appendix E.2 and memory use is shown in Appendix Fig. E.2. To summarize the differences between SLAG and KnowledgeEditor: (1) we use $K_{\text{train}} = K_{\text{test}}$ rather than $K_{\text{train}} = 1$; (2) we adopt training labels using real data labels rather than alternatives from the model's beam search; and (3) our objective terms differ following tuning.

Baselines. We use off-the-shelf optimizers as baselines. We tune the baseline hyperparameters separately for each dataset, selecting among several kinds of optimizers, learning rates, and the number of update steps. The selection criterion is the same as the criterion outlined for learned optimizers above. The resulting baselines are surprisingly strong (see Appendix Table E.4 for final selections).

Hypothesis testing. We obtain 95% confidence intervals and perform hypothesis tests via block bootstrap, resampling model seeds and data points [50]. For ablation experiments, we run only one model seed per condition.

8.5 Experiment Results

8.5.1 Do LMs have beliefs about the world?

We measure Paraphrase Consistency, Entailment Acc, and Contrapositive Acc for our finetuned task models. Paraphrase Consistency is the fraction of paraphrase pairs for which a model produces the same output [52]. Entailment Acc is the accuracy of a model on data that is entailed by the Main Input. For LeapOfThought (see Table 8.1), "Main Input x_i is true" implies "entailed input x_E has label y_E," but the inverse ($\neg A \Rightarrow \neg B$) does not necessarily hold. Therefore, we compute Entailment Acc only where the Main Input prediction is correct. We do know that the contrapositive holds: "Entailed input x_E does not have label y_E" implies that "Main Input x_i is false." So for Contrapositive Acc, we measure how often the model follows this rule, when the antecedent holds of its prediction.

Single-Update Setting		Update Success Rate			Retain Rate		Δ-Acc
Dataset	Method	Main Input	Paraphrases	Entailed Data	Local Neutral	All Data	All Data
FEVER	AdamW	100 (0.0)	-	-	-	**98.80** (0.2)	**0.22** (0.1)
	KE	99.98 (0.1)	-	-	-	98.28 (0.3)	-0.24 (0.1)
	SLAG	99.99 (0.1)	-	-	-	98.41 (0.2)	-0.20 (0.1)
LeapOfThought	SGD	100 (0.0)	-	72.48 (4.6)	-	95.52 (0.4)	**1.23** (0.8)
	KE	99.78 (0.4)	-	74.48 (4.4)	-	93.50 (1.3)	-1.33 (1.1)
	SLAG	100 (0.0)	-	75.50 (4.3)	-	94.92 (1.4)	-1.31 (1.2)
zsRE	SGD	**99.36** (0.1)	94.44 (0.6)	-	-	74.73 (0.4)	-0.43 (0.1)
	KE	84.73 (1.4)	89.26 (1.8)	-	-	71.55 (2.4)	-2.19 (0.4)
	SLAG	94.29 (0.4)	94.71 (0.5)	-	-	**80.48** (1.3)	-0.29 (0.1)
Wikidata5m	SGD	**98.05** (0.3)	68.78 (0.8)	-	41.46 (1.0)	58.62 (0.6)	-1.97 (0.3)
	KE	74.57 (2.9)	58.05 (2.2)	-	40.84 (1.8)	53.58 (2.2)	-3.03 (0.5)
	SLAG	87.59 (0.6)	**80.70** (0.9)	-	**47.85** (1.0)	**63.51** (1.3)	-1.71 (0.3)

Table 8.4: Belief update metrics for off-the-shelf optimizers, KnowledgeEditor (KE) from De Cao et al. [38], and SLAG, with $r_{\text{test}} = 1$. Bolded numbers are the best in their group at a statistical significance threshold of $p < .05$ (or lower). Our SLAG objective improves over KE, but off-the-shelf optimizers perform surprisingly well.

	Update Success Rate ↑		Δ-Acc ↑
Desired Label	Main Input	Paraphrase	All Data
Beam Label	97.41 (0.3)	97.03 (0.4)	-0.30 (0.1)
Correct Label	94.46 (0.7)	94.45 (0.7)	-0.24 (0.1)

Table 8.5: Evaluation difficulty by desired model output, for a learned optimizer trained with SLAG on zsRE.

Belief measurement results. Table 8.2 shows the belief metrics for each dataset. We find that ∼100M parameter models show limited evidence of having beliefs about the world. Paraphrase consistency is 69.50% (± 1.09) for zsRE and much lower for Wikidata5m (25.84%±0.53). While entailment accuracy is high for LeapOfThought (85.63%±1.08), the model is consistent under the contrapositive only 16.51% (± 2.71) of the time. One might reasonably set the bar for qualifying as a "belief" higher than these scores. But since belief-likeness comes in degrees, we continue to refer to model beliefs for the rest of the paper. Interestingly, the metrics are much higher when the model prediction on the Main Input is correct (Table 8.3).

8.5.2 Can we update beliefs in LMs?

First, we compare two evaluation procedures for sequence prediction tasks: correcting model beliefs versus changing them to an alternative from the model's beam search. We do so for zsRE using SLAG. Next, we compare belief update metrics across datasets using KnowledgeEditor, SLAG, and off-the-shelf optimizers as baselines. We report results in single-update ($r_{\text{test}} = 1$) and sequential-update ($r_{\text{test}} = 10$) settings. See Appendix Fig. E.3 for an ablation across r_{test}.

Correcting beliefs vs. changing beliefs. Given the results in Table 8.5, we find that correcting model outputs is harder than simply changing them to a plausible alternative. Update Success can rise by a full 2.96 (±0.48; p<1e−4) points for Main Inputs and 2.58

Sequential-Update Setting		Update Success Rate			Retain Rate		Δ-Acc
Dataset	Method	Main Input	Paraphrases	Entailed Data	Local Neutral	All Data	All Data
FEVER	AdamW	92.81 (1.3)	-	-	-	**91.86** (1.4)	**1.16** (0.6)
	$SLAG_1$	74.13 (1.8)	-	-	-	39.86 (0.7)	-27.13 (1.3)
	$SLAG_{10}$	91.27 (2.9)	-	-	-	70.30 (5.8)	-11.96 (4.5)
LeapOfThought	SGD	100 (0.0)	-	**61.34** (5.0)	-	**82.62** (0.8)	**-4.93** (1.0)
	$SLAG_1$	96.14 (2.3)	-	49.27 (6.0)	-	72.45 (0.9)	-15.03 (1.0)
	$SLAG_{10}$	100 (0.0)	-	50.46 (5.5)	-	74.02 (1.1)	-13.03 (1.3)
zsRE	SGD	82.71 (0.6)	90.81 (0.7)	-	-	40.49 (0.6)	-2.38 (0.3)
	$SLAG_1$	0.10 (0.1)	36.55 (1.4)	-	-	0.05 (0.1)	-20.98 (0.7)
	$SLAG_{10}$	**87.57** (0.6)	**92.20** (0.7)	-	-	**47.19** (0.7)	**-1.74** (0.3)
Wikidata5m	SGD	56.82 (0.8)	54.49 (0.7)	-	6.40 (0.4)	26.37 (0.6)	-3.96 (0.4)
	$SLAG_1$	0 (0.0)	40.84 (0.9)	-	0 (0.0)	0 (0.0)	-10.05 (0.6)
	$SLAG_{10}$	**58.27** (1.0)	**65.51** (0.9)	-	**7.36** (0.5)	**27.76** (0.7)	-3.62 (0.4)

Table 8.6: Belief update results when a model is sequentially updated r_{test}=10 times. $SLAG_R$ uses r_{train}=R. On sequence prediction tasks in this setting, SLAG can outperform the off-the-shelf optimizers across metrics.

(±0.81; p<1e−4) for Paraphrases, while Δ-Acc is virtually unchanged. This suggests that that past work has overestimated the efficacy of belief update methods for actually fixing models. Henceforth we evaluate methods according to their ability to update model beliefs to be true.

Update method results (single update). Table 8.4 shows the results in a single-update setting. First, we find that off-the-shelf optimizers are very effective across the board. The baselines show Main Input Update Success Rates of 100% for binary tasks with positive Δ-Acc scores.[1] On sequence prediction tasks, SGD achieves 98%+ Main Input Update Success with competitive Δ-Acc scores. When strongly tuned, these baselines outperform learned optimizers on most metrics here.

However, SLAG does surpass the baselines in a few places. All Data Retain Rate on zsRE rises by 5.77 points (±1.43; p<1e−4), and on Wikidata5m we improve Paraphrase Update Success by 11.92 points (±1.20; p<1e−4) and the Local Neutral Retain Rate by 6.40 (±1.41; p<1e−4) points. The gain on Entailed Data Update Success is 3.02 points, but it is not significant (±6.26; p=.345). The SLAG objective also greatly improves performance over KE for sequence prediction tasks.

Update method results (sequential updates). We give results for a sequential update setting (r_{test}=10) in Table 8.6. Immediately we see this is a much more difficult setting for updating model beliefs, as update metrics are generally much lower for each dataset. Next, we observe that learned optimizers with $SLAG_{10}$ (r_{train}=10) now outperform baselines on sequence prediction tasks. On zsRE, we improve Update Success for Main Inputs by 4.86 (±0.83; p=1e−4) and for Paraphrases by 1.39 (±0.93; p=.004), with better Δ-Acc by 0.64 (±0.35; p=.0005). Improvements trend in the same direction for Wikidata5m and are all statistically significant except for the gain in Δ-Acc. The jump on Paraphrases in particular is very large (11.02±1.17; p<1e−4). In comparison, using a non-sequential ($r_{\text{train}} = 1$) training

[1]Positive Δ-Acc values are possibly due to distribution shift in the test split. In FEVER, for instance, the train and dev data are 73% True, while test data is 50% True. On the dev split, AdamW achieves a negative Δ-Acc, -0.18 (±0.11).

Metric	Before Update	After Update
Entailment Acc	58.30 (5.7)*	75.50 (4.3)
Para. Cons (zsRE)	61.39 (1.3)	94.53 (0.6)
Para. Cons (Wiki)	24.69 (0.5)	84.56 (0.9)

Table 8.7: Entailment Acc and Paraphrase Consistency rise considerably after model updates to incorrect points. *All Main Inputs in this subset are wrongly predicted as false, so the entailment does not actually hold.

objective leads to drastic drops in performance.

Learned optimizers still struggle with the binary datasets compared to the off-the-shelf optimizers. The baselines achieve high update update success with much better Δ-Acc scores, by 13.12 ($\pm$4.51; p=1e$-$4) on FEVER and 8.16 ($\pm$1.63; p=1e$-$4) on LeapOfThought. Also on LeapOfThought, the baseline's update success with entailed data is over 10 points higher ($\pm$7.38; p=.004).

8.5.3 How does the learned optimizer objective influence performance?

Here, we discuss ablations with respect to the terms in the training objective, Eq. 8.1. We show the effect of K_{train} in Appendix Fig. E.6 and the choice of optimizer training labels in Appendix Table E.8.

Training objective ablation. We give objective ablation results in Appendix Table E.9. Surprisingly, we do not always see that the objective terms help for the data they are intended to help with. First, we obtain mixed results for the paraphrase objective. On zsRE, the objective term seems to hinder performance, with update success dropping on Main Inputs by 0.71 ($\pm$0.60; p=.021) and Δ-Acc dropping by 0.18 ($\pm$0.19; p=.069), while the paraphrase Update Success Rate itself is unaffected. With Wikidata5m, however, the paraphrase term improves paraphrase update success by a large margin of 16.94 ($\pm$1.03; p<1e$-$4) points. Adding the Local Neutral (LN) term with the paraphrase term greatly improves the LN Retain Rate for Wikidata5m, by 9.71 points ($\pm$1.44; p<1e$-$4), though both of these terms come at a cost to Main Input Update Success, similar to zsRE. Lastly, we do not find that the entailment objective improves Entailed Data Update Success; in fact, this metric falls by 4.56 ($\pm$7.22; p=.213) points with the objective.

8.6 Analysis

8.6.1 Belief updates improve consistency

In Table 8.7, we show belief metrics before and after model updates using SLAG with r_{test}=1. We observe that belief updates greatly improve paraphrase consistency and entailment accuracy for updated data. Paraphrase consistency rises by 33.14$\pm$1.46 on zsRE and 59.87$\pm$1.09 on Wikidata5m, while Entailment Acc rises by 17.20$\pm$7.10 points. To see if these improvements depend on pre-update consistency, we plot paraphrase consistency before and after updating in Fig. 8.2. For zsRE, consistency rises irrespective of pre-update consistency. There is a noticeable trend for Wikidata5m paraphrases, where post-update consistency is 90.1% when pre-update consistency is maxed out, versus 77.1% for totally inconsistent pre-update beliefs. We conclude that learned optimizers can induce a fairly consistent model belief even where there is no consistent belief to begin with.

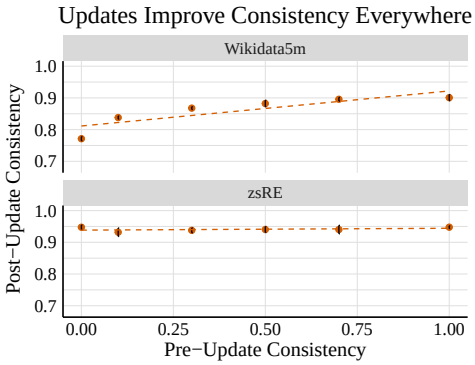


Figure 8.2: Post-update consistency under paraphrase is high even for points with low pre-update consistency.

8.6.2 Which beliefs are hard to retain when updating other beliefs?

We find that the Retain Rate depends heavily on whether the predictions on that data are correct to begin with. On zsRE for instance, the retain rate on correct inputs is about 96%, while for incorrect predictions, it is about 75%. So it appears that incorrect predictions are the most sensitive to model updates, and these points merely change from one incorrect prediction to another. On FEVER, incorrect beliefs change around 4% of the time, while correct beliefs change only 2.5% of the time.

We also find that Local Neutral beliefs are much harder to avoid changing than simply random data. For Wikidata5m in Table 8.4, the Retain Rate on All Data is 61.51±1.33, while for Local Neutral data it is a full 15.66 points lower, at 47.85±0.96.

8.6.3 Belief Graphs

We now construct *belief graphs* for the purpose of better understanding the connections between model beliefs. We form the graphs from a set of datapoints by updating each prediction and checking what other predictions change. We represent each datapoint as its own node in a belief graph. Whenever updating a datapoint u changes the model prediction for point v, we draw a directed edge from u to v. Following our results in Sec. 8.5.2, we use off-the-shelf optimizers to change the model output to the opposite of its original prediction for every datapoint. The resulting graphs have up to $n^2 - n$ edges (no self edges). For FEVER we obtain a graph of 10,444 nodes, and for LeapOfThought we obtain a graph with 8642 nodes, which is double the original test set size because we include both Main Inputs and Entailed Data as their own nodes.

We visualize part of a belief graph in Fig. 8.3. This figure shows a non-random subgraph intended to give a representative view of the data (we give three random subgraphs of 20 nodes in Appendix E.5). On inspection, we see no reason that beliefs are connected or not connected. Whether or not changing one belief changes another appears essentially random. We come to same conclusion looking at other random subgraphs (see Appendix Figures E.7, E.8, E.9). However, we do know of some aggregate trends from earlier results. Sec. 8.6.2 suggests that a model's incorrect beliefs are most likely to change after model updates, and following Sec. 8.5, we have reason to believe that Local beliefs are more likely than others to change with model updates.

We highlight a few summary statistics here from Table 8.8 for a broader view of the graphs. First, % Edgeless is the proportion of nodes which have no in or out edges. Since this is 0

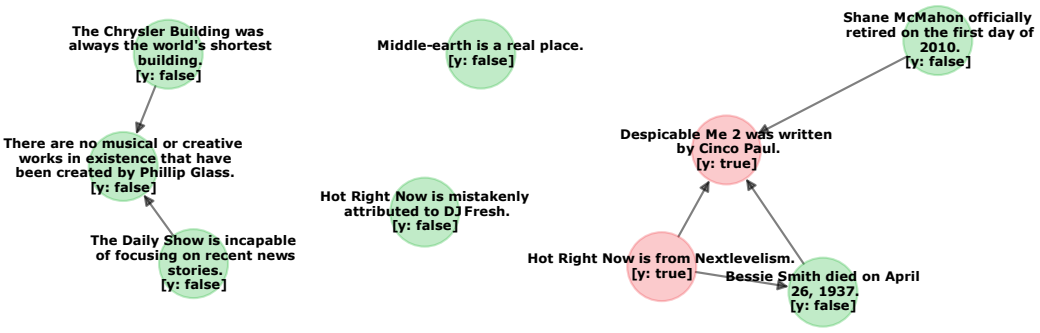


Figure 8.3: A non-random subgraph of the belief graph for a model trained on FEVER. Directed edges from u to v indicate that changing the model belief in u causes the belief in v to change. The ground-truth label is given in brackets for each point, and node color shows the model's accuracy before any updates (green=correct).

	Dataset	
Metric	FEVER	LeapOfThought
# Nodes	10,444	8,642
% Edgeless	0.0	0.0
# Edges Total	1.88m	9.71m
# In Edges (95^{th} perc.)	1,088	5,347
# Out Edges (95^{th} perc.)	390	3,087
# Corrupted (95^{th} perc.)	211	2,752
% Update-Transitivity	66.64	24.38*

Table 8.8: Belief graph summary statistics. *We compute Update-Transitivity for LeapOfThought with $n = 4000$ points due to computational cost.

for both datasets, every belief can be changed by editing the right belief. # In Edges is the number of in edges at the 95^{th} percentile, meaning 5% of beliefs have more in edges than this value, and the same holds of # Out Edges. These values grow to a rather large fraction of the overall datasets, suggesting that (1) some beliefs are sensitive to changes in a large fraction of all beliefs, and (2) some beliefs are influential to hundreds of other beliefs when changed. # Corrupted is the number of correct predictions changed to be incorrect following a model update. For 5% of the data, model updates cause at least 211 points to become incorrectly predicted on FEVER, and 2,752 points for LeapOfThought. Lastly, % Update-Transitivity represents the answer to the question: if updating belief A changes belief B, and updating belief B changes belief C, what proportion of the time does updating A change C? For these datasets, a logically consistent model should display 100% Update-Transitivity (see Appendix E.4 for a caveat on this metric). We find that belief updates often yield intransitive results for both datasets.

8.7 Discussion and Conclusion

Degrees of commitment to beliefs. The data we use comes in the form of declarative statements and answers to questions. These utterances take what is called a veridical stance toward a proposition, displaying a "full commitment" to that proposition's truthfulness [60]. It will be valuable for future work to explore two dimensions of uncertainty in beliefs: (1)

expression of uncertainty in language, via partial or trivial commitments (like "X might be Y") and (2) expression of uncertainty mathematically, via probabilities assigned by a model to utterances or True/False values. In this chapter we treat a belief as "updated" when the model output changes, but this ignores any underlying change in the distribution $p_\theta(y|x)$ that could occur even if its mode does not change.

Ethics and dual use concerns. Belief update methods may be used to either correct undesired beliefs or induce problematic beliefs in LMs, and it is not clear whether these capabilities could be separated. We propose to evaluate methods only on the basis of their ability to correct mistaken model beliefs, but the malicious use case remains. We are uncertain about how a bad belief would influence the general behavior of a model (e.g. answers to many questions), but it is possible that a belief update method could instill bad beliefs in a generally capable LM with far-reaching implications for model behavior. That said, we hope that these methods will instead be used to update undesirable moral, social, and factual beliefs in large LMs.

Conclusion. We first discuss criteria for detecting when LMs have *beliefs* about the world. Next, we argue for evaluating belief update methods by their ability to correct mistaken beliefs, which is harder than the evaluation done in past work. We show that strongly tuned off-the-shelf optimizers make for surprisingly good belief update methods, even surpassing specialized learned optimizers in several settings. But with a new training objective (SLAG), we are able to outperform these baselines on sequence prediction tasks when updating multiple beliefs one after another. Finally, we introduce *belief graphs* as a means of understanding the connections between model beliefs. We find that model beliefs are highly interconnected, with some beliefs influencing hundreds of other beliefs. While it is hard to point to concrete reasons for individual connections between beliefs, we identify several patterns in the dependencies between beliefs.

9 Localization and Editing of Knowledge in LMs

Lastly, I present work at the intersection of interpretability and controllability for language models. The question is: does understanding how language models store factual knowledge help us update what knowledge they store?

9.1 Introduction

Language models learn a variety of facts about the world during pretraining that can be elicited via natural language prompts [151]. Recent work explores how these facts are stored in model weights and expressed in response to particular prompts, suggesting that MLP weights act as key-value memories that support factual association [57, 128, 58]. Besides improving our scientific understanding of pretrained language models, this kind of investigative work may enable the design of better model editing methods for injecting new facts into model weights, and indeed it has been used to motivate the ROME and MEMIT model-editing methods [128, 129]. These recent methods set a new state of the art for weight edits that successfully rewrite stored facts in language models. Model editing methods could be broadly useful for correcting factual errors in pretrained models, avoiding morally undesirable outputs, and updating models with changing knowledge over time.

The connection between *localization* (identifying components of a model responsible for a certain behavior) and *editing* (changing model components in order to change model behavior) is predicated on the reasonable assumption that one should go about editing a model by first localizing a behavior to a specific component and then choosing to edit that particular component. In the case of ROME and MEMIT, localization is done via Causal Tracing, which measures the information content of hidden representations, and editing is done by treating MLP weights as linear associative memories and injecting new key-value memories into the weights. Meng et al. [128, 129] choose to edit early MLP layer(s) based on results from Causal Tracing showing the largest causal effects on average in early layers.

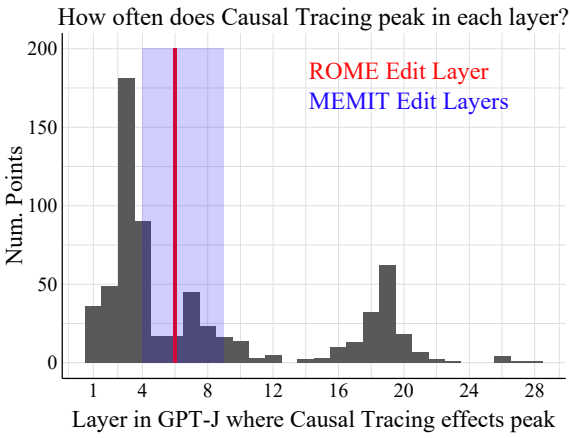


Figure 9.1: We visualize where 652 facts known by GPT-J are stored within the model, as localized by Causal Tracing. Model editing methods like ROME and MEMIT can successfully change knowledge in LMs by editing layers 4-9. But many facts appear to be stored outside of this range, e.g. at layers 1-3 and 16-20. What about these facts?

Surprisingly, the assumption that one should change the knowledge in a model by editing the weights where it is stored turns out to be false. In fact, localization results from Causal Tracing are statistically uncorrelated with the success of an edit injecting a new fact into MLP weights. Using the CounterFact dataset from Meng et al. [128] with a GPT-J model [209], we show that (1) not only is a substantial fraction of factual

knowledge stored outside of the range of layers edited by ROME/MEMIT (see Fig. 9.1), (2) the correlation between Causal Tracing results and edit success is near zero (for several editing methods including ROME, MEMIT, and Adam-based finetuning). We note that this is surprising largely because ROME and MEMIT *do* work well for editing facts, in spite of Causal Tracing often suggesting knowledge is stored elsewhere than early-to-mid-layer MLP weights.

In the face of this result, we attempt to recover the connection between tracing-based localization and editing by introducing four variants of the default model editing problem. Each variant differs in terms of the input, target, or objective used in the editing problem. One variant we introduce, called Fact Forcing, is designed to match Causal Tracing along these three factors. Specifically, Fact Forcing uses a noised input and involves maximizing the probability of the correct target output, just like Causal Tracing. We find that tracing results *are* related to edit success for Fact Forcing. However, even for this variant, it is still better to ignore the tracing results and always choose an early-to-mid-layer MLP weight for editing. We conclude that, although Causal Tracing is a reasonable localization method that has yielded insight into how models store factual information, this insight does not actually indicate which model layers we should edit in order to manipulate what facts are stored in language models.

To summarize, our conclusions are as follows:

1. We find that model edit success is essentially unrelated to where factual information is stored in models, as measured by Causal Tracing. Robustness experiments generalize this result across causal localization methods, editing methods, editing metrics, models, and datasets.
2. To reconnect localization with editing performance, we introduce four variants of a standard model editing problem, including Tracing Reversal, Fact Erasure, Fact Amplification, and Fact Forcing.
3. Edit success and tracing effects correlate best in the Fact Forcing setting. However, tracing effects explain only a small fraction of the variance in editing performance, while the choice of edit layer is a much more important factor. This suggests that, surprisingly, localization insights from Causal Tracing are not useful for choosing which model layer to edit.

9.2 Related Work

Localization. A long line of work aims to interpret what certain hidden representations represent, or, in the reverse direction, to understand how a given concept is represented in a model. Both of these efforts aim to localize behaviors to specific model components. We group these methods based on the kinds of model components they consider (e.g. layers, neurons, etc.).

Many works focus on individual layers or weight matrices [229, 170, 38, 181, 53]. In this chapter, we adopt the layer-wise localization method from Meng et al. [128] known as Causal Tracing, which estimates the information content of a set of representations via a denoising operation. We specifically focus on MLP layers given evidence of their role in factual association [57, 128, 58].

Related to analysis at the layer level, other work aims to localize concepts to directions in a latent space, dating back to work interpreting directions in word vector space [130, 98, 237, 59, 231, 29]. One might also place "key-value memory" theories of weight matrices in this category since a key vector represents a direction in the latent space [14, 178, 57, 128].

Neurons, meanwhile, are the most common focus of localization analysis. Past work explores the functions of groups of neurons and subnetworks [142, 34, 39, 23] or simply individual neurons [159, 236, 104, 15, 207, 137, 36, 105, 19, 78, 35, 213].

Relating Localization to Editing. Many works on localization validate the quality of their conclusions by editing neuron activations or layer weights corresponding to a particular concept, then checking that the network behavior changes appropriately. For example, Dai et al. [36] check that their "knowledge neurons" have localized a specific fact by amplifying or suppressing the expression of that fact via adjusting the corresponding neuron activations. Altogether, we find many localization analyses are validated by editing models in suggested locations [159, 104, 15, 137, 207, 36, 105, 35, 213, 23] or directions in the latent space [130, 14, 178, 128].

Changing model behavior by editing components suggested by localization seems like a reasonable validation step. However, in isolation, it paints an incomplete picture that has led to misleading interpretations about the connections between localization and editing. Such experiments alone do not show whether editing *that specific component* is (1) successful in proportion to the strength of the localization, (2) necessary to achieve the desired behavior, or (3) the best option for editing. In particular, these experiments do not show whether the same change in behavior can be achieved *elsewhere in the network*. Meng et al. [128] consider this question by measuring editing success across layers, averaged across data, then comparing the results with Causal Tracing conclusions also averaged across data. However, as we show, more fine-grained analysis at the datapoint level reveals the unexpected result that tracing results are unrelated to edit success. We are not aware of any work that primarily investigates the connection between localization and editing or that demonstrates better model editing at locations elsewhere in the network than those suggested by localization analysis.

9.3 Notation and Background

9.3.1 Data Notation

Following Meng et al. [128], we consider facts of the form (s, r, o), where s represents a subject entity (e.g. *Paris*), r a binary relation (e.g. *is located in*), and o an object (e.g. *France*) for which the tuple (s, r, o) represents a factual assertion about the world. In the CounterFact dataset [128], each datapoint is a prompt P for some fact (s, r, o). So, P might be "Paris is located in" or "Paris is situated in," to be completed by the object o to form a true statement. In an abuse of notation, we will often use s and r to refer to textual representations of a subject and relation, for instance by writing a model's conditional probability as $p_\theta(\cdot|s, r)$ instead of $p_\theta(\cdot|P)$. We do so in order to more easily indicate when an input is provided where the subject or relation has been manipulated (described next).

We make use of a few variations of the data for the fact (s, r, o). The additional variables include:

1. s^* is a "neighboring" entity to the subject s (similar to s) for which (s^*, r, o) is a true fact like (s, r, o). In CounterFact, "Marseille" is a neighboring entity to "Paris."
2. r^* is a paraphrase of the relation r, such as "is situated in" for "is located in."
3. s_{noise} is a noised representation of the subject s. We add Gaussian noise to the token embeddings of s, following Meng et al. [128].
4. o_{false} is an object that incorrectly completes the tuple $(s, r, \cdot)$. CounterFact contains an o_{false} for each datapoint, intended to be the new model output when evaluating model editing methods.

5. o_{true}, for clarity, is the object that correctly completes the fact $(s, r, \cdot)$, from CounterFact.

9.3.2 Causal Tracing

We give a brief description of Causal Tracing here and refer readers to Meng et al. [128] for more information (see Fig. 9.2 for an example visualization). Causal Tracing is a method for localizing information in the forward pass of an autoregressive Transformer to specific hidden representations. For a model with L layers, the input is a prompt containing T tokens (including a subject s and relation r). Given this input, the forward pass produces $T \times L$ layer outputs (one representation per T tokens and L layers). The algorithm aims to estimate the amount of information about the fact (s, r, o_{true}) that is contained in each of these representations. We denote the representation at token t and layer ℓ as $v_{(t,\ell)}$.

The amount of factual information in $v_{(t,\ell)}$ is estimated by copying this representation into a different forward pass obtained from using a noised subject in the input:

$$\text{Tracing Effect} = p_\theta(o_{true}|s_{noise}, r, v_{(t,\ell)}) - p_\theta(o_{true}|s_{noise}, r)$$

where s_{noise} indicates that we add Gaussian noise with $\sigma = 0.094$ to the token embeddings of s following Meng et al. [128], and $v_{(t,\ell)}$ is the representation at token t and layer ℓ in the forward pass on the original prompt $P = (s, r)$. The probability $p_\theta(o_{true}|s_{noise}, r, v_{(t,\ell)})$ is computed by (1) running the model forward pass on the noised prompt $P^* = (s_{noise}, r)$ until layer ℓ, (2) *overwriting* the existing representation at token t and layer ℓ with the representation $v_{(t,\ell)}$, then (3) computing the remaining $L - \ell$ layers as normal using this adjusted set of T representations as input (adjusted at token index t). Thus, Causal Tracing estimates the information content of a representation in terms of its effect on the probability of the true target. The results from Causal Tracing show where the representations containing information about the true target are in the model forward pass.

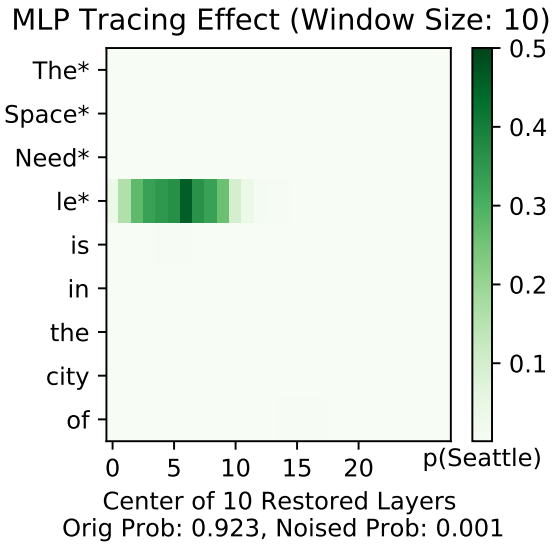


Figure 9.2: Visualizing Causal Tracing results over MLP layers with window size 10. Tokens with an asterisk are the noised subject tokens. Here, $p_\theta(o_{true}|s, r)$=.923 and $p_\theta(o_{true}|s_{noise}, r)$=.001.

In practice, a *set* of representations from multiple adjacent layers is copied from the clean forward pass rather than a single layer's representation (for instance, ten layers in Fig. 9.2). The size of this set is referred to as the *tracing window size*. A window size of, e.g., three implies that the tracing effect at layer ℓ estimates the amount of information contained in the three representations $v_{(t,\ell-1)}$, $v_{(t,\ell)}$, and $v_{(t,\ell+1)}$. See Appendix Figs. F.4 and F.5 for analysis of the parameter's effect. In this chapter, we use a tracing window size of 5 by default, and we apply Causal Tracing exclusively to MLP layers, given evidence of their role in factual association [57, 128].

9.3.3 Model Editing with ROME

We describe the ROME editing method here since we use it in our analysis in Sec. 9.4, and later in Sec. 9.5 we outline additional editing methods we consider. For mathematical detail, see Meng et al. [128].

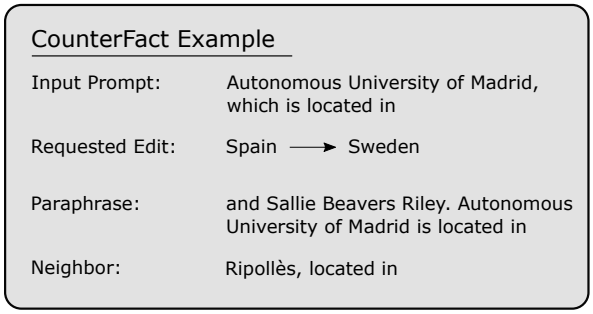


Figure 9.3: An example CounterFact datapoint.

The input to ROME includes a prompt $P = (s, r)$ and a new desired output, which is always a false target o_{false} in the CounterFact dataset. To change the model prediction to o_{false}, ROME applies a rank one edit to the down-projection matrix in a prespecified MLP layer in the model. The default layer in GPT-J is layer 6, following from averaged Causal Tracing results. ROME also makes use of covariance statistics of different subject representations obtained from a larger corpus as it edits individual facts. Overall, the method is designed to optimize the quantity $p_\theta(o_{false}|s, r)$ while aiming to satisfy some other constraints reflecting what a desirable model edit is (described in Sec. 9.3.4 next).

9.3.4 Editing Metrics

Editing methods are typically evaluated according to their ability to (1) change the model prediction on the input P provided at runtime, (2) generalize appropriately to paraphrases of the prompt P, and (3) avoid over-generalizing to unrelated data [240, 38, 132, 70, 133]. We adopt metrics for each desideratum that we compute with available CounterFact data. Instead of the exact "magnitude" metrics from Meng et al. [128], we use normalized versions of each metric that we design to scale from 0 to 1 depending on whether the edit was maximally (un)successful, for purposes of making scores more comparable across data points. We denote the new edited weights of the LM as θ^* and its pre-edit weights as θ. See Fig. 9.3 for an example of the kinds of data these metrics are computed on.

1. *Rewrite Score.* The rewrite score measures how much an edit improves the target probability $p(o_{false}|s, r)$ as a fraction of the maximum possible improvement:

$$\frac{p_{\theta^*}(o_{false}|s, r) - p_\theta(o_{false}|s, r)}{1 - p_\theta(o_{false}|s, r)}$$

2. *Paraphrase Score.* The paraphrase score measures the target probability using syntactical paraphrases as inputs, always preserving the exact subject wording:

$$\frac{p_{\theta^*}(o_{false}|s, r^*) - p_\theta(o_{false}|s, r^*)}{1 - p_\theta(o_{false}|s, r^*)}$$

   which is averaged over multiple available paraphrases per input P. The score measures whether edits properly generalize across semantically equivalent prompts.

3. *Neighborhood Score.* The neighborhood score measures whether edits change predictions for prompts with a similar subject s^*, the same relation r, and the same (true) objects.

We scale the difference in probabilities so that 1 means the probability did not change (good), and 0 means it changed to the maximum extent possible (bad):

$$1 - \frac{|p_{\theta^*}(o_{false}|s^*, r) - p_\theta(o_{false}|s^*, r)|}{.5 + |p_\theta(o_{false}|s^*, r) - .5|}$$

The score measures whether edits avoid *over*-generalizing from the prompt P to different subjects.

9.4 Does Edit Success Follow From Localization?

Ostensibly, localization results should inform editing methods because it should help to know where information is stored in a model if you are going to manipulate the model's expression of that information. More specifically, if you wanted to inject a false belief (s, r, o_{false}) into a model (as defined in the ROME editing problem), it seems helpful to know which weights store the true fact (s, r, o_{true}), so that you could replace some stored representation of o_{true} with that of o_{false}. This underlying assumption about editing models appears in much past work on localization, where editing is used to verify localization analysis (see Sec. 9.2). In this section, we investigate the validity of this assumption as it applies to autoregressive Transformers.

9.4.1 Experiment Design

The goal of our experiments is to determine, for a given datapoint, whether edit success *at a specific layer* aligns with the results from Causal Tracing at that layer (see Causal Tracing description in Sec. 9.3.2). We operationalize this outcome and explanatory variable as follows:

1. *Edit Success.* We primarily consider Rewrite Score as our measure of edit success, given that this is the main optimization objective of ROME. Note ROME achieves an average rewrite score of 99% at layer 6 of GPT-J and above 96% at layers besides the last layer of the model.

2. *Tracing Effect at layer ℓ.* Since the output of Causal Tracing is a $T \times L$ grid of estimates, we obtain a single tracing effect per layer by taking the max across the T token effects at each layer (i.e., we collapse the grid in Fig. 9.2 down to a single curve across layers). Like our other metrics, we use a *fractional* tracing effect where 0 means the intervention had no effect and 1 means it fully restored the original probability $p_\theta(o_{true}|s, r)$:

$$\frac{p_\theta(o_{true}|s_{noise}, r, v_{(t,\ell)}) - p_\theta(o_{true}|s_{noise}, r)}{p_\theta(o_{true}|s, r) - p_\theta(o_{true}|s_{noise}, r)}$$

   Lastly, note we use a tracing window size of 5 (smaller than the value of 10 used in Fig. 9.2).

9.4.2 Model and Data

We conduct our analysis with GPT-J [209] using the CounterFact dataset, similar to Meng et al. [128]. GPT-J is a 6 billion parameter autoregressive language model. We record editing performance at layers in {1, 5, 9, 13, 17, 21, 25, 28} as well as layer 6 (the default for ROME). Note ROME achieves an average rewrite score of 99% at layer 6 and above 96% at layers besides layer 28.

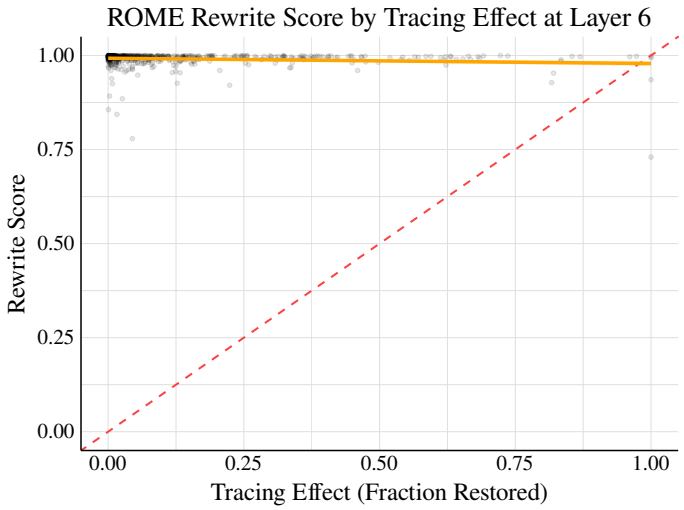


Figure 9.4: The correlation between ROME edit success and the tracing effect at layer 6 in GPT-J is not positive but in fact slightly negative ($\rho = -0.13$; p <1e−3). The dashed red line shows a hypothetical perfect relationship.

The CounterFact dataset includes datapoints consisting of a prompt, paraphrases, and neighboring points. For each point, a new (false) target is supplied for editing purposes. We show an example datapoint in Fig. 9.3. Note paraphrases intentionally include unrelated text preceding a syntactical paraphrase of the input, with the idea that this text should not affect the output. We select data for experiments from 10% of CounterFact, additionally filtering to a subset of facts that are correctly completed by GPT-J, in order to ensure that there is knowledge to localize in the model for each point (details in Appendix F.1). Our final sample size is $n = 652$.

9.4.3 Experiment Results

We present results in two ways. First, in Fig. 9.4, we show Rewrite Score as a function of the (fractional) tracing effect. The red dotted line shows a hypothetical perfect relationship between tracing and edit success. Surprisingly, there is not a positive relationship but a *negative* relationship between the rewrite score and the tracing effect (linear correlation of $\rho = -0.13$; p <1e−3). This seems to fully invalidate the assumption that editing should be most effective when it occurs at a layer where information is stored about the edited fact. We wish to emphasize, however, that in most layers we simply see a near-zero rather than negative correlation, as shown in Appendix Fig. F.9.

Our second mode of analysis is though linear regression models predicting rewrite score based on (1) the tracing effect, (2) the choice of edit layer treated as a categorical variable, or (3) both terms interacted, again treating edit layer as a categorical variable. The purpose of the models is to show how much of the variance in rewrite score is explained by one variable versus the other. We show the resulting R^2 values in Table 9.1. We see that the choice of layer explains almost

Table 9.1: R^2 values for predicting ROME edit success. Tracing effects explain essentially none of the variance in rewrite score, while the choice of edit layer is very important.

	R^2 Values		
Method	Layer	Tracing Effect	Both
ROME	0.947	0.016	0.948

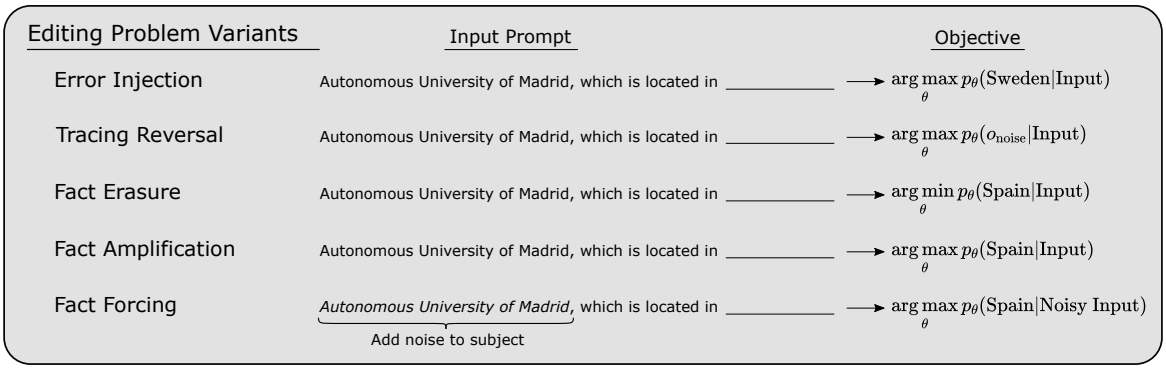


Figure 9.5: Depiction of editing problem variants. Rather than inject a new false fact into a model (Error Injection), we consider injecting the output obtained from noising the subject entity (Tracing Reversal), erasing a stored fact (Fact Erasure), amplifying a stored fact (Fact Amplification), or forcing a known fact onto the same kind of noisy input as used in Causal Tracing (Fact Forcing).

all of the variance in rewrite score (94.7%), while adding the tracing effect to the model raises the R^2 only to 94.8%. This means that **the tracing effect is able to explain only 0.1% of the variance in edit success** when accounting for the choice of edit layer. These results suggest that the tracing effect is essentially unrelated to the success of model editing.

This is a surprising conclusion, and it naturally raises the question of why applying ROME at layer 6 works well in the first place (see average rewrite, paraphrase, and neighborhood scores across layers in Appendix Fig. F.1). We suggest a possible answer to this question in Sec. 9.6.

Additional Robustness Experiments. We include additional results in Appendix F.2 using another dataset, ZSRE [111] (Figs. F.13 and F.14, Table F.7), and another localization method, representation zeroing [15] (Figs. F.15 and F.16). Further robustness experiments in Appendix F.3 include results with (1) other measures of edit success including Paraphrase Score, Neighborhood Score, and an Overall Score (Tables F.3, F.4 and F.5), (2) different values of the tracing window size (Fig. F.6), (3) GPT2-XL rather than GPT-J (Fig. F.7), (4) the original unscaled metrics from Meng et al. [128] (Fig. F.8), and (5) tracing effects measured at the last subject token rather than the max across tokens (Fig. F.10). We find that **all of these experiments corroborate our results comparing Causal Tracing to Rewrite Score for GPT-J on CounterFact.** Considering these robustness results alongside additional editing method experiments that we consider in Sec. 9.5 below, we note that our main conclusions generalize across different causal localization methods, editing methods, editing metrics, models, and datasets.

9.5 Reconciling Localization and Editing

If injecting a new fact has little to do with where an existing fact is stored in the model, perhaps there is some other editing intervention that would be more closely related to insights from tracing analysis. In this section, we propose a few variants of the model editing problem that appear more and more like Causal Tracing in terms of their input, target, and objective. Then, we repeat and extend our analysis from Sec. 9.4 for all of these editing problems.

9.5.1 Editing Problem Variants

We summarize the following editing problems in Fig. 9.5.

1. *Error Injection.* The editing problem considered in Sec. 9.4, the objective being to maximize $p_\theta(o_{false}|s,r)$.
2. *Tracing Reversal.* We maximize $p_\theta(o_{noise}|s,r)$, aiming to change the model output from o_{true} back to the output for the "original" noised input $P = (s_{noise}, r)$ in Causal Tracing, o_{noise}.
3. *Fact Erasure.* Knowing where a fact is stored could be more useful for erasing the fact rather than injecting a new one. Hence, we consider erasing a fact by minimizing $p_\theta(o_{true}|s,r)$.
4. *Fact Amplification.* We reinforce known facts in the model by maximizing $p_\theta(o_{true}|s,r)$. Even for correctly predicted points, this value is often not near 1, leaving room for it to be increased.
5. *Fact Forcing.* As in Causal Tracing, this method uses a noised subject representation s_{noise}. We force the model to output o_{true} for this input by maximizing $p_\theta(o_{true}|s_{noise}, r)$. Though this problem is of little practical significance, it is the most similar to Causal Tracing in its design, since it uses the same input as Causal Tracing and matches the goal of increasing the probability of o_{true} (see Sec. 9.3.2).

Note that solutions to each of these problems are evaluated according to our Rewrite Score, Paraphrase Score, and Neighborhood Score metrics from Sec. 9.3.4. The only difference is in the target output for the rewrite and paraphrase metrics (neighborhood is entirely identical).

9.5.2 Experiment Design and Additional Edit Methods

We use the same experimental procedure as in Sec. 9.4, except that we consider a broader set of editing methods besides ROME. We list the four methods below:

1. ROME. The edit method from Sec. 9.4, ROME edits a single MLP layer's down-projection weight.
2. MEMIT. Though designed to edit multiple facts at once, when editing a single fact this method differs from ROME only by spreading out its update over several layers rather than one layer [129].
3. Constrained Finetuning (window size 1). We adopt a simple Adam-based optimization approach with an ℓ_∞-norm constraint, following Zhu et al. [240]. The window size of 1 indicates we apply this method at a single layer.
4. Constrained Finetuning (window size 5). The above finetuning method on five adjacent layers.

We select these methods for their simplicity and since ROME and MEMIT are designed specifically to edit MLP layers. Note that we report results for Causal Tracing with a window size of five, so when we use MEMIT or constrained finetuning to edit five layers, these five layers can exactly match the range of restored layers from Causal Tracing.

9.5.3 Experiment Results

Main Results. As in our analysis in Sec. 9.4, we report R^2 values for a linear regression model predicting the rewrite score based on (1) the choice of edit layer treated as a categorical

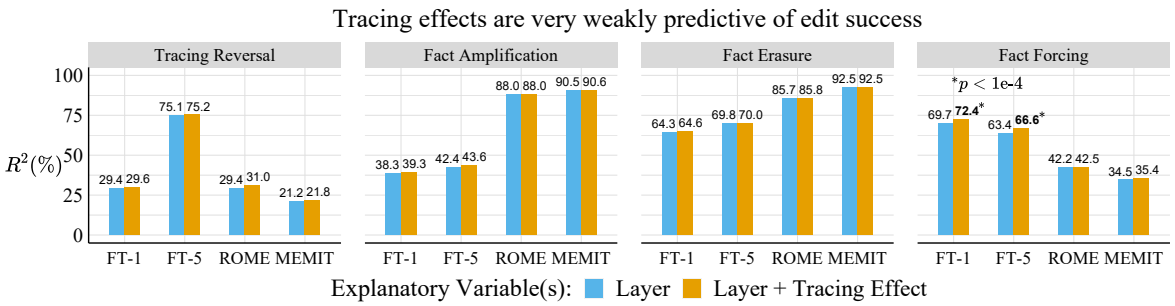


Figure 9.6: Tracing effects are very weakly predictive of edit success across editing problems and methods. Relative to the R^2 of a regression predicting rewrite score based on the edit layer (blue), a regression with edit layer and tracing effects (orange) improves the R^2 by at most .03 points (bolded). The choice of edit layer is a much better predictor of the rewrite score.

variable, or (2) that variable interacted with the tracing effect. We show the results in Fig. 9.6, with R^2 values for each regression above their respective bars (numbers also in Appendix Table F.2). We find that, relative to the Layer-only regression, **tracing effects explain at most an additional 3.2% of the variance in edit success** across our different editing problems and editing methods. This is a very small effect, especially compared to R^2 values from the Layer-only regression, which explains most of the variance in the outcome (58.5% on average across conditions in Fig. 9.6). We believe this is surprising given how the editing problem variants are designed. **It would seem that knowing where a fact is stored should help with amplifying or erasing that fact, but our results appear to fully disconfirm this hypothesis.** Interestingly, it also appears that it makes little difference whether we edit at one layer or five layers in order to match the number of representations restored by Causal Tracing. Based on comparisons between finetuning methods (FT-1 and FT-5) and between ROME and MEMIT (applied to 5 layers), editing at five layers does not improve the alignment between tracing and editing. In addition to our robustness results listed in Sec. 9.4.3, we also repeat our analysis using a subset of points where tracing effects are concentrated to a small number of layers, in order to focus on points where MEMIT and FT-5 edit *all* of the layers where the fact is stored. Results are nearly identical for this subset of the data (see Appendix F.2).

One Successful Case. We see the strongest positive relationship between edit success and tracing effects for Fact Forcing with finetuning methods. Here, we find that tracing effects explain an additional 3% of the variance in edit success (up from 1.5% for other experiments). This effect is statistically significant at $p < 1e{-}4$ according to an F-test[1] comparing the two models (see visualization in Appendix Fig. F.11). The result for Fact Forcing suggests that using s_{noise} rather than s in the model input is the cause of the positive relationship between editing and localization. We rule out the choice of target and maximizing vs. minimizing the target probability as possible causes based on the design of each problem variant (see Fig. 9.5): (1) the choice of target is not important since results are similar for Error Injection, Tracing Reversal, and Fact Amplification, and (2) maximizing vs. minimizing the target

[1]This tests if one model explains more of the variance than another model which has only a subset of the first's covariates (here, tracing effect and edit layer vs. only edit layer).

probability is not important since results are similar for Fact Erasure and Fact Amplification. Yet, tracing effects are still weakly informative of Fact Forcing editing if they explain only 3% of the variance in edit success. This points to there being other deeper reasons for localization results being unrelated to editing success.

9.6 Discussion

Does Causal Tracing tell us anything? We show that Causal Tracing is not indicative of which layer to select for model editing. However, this does not mean that localization insights from Causal Tracing have been useless. Causal Tracing has helped reveal the role that early-to-mid-range MLP representations *at the last subject token index* play in factual association in autoregressive language models, and ROME does perform better on average when optimizing the last subject token representation rather than another token representation [128].[2] Past work finds that both MLP and attention layers can show large Causal Tracing effects, and additional empirical editing experiments then demonstrate that it is preferable to edit MLP weights [128].

Why is edit success high at layers where the edited fact is not actually stored? First, we note that information is gradually accumulated across layers in a Transformer forward pass, as discovered by past work [170, 57, 128, 129, 58]. We suggest that it is possible to "override" the information in layer ℓ with an edit to another layer k (where $k < \ell$ or $k > \ell$). Since ROME is typically effective across a large range of layers (see Fig. F.3), it appears that ROME can override the information accrued across 5 or 10 layers of a forward pass with an edit to a single layer outside of that range of layers. We summarize this hypothesis as follows: *Many layers could store a fact, and it happens that some do.*

If this hypothesis were true, it would be surprising because one cannot arbitrarily swap layers in a Transformer model without greatly damaging model performance [233]. That is, it should matter where information enters the residual stream, since later layers strongly depend on receiving the right incoming information from prior layers. We leave it to future work to further investigate this hypothesis.

What do our results imply about using model editing to validate localization claims? We interpret our results to suggest that Causal Tracing *answers a different question* than model editing does. That is, Causal Tracing answers a question about where factual information is carried in representations in a Transformer forward pass, and this question turns out to be a different question than the *editing* question of where is best to intervene in the Transformer in order to change the factual information it expresses. It seems critical, then, to carefully formalize the questions that one wishes to answer before (1) validating the results of localization via editing or (2) motivating the design of an editing method via localization, because the conclusions that can be drawn from a particular localization method might not be relevant for the performance of a given model editing method. This would not imply the conclusions from the localization analysis are invalid, though. For instance, we believe Causal Tracing reveals interesting insights about where MLP representations contain factual information (see Figs. 9.1 and 9.2). We only wish to suggest that localization analysis might answer a different question than the question answered by model editing.

These observations may have implications for the array of studies that validate their localization analysis by manipulating a certain model behavior via an intervention on the model component recommended by the analysis [159, 104, 15, 14, 137, 207, 36, 105, 35, 213, 23, 128].

[2] Although, datapoint-level regression would provide stronger evidence that tracing effects predict which token representation is best to optimize with ROME (and rule out other confounders such as the edit layer).

Do model editing experiments provide *additional* evidence for claims about which model components are responsible for certain behaviors? If localization and editing answer different questions, editing experiments will not provide further evidence for localization conclusions.

9.7 Conclusion

We obtain the surprising result that model edit success is essentially unrelated to where factual information is stored in models, as measured by Causal Tracing. Faced with this result, we attempt to reconnect tracing-based localization with edit success by introducing four variants of the Error Injection problem using the CounterFact dataset. We find that edit success and tracing effects correlate best in our Fact Forcing setting. However, even in this case, tracing effects explain only a small fraction of the variance in editing performance, while the choice of edit layer is a much more important factor. This suggests that, counterintuitively, better mechanistic understanding of how pretrained language models work may not always translate to insights about how to best change their behavior.

10 Conclusion

This concludes the foregoing work on interpretable and controllable models. To summarize:

First, I discussed *Human Evaluation of ML Explanations* and work on evaluation protocols for one-size-fits-all tests for model explanation faithfulness.

Second, I covered *Natural Language Explanation Methods* and describe faithfulness tests specifically for model-based evaluation of natural language explanations.

Third, I surveyed approaches for *Adding Explanation Data to Traditional Discriminative Learning*. Here, I argue that explanation data is best utilized as model inputs rather than as model targets or a prior over model weights in the context of discriminative learning.

Fourth, I introduced new approaches for *Feature Attribution Methods and Evaluation.* I suggest that, rather than using a linear attribution method on an arbitrary blackbox model, one should preferably explain only models that sometimes see ablated inputs during training (i.e. partly missing features), while obtaining explanations with a compute-adjustable search method to search specifically for features that would be sufficient or necessary for a model's test-time prediction.

Fifth, I explored *Model Editing and Belief Graphs for LMs.* Since this work, model editing has become an increasingly popular problem area, and tools for manipulating and visualizing model beliefs will be especially important as language models develop (somewhat) coherent world models.

Lastly, I described work on *Localization and Editing of Knowledge in LMs.* This conceptual work highlights important subtleties regarding (1) localization findings' utility for model editing, and (2) the use of model editing to substantiate the validity of a localization claim.

In total, this thesis includes new evaluation procedures, explainability methods, approaches to model editing and model control, and theoretical contributions to interpretability.

11 Published Work

Below is a list of published papers and preprints which I have authored or co-authored during my graduate research:

2024

[73] Peter Hase, Mohit Bansal, Peter Clark, and Sarah Wiegreffe. The unreasonable effectiveness of easy training data for hard tasks. *arXiv preprint arXiv:2401.06751*, 2024. URL https://arxiv.org/pdf/2401.06751.pdf

[4] Usman Anwar, Abulhair Saparov, Javier Rando, Daniel Paleka, Miles Turpin, Peter Hase, Ekdeep Singh Lubana, Erik Jenner, Stephen Casper, Oliver Sourbut, et al. Foundational challenges in assuring alignment and safety of large language models. *arXiv preprint arXiv:2404.09932*, 2024. URL https://arxiv.org/pdf/2404.09932.pdf

[122] Sijia Liu, Yuanshun Yao, Jinghan Jia, Stephen Casper, Nathalie Baracaldo, Peter Hase, Xiaojun Xu, Yuguang Yao, Hang Li, Kush R Varshney, et al. Rethinking machine unlearning for large language models. *arXiv preprint arXiv:2402.08787*, 2024. URL https://arxiv.org/pdf/2402.08787.pdf

2023

[148] Vaidehi Patil, Peter Hase, and Mohit Bansal. Can sensitive information be deleted from llms? objectives for defending against extraction attacks. *arXiv preprint arXiv:2309.17410*, 2023. URL https://arxiv.org/pdf/2309.17410.pdf

[72] Peter Hase, Mohit Bansal, Been Kim, and Asma Ghandeharioun. Does localization inform editing? surprising differences in causality-based localization vs. knowledge editing in language models. *arXiv preprint arXiv:2301.04213*, 2023. URL https://arxiv.org/pdf/2301.04213.pdf

[24] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023. URL https://arxiv.org/pdf/2307.15217.pdf

[176] Swarnadeep Saha, Peter Hase, and Mohit Bansal. Can language models teach weaker agents? teacher explanations improve students via theory of mind. *arXiv preprint arXiv:2306.09299*, 2023. URL https://arxiv.org/pdf/2306.09299.pdf

[227] Zhuofan Ying, Peter Hase, and Mohit Bansal. Adaptive contextual perception: How to generalize to new backgrounds and ambiguous objects. *arXiv preprint arXiv:2306.05963*, 2023. URL https://arxiv.org/pdf/2306.05963.pdf

2022

[174] Swarnadeep Saha, Peter Hase, Nazneen Rajani, and Mohit Bansal. Are hard examples also harder to explain? a study with human and model-generated explanations. *arXiv preprint arXiv:2211.07517*, 2022. URL https://arxiv.org/pdf/2211.07517.pdf

[226] Zhuofan Ying, Peter Hase, and Mohit Bansal. Visfis: Visual feature importance supervision with right-for-the-right-reason objectives. *Advances in Neural Information Processing*

Systems, 35:17057–17072, 2022. URL https://arxiv.org/pdf/2206.11212.pdf

[175] Swarnadeep Saha, Shiyue Zhang, Peter Hase, and Mohit Bansal. Summarization programs: Interpretable abstractive summarization with neural modular trees. *arXiv preprint arXiv:2209.10492*, 2022. URL https://arxiv.org/pdf/2209.10492.pdf

[155] Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. Grips: Gradient-free, edit-based instruction search for prompting large language models. *arXiv preprint arXiv:2203.07281*, 2022. URL https://arxiv.org/pdf/2203.07281.pdf

2021

[71] Peter Hase, Harry Xie, and Mohit Bansal. The out-of-distribution problem in explainability and search methods for feature importance explanations. In *Advances in Neural Information Processing Systems*, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/1def1713ebf17722cbe300cfc1c88558-Paper.pdf

[70] Peter Hase, Mona Diab, Asli Celikyilmaz, Xian Li, Zornitsa Kozareva, Veselin Stoyanov, Mohit Bansal, and Srinivasan Iyer. Do language models have beliefs? methods for detecting, updating, and visualizing model beliefs. *arXiv preprint arXiv:2111.13654*, 2021. URL https://arxiv.org/pdf/2111.13654.pdf

[222] Prateek Yadav, Peter Hase, and Mohit Bansal. INSPIRE: Incorporating diverse feature preferences in recourse. *arxiv*, 2021. URL https://openreview.net/pdf?id=6yzIuqKGnq

[66] Peter Hase and Mohit Bansal. When can models learn from explanations? a formal framework for understanding the roles of explanation data. *arXiv preprint arXiv:2102.02201*, 2021. URL https://arxiv.org/pdf/2102.02201.pdf

2020

[62] Han Guo, Nazneen Fatema Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. Fastif: Scalable influence functions for efficient model interpretation and debugging. *arXiv preprint arXiv:2012.15781*, 2020. URL https://arxiv.org/pdf/2012.15781.pdf

[68] Peter Hase, Shiyue Zhang, Harry Xie, and Mohit Bansal. Leakage-adjusted simulatability: Can models generate non-trivial explanations of their behavior in natural language? In *Findings of EMNLP*, 2020. URL https://arxiv.org/abs/2010.04119

[64] Peter Hase and Mohit Bansal. Evaluating explainable ai: Which algorithmic explanations help users predict model behavior?, 2020. URL https://arxiv.org/pdf/2005.01831.pdf

References

[1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018. URL https://arxiv.org/abs/1810.03292.

[2] David Alvarez-Melis, Hal Daumé III, Jennifer Wortman Vaughan, and Hanna Wallach. Weight of evidence as a basis for human-oriented explanations. *arXiv preprint arXiv:1910.13503*, 2019. URL https://arxiv.org/pdf/1910.13503.pdf.

[3] Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *NAACL-HLT 2018*, 2018. doi: 10.18653/v1/n18-1197. URL https://doi.org/10.18653/v1/n18-1197.

[4] Usman Anwar, Abulhair Saparov, Javier Rando, Daniel Paleka, Miles Turpin, Peter Hase, Ekdeep Singh Lubana, Erik Jenner, Stephen Casper, Oliver Sourbut, et al. Foundational challenges in assuring alignment and safety of large language models. *arXiv preprint arXiv:2404.09932*, 2024. URL https://arxiv.org/pdf/2404.09932.pdf.

[5] Leila Arras, Ahmed Osman, Klaus-Robert Müller, and Wojciech Samek. Evaluating recurrent neural network explanations. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 113–126, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4813. URL https://www.aclweb.org/anthology/W19-4813.

[6] Abhijeet Awasthi, Sabyasachi Ghosh, Rasna Goyal, and Sunita Sarawagi. Learning from rules generalizing labeled exemplars. In *ICLR 2020*, 2020. URL https://arxiv.org/pdf/2004.06025.pdf.

[7] Lei Jimmy Ba, Kevin Swersky, Sanja Fidler, and Ruslan Salakhutdinov. Predicting deep zero-shot convolutional neural networks using textual descriptions. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 4247–4255. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.483. URL https://doi.org/10.1109/ICCV.2015.483.

[8] Livio Baldini Soares, Nicholas FitzGerald, Jeffrey Ling, and Tom Kwiatkowski. Matching the blanks: Distributional similarity for relation learning. In *ACL*, pages 2895–2905, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1279. URL https://www.aclweb.org/anthology/P19-1279.

[9] Seojin Bang, Pengtao Xie, Heewook Lee, Wei Wu, and Eric Xing. Explaining a black-box using Deep Variational Information Bottleneck Approach. *arXiv:1902.06918 [cs, stat]*, abs/1902.06918, February 2019. URL http://arxiv.org/abs/1902.06918. arXiv: 1902.06918.

[10] Yujia Bao, Shiyu Chang, Mo Yu, and Regina Barzilay. Deriving machine attention from human rationales. In *EMNLP*, pages 1903–1913, Brussels, Belgium, October-November

2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1216. URL https://www.aclweb.org/anthology/D18-1216.

[11] Ricardo Baptista and Matthias Poloczek. Bayesian optimization of combinatorial structures. In *International Conference on Machine Learning*, pages 462–471. PMLR, 2018. URL http://proceedings.mlr.press/v80/baptista18a/baptista18a.pdf.

[12] Jasmijn Bastings, Wilker Aziz, and Ivan Titov. Interpretable neural predictions with differentiable binary variables. In *ACL 2019*, pages 2963–2977, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1284. URL https://www.aclweb.org/anthology/P19-1284.

[13] Jasmijn Bastings, Wilker Aziz, and Ivan Titov. Interpretable neural predictions with differentiable binary variables. In *ACL 2019*, pages 2963–2977, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1284. URL https://www.aclweb.org/anthology/P19-1284.

[14] David Bau, Steven Liu, Tongzhou Wang, Jun-Yan Zhu, and Antonio Torralba. Rewriting a deep generative model. In *European conference on computer vision*, pages 351–369. Springer, 2020. URL https://arxiv.org/pdf/2007.15646.pdf.

[15] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 117(48):30071–30078, 2020. URL https://www.pnas.org/doi/pdf/10.1073/pnas.1907375117.

[16] David Belanger, Bishan Yang, and Andrew McCallum. End-to-end learning for structured prediction energy networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 429–439. PMLR, 2017. URL http://proceedings.mlr.press/v70/belanger17a.html.

[17] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020. URL https://arxiv.org/abs/2004.05150.

[18] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623, 2021. URL https://dl.acm.org/doi/10.1145/3442188.3445922.

[19] Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Viégas, and Martin Wattenberg. An interpretability illusion for bert. *arXiv preprint arXiv:2104.07143*, 2021. URL https://arxiv.org/pdf/2104.07143.pdf.

[20] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *EMNLP 2015*, 2015. URL https://arxiv.org/abs/1508.05326.

[21] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya

Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS*, 2020. URL https://arxiv.org/abs/2005.14165.

[22] Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. e-snli: Natural language inference with natural language explanations. In *NeurIPS 2018*, 2018. URL https://arxiv.org/pdf/1812.01193.pdf.

[23] Stephen Casper, Shlomi Hod, Daniel Filan, Cody Wild, Andrew Critch, and Stuart Russell. Graphical clusterability and local specialization in deep neural networks. In *ICLR 2022 Workshop on PAIR*, 2022. URL https://arxiv.org/pdf/2110.08058v2.pdf.

[24] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023. URL https://arxiv.org/pdf/2307.15217.pdf.

[25] Arjun Chandrasekaran, Viraj Prabhu, Deshraj Yadav, Prithvijit Chattopadhyay, and Devi Parikh. Do explanations make vqa models more predictable to a human? In *EMNLP*, pages 1036–1042, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1128. URL https://arxiv.org/pdf/1810.12366.pdf.

[26] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation. In *ICLR*, 2019. URL https://arxiv.org/pdf/1807.08024.pdf.

[27] Chaofan Chen, Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. This Looks Like That: Deep Learning for Interpretable Image Recognition. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, June 2019. URL http://arxiv.org/abs/1806.10574.

[28] Hanjie Chen and Yangfeng Ji. Learning variational word masks to improve the interpretability of neural text classifiers. In *EMNLP*, pages 4236–4251, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.347. URL https://www.aclweb.org/anthology/2020.emnlp-main.347.

[29] Pattarawat Chormai, Jan Herrmann, Klaus-Robert Müller, and Grégoire Montavon. Disentangled explanations of neural network predictions by finding relevant subspaces. *arXiv preprint arXiv:2212.14855*, 2022. URL https://arxiv.org/pdf/2212.14855.pdf.

[30] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015. URL https://arxiv.org/pdf/1502.02127.pdf.

[31] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. 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 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://www.aclweb.org/anthology/N19-1300.

[32] John D. Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, Jacob Andreas, John DeNero, Pieter Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. In *ICLR 2019*, 2019. URL https://openreview.net/forum?id=HkgSEnA5KQ.

[33] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, 1988. ISBN 9780805802832.

[34] Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Are neural nets modular? inspecting functional modularity through differentiable weight masks. *arXiv preprint arXiv:2010.02066*, 2020. URL https://arxiv.org/pdf/2010.02066.pdf.

[35] Audrey Cui, Ali Jahanian, Agata Lapedriza, Antonio Torralba, Shahin Mahdizadehaghdam, Rohit Kumar, and David Bau. Local relighting of real scenes. *arXiv preprint arXiv:2207.02774*, 2022. URL https://arxiv.org/pdf/2207.02774.pdf.

[36] Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, and Furu Wei. Knowledge neurons in pretrained transformers. In *ACL*, 2022. URL https://arxiv.org/pdf/2104.08696.pdf.

[37] Nicola De Cao, Michael Sejr Schlichtkrull, Wilker Aziz, and Ivan Titov. How do decisions emerge across layers in neural models? interpretation with differentiable masking. In *EMNLP*, pages 3243–3255, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.262. URL https://www.aclweb.org/anthology/2020.emnlp-main.262.

[38] Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. In *EMNLP*, pages 6491–6506. Association for Computational Linguistics, November 2021. URL https://aclanthology.org/2021.emnlp-main.522.

[39] Nicola De Cao, Leon Schmid, Dieuwke Hupkes, and Ivan Titov. Sparse interventions in language models with differentiable masking. In *EMNLP BlackboxNLP Workshop*, 2021. URL https://arxiv.org/pdf/2112.06837.pdf.

[40] Daniel Dennett. Do animals have beliefs? *Comparative approaches to cognitive science*, 111, 1995. URL https://dl.tufts.edu/concern/pdfs/rj430g708.

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *ACL 2019*, 2019. URL https://arxiv.org/pdf/1810.04805.pdf.

[42] Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C. Wallace. Eraser: A benchmark to evaluate rationalized nlp models. In *ACL 2020*, volume abs/1911.03429, 2020. URL https://arxiv.org/pdf/1911.03429.pdf.

[43] Bhuwan Dhingra, Jeremy R Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W Cohen. Time-aware language models as temporal knowledge bases. *arXiv preprint arXiv:2106.15110*, 2021. URL https://arxiv.org/pdf/2106.15110.pdf.

[44] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping. *arXiv:2002.06305 [cs]*, February 2020. URL http://arxiv.org/abs/2002.06305. arXiv: 2002.06305.

[45] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah A. Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *CoRR*, abs/2002.06305, 2020. URL https://arxiv.org/abs/2002.06305.

[46] Finale Doshi-Velez and Been Kim. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv:1702.08608 [cs, stat]*, February 2017. URL http://arxiv.org/abs/1702.08608. arXiv: 1702.08608.

[47] Qingfeng Du and Jincheng Xu. Model-agnostic local explanations with genetic algorithms for text classification. In *The 33rd International Conference on Software Engineering & Knowledge Engineering*, 2021. URL https://ksiresearch.org/seke/seke21paper/paper040.pdf.

[48] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[49] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-box adversarial examples for text classification. In *ACL*, pages 31–36, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2006. URL https://www.aclweb.org/anthology/P18-2006.

[50] Bradley Efron and Robert J Tibshirani. *An Introduction to the Bootstrap*. CRC press, 1994.

[51] Upol Ehsan, Samir Passi, Q Vera Liao, Larry Chan, I Lee, Michael Muller, Mark O Riedl, et al. The who in explainable ai: How ai background shapes perceptions of ai explanations. *arXiv preprint arXiv:2107.13509*, 2021. URL https://arxiv.org/pdf/2107.13509.pdf.

[52] Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. Measuring and improving consistency in pretrained language models. *Transactions of the Association for Computational Linguistics*, 9: 1012–1031, 2021. URL https://arxiv.org/pdf/2102.01017.pdf.

[53] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

[54] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3429–3437, 2017. URL https://arxiv.org/pdf/1704.03296.pdf.

[55] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. In *Findings of EMNLP*, 2020. URL https://arxiv.org/pdf/2009.11462.pdf.

[56] Atticus Geiger, Chris Potts, and Thomas Icard. Causal abstraction for faithful model interpretation. *arXiv preprint arXiv:2301.04709*, 2023.

[57] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *EMNLP*, 2021. URL https://arxiv.org/pdf/2012.14913.pdf.

[58] Mor Geva, Avi Caciularu, Kevin Ro Wang, and Yoav Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *arXiv preprint arXiv:2203.14680*, 2022. URL https://arxiv.org/pdf/2203.14680.pdf.

[59] Amirata Ghorbani, James Wexler, James Y Zou, and Been Kim. Towards automatic concept-based explanations. *Advances in Neural Information Processing Systems*, 32, 2019. URL https://arxiv.org/pdf/1902.03129.pdf.

[60] Anastasia Giannakidou and Alda Mari. A linguistic framework for knowledge, belief, and veridicality judgement. *HAL*, 2020. URL https://halshs.archives-ouvertes.fr/halshs-03088697/document.

[61] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining Explanations: An Overview of Interpretability of Machine Learning. *The 5th IEEE International Conference on Data Science and Advanced Analytics (DSAA 2018)*, May 2018. URL http://arxiv.org/abs/1806.00069.

[62] Han Guo, Nazneen Fatema Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. Fastif: Scalable influence functions for efficient model interpretation and debugging. *arXiv preprint arXiv:2012.15781*, 2020. URL https://arxiv.org/pdf/2012.15781.pdf.

[63] Braden Hancock, Paroma Varma, Stephanie Wang, Martin Bringmann, Percy Liang, and Christopher Ré. Training classifiers with natural language explanations. In *ACL*, 2018. URL https://pubmed.ncbi.nlm.nih.gov/31130772/.

[64] Peter Hase and Mohit Bansal. Evaluating explainable ai: Which algorithmic explanations help users predict model behavior?, 2020. URL https://arxiv.org/pdf/2005.01831.pdf.

[65] Peter Hase and Mohit Bansal. Evaluating explainable ai: Which algorithmic explanations help users predict model behavior? In *ACL 2020*, 2020. URL https://arxiv.org/pdf/2005.01831.pdf.

[66] Peter Hase and Mohit Bansal. When can models learn from explanations? a formal framework for understanding the roles of explanation data. *arXiv preprint arXiv:2102.02201*, 2021. URL https://arxiv.org/pdf/2102.02201.pdf.

[67] Peter Hase, Chaofan Chen, Oscar Li, and Cynthia Rudin. Interpretable Image Recognition with Hierarchical Prototypes. In *Proceedings of the Seventh AAAI Conference on Human Computation and Crowdsourcing (HCOMP-19)*, pages 32–40, June 2019. URL http://arxiv.org/abs/1906.10651.

[68] Peter Hase, Shiyue Zhang, Harry Xie, and Mohit Bansal. Leakage-adjusted simulatability: Can models generate non-trivial explanations of their behavior in natural language? In *Findings of EMNLP*, 2020. URL https://arxiv.org/abs/2010.04119.

[69] Peter Hase, Shiyue Zhang, Harry Xie, and Mohit Bansal. Leakage-adjusted simulatability: Can models generate non-trivial explanations of their behavior in natural language?, 2020. URL https://arxiv.org/pdf/2010.04119.pdf.

[70] Peter Hase, Mona Diab, Asli Celikyilmaz, Xian Li, Zornitsa Kozareva, Veselin Stoyanov, Mohit Bansal, and Srinivasan Iyer. Do language models have beliefs? methods for detecting, updating, and visualizing model beliefs. *arXiv preprint arXiv:2111.13654*, 2021. URL https://arxiv.org/pdf/2111.13654.pdf.

[71] Peter Hase, Harry Xie, and Mohit Bansal. The out-of-distribution problem in explainability and search methods for feature importance explanations. In *Advances in Neural Information Processing Systems*, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/1def1713ebf17722cbe300cfc1c88558-Paper.pdf.

[72] Peter Hase, Mohit Bansal, Been Kim, and Asma Ghandeharioun. Does localization inform editing? surprising differences in causality-based localization vs. knowledge editing in language models. *arXiv preprint arXiv:2301.04213*, 2023. URL https://arxiv.org/pdf/2301.04213.pdf.

[73] Peter Hase, Mohit Bansal, Peter Clark, and Sarah Wiegreffe. The unreasonable effectiveness of easy training data for hard tasks. *arXiv preprint arXiv:2401.06751*, 2024. URL https://arxiv.org/pdf/2401.06751.pdf.

[74] Johannes Haug, Stefan Zürn, Peter El-Jiz, and Gjergji Kasneci. On baselines for local feature attributions. In *AAAI*, 2021. URL https://arxiv.org/pdf/2101.00905.pdf.

[75] Benjamin Heinzerling and Kentaro Inui. Language models as knowledge bases: On entity representations, storage capacity, and paraphrased queries. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1772–1791, Online, April 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.eacl-main.153.

[76] Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. Generating visual explanations. In *ECCV 2016*, 2016. URL https://arxiv.org/pdf/1603.08507.pdf.

[77] Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. Grounding visual explanations. In *ECCV 2018*, 2018. URL https://arxiv.org/pdf/1807.09685.pdf.

[78] Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob Andreas. Natural language descriptions of deep visual features. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/pdf?id=NudBMY-tzDr.

[79] John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *EMNLP*, 2019. URL https://arxiv.org/pdf/1909.03368.pdf.

[80] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1806.10758.

[81] Cheng-Yu Hsieh, Chih-Kuan Yeh, Xuanqing Liu, Pradeep Ravikumar, Seungyeon Kim, Sanjiv Kumar, and Cho-Jui Hsieh. Evaluations and methods for explanation through robustness analysis. *arXiv preprint arXiv:2006.00442*, 2020. URL https://arxiv.org/pdf/2006.00442.pdf.

[82] Amanda Hutton, Alexander Liu, and Cheryl Martin. Crowdsourcing Evaluations of Classifier Interpretability. In *AAAI Spring Symposium: Wisdom of the Crowd*, pages 21–26, 2012.

[83] Alon Jacovi and Yoav Goldberg. Aligning faithful interpretations with their social attribution. *arXiv preprint arXiv:2006.01067*, 2020. URL https://arxiv.org/abs/2006.01067.

[84] Alon Jacovi and Yoav Goldberg. Towards faithfully interpretable nlp systems: How should we define and evaluate faithfulness? In *ACL 2020*, 2020. URL https://www.aclweb.org/anthology/2020.acl-main.386.pdf.

[85] Alon Jacovi and Yoav Goldberg. Aligning faithful interpretations with their social attribution. *Transactions of the Association for Computational Linguistics*, 9:294–310, 2021. URL https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00367/98620/Aligning.

[86] Alon Jacovi, Ana Marasović, Tim Miller, and Yoav Goldberg. Formalizing trust in artificial intelligence: Prerequisites, causes and goals of human trust in ai. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 624–635, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445923. URL https://doi.org/10.1145/3442188.3445923.

[87] Sarthak Jain and Byron C Wallace. Attention is not explanation. 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 3543–3556, 2019.

[88] Sarthak Jain, Sarah Wiegreffe, Yuval Pinter, and Byron C. Wallace. Learning to faithfully rationalize by construction. In *ACL 2020*, pages 4459–4473, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.409. URL https://www.aclweb.org/anthology/2020.acl-main.409.

[89] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2016. URL https://arxiv.org/pdf/1611.01144.pdf.

[90] Dominik Janzing, Lenon Minorics, and Patrick Blöbaum. Feature relevance quantification in explainable ai: A causal problem. In *International Conference on Artificial Intelligence and Statistics*, pages 2907–2916. PMLR, 2020. URL https://arxiv.org/pdf/1910.13413.pdf.

[91] Neil Jethani, Mukund Sudarshan, Yindalon Aphinyanaphongs, and Rajesh Ranganath. Have we learned to explain?: How interpretability methods can learn to encode predictions in their interpretations. In *International Conference on Artificial Intelligence and Statistics*, pages 1459–1467. PMLR, 2021. URL https://arxiv.org/pdf/2103.01890.pdf.

[92] Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

[93] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2017. URL https://arxiv.org/pdf/1702.08734.pdf.

[94] Shalmali Joshi, Oluwasanmi Koyejo, Been Kim, and Joydeep Ghosh. xGEMs: Generating Examplars to Explain Black-Box Models. *arXiv:1806.08867 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1806.08867. arXiv: 1806.08867.

[95] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://www.aclweb.org/anthology/2020.emnlp-main.550.

[96] Nora Kassner, Oyvind Tafjord, Hinrich Schütze, and Peter Clark. Beliefbank: Adding memory to a pre-trained language model for a systematic notion of belief. *arXiv preprint arXiv:2109.14723*, 2021. URL https://arxiv.org/pdf/2109.14723.pdf.

[97] Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. 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 252–262, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1023. URL https://www.aclweb.org/anthology/N18-1023.

[98] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2668–2677. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/kim18d.html.

[99] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). In *Proceedings of the 35th International Conference on Machine Learning*, pages 2668–2677. PMLR, June 2018. URL http://arxiv.org/abs/1711.11279. arXiv: 1711.11279.

[100] Jinkyu Kim, Anna Rohrbach, Trevor Darrell, John F. Canny, and Zeynep Akata. Textual explanations for self-driving vehicles. In *ECCV 2018*, 2018. URL https://arxiv.org/pdf/1807.11546.pdf.

[101] Siwon Kim, Jihun Yi, Eunji Kim, and Sungroh Yoon. Interpretation of NLP models through input marginalization. In *EMNLP*, pages 3154–3167, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.255. URL https://www.aclweb.org/anthology/2020.emnlp-main.255.

[102] Sawan Kumar and Partha Talukdar. Nile : Natural language inference with faithful natural language explanations. In *ACL 2020*, 2020. URL https://arxiv.org/abs/2005.12116.

[103] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Jure Leskovec. Faithful and customizable explanations of black box models. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 131–138, 2019. URL https://papers.nips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[104] Yair Lakretz, German Kruszewski, Theo Desbordes, Dieuwke Hupkes, Stanislas Dehaene, and Marco Baroni. The emergence of number and syntax units in lstm language models. In *NAACL-HLT*, 2019. URL https://arxiv.org/pdf/1903.07435.pdf.

[105] Yair Lakretz, Dieuwke Hupkes, Alessandra Vergallito, Marco Marelli, Marco Baroni, and Stanislas Dehaene. Mechanisms for handling nested dependencies in neural-network language models and humans. *Cognition*, 213:104699, 04 2021. doi: 10.1016/j.cognition.2021.104699. URL https://arxiv.org/ftp/arxiv/papers/2006/2006.11098.pdf.

[106] Andrew K Lampinen, Ishita Dasgupta, Stephanie CY Chan, Kory Matthewson, Michael Henry Tessler, Antonia Creswell, James L McClelland, Jane X Wang, and Felix Hill. Can language models learn from explanations in context? *arXiv preprint arXiv:2204.02329*, 2022. URL https://arxiv.org/pdf/2204.02329.pdf.

[107] Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. In *ICLR 2017*, 2017. URL https://arxiv.org/pdf/1612.07182.pdf.

[108] Angeliki Lazaridou, Anna Potapenko, and Olivier Tieleman. Multi-agent communication meets natural language: Synergies between functional and structural language learning. In *ACL 2020*, pages 7663–7674. Association for Computational Linguistics, 2020. URL https://www.aclweb.org/anthology/2020.acl-main.685/.

[109] Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Sebastian Ruder, Dani Yogatama, et al. Mind the gap: Assessing temporal generalization in neural language models. In *NeurIPS*, 2021. URL https://arxiv.org/pdf/2102.01951.pdf.

[110] Eric Lehman, Jay DeYoung, Regina Barzilay, and Byron C. Wallace. Inferring which medical treatments work from reports of clinical trials. 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 3705–3717, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1371. URL https://www.aclweb.org/anthology/N19-1371.

[111] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational*

Natural Language Learning (CoNLL 2017), pages 333–342, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-1034. URL https://aclanthology.org/K17-1034.

[112] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS*, 2020. URL https://arxiv.org/abs/2005.11401.

[113] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015. URL https://arxiv.org/pdf/1506.01066.pdf.

[114] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and Understanding Neural Models in NLP. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 681–691. Association for Computational Linguistics, June 2016. doi: 10.18653/v1/N16-1082. URL https://www.aclweb.org/anthology/N16-1082.

[115] Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220*, 2016. URL https://arxiv.org/pdf/1612.08220.pdf.

[116] Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. *arXiv preprint arXiv:2306.03341*, 2023.

[117] Weixin Liang, James Zou, and Zhou Yu. ALICE: active learning with contrastive natural language explanations. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *EMNLP*, pages 4380–4391. Association for Computational Linguistics, 2020. URL https://www.aclweb.org/anthology/2020.emnlp-main.355/.

[118] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021. URL https://arxiv.org/pdf/2109.07958.pdf.

[119] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *ACL 2017*, 2017. URL https://arxiv.org/pdf/1705.04146.pdf.

[120] Zachary C. Lipton. The Mythos of Model Interpretability. *2016 ICML Workshop on Human Interpretability in Machine Learning*, June 2016. URL http://arxiv.org/abs/1606.03490.

[121] Nelson F. Liu, Tony Lee, Robin Jia, and Percy Liang. Can small and synthetic benchmarks drive modeling innovation? a retrospective study of question answering modeling approaches. *CoRR*, 2021. URL https://arxiv.org/pdf/2102.01065.pdf.

[122] Sijia Liu, Yuanshun Yao, Jinghan Jia, Stephen Casper, Nathalie Baracaldo, Peter Hase, Xiaojun Xu, Yuguang Yao, Hang Li, Kush R Varshney, et al. Rethinking machine unlearning for large language models. *arXiv preprint arXiv:2402.08787*, 2024. URL https://arxiv.org/pdf/2402.08787.pdf.

[123] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019. URL https://arxiv.org/pdf/1907.11692.pdf.

[124] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. URL https://arxiv.org/pdf/1711.05101.pdf.

[125] Charles Lovering, Rohan Jha, Tal Linzen, and Ellie Pavlick. Predicting inductive biases of pre-trained models, 2021. URL https://openreview.net/pdf/5f8e7508b216ea50a36e7f4584e4e6d8953917be.pdf.

[126] Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774, 2017. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[127] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR 2017*, 2017. URL https://arxiv.org/abs/1611.00712.

[128] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual knowledge in gpt. In *NeurIPS 2022*, 2022. URL https://arxiv.org/pdf/2202.05262.pdf.

[129] Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. *arXiv preprint arXiv:2210.07229*, 2022. URL https://arxiv.org/pdf/2210.07229.pdf.

[130] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *arXiv preprint arXiv:1301.3781*, 2013. URL https://arxiv.org/pdf/1301.3781.pdf.

[131] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019. doi: 10.1016/j.artint.2018.07.007. URL https://doi.org/10.1016/j.artint.2018.07.007.

[132] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021. URL https://arxiv.org/pdf/2110.11309.pdf.

[133] Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. Memory-based model editing at scale. In *International Conference on Machine Learning*, pages 15817–15831. PMLR, 2022. URL https://arxiv.org/pdf/2206.06520.pdf.

[134] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018. URL https://arxiv.org/pdf/1704.03976.pdf.

[135] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern recognition*, 45(1): 521–530, 2012. URL https://rtg.cis.upenn.edu/cis700-2019/papers/dataset-shift/dataset-shift-terminology.pdf.

[136] Ramaravind Kommiya Mothilal, Amit Sharma, and Chenhao Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In Mireille Hildebrandt, Carlos Castillo, Elisa Celis, Salvatore Ruggieri, Linnet Taylor, and Gabriela Zanfir-Fortuna, editors, *FAT* '20: Conference on Fairness, Accountability, and Transparency*, pages 607–617. ACM, 2020. doi: 10.1145/3351095.3372850. URL https://doi.org/10.1145/3351095.3372850.

[137] Jesse Mu and Jacob Andreas. Compositional explanations of neurons. *Advances in Neural Information Processing Systems*, 33:17153–17163, 2020. URL https://proceedings.neurips.cc/paper/2020/file/c74956ffb38ba48ed6ce977af6727275-Paper.pdf.

[138] Shikhar Murty, Pang Wei Koh, and Percy Liang. Expbert: Representation engineering with natural language explanations. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *ACL*, pages 2106–2113. Association for Computational Linguistics, 2020. URL https://www.aclweb.org/anthology/2020.acl-main.190/.

[139] Sharan Narang, Colin Raffel, Katherine J. Lee, Adam Roberts, Noah Fiedel, and Karishma Malkan. WT5?! training text-to-text models to explain their predictions. *ArXiv*, abs/2004.14546, 2020. URL https://arxiv.org/pdf/2004.14546.pdf.

[140] Albert Newen and Tobias Starzak. How to ascribe beliefs to animals. *Mind & Language*, 2020. URL https://onlinelibrary.wiley.com/doi/full/10.1111/mila.12302.

[141] Dong Nguyen. Comparing Automatic and Human Evaluation of Local Explanations for Text Classification. 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 1069–1078, June 2018. doi: 10.18653/v1/N18-1097. URL https://www.aclweb.org/anthology/N18-1097.pdf.

[142] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. doi: 10.23915/distill.00010. https://distill.pub/2018/building-blocks.

[143] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - EMNLP '02*, volume 10, pages 79–86. Association for Computational Linguistics, 2002. doi: 10.3115/1118693.1118704. URL http://portal.acm.org/citation.cfm?doid=1118693.1118704.

[144] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL 2002*, 2002. URL https://www.aclweb.org/anthology/P02-1040.pdf.

[145] Bhargavi Paranjape, Mandar Joshi, John Thickstun, Hannaneh Hajishirzi, and Luke Zettlemoyer. An information bottleneck approach for controlling conciseness in rationale extraction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural*

Language Processing (EMNLP), pages 1938–1952, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.153. URL https://www.aclweb.org/anthology/2020.emnlp-main.153.

[146] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. URL https://www.sciencedirect.com/science/article/pii/S0893608019300231.

[147] Dong Huk Park, Lisa Anne Hendricks, Zeynep Akata, Anna Rohrbach, Bernt Schiele, Trevor Darrell, and Marcus Rohrbach. Multimodal explanations: Justifying decisions and pointing to the evidence. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8779–8788, 2018. URL https://ieeexplore.ieee.org/document/8579013/.

[148] Vaidehi Patil, Peter Hase, and Mohit Bansal. Can sensitive information be deleted from llms? objectives for defending against extraction attacks. *arXiv preprint arXiv:2309.17410*, 2023. URL https://arxiv.org/pdf/2309.17410.pdf.

[149] Ellie Pavlick and Tom Kwiatkowski. Inherent disagreements in human textual inferences. *Transactions of the Association for Computational Linguistics*, 7:677–694, 2019. doi: 10.1162/tacl_a_00293. URL https://doi.org/10.1162/tacl_a_00293.

[150] Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 3(0):96–146, 2009. ISSN 1935-7516. doi: 10.1214/09-SS057. URL http://projecteuclid.org/euclid.ssu/1255440554.

[151] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? 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 2463–2473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1250. URL https://aclanthology.org/D19-1250.

[152] Marc Pirlot. General local search methods. *European journal of operational research*, 92(3):493–511, 1996.

[153] John Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Adv. Large Margin Classif.*, 10, 06 2000.

[154] Matt Post. A call for clarity in reporting bleu scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, 2018.

[155] Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. Grips: Gradient-free, edit-based instruction search for prompting large language models. *arXiv preprint arXiv:2203.07281*, 2022. URL https://arxiv.org/pdf/2203.07281.pdf.

[156] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *J. Mach. Learn. Res.*, 20(53):1–32, 2019. URL https://arxiv.org/pdf/1802.09596.pdf.

[157] Danish Pruthi, Bhuwan Dhingra, Livio Baldini Soares, Michael Collins, Zachary C. Lipton, Graham Neubig, and William W. Cohen. Evaluating explanations: How much

do explanations from the teacher aid students? *TACL*, abs/2012.00893, 2021. URL https://arxiv.org/abs/2012.00893.

[158] Luyu Qiu, Yi Yang, Caleb Chen Cao, Jing Liu, Yueyuan Zheng, Hilary Hei Ting Ngai, Janet Hsiao, and Lei Chen. Resisting out-of-distribution data problem in perturbation of xai. *arXiv preprint arXiv:2107.14000*, 2021.

[159] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017. URL https://arxiv.org/pdf/1704.01444.pdf.

[160] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. In *OpenAI Technical Report*, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[161] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. URL https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[162] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683, 2019. URL https://arxiv.org/pdf/1910.10683.pdf.

[163] Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. Explain yourself! leveraging language models for commonsense reasoning. In *ACL 2019*, 2019. URL https://arxiv.org/pdf/1906.02361.pdf.

[164] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *EMNLP-IJCNLP*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://www.aclweb.org/anthology/D19-1410.

[165] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Knowledge Discovery and Data Mining (KDD)*, February 2016. URL http://arxiv.org/abs/1602.04938.

[166] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI 2018*, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16982.

[167] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence*, 2018. URL https://homes.cs.washington.edu/~marcotcr/aaai18.pdf.

[168] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.437. URL https://aclanthology.org/2020.emnlp-main.437.

[169] M. Robnik-Sikonja and I. Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20:589–600, 2008. URL http://lkm.fri.uni-lj.si/rmarko/papers/RobnikSikonjaKononenko08-TKDE.pdf.

[170] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020. doi: 10.1162/tacl_a_00349. URL https://aclanthology.org/2020.tacl-1.54.

[171] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *IJCAI*, pages 2662–2670, 2017. doi: 10.24963/ijcai.2017/371. URL https://doi.org/10.24963/ijcai.2017/371.

[172] Cynthia Rudin. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 1:206–215, May 2019. URL https://www.nature.com/articles/s42256-019-0048-x.

[173] Christian Rupprecht, Iro Laina, Nassir Navab, Gregory D. Harger, and Federico Tombari. Guide me: Interacting with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*, 2018. URL https://arxiv.org/abs/1803.11544.

[174] Swarnadeep Saha, Peter Hase, Nazneen Rajani, and Mohit Bansal. Are hard examples also harder to explain? a study with human and model-generated explanations. *arXiv preprint arXiv:2211.07517*, 2022. URL https://arxiv.org/pdf/2211.07517.pdf.

[175] Swarnadeep Saha, Shiyue Zhang, Peter Hase, and Mohit Bansal. Summarization programs: Interpretable abstractive summarization with neural modular trees. *arXiv preprint arXiv:2209.10492*, 2022. URL https://arxiv.org/pdf/2209.10492.pdf.

[176] Swarnadeep Saha, Peter Hase, and Mohit Bansal. Can language models teach weaker agents? teacher explanations improve students via theory of mind. *arXiv preprint arXiv:2306.09299*, 2023. URL https://arxiv.org/pdf/2306.09299.pdf.

[177] Pouya Samangouei, Ardavan Saeedi, Liam Nakagawa, and Nathan Silberman. ExplainGAN: Model Explanation via Decision Boundary Crossing Transformations. In *ECCV 2018*. Springer International Publishing, 2018. ISBN 978-3-030-01248-9 978-3-030-01249-6. doi: 10.1007/978-3-030-01249-6_41. URL http://link.springer.com/10.1007/978-3-030-01249-6_41.

[178] Shibani Santurkar, Dimitris Tsipras, Mahalaxmi Elango, David Bau, Antonio Torralba, and Aleksander Madry. Editing a classifier by rewriting its prediction rules. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 23359–23373. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/c46489a2d5a9a9ecfc53b17610926ddd-Paper.pdf.

[179] Soumya Sanyal and Xiang Ren. Discretized integrated gradients for explaining language models. *arXiv preprint arXiv:2108.13654*, 2021. URL https://arxiv.org/pdf/2108.13654.pdf.

[180] Christian Schäfer. Particle algorithms for optimization on binary spaces. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 23(1):1–25, 2013. URL https://arxiv.org/pdf/1111.0574.pdf.

[181] Johannes Schneider and Michalis Vlachos. Explaining neural networks by decoding layer activations. In *International Symposium on Intelligent Data Analysis*, pages 63–75. Springer, 2021. URL https://arxiv.org/pdf/2005.13630.pdf.

[182] Eric Schwitzgebel. Belief. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2019 edition, 2019. URL https://plato.stanford.edu/entries/belief/.

[183] Ramprasaath Ramasamy Selvaraju, Stefan Lee, Yilin Shen, Hongxia Jin, Shalini Ghosh, Larry P. Heck, Dhruv Batra, and Devi Parikh. Taking a HINT: leveraging explanations to make vision and language models more grounded. In *ICCV*, pages 2591–2600. IEEE, 2019. doi: 10.1109/ICCV.2019.00268. URL https://doi.org/10.1109/ICCV.2019.00268.

[184] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pages 3145–3153, 2017. URL https://arxiv.org/pdf/1704.02685.pdf.

[185] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *Workshop at International Conference on Learning Representations.*, 2013. URL https://arxiv.org/pdf/1312.6034.pdf.

[186] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*, 2014. URL https://arxiv.org/abs/1312.6034.

[187] Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks. In *ICLR*, 2020. URL https://openreview.net/pdf?id=HJedXaEtvS.

[188] Kevin Small, Byron C Wallace, Carla E Brodley, and Thomas A Trikalinos. The constrained weight space svm: learning with ranked features. In *ICML*, pages 865–872, 2011.

[189] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017. URL https://arxiv.org/pdf/1706.03825.pdf.

[190] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 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, October 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1170.

[191] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 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, 2013. URL https://www.aclweb.org/anthology/D13-1170.pdf.

[192] Shashank Srivastava, I. Labutov, and T. Mitchell. Learning classifiers from declarative language. In *NeurIPS 2017*, 2017. URL http://www.cs.cmu.edu/~shashans/papers/srivastava17-lldworkshop.pdf.

[193] Shashank Srivastava, Igor Labutov, and Tom Mitchell. Zero-shot learning of classifiers from natural language quantification. In *ACL 2018*, July 2018. doi: 10.18653/v1/P18-1029. URL https://www.aclweb.org/anthology/P18-1029.

[194] Joe Stacey, Yonatan Belinkov, and Marek Rei. Supervising model attention with human explanations for robust natural language inference. In *AAAI*, 2022. URL https://arxiv.org/pdf/2104.08142.pdf.

[195] Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. *CoRR*, abs/2011.12854, 2020. URL https://arxiv.org/abs/2011.12854.

[196] Pascal Sturmfels, Scott Lundberg, and Su-In Lee. Visualizing the impact of feature attribution baselines. *Distill*, 5(1):e22, 2020. URL https://distill.pub/2020/attribution-baselines/.

[197] Mukund Sundararajan and Amir Najmi. The many shapley values for model explanation. In *International Conference on Machine Learning*, pages 9269–9278. PMLR, 2020. URL https://arxiv.org/pdf/1908.08474.pdf.

[198] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pages 3319–3328, 2017. URL https://arxiv.org/pdf/1703.01365.pdf.

[199] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, June 2017. URL http://arxiv.org/abs/1703.01365. arXiv: 1703.01365.

[200] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *NAACL-HLT 2019*, 2019. URL https://arxiv.org/pdf/1811.00937.pdf.

[201] Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. Leap-of-thought: Teaching pre-trained models to systematically reason over implicit knowledge. In *NeurIPS*, 2020. URL http://128.84.4.27/pdf/2006.06609.

[202] Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. Leap-of-thought: Teaching pre-trained models to systematically reason over implicit knowledge. In *NeurIPS 2020*, 2020. URL https://arxiv.org/abs/2006.06609.

[203] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. FEVER: a large-scale dataset for fact extraction and VERification. 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 809–819, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1074. URL https://www.aclweb.org/anthology/N18-1074.

[204] Marcos Vinícius Treviso and André F. T. Martins. Towards prediction explainability through sparse communication. *ArXiv*, abs/2004.13876, 2020. URL https://arxiv.org/pdf/2004.13876.pdf.

[205] Keyon Vafa, Yuntian Deng, David M Blei, and Alexander M Rush. Rationales for sequential predictions. In *EMNLP*, 2021. URL https://arxiv.org/pdf/2109.06387.pdf.

[206] Jan N Van Rijn and Frank Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2367–2376, 2018. URL https://arxiv.org/pdf/1710.04725.pdf.

[207] Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Simas Sakenis, Jason Huang, Yaron Singer, and Stuart Shieber. Causal mediation analysis for interpreting neural nlp: The case of gender bias. *arXiv preprint arXiv:2004.12265*, 2020. URL https://arxiv.org/pdf/2004.12265.pdf.

[208] Elena Voita and Ivan Titov. Information-theoretic probing with minimum description length. In *EMNLP*, 2020. URL https://aclanthology.org/2020.emnlp-main.14.pdf.

[209] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

[210] Cunxiang Wang, Shuailong Liang, Yue Zhang, Xiaonan Li, and Tian Gao. Does it make sense? and why? a pilot study for sense making and explanation. In *ACL 2019*, 2019. URL https://arxiv.org/pdf/1906.00363.pdf.

[211] Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Guihong Cao, Daxin Jiang, Ming Zhou, et al. K-adapter: Infusing knowledge into pre-trained models with adapters. In *Findings of ACL*, 2021. URL https://aclanthology.org/2021.findings-acl.121.pdf.

[212] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021. URL https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00360/98089/KEPLER-A-Unified-Model-for-Knowledge-Embedding-and.

[213] Xiaozhi Wang, Kaiyue Wen, Zhengyan Zhang, Lei Hou, Zhiyuan Liu, and Juanzi Li. Finding skill neurons in pre-trained transformer-based language models. *arXiv preprint arXiv:2211.07349*, 2022. URL https://arxiv.org/pdf/2211.07349.pdf.

[214] Ziqi Wang, Yujia Qin, Wenxuan Zhou, Jun Yan, Qinyuan Ye, Leonardo Neves, Zhiyuan Liu, and Xiang Ren. Learning from explanations with neural execution tree. In *ICLR*, 2019. URL https://openreview.net/pdf?id=rJlUt0EYwS.

[215] Peter West, Chandra Bhagavatula, Jack Hessel, Jena D Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. Symbolic knowledge distillation: from

general language models to commonsense models. *arXiv preprint arXiv:2110.07178*, 2021. URL https://arxiv.org/pdf/2110.07178.pdf.

[216] Sarah Wiegreffe and Yuval Pinter. 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, 2019. URL https://arxiv.org/pdf/1908.04626.pdf.

[217] Sarah Wiegreffe, Ana Marasovic, and Noah A. Smith. Measuring association between labels and free-text rationales. *CoRR*, abs/2010.12762, 2020. URL https://arxiv.org/abs/2010.12762.

[218] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. URL https://link.springer.com/article/10.1023/A:1022672621406.

[219] Maksymilian Wojtas and Ke Chen. Feature importance ranking for deep learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5105–5114. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/36ac8e558ac7690b6f44e2cb5ef93322-Paper.pdf.

[220] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, pages arXiv–1910, 2019.

[221] 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. 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, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[222] Prateek Yadav, Peter Hase, and Mohit Bansal. INSPIRE: Incorporating diverse feature preferences in recourse. *arxiv*, 2021. URL https://openreview.net/pdf?id=6yzIuqKGnq.

[223] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical Attention Networks for Document Classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489. Association for Computational Linguistics, June 2016. doi: 10.18653/v1/N16-1174. URL https://www.aclweb.org/anthology/N16-1174.

[224] Jihun Yi, Eunji Kim, Siwon Kim, and Sungroh Yoon. Information-theoretic visual explanation for black-box classifiers. *arXiv preprint arXiv:2009.11150*, 2020. URL https://arxiv.org/pdf/2009.11150.pdf.

[225] Fan Yin, Zhouxing Shi, Cho-Jui Hsieh, and Kai-Wei Chang. On the faithfulness measurements for model interpretations, 2021. URL https://arxiv.org/pdf/2104.08782.pdf.

[226] Zhuofan Ying, Peter Hase, and Mohit Bansal. Visfis: Visual feature importance supervision with right-for-the-right-reason objectives. *Advances in Neural Information Processing Systems*, 35:17057–17072, 2022. URL https://arxiv.org/pdf/2206.11212.pdf.

[227] Zhuofan Ying, Peter Hase, and Mohit Bansal. Adaptive contextual perception: How to generalize to new backgrounds and ambiguous objects. *arXiv preprint arXiv:2306.05963*, 2023. URL https://arxiv.org/pdf/2306.05963.pdf.

[228] Omar Zaidan, Jason Eisner, and Christine Piatko. Using "Annotator Rationales" to Improve Machine Learning for Text Categorization. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 260–267, Rochester, New York, April 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N07-1033.

[229] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. URL https://arxiv.org/pdf/1311.2901.pdf.

[230] Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. From recognition to cognition: Visual commonsense reasoning. In *IEEE/CVF 2019*, 2019. URL https://ieeexplore.ieee.org/document/8953217.

[231] Ruihan Zhang, Prashan Madumal, Tim Miller, Krista A Ehinger, and Benjamin IP Rubinstein. Invertible concept-based explanations for cnn models with non-negative concept activation vectors. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. URL https://arxiv.org/pdf/2006.15417.pdf.

[232] Ye Zhang, Iain Marshall, and Byron C. Wallace. Rationale-Augmented Convolutional Neural Networks for Text Classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 795–804, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1076. URL https://www.aclweb.org/anthology/D16-1076.

[233] Sumu Zhao, Damián Pascual, Gino Brunner, and Roger Wattenhofer. Of non-linearity and commutativity in bert. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021. URL https://arxiv.org/pdf/2101.04547.pdf.

[234] Xinyan Zhao and VG Vydiswaran. Lirex: Augmenting language inference with relevant explanation. In *AAAI*, 2021.

[235] Ruiqi Zhong, Steven Shao, and Kathleen McKeown. Fine-grained sentiment analysis with faithful attention. *arXiv preprint arXiv:1908.06870*, 2019. URL https://arxiv.org/pdf/1908.06870.pdf.

[236] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *IEEE transactions on pattern analysis and*

machine intelligence, 41(9):2131–2145, 2018. URL https://arxiv.org/pdf/1711.05611.pdf.

[237] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Interpretable basis decomposition for visual explanation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 119–134, 2018. URL https://people.csail.mit.edu/bzhou/publication/eccv18-IBD.

[238] Wangchunshu Zhou, Jinyi Hu, Hanlin Zhang, Xiaodan Liang, Maosong Sun, Chenyan Xiong, and Jian Tang. Towards interpretable natural language understanding with explanations as latent variables. In *NeurIPS*, 2020. URL https://arxiv.org/pdf/2011.05268.pdf.

[239] Yilun Zhou, Serena Booth, Marco Tulio Ribeiro, and Julie Shah. Do feature attribution methods correctly attribute features? *arXiv preprint arXiv:2104.14403*, 2021. URL https://arxiv.org/pdf/2104.14403.pdf.

[240] Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. Modifying memories in transformer models. *arXiv preprint arXiv:2012.00363*, 2020. URL https://arxiv.org/pdf/2012.00363.pdf.

[241] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. In *ICLR*, 2017. URL https://arxiv.org/pdf/1702.04595.pdf.

[242] Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023. URL https://arxiv.org/pdf/2310.01405.pdf.

A Additional Results and Details for Chapter 4

A.1 Method Implementations

Explanation methods. For our tabular data, we use the implementations of Anchor and LIME provided in the code for Ribeiro et al. [167]. We implement our prototype and decision boundary methods. With text data, we use the implementation of Anchor provided by Ribeiro et al. [167], and for LIME we use the code provided with Ribeiro et al. [165]. As before, we implement our prototype and decision boundary methods.

Text and Tabular Models. We train neural networks for both tasks as follows: for our tabular task model, we use a neural network with two hidden layers, each of width 50, as Ribeiro et al. [167] do. For our text task model, we use a BiLSTM of the kind introduced by Yang et al. [223], who reported state of the art results on a number of sentiment analysis tasks. Since their network is designed for classification of documents, we limit our network components to those relevant to classification of single sentences. We build our prototype models on top of the feature extractor layers of each of these models, meaning that we only replace the final classifier layer of the neural task model with a prototype layer. Accuracies for each model are shown in Table A.1. The task models are trained with stochastic gradient descent and a cross-entropy loss function, using early stopping on a validation dataset and l_2 regularization with a coefficient of 1e−4. See training details for the prototype models below.

Prototype Model Training. Here we describe our prototype training algorithm, beginning with weight initialization. We initialize 1) feature extraction layers using the pretrained weights of our neural task model, 2) prototype vectors via k-means clustering on the latent representations of the entire training set, and 3) final classifier weights as 1 where the corresponding prototype's class matches the weight vector's class, and −.5 elsewhere. The objective function for our prototype models contains three terms: 1) a cross entropy loss, 2) l_1 regularization on off-class weights in the classifier, and 3) a separation cost term, which is the minimum distance between a latent representation and any prototype not belonging to the input's class.

Importance Scores in Protoype Model. For a given feature, we compute an importance score by taking the difference in function output with that feature present in the input, relative to when that feature is omitted. With text data, there are a number of mechanisms by which one can omit a word from an input; we opt for setting that word's embedding to the zero vector. For tabular data, to estimate a variable value's importance we compute a measure of *evidence gain* from knowing the value, relative to not knowing it. Formally, our importance function is the difference between the function value at the original input and the expected function value for the input with variable j removed. The expectation is taken over a distribution generated by an imputation model conditioned on the remaining covariates.

$$\begin{aligned} &Importance(\mathbf{x}_{i,j}) = \\ &\quad f(\mathbf{x}_i) - \mathbb{E}_{p(\mathbf{x}_{i,j}|\mathbf{x}_{i,-j})} f(\mathbf{x}_{i,-j} \cup \mathbf{x}_{i,j}) \end{aligned}$$

where $p(\mathbf{x}_{i,j}|\mathbf{x}_{i,-j})$ is given by a multinomial logistic regression fit to the training data, and $\mathbf{x}_{i,-j}$ is the data point without feature j, and $f(\mathbf{x}_{i,-j} \cup \mathbf{x}_{i,j})$ is the data point $\mathbf{x}_{i,-j}$ with feature value $\mathbf{x}_{i,j}$ imputed at index j. We choose to use logistic regressions with no feature engineering

Model Accuracies	
Data & Model	Test Acc
Text	
Task Model	80.93
Prototype	80.64
Tabular	
Task Model	83.49
Prototype	81.90

Table A.1: Model accuracies on each data domain. Text data is split into partitions of 70%, 10%, and 20% for the train, validation, and test sets, respectively. We use the same data processing scheme as Ribeiro et al. [167] for tabular data.

	User Ratings			
Model Correctness	n	μ	CI	σ
Text				
Correct	464	4.44	.49	1.89
Incorrect	468	4.12	.67	1.81
Tabular				
Correct	391	5.09	.27	1.64
Incorrect	394	4.64	.27	1.69

Table A.2: User simulatability ratings grouped by model correctness and data domain. Users do not seem to be rating explanations simply based on model correctness, as the differences in group means based on model correctness are not significant at a level of $p < .05$.

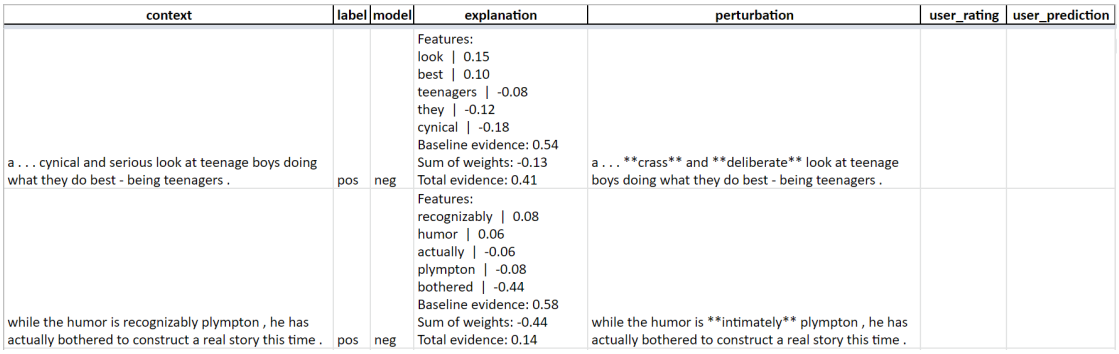
in order to 1) generate calibrated probability distributions, and 2) scale straightforwardly with dataset size.

Decision Boundary Algorithm. In detail, the algorithm takes as input a data point x^*, the classifier f, a perturbation distribution $\mathcal{D}(\cdot|x^*)$, and a measure of distance between inputs $d(x_1, x_2)$. We first sample $\{\tilde{x}\}_{i=1}^{10,000}$ from the perturbation distribution around x^*. The eligible perturbations to choose from are those with the opposite prediction from the original: $E = \{\tilde{x}_i | f(\tilde{x}_i) \neq f(x^*)\}$. Then using a distance function d, we select a counterfactual input as

$$x^{(c)} = \min_{\tilde{x}_i \in E} d(x^*, \tilde{x}_i)$$

We provide a path from x^* to $x^{(c)}$ by greedily picking the single edit from the remaining edits that least changes the model's evidence margin, which is the difference between positive and negative class scores. Our distance function is the count of different features between inputs, plus the squared Euclidean distance between latent representations. The Euclidean distance is on a scale such that it serves as a tie-breaker:

$$d(x_1, x_2) = \sum_j \mathbb{1}(x_{1j} \neq x_{2j}) + ||f(x_1) - f(x_2)||_2^2.$$



context	label	model	explanation	perturbation	user_rating	user_prediction
a . . . cynical and serious look at teenage boys doing what they do best - being teenagers .	pos	neg	Features: look | 0.15 best | 0.10 teenagers | -0.08 they | -0.12 cynical | -0.18 Baseline evidence: 0.54 Sum of weights: -0.13 Total evidence: 0.41	a . . . **crass** and **deliberate** look at teenage boys doing what they do best - being teenagers .		
while the humor is recognizably plympton , he has actually bothered to construct a real story this time .	pos	neg	Features: recognizably | 0.08 humor | 0.06 actually | -0.06 plympton | -0.08 bothered | -0.44 Baseline evidence: 0.58 Sum of weights: -0.44 Total evidence: 0.14	while the humor is **intimately** plympton , he has actually bothered to construct a real story this time .		

Figure A.1: A screenshot of our user testing interface. This example is of the counterfactual Post test with LIME for text data.

A.2 Perturbation Distributions

We design perturbation distributions for two points in our experiments: 1) selecting counterfactual inputs in simulation tests, and 2) generating decision boundary explanations. First, we describe our approaches for selecting counterfactual inputs, which are conditioned on the need for a certain prediction type: either the same prediction as the original input or the alternative class. In both data domains, we sample $10,000$ local perturbations around the input and then randomly pick a sample that the model predicts to be of the needed prediction type. While working with tabular data, we sample perturbations as follows: we randomly choose to make between 1 and 3 edits, then choose the features to edit uniformly at random, and finally pick new feature values uniformly at random. The only sampling constraint is that a variable cannot be set as its original value.

For text data, we use a strategy that is similar to sampling from the perturbation distribution in Ribeiro et al. [167], which is to randomly substitute words with their neighbors in GloVe word embedding space, sampling neighbors with probability proportional to their similarity. We make a few changes: we 1) decrease probability of token change with the length of sentence, 2) cap the number of edited words at 5 in the chosen perturbation if possible, and 3) limit edited tokens to be nouns, verbs, adjectives, adverbs, and adpositions. Example perturbations are shown in the example of the user testing interface in Figure A.1, which is given for a counterfactual test with text data.

A.3 Testing Environment

We show a screenshot of our user testing interface in Figure A.1. This example is of the counterfactual Post test with LIME for text data. Tests are administered through spreadsheets, wherein users read test material and place responses. Users are guided from file to file by the experimenter.

B Additional Results and Details for Chapter 5

B.1 Experimental Details

B.1.1 Datasets and Examples

We conduct experiments with each method using two datasets. The first is the CommonSenseQA[1] dataset of Talmor et al. [200], with explanations collected by Rajani et al. [163] to make a combined CoS-E dataset.[2] We opt for the Version 1.0 of this dataset since it has higher-quality explanations than Version 1.1.[3] The dataset split sizes are 7610, 950, and 940 for the train, dev, and test, respectively. Next, we use the e-SNLI dataset of Camburu et al. [22],[4] which includes explanations for the SNLI benchmark [20].[5] The split sizes are 549,339, 9842, and 9824, for train, dev, and test. Three explanations per data point are available for the test data in e-SNLI; to compute BLEU, we use the first explanation in the data for each data point; we use the `sacrebleu` Python package [154].[6]

Note that explanations for the CQA test split were not collected for the CoS-E dataset, as the CQA test split itself is withheld as a leaderboard test set. Meanwhile, we report results using 10% of the SNLI training data, since training our multi-task T5 models with the full e-SNLI dataset can take over 24 hours per epoch on a single T4 GPU. These accuracy results are shown in Table B.2. We report test set statistics here for simulation-related experiments for CQA, shown in Table 5.3, along with dev statistics for SNLI. Trends across models remain the same as with the data split statistics reported in the main paper. In Table B.6, we confirm trends observed with the SNLI training data subset using models trained with the entire dataset. Finally, Table B.1 shows additional examples from CQA and SNLI plus model-generated explanations.

B.1.2 Hypothesis Testing

We describe results as statistically significant when p-values are below .05, where p-values are calculated by bootstrap for LAS, a difference in the binomial means test for model accuracies, and by linear regression with i.i.d. normal noise for associations between human ratings and simulator correctness. Note that confidence intervals for LAS vary in width based on how many data points are in each leakage bin. With the expert evaluation, we compute Spearman's rank correlation between proxy and human simulation variables (with a corresponding p-value). For our data, the results are nearly identical to Pearson's linear correlation and Kendall's Tau.

B.1.3 Model Selection and Training Details

Our model selection procedure is to train each task model five times with differing seeds, then select the model with the best development performance. We train one simulator model per

[1] https://www.tau-nlp.org/commonsenseqa

[2] https://github.com/nazneenrajani/CoS-E

[3] In Version 1.1, 20% of explanations were found to belong to a small set of duplicates that are unrelated to the data point. See https://github.com/salesforce/cos-e/issues/2.

[4] https://github.com/OanaMariaCamburu/e-SNLI

[5] https://nlp.stanford.edu/projects/snli/

[6] https://github.com/mjpost/sacreBLEU

condition. Since the two-agent experiments have far increased computational load, we run one seed using a T5-Small during training, selecting the best task model according to its LAS with this weaker simulator. Afterward, we retrain with a T5-Base simulator.

Our training procedures result in the following (approximate) experimental times for each model when training on a single NVIDIA T4 GPU. With a T5-Base model and CQA data, our baseline takes about 10 hours for 20 epochs; ST-Re about 10 hours for 20 epochs; ST-Ra about 20 hours for 20 epochs; MT-Re about 12 hours for 20 epochs; MT-Ra about 12 hours for 20 epochs. Multi-agent RL optimization with a T5-Small simulator takes about 16 hours for 10 epochs, and SGD takes 24 hours for 10 epochs. Now with a T5-Base model and SNLI data (using 10% of the training data), our baseline takes about 24 hours for 10 epochs; ST-Re about 24 hours for 10 epochs; ST-Ra about 48 hours for 10 epochs; MT-Re about 30 hours for 10 epochs; MT-Ra about 30 hours for 10 epochs. Multi-agent RL optimization with a T5-Small simulator takes about 3 days for 5 epochs, and SGD takes 5 days for 5 epochs. Using the full SNLI dataset, the baseline took four days to train five epochs, and either MT model took 5 days for 5 epochs. We train generators for the ST conditions for 5 epochs on the 10% subset, which takes under 6 hours. Note that to follow our model selection procedure, experimental times should be multiplied by five here, and further extended to include training simulators.

Lastly, we note that T5-Base has 220 million parameters, while T5-Small as 60 million parameters [162]. In general, this means our model sizes are 220 million parameters, although, for multi-agent training, our effective model size is 280 million parameters.

B.1.4 Training Simulator Models

When training simulators, it is critical that the model can approximate the three distributions used in LAS computation: $p_\phi(\hat{y}_i|x_i, \hat{e}_i)$, $p_\phi(\hat{y}_i|x_i)$, and $p_\phi(\hat{y}_i|\hat{e}_i)$. This is achieved by applying dropout at the input token level to either (1) the entire x subsequence, or (2) the entire $\hat{e}$ subsequence. The same proportion of inputs in each batch are affected by the dropout, with the subset being chosen randomly. Without this technique, simulator models rely too heavily on explanations, and when conditioned only on x, they underperform baseline models that are trained only with x. In our multi-agent experiments, we take a nearly identical approach, but we make use of the fact that each of the three simulator predictions is made for each batch ($p_\phi(\hat{y}_i|x_i, \hat{e}_i)$, $p_\phi(\hat{y}_i|x_i)$, and $p_\phi(\hat{y}_i|\hat{e}_i)$). That is, we weight these terms in the simulator objective by ratios implied by our dropout technique, rather than using dropout directly. See the Section B.1.5 for the relevant hyperparameters.

B.1.5 Hyperparameter Tuning

For baselines, we tune hyperparameters such as the learning rate and batch size for accuracy, selecting from $[1e-5, 1e-4, 1e-3]$ for LR and $[4, 6, 12, 24, 36]$ for batch size, finally using $1e-4$, with CQA batch size 12 and SNLI batch size 36.

For multi-task models, we tune the mixing weight α based on task performance, searching over values in $[.3, .4, .5, .6, .7, .8]$, settling on .5.

For simulator models, we tune mixing weights (or dropout proportions) by selecting based on each of the three predictions' accuracies, relative to baseline models trained on one input type only. Specifically, we select based on the max accuracy of the subsequence (x and e) predictions (with accuracies added together), under the constraint that models must achieve within 1 percentage point accuracy of the overall $p_\phi(\hat{y}_i|x_i, \hat{e}_i)$ accuracy. Now taking $\lambda_{x,e}, \lambda_x$,

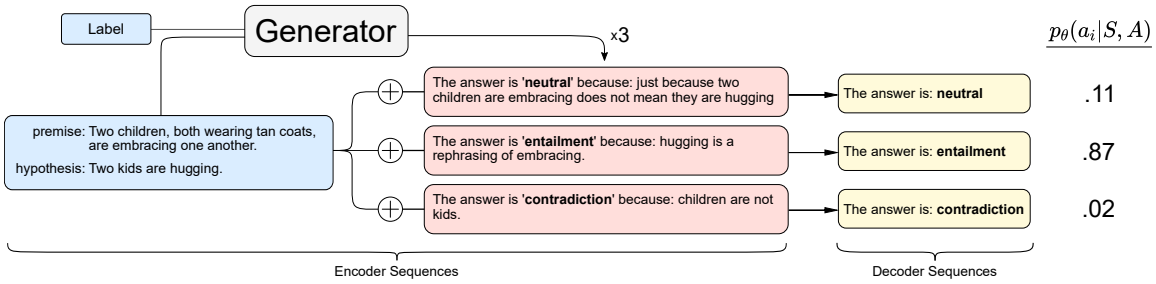


Figure B.1: Inputs and outputs for the sequence to sequence ST-Ra framework. One explanation is generated for each answer choice, conditioned on the choice. The sequences and answers are supplied to a sequence-to-sequence task model for scoring. We use separate T5 models for the generator and task model.

and λ_e as loss function weights for predictions conditioned on their subscripts, the effective loss function weights for CoS-E data are: $\lambda_{x,e} = .5, \lambda_x = .5$, and $\lambda_e = 0$; and for NLI, we use $\lambda_{x,e} = .4, \lambda_x = .4, \lambda_e = .2$.

The most complex set-up for tuning is our multi-agent method. Here, we must tune mixing weights for the task, LM, and explanation objectives, as well as the weight for penalizing leaking explanations. First, we tune the task, LM, and simulatability weights directly for overall simulator accuracy, without applying a penalty for leaking. We search each parameter over the range $[.2, .5]$ spaced by .05, with constraints that the three terms must add to 1, task weight must be as high as LM weight, and sim weight must be as high as task weight). Lastly, we tune the α trading off between explanation rewards and penalties by selecting directly for LAS scores; we search the unit interval spaces by .1. For SGD, α is set to .8 for CQA and .9 for SNLI; the task loss is .35, LM loss is .15, explanation loss is .5, and the simulator model objective adopts the same weights as described above. For RL, this mixing weight α is set to .8 for both datasets; the task loss is .025, LM loss is .025, explanation loss is .95, and the simulator model objective also adopts the same weights as described above.

B.2 LAS Robustness Checks

B.2.1 Continuous Leakage Scores and LAS Metric

While we binarize our proxy for label leakage based on prediction correctness and take the raw average of explanation effects across two leakage bins, a continuous measure of leakage can be obtained directly from $p(\hat{y}|\hat{e})$. Then, an arbitrary number of bins can be used. Interestingly, for a T5 model fine-tuned by decoder sequence likelihood maximization, these probabilities are tightly concentrated around values just above random chance performance (.33 for both CQA v1.0 and SNLI), taking a roughly normal distribution. As a result, they are easily calibrated via Platt scaling [153]. To check for our results' robustness, we perform sensitivity analysis with respect to the number of evenly spaced leakage bins chosen to subset, after calibrating our leakage probabilities. Across bin counts between 2 and 100, LAS estimates typically vary by less than 1 point, and as a result, method ranking is almost always preserved. In the limit of the number of bins, our metric becomes the integral of the explanation effect as a function of leakage probability. To ensure the robustness of LAS scores, this type of sensitivity analysis should be performed whenever possible, but especially when explanation effectiveness is not linearly related to the leakage probability.

B.2.2 Robustness to Seed and Model Choice

We check LAS scores across three random seeds since random seeds tend to have a large influence on all statistics derived from pretrained neural language models [45]. Results are shown in Table B.4. The rank ordering of scores is typically preserved, and in most cases, scores display relatively low variance, although there are some outlying values.

We also check the effect of using a different simulator model, shown in Table B.5. We compare between our primary choice of T5-Base and RoBERTa-Large models for SNLI data. For ST models, the task model and simulator are of the same architecture, but we do not evaluate MT conditions since RoBERTa is not generative. RoBERTa produces lower LAS scores than T5, and their rank ordering is not necessarily the same, though ST-RA is the highest on average in both cases. The differences between them could result from their pretraining procedures, architectural differences, finetuning sample efficiency, or another cause.

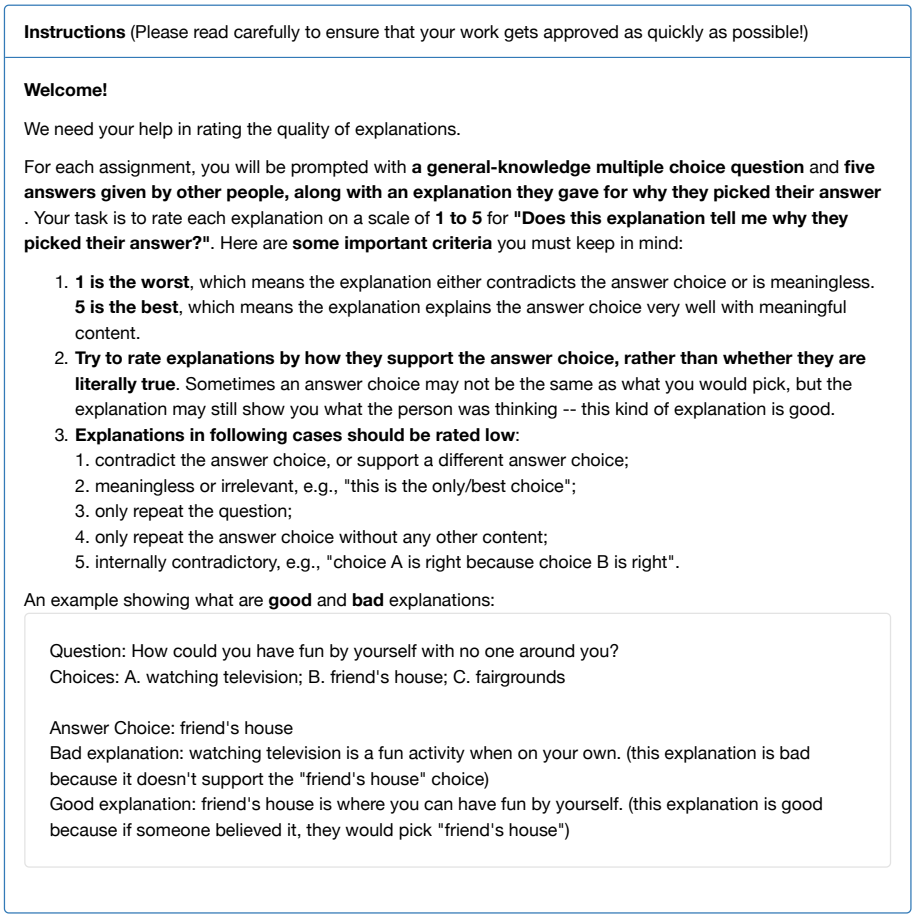
B.3 Alternative Computational Models and Language Modeling Objectives

Our generative models neither gained nor lost accuracy relative to their baselines when implemented with T5 models. Since learning from explanations to improve accuracy is another goal in collecting human explanations as data, we seek to assess this trend with alternative computational models and language modeling objectives. Hence, we test our MT models with Masked Language Modeling (MLM) objectives in place of the Causal objectives used for the generation, and wherever a generator or task model appears in current experiments, we test the effect of substituting GPT2 and BERT in their place. We show results for these models in Table B.8; GPT2+BERT methods are tagged as ENC methods. Just as with our generative approaches, we observe no differences in accuracies between baselines and other methods.

B.4 Human Quality Rating Collection

We collected the human ratings of explanation quality from Amazon Mechanical Turk. For CQA or SNLI, we sample 200 examples from the development or testing set (CQA's testing set does not contain human explanations). Each example has five explanations that are generated by the four models we introduced in the main paper as well as humans. We anonymously shuffle the five explanations and ask turkers to rate them separately on a 5-point Likert scale. Meanwhile, we give them some instructions about "rate explanations by how they support the answer choice, rather than whether they are literally true" and "explanations in which cases should be rated low". Figure B.2 shows the full instructions we used for collecting explanation ratings for CQA, and Figure B.3 shows one CQA question and its answer choices plus the first model's choice and its explanation. SNLI has a similar GUIs. Turkers will be required to rate five (choice, explanation) pairs on one page.

We collected 3 responses for each example, so there are 600 responses in total for each dataset. We apply a simply quality filter to filter the responses from bad turkers. We first manually picked 10 explanations from both CQA and SNLI that contradict their corresponding model outputs (choices). As we know, these explanations are sure to be bad. So, we filter the responses from those turkers who rated high (> 2 for CQA, > 3 for SNLI, since SNLI has



Instructions (Please read carefully to ensure that your work gets approved as quickly as possible!)

Welcome!

We need your help in rating the quality of explanations.

For each assignment, you will be prompted with **a general-knowledge multiple choice question** and **five answers given by other people, along with an explanation they gave for why they picked their answer** . Your task is to rate each explanation on a scale of **1 to 5** for **"Does this explanation tell me why they picked their answer?"**. Here are **some important criteria** you must keep in mind:

1. **1 is the worst**, which means the explanation either contradicts the answer choice or is meaningless. **5 is the best**, which means the explanation explains the answer choice very well with meaningful content.
2. **Try to rate explanations by how they support the answer choice, rather than whether they are literally true**. Sometimes an answer choice may not be the same as what you would pick, but the explanation may still show you what the person was thinking -- this kind of explanation is good.
3. **Explanations in following cases should be rated low**:
   1. contradict the answer choice, or support a different answer choice;
   2. meaningless or irrelevant, e.g., "this is the only/best choice";
   3. only repeat the question;
   4. only repeat the answer choice without any other content;
   5. internally contradictory, e.g., "choice A is right because choice B is right".

An example showing what are **good** and **bad** explanations:

Question: How could you have fun by yourself with no one around you?
Choices: A. watching television; B. friend's house; C. fairgrounds

Answer Choice: friend's house
Bad explanation: watching television is a fun activity when on your own. (this explanation is bad because it doesn't support the "friend's house" choice)
Good explanation: friend's house is where you can have fun by yourself. (this explanation is good because if someone believed it, they would pick "friend's house")

Figure B.2: The instruction shown on Amazon Mechanical Turk page for human rating collection on CQA.

a higher average rating) for these bad explanations. After filtering, we finally obtained 466 responses for CQA and 436 responses for SNLI.

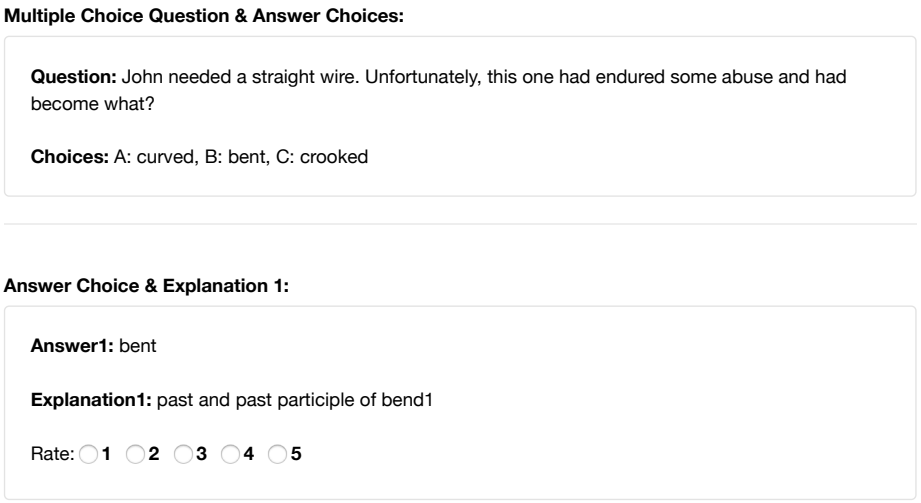


Figure B.3: A part of the questions for human rating collection on CQA.

Input, Output, and Explanation	Model		Human	
	Leaking?	LAS	Leaking?	LAS
Question: Marathoners feel fatigued after running twenty six miles, but some that have pushed them self too hard might be prone to what? Choices: A. passing out; B. death; **C. exhaustion** STRA explanation: if you are running too hard, you are likely to be exhausted.	Yes	1	Yes	1
Question: Where is likely to not just have a kosher restaurant? Choices: **A. new york city**; B. jewish neighborhoods; C. jerusalem HUMAN explanation: kosher restaurant is not in new york city.	Yes	0	No	0
Question: When are people buying products more? Choices: **A. economic boom**; B. disagreements; C. being able to use HUMAN explanation: being able to use.	No	-1	No	-1
Question: John bought a new water hose. But he found his old one near his car. Where did he find the old one? Choices: A. garden shed; B. hardware store; **C. garage** STRA explanation: garage is the only place where you can find old water hoses.	Yes	1	Yes	0
Premise: A man of the cloth puts a black substance on a man 's forehead. Hypothesis: The men are at church. Choices: A. entailment; **B. neutral**; C. contradiction HUMAN explanation: You can not infer they are at church .	Yes	1	Yes	1
Premise: One tan girl with a wool hat is running and leaning over an object , while another person in a wool hat is sitting on the ground. Hypothesis: A boy runs into a wall. Choices: A. entailment; B. neutral; **C. contradiction** STRA explanation: A girl is not a boy.	Yes	0	Yes	0
Premise: A man dressed in a light blue shirt dumping items from a bin into another bin , while standing in a room full of food donations. Hypothesis: Foods are not stored in room by a man. Choices: A. entailment; B. neutral; **C. contradiction** STRA explanation: Food donations are not stored.	Yes	-1	Yes	-1
Premise: Taking a break to watch some TV Hypothesis: Taking a neverending break Choices: A. entailment; B. neutral; **C. contradiction** HUMAN explanation: Some TV is not enough to be on a neverending break.	No	-1	No	0

Table B.1: Example data points from both CQA and SNLI with HUMAN or STRA label (**bold** in text) and explanation. Leakage predictions and example-level LAS scores from both model-based (T5) and human simulators are given.

	SNLI		CQA
Method	Dev Acc	Test Acc	Dev Acc
T5-Base	88.58	88.14 (.63)	68.84 (2.95)
MT-Re	88.91	88.44 (.62)	69.26 (2.93)
MT-Ra	88.95	87.98 (.63)	68.95 (2.94)
ST-Re	87.67	87.67 (.64)	66.74 (3.00)
ST-Ra	87.69	87.69 (.64)	68.84 (2.95)
Multi-Agent			
MT-Re (SGD)	88.24	87.94 (.64)	68.00 (2.97)
MT-Ra (SGD)	88.04	87.68 (.64)	65.58 (3.02)
MT-Re (RL)	88.31	87.91 (.64)	68.31 (2.96)
MT-Ra (RL)	87.99	87.72 (.65)	67.47 (2.98)

Table B.2: Model accuracies for the CQA and SNLI tasks. Generative models perform as well as non-generative baselines. CQA results are for dev data and SNLI are dfor test.

	Dev. SNLI			Test CQA		
Explanations	LAS Score (CI)	Acc($\hat{y} \mid x, \hat{e}$)	BLEU	LAS Score (CI)	Acc($\hat{y} \mid x, \hat{e}$)	BLEU
Human	4.36 (2.10)	98.40	-	-	-	-
MT-Re	-14.08 (1.78)	94.05	-	-5.40 (3.73)	80.00	-
MT-Ra	2.70 (8.59)	99.92	-	2.25 (4.60)	91.91	-
ST-Re	1.52 (0.90)	94.44	-	2.78 (2.10)	82.23	-
ST-Ra	7.26 (3.20)	99.90	-	10.33 (3.34)	86.70	-
Multi-Agent						
MT-Re (SGD)	-9.56 (1.64)	94.44	-	-2.16 (3.56)	77.23	-
MT-Ra (SGD)	5.06 (5.97)	99.90	-	4.53 (3.51)	84.79	-
MT-Re (RL)	-12.08 (1.51)	93.52	-	-6.55 (3.38)	80.95	-
MT-Ra (RL)	-0.52 (0.45)	93.18	-	-9.59 (2.93)	70.31	-

Table B.3: Evaluations of human and model-generated explanations by LAS score, overall simulator accuracy, and BLEU. We show the opposite data split relative to the main paper, for reproducibility. 95% confidence intervals as calculated by bootstrap are shown in parentheses. Confidence intervals are wider when the nonleaking subset is very small, and smaller when leaking and nonleaking subsets are both large.

Method	Seed		
	Seed 1	Seed 2	Seed 3
SNLI			
Human	4.31	1.68	5.34
MT-Re	-15.83	-5.55	-4.66
MT-Ra	4.34	2.12	2.21
ST-Re	0.55	1.19	1.35
ST-Ra	6.74	4.93	5.14
CQA			
Human	14.73	15.46	16.16
MT-Re	-7.07	-5.38	-3.53
MT-Ra	-1.31	0.32	6.33
ST-Re	3.76	1.82	2.46
ST-Ra	10.32	7.24	13.43

Table B.4: We check LAS scores across three random seeds, since random seeds tend to have a large influence on all statistics derived from pretrained neural language models [45]. Seed 1 is the result reported in the main body. We test two additional seeds for our primary experiments, retraining all models involved in the LAS score (including task model, simulator, and ST generators).

Method	Model	
	T5-Base	RoBERTa-Large
Human	4.31 (1.97)	-1.09 (2.69)
ST-Re	0.55 (0.87)	-0.44 (0.95)
ST-Ra	6.74 (4.53)	4.74 (9.68)

Table B.5: LAS score comparison between T5-Base and RoBERTa-Large models with SNLI data (95% confidence intervals obtained by bootstrap). For ST models, the task model and simulator are of the same architecture. RoBERTa produces lower LAS scores than T5, and their rank ordering is not necessarily the same. The differences between them could result from their pretraining procedures, architectural differences, finetuning sample efficiency, or another cause.

Method	SNLI	
	Dev. Acc (CI)	Test Acc (CI)
T5-Base	91.31 (.56)	91.01 (.57)
MT-Re	91.62 (.55)	91.14 (.56)
MT-Ra	91.56 (.55)	91.20 (.56)

Table B.6: NLI results using the full training dataset. Generative models of explanations can maintain task accuracy.

LAS	Human		
Model	-1	0	1
-1	0.271	0.659	0.071
0	0.082	0.781	0.138
1	0.031	0.654	0.315

Table B.7: Row-normalized contingency table between model-based and human variables resulting from the expert simulation analysis. Model scores of -1 and 1 tend to shrink toward human ratings of 0.

	e-SNLI	CQA
Method	Test Acc (CI)	Dev Acc (CI)
BERT-Base	87.01 (0.66)	67.89 (2.97)
ST-Re-Enc	85.67 (0.69)	63.16 (3.07)
ST-Ra-Enc	85.62 (0.69)	64.84 (3.04)
MT-Re-Enc	87.25 (0.66)	70.74 (2.89)
MT-Ra-Enc	87.23 (0.66)	69.79 (2.92)
T5-Base	88.14 (0.63)	68.84 (2.95)
MT-Re-MLM	88.26 (0.63)	69.05 (2.94)
MT-Ra-MLM	88.43 (0.63)	70.11 (2.91)

Table B.8: Task results table with alternative computational models and language modeling objectives.

C Additional Results and Details for Chapter 6

C.1 Additional Experiments

We give additional experimental results with our synthetic dataset in an extended technical report on this topic, available here: https://arxiv.org/abs/2102.02201. Additional experiments are conducted to answer a research questions including:

1. Can explanations help models learn to use strong (causal, generalizable) features rather than weak ones?
2. What is the best way to compute explanation representations for prediction?
3. Can models aggregate information across several retrieved explanations?
4. What makes an explanation relevant across data points? What enables a retrieval model to find relevant explanations for a new data point?
5. How does the co-dependence between classifier and retrieval model influence the viability of joint training?
6. Does retrieval of explanations improve model performance on existing natural language datasets?

C.2 Our Model for Initial Experiments

Here, we introduce our chosen model for incorporating explanation data, which makes use of explanations as *model inputs* after they are retrieved from the training data (the "Retrieval" graphical model in Fig. 6.2). Our approach is similar to Lewis et al. [112], who marginalize over latent documents retrieved from Wikipedia for question answering, question generation, and fact verification. The marginal distribution is given as:

$$p_\Theta(y|x) = \sum_{e \in \text{top-}k(p_\eta(\cdot|x))} p_\theta(y|x,e) p_\eta(e|x)$$

where top-k gets the top k texts as ranked by the retrieval model, $p_\eta(e|x)$. Note that we never retrieve a data point's own explanation when predicting its label. We do so because explanations can leak the label [68] and this approach matches the test-time distribution, where we assume explanations are not collected for new data points (see discussion in Sec. 6.2).

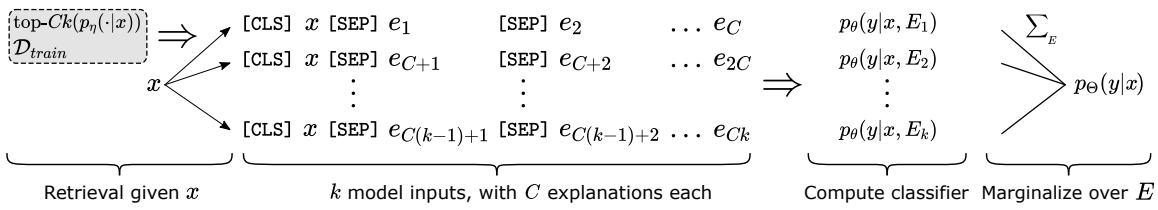


Figure C.1: A depiction of our retrieval-based method TextCat. A total of Ck explanations are retrieved and allocated into k latent variables, each a set of explanations E, which are marginalized over to produce a final prediction.

Zhou et al. [238] also propose to use explanations as latent variables and retrieve explanations at inference time, but they do not learn the retrieval model, marginalize over the latents during inference, or prohibit data point's own explanations from being retrieved. In our experiments, we compare with their original approach and a version where we marginalize over the latents and learn the retrieval model.

The form of $p_\eta(e|x)$ follows Lewis et al. [112] and Karpukhin et al. [95]. Given a query x, unnormalized probabilities are computed as:

$$p_\eta(e|x) \propto \exp\left(f_\eta(e)^T f_\eta(x)\right)$$

where f_η embeds each sequence into a vector. To compute top-$k(p_\eta(\cdot|x))$, we search through the training explanations using FAISS [93]. We discuss methods for computing $p_\theta(y|x,e)$ and $f_\eta(e|x)$ in Sec. C.2.1. Because it may be helpful to reason over multiple explanations at once, we extend this model to allow for explanations to be composed into a single "document." Assuming explanations to be conditionally independent given x, we can compute the probability of a set of explanations $E = \{e_c\}_{c=1}^C$ as

$$p(E|x) \propto \exp\left(\sum_{e \in E} f_\eta(e)^T f_\eta(x)\right),$$

where (1) a *context size* C will control the size of the explanation set, (2) a value of k implies that the top Ck will be retrieved, and (3) we sort these Ck explanations into sets in order of their probability $p_\eta(e|x)$.

We represent the overall approach in Fig. C.1 for one method of computing $p_\theta(y|x,E)$ (described fully in Sec. C.2.1), where explanations are concatenated with the query sequence. Flowing from left to right, Fig. C.1 shows how explanations are retrieved from the training data conditioned on a query sequence x, then allocated into k classifier inputs with C explanations each. The k classifier predictions are aggregated by marginalizing over the latent variable, $Z = E$.

Modeling Assumptions. In using retrieval, we make a few assumptions. First, since the number of forward passes per data point scales with k, we require a relatively small value of k, i.e. $k \leq 10$, for reasonable computational efficiency in SGD-based training. Hence, we must assume that this summation is sufficiently similar to the full summation over latent variables. This assumption is more likely to hold when (1) a small number of documents account for most of the probability mass in $p_\eta(e|x)$, and (2) a pretrained model $p_\eta(e|x)$ yields a decent initial rank-ordering, such that some of the best documents are in the top-k. The exact value of k we use depends on the experiment. A second, more basic assumption is that explanations will be useful in predicting other data points' labels. Such an assumption is needed since we never condition on a data point's own explanation. Lastly, during retrieval we assume that explanations are independent given x, i.e. $p(E|x) = \prod_{e \in E} p(e|x)$. This could be a poor assumption when, for instance, explanations each contribute one of a number of needed facts, in which case it would be helpful to retrieve additional explanations conditioned on what has already been retrieved.

C.2.1 Conditioning Mechanisms

In this section we describe the methods used to compute $p_\theta(y|x,E)$ and $p_\eta(e|x)$ (see Sec. C.2 for the overall model description). For the classifier $p_\theta(y|x,E)$, we use two methods, TextCat

and H-MEAN, which are described below. Then we describe the retrieval model, which is based on Sentence-BERT [164].

TextCat. Represented in Figure C.1, this method takes a straightforward approach to conditioning on a set of explanations: concatenating C explanations and the input x to form a longer sequence of text. Each of the original sequences is separated by a special token, e.g. `[SEP]` for BERT. In our experiments, we pass this longer sequence into a RoBERTa-base model. After pooling the output token representations, we pass the resulting vector to a 1-layer MLP for classification. We use mean pooling for our synthetic task and NLI; for relation extraction tasks, we concatenate the representations corresponding to the initial tokens in the *subject* and *object* words, since this is an especially effective pooling technique [8].

This approach allows the model to reason over all of the explanations and the input together. While the method may be limited by the fact that some models can face difficulties in processing long pieces of text [17], this issue is partly mitigated by marginalizing over k sets of explanations. As a result of the marginalization, the final prediction can be conditioned on a far higher number (Ck) of individual explanations than could fit in the context alone.

H-Mean. By H-MEAN, we refer to the kind of unweighted hidden representation averaging used in Co-Reyes et al. [32] and Zhou et al. [238]. H-MEAN works by first obtaining representations of the input x and a single explanation e at a time, then passing the unweighted average of these representations to an MLP. For a fair comparison with TEXTCAT, we use the same token pooling and a 1-layer MLP. So with C explanations to condition on, $x' = concatenate(x, e)$, and vector representations from RoBERTa(x'), H-MEAN obtains a single representation as

$$h = \frac{1}{C} \sum_{c=1}^{C} \text{RoBERTa}(x')$$

which is then passed to the MLP for classification. H-MEAN does not face the same sequence length limitations as TEXTCAT, but by separately processing of each explanations H-MEAN may fail to integrate information across explanations. This method also becomes expensive when we marginalize over E (which is what allows retrieval to be learned), as it requires Ck forward passes for a single prediction.

C.2.2 Retrieval

We use a similar approach to retrieval as in Lewis et al. [112], namely using vector representations of sequences from a pretrained transformer to compute

$$p_\eta(e|x) \propto \exp\left(f_\eta(e)^T f_\eta(x)\right),$$

which is followed by computing top-$Ck(p_\eta(\cdot|x)$. We use an approximate but sub-linear time search method (FAISS) to find the top-Ck points [93]. In our experiments we find that it is necessary to use Sentence-BERT [164] as our pretrained f_η, rather than simply a pretrained RoBERTa model. Sentence-BERT is a network trained to produce semantic representations of sentences that can be compared under cosine similarity. In our experiments, we use the Sentence-RoBERTa-base model trained on a combination of several NLI and semantic textual similarity tasks, with mean pooling of token representations. We normalize the representations we obtain from this model, so that our inner product is equivalent to a cosine similarity.

Note that during training, we never condition on a data point's own explanation when predicting its label. This is an important constraint for matching the train and test-time distributions. At test time, we assume we have access only to past (training) explanations, since they can be expensive to collect and conditioning on explanations at test time can lead to label leakage, meaning what is essentially the benefit of human labeling could be mistaken as improvements in model performance.

C.3 Training Details

C.3.1 Runtimes.

Regarding training times, we run most experiments on a single NVIDIA RTX 2080 GPU, with runtimes as follows: 4.0 hours for 40 epochs of the no-retrieval RoBERTa-base using the synthetic dataset; 5.7 hours for 40 epochs of RoBERTa-large in the same setting; 8.6 hours for 20 epochs of learned retrieval with RoBERTa-base models on synthetic data.

C.3.2 Training Hyperparameters and Analysis

For optimization, we use AdamW with a learning rate of 1e−5 and gradient norm clipping at norm 1. For the LR, we use a linear warmup and decay schedule peaking at 10% of the training steps for experiments with synthetic data and at 1% for experiments with existing datasets (given the larger training set sizes). The batch size is set to 10 across all experiments.

We decide how often to rebuild the representations of training explanations while learning the retrieval model by tuning across frequency values in the range {10%, 20%, 33%, 50%, 100%} (i.e. to rebuild at this percentage of every epoch), as well as never rebuilding. In our synthetic setting, the only noticeable drop in performance comes from never rebuilding. As long as representations are re-encoded at least as often as every epoch, we notice no difference in final test accuracy, though in early experiments we observed that rebuilding more often improved training stability. To err on the safe side of training stability, we re-encode the representations every 20% of each epoch in all experiments except e-SNLI with full data, where we re-encode every 30% of each epoch.

Additionally, we use the stop-gradient function when computing the gradient of $p_\eta(e|x)$ as follows:

$$\nabla_\eta \exp\left(\text{sg}[f_\eta(e)]^T f_\eta(x)\right),$$

meaning that we do not differentiate through the explanation embeddings, but only through the query data point embeddings. In early experiments, we found that this decision contributed to training stability, while improving computational efficiency, and we confirm that we observe no differences in model accuracy as a result.

C.3.3 Experiment Confidence Intervals

We compute confidence intervals for our synthetic data tasks to represent *seed variance* around some mean seed performance. We represent seed variance in figures rather than sample variance because the sample variance is fairly low with 50,000 test points and could be driven arbitrarily low with more generated test points. For instance, the 95% confidence interval for a model accuracy of 90% would be ±0.26. To calculate seed variance, we run 10 random seeds for our baseline condition (no-retrieval) with the default synthetic task setup.

C.4 Synthetic Task Generative Process

The required parameters to the data generation include: (1) a training sample size *sample-size* and (2) *num-tasks*, the number of unique integer pairs to be counted, or, equivalently, the number of points per *index*, n_{task}. In all experiments, we use a maximum integer value of 100 to appear in the sequences, and a maximum *index* value of 10,000. We give the general generative process below. Note that the dev and test sets are constructed with the extra constraint that sequences must not appear in the training data. Further note that this is the generic version of generative process, and in some experiments the process is altered. For example, in RQ3, *indicator* is always 1 and the construction of the map from *index* values to (m, n) tuples occurs in a special way described in the experimental design for RQ3.

1. Sample $\{index_t\}_{\tau=1}^{num\text{-}tasks}$ from the uniform distribution over integers {1,...,10000} without replacement.
2. Sample $\{(m, n, r, d)_t\}_{\tau=1}^{num\text{-}tasks}$ from the uniform distribution over integers, $unif([1, 100]^4)$, without replacement and requiring that $m \neq n \neq r \neq d$.
3. Define the set $\{(index, m, n, r, d)_{index})\}$ for *index* and (m, n, r, d) drawn from their respective sets, without replacement, in an arbitrary order.
4. Compute the number of points per *index*, $n_{task} = sample\text{-}size \;//\; num\text{-}tasks$.
5. For each $index \in \{index_t\}_{\tau=1}^{num\text{-}tasks}$:
   (a) Sample a vector of length n_{task}, balanced between 1s and 2s, that gives the values of $\{indicator_p\}_{p=1}^{P}$ for the P points with that *index*.
   (b) Sample a vector of length n_{task}, balanced between 0s and 1s, representing whether the features $\mathbb{1}[\#m{>}\#n]$ and $\mathbb{1}[\#r{>}\#d]$ should correlate (1 implies they are equal, and 0 unequal). This balance changes when the strong-weak correlation is intended to change.
   (c) Sample a vector of length n_{task}, balanced between 0s and 1s, representing whether (m, n) or (r, d) should be the more *numerous* integers in the sequence (so that there is no bias, even randomly, between features by size).
   (d) For $i \in 1 : n_{task}$:
      i. Place the *index* in the first element of an empty array, and the *indicator* in the second.
      ii. Based on the i^{th} elements of the three vectors described above, allocate samples of the integers in $(m, n, r, d)_{index}$ into the remaining 18 slots.
      iii. If there are any remaining slots after these integers are randomly allocated, fill them with i.i.d. samples from $unif(1, 100)$.

D Additional Results and Details for Chapter 7

D.1 Method Implementation and Hyperparameter Tuning Details

D.1.1 `Replace` Functions

1. **Attention Mask.** To make this a differentiable function, we compute the function by taking the element-wise product between the attention distribution and the binary attention mask, then renormalizing the attention probabilities to sum to 1. The difference between this approach and deleting a token from an input text is that positional embeddings for retained tokens are unchanged.
2. **Marginalize.** As in Kim et al. [101], we use a pretrained BERT-Base as our generative model (or a RoBERTa-Base model, when the classifier is a RoBERTa model). The final prediction is obtained from the marginal log probability as $\arg\max_y \ln \sum_{\tilde{x} \sim p_\phi(\tilde{x}|x,e)} p_\theta(y|\tilde{x}) p_\phi(\tilde{x}|x,e)$, where $p_\phi(\tilde{x}|x,e)$ is the distribution over imputed tokens. Since computing this marginal distribution is quite expensive, we adopt a Monte Carlo approximation common to past work [101, 224]. Using a subset of SNLI validation data, we tune the number of samples over sizes in {10, 25, 50, 100}, selecting for maximum robustness. Surprisingly, 10 samples performed the best in terms of robustness, though the margin was small over the other values. Consequently, we select a value of 10, which also allows us to evaluate this method at scale due to its relative computational efficiency. This finding is similar to the results in Yi et al. [224], who ultimately use a value of 8 samples for Monte Carlo estimation of the marginal distribution. This method is still over ten times slower than other `Replace` functions given the need to perform many MLM forward passes.
3. **MASK Token**. Described in main paper.
4. **Slice Out**. Desribed in main paper.
5. **Zero Vector**. Described in main paper.

D.1.2 Explanation Methods

LIME. For a data point x, we train a model m_ϕ minimizing an MSE weighted by the kernel π and regularized by Ω,

$$\sum_{i=1}^{N} \pi(x, \tilde{x}_i)(m_\phi(\tilde{x}_i) - f_\theta(\tilde{x}_i)_{\hat{y}})^2 + \Omega(\phi)$$

where $f_\theta(x)_{\hat{y}}$ is the task model's predicted probability, local samples $\tilde{x}_i$ have attention masks that are imputed with a random number of 0s, and Ω is the default `auto` regularization in the LIME package.

We next specify the form of the weight function π, the regularization method Ω, and the distribution of perturbed data points $p(\tilde{x}|x_i)$, which are all set to the default LIME package settings. The weight function π is an exponential kernel on the negative cosine similarity between data points multiplied by 100. The perturbation distribution is over binary vectors:

in every sample, a uniformly random number of randomly located elements are set to 0, and the remainder are kept as 1. Lastly, Ω is to perform forward selection when there are no more than 6 features (i.e. perform greedy local search in the space of possible feature sets, starting with no features and adding one feature at a time). When there are more than six features, ridge regression is used, then the top k features according to the product of their feature weight and the observed feature value (0 or 1 in our case). We use the regression weights as the final salience scores.

Vanilla Gradient. We obtain model gradients w.r.t. the model input as salience scores, one early method for explanation [113]. We compute the gradient of the predicted probability w.r.t. input token embeddings, and we obtain a single value per token by summing along the embedding dimension.

Integrated Gradients. The salience for an input x with baseline $\tilde{x}$ is given as

$$(x - \tilde{x}) \times \int_{\alpha=0}^{1} \frac{\partial f(\tilde{x} + \alpha(x - \tilde{x}))}{\partial x} d\alpha.$$

We use the input embeddings of a sequence as x. By the Completeness property of IG, token-level salience scores still sum to the difference in predicted probabilities between the observed input and the baseline.

Random Search. Using a subset of SNLI validation points, we tune this method over two possible search spaces: the space of all k-sparse explanations, when the sparsity levels allows up to k tokens to be retained (or no lower than k tokens, for Comprehensiveness), and the space of all allowably sparse explanations. We find it preferable to restrict the search space to exactly k-sparse explanations. We adopt this same search space for all other search methods.

Anchors. We use the `anchor-exp` package made available by Ribeiro et al. [167] for our experiments, with two modifications. First, we limit the compute budget used in this method to 1000 forward passes (as with all search methods). Second, thoughwe sample locally around inputs using the default argument `masking_str='UNKWORDZ'`, we use the Attention Mask `Replace` function for computing the model forward pass, as we do with all search methods. Besides this, we call `explain_instance` with default parameters, and we refer the reader to Ribeiro et al. [167] for additional details. Note that we distinguish results on Sufficiency and Comprehensiveness in terms of the maximum number of features selected by the explanation. Additionally, this method has over a 3x slower wall-clock runtime compared to our search methods used with the same compute budget (in terms of model forward passes), and as a result we are constrained to reporting results across a smaller number of model seeds for each dataset (between 3 and 10, rather than always 10).

Gradient Search. For an input of length L, this method sequentially updates a continuous state vector $s \in \mathbb{R}^L$ via gradient descent in order to minimize a regularized cross-entropy loss between the original model prediction $\hat{y}$ and the predicted probability given the input $\tilde{x} = \texttt{Replace}(x, e_t)$. The explanation e_t is sampled as follows: $e^{(d)} \sim \text{Gumbel-Softmax}(s^{(d)})$, for $d = 1 : L$. The new state is $s_{t+1} \leftarrow s_t - \alpha \nabla_s \mathcal{L}(\hat{y}, f(\tilde{x})_{\hat{y}})$, though note that we use an AdamW optimizer for this step. By virtue of the differentiable Attention Mask `Replace` function and the Gumbel-Softmax estimator [127, 89], this loss is differentiable w.r.t. s. The

regularizer is an ℓ_2 penalty on the difference between the expected sparsity $\sum_{d=1}^{L} \sigma(s^{(d)})$ and a target sparsity, set to ceiling$(.05 \cdot L)$, which is designed to encourage searching through sparse explanations. The final salience scores are given by s, with the probabilistic interpretation that $\sigma(s_j)$ is the probability that token j is in the optimal explanation.

We observe that this search method is equivalent to fitting a non-parametric model to the dataset with the objective $\mathcal{L}$ given above. Recently, many parametric models have been proposed for sampling explanations for individual data points [13, 9, 219, 145, 28, 37]. In early experiments, we found that a parametric model performed far worse than this non-parametric approach, and hence we leave out parametric models from further consideration. This is perhaps unsurprising given how hard it may be to learn a map from inputs to good explanations for all data.

Now we give more details to checkpoint selection, weight initialization, regularization, and tuning for Gradient Search. For checkpoint selection: we select the search state that achieves the best Sufficiency (or Comprehensiveness) as measured once every m gradient updates. We do so because checking these metrics consumes some of the available compute budget (see Supplement D.2.3), and therefore we check the metric value at intervals for purposes of checkpointing. In our experiments, we check the metric every 20 gradient updates and search until the total budget has been consumed. For initialization: a random initial starting point is sampled from a Multivariate Normal distribution centered around 0, with $\Sigma = I$. For regularization and other tuning details, we perform sequential line searches over hyperparameters, according to Sufficiency scores on a subset of BoolQ data points. To tune a specific hyperparameter, we set all other hyperparameters to some default values. We refer to the hyperparameters we use after tuning as "final" hyperparameters, which are listed in the table below (note: Number of Samples is the number of sampled explanations per gradient update).

Hyperparameter	Default	Final	Range
Number of Samples	10	1	1, 10, 20, 40
Optimizer	AdamW	AdamW	AdamW, SGD
Scheduler	None	None	None, Linear, Step, Cosine
Learning Rate	0.2	0.1	0.01, 0.05, 0.1, 0.2, 0.4
Sparsity Weight	1e-3	1e-3	1e-1, 5e-2, 1e-2, 5e-3, 1e-3, 5e-4, 0
Target Sparsity	0.1	0.05	0.03, 0.05, 0.1, 0.2, 0.3, 0.4

Taylor Search. At time step t, the state is an explanation e_t, and a heuristic is evaluated on neighboring states in order to select the next state to compute the objective on. The search space is all k-sparse explanations, and therefore neighboring states are those with Hamming distance 2 to the current state (with one retained token being hidden and one hidden token being retained). The heuristic is the projected change in the cross-entropy loss between the model's original prediction $\hat{y}$ and this label's probability given the input $\texttt{Replace}(x, e)$, for a proposed explanation e, which is computed as such: We first calculate the gradient $g \in \mathbb{R}^L$ of the cross-entropy loss with respect to the explanation e_t, which is possible with the differentiable Attention Mask `Replace` function. Then, when optimizing for Sufficiency, we find the two indices i and j as the solution to $\arg\max_{i,j}\ g^{(i)} - g^{(j)}$ such that the sparsity is maintained by flipping both tokens, meaning $e^{(i)} = 1$ ($x^{(i)}$ is retained) and $e^{(j)} = 0$ ($x^{(j)}$ is hidden). The next state is obtained by setting $e_t^{(i)}$ and $e_t^{(j)}$ to these values. This is a first-order approximation to the change in loss between the new and old state, based

on the Taylor expansion of the loss [49]. Note when optimizing for Comprehensiveness, we use the arg min. Following Ebrahimi et al. [49], we ultimately use this search heuristic within a beam search, starting from a random explanation, with width $w = 3$.

Hyperparameters for Taylor Search are listed below. We performed tuning with Taylor Search for Sufficiency on a subset of BoolQ validation points, and ultimately we selected the largest, best performing pair of values given the available compute budget.

Hyperparameter	Default	Final	Range
Beam Width	2	5	1,2,3,4,5
Number of Steps	50	50	50, 100, 200

Ordered Search. More complicated forms for f, such as being quadratic in e, would make finding the optimum of the function computationally intractable, let alone a full rank-ordering of solutions [11].

Using Sufficiency scores on a subset of SNLI validation data, we tune over the ratio between compute budget used in estimating the model f_θ (i.e. salience scores) and the budget used for the search. Out of 1000 steps, we consider using up to m steps for estimating the salience scores via LIME, where $m \in \{10, 100, 200, 250, 500, 750\}$, ultimately using $m = 250$.

Parallel Local Search. Here we specify the `Propose` function used in Parallel Local Search, and we describe some additional implementation details, some comparisons we performed with Simulated Annealing, and tuning for the number of parallel searches r. The `Propose` function samples new explanations by starting a random walk from the current explanation that ends when a not-before-seen explanation is encountered. As in Taylor Search, neighboring states have Hamming distance 2. Though we parallelize this search method, we maintain a shared set of previously-seen explanations and compute the `Propose` function serially at each step so that we never compute the more expensive objective function on the same explanations.

A similar algorithm, Simulated Annealing, uses a probabilistic update condition that favors exploration early on in the search and exploitation later in the search [11]. We find it preferable to use a deterministic update rule, moving to the new state if and only if its objective value is better than the old state. Lastly, following tuning results, we use $r = 10$ parallel runs for all experiments, meaning that each run has a budget $b = 100$ when the overall method budget is 1000 forward passes. The value of r is tuned over the set {1, 5, 10, 25}. We note that using a value greater than 1 significantly improves the wall-clock runtime of this algorithm, as the batched forward passes performed when r searches are done in parallel are much more efficient than performing a greater number of forward passes with only 1 input.

D.1.3 Model Training Details and Experiment Runtimes

We now give implementation details for training models on our six datasets. The models include BERT-Base or RoBERTa-Base models drawn from the Hugging Face Transformers library [221], trained with AdamW [124] using a learning rate of 1e-5, and in general we select the best model training checkpoint according to the validation set accuracy. When training models for our analysis of counterfactual OOD-ness in Sec. 7.5, we train Standard models for 20 epochs and Counterfactual-Trained models for 10 epochs, since in the latter case we effectively double the number of inputs per batch. For our explanation evaluation experiments

in Sec. 7.6, we train all models for 10 epochs. Note that in every experiment we train ten models using ten different random seeds.

All experiments are conducted on a NVIDIA RTX 2080 GPU. Wall-clock runtimes for training models are: for Sec. 7.5 experiments, .6 hours per SNLI model and 1 hour per SST-2 model; for Sec. 7.6, training one model takes 4.8 hours for FEVER, .7 hours for BoolQ, 1.5 hours for SNLI, 2.9 hours for MultiRC, 1.7 for Evidence Inference, and 3.9 hours for SST-2.

For analysis and explanation evaluation experiments, we report the following runtimes: Sec. 7.5 experiments take up to 12 hours for robustness analysis for each `Replace` function (across seeds, models, and datasets), except for Marginalize which takes close to 48 hours. We give max runtimes across datasets for obtaining and evaluating explanations to provide an upper bound on wall-clock runtime given the variable sequence lengths between datasets, meaning we report runtimes for the Evidence Inference dataset. With a compute budget of 1000 forward/backward passes, we find observe that obtaining a set of explanations at one sparsity level using 500 points for one BERT-Base model takes 2.5 hours for LIME, 2.5 minutes for Vanilla Gradient, 11 hours for Integrated Gradients, 10 hours for Gradient Search, 17 hours for Anchors, 5 hours for Taylor Search, 3 hours for Ordered Search, 5 hours for Random Search, and 5 hours for Parallel Local Search.

D.2 Experimental Details

D.2.1 Data Preprocessing

We use datasets as prepared by the ERASER dataset suite [42], which are publicly available and distributed under the original licenses for each dataset (including Creative Commons and MIT License),[1] as well as SST-2 which is publicly available under a Creative Commons license [191].[2] Note that BoolQ has only one split for evaluation. Additionally, note that for experiments in Sec. 7.6, we use a subset of 50k points for training models on SNLI, and we use the 500 shortest SST-2 test points according to the number of tokens given by the BERT tokenizer [220] in order to compare with exhaustive search for this dataset.

For preprocessing, input text is split by spaces into a list of words. We tokenize each word independently and concatenate the resulting tokens. Each input consists of a document section and a query section. The document section contains the document while the query section contains the question. The two sections are separated by [SEP]. The entire input is preceded by a [CLS] token and followed by a [SEP] token. For inputs longer than 512 tokens (the maximum input length for our task model Bert), we truncate the input document so that the entire input is shorter or equal to 512 tokens.

D.2.2 Analysis of Counterfactual Input OOD-ness Details

In this evaluation, the tokens to be hidden from the model are selected uniformly at random without replacement. We sample 10 random masks per data point and take the majority prediction on the corresponding 10 ablated data points as the model's overall prediction. The exception to this procedure is Marginalize, since it is much more computationally expensive than other methods, and therefore we use only one random explanation per input. Note that we use a subset of 10k train points for each task, and we perform this experiment on validation splits because the experiment motivates method design choices in Sec. 7.6.1.

[1] http://www.eraserbenchmark.com/

[2] https://www.kaggle.com/atulanandjha/stanford-sentiment-treebank-v2-sst2

Table D.1: Dataset statistics

Dataset	# Classes	Split	Size	Avg. document length	Avg. query length
SNLI	3	Train	5000	24.8	-
		Validation	9823	24.4	-
		Test	9807	24.4	-
BoolQ	2	Train	9427	121.9	9.4
		Validation	3270	119.8	9.3
FEVER	2	Train	97957	342.3	10.4
		Validation	6122	291.2	10.7
		Test	6111	278.7	10.7
Evidence Inference	3	Train	7958	483.1	25.3
		Validation	972	484.4	23.6
		Test	959	482.0	27.0
SST-2 2	2	Train	67349	11.3	-
		Validation	872	23.2	-
		Test	1821	10.8*	-
MultiRC 2	2	Train	24029	326.9	21.2
		Validation	3214	326.1	20.7
		Test	4848	314.8	20.8

Table D.2: Model accuracies

Dataset	Standard Acc.	CT Acc.
SNLI	85.84 (0.69)	85.08 (0.71)
BoolQ	74.16 (1.62)	73.76 (1.62)
FEVER	89.66 (0.76)	89.72 (0.76)
Evidence Inference	58.81 (3.12)	57.35 (3.13)
SST-2	92.89 (1.18)	92.43 (1.21)
MultiRC	68.96 (1.30)	67.76 (1.32)

D.2.3 Compute Budget Details

In this section we describe how the compute budget is spent by each explanation method. Note that we consider both creating and evaluating explanations to draw from the available compute budget, because some methods compute the Suff and Comp metrics while obtaining an explanation, whereas others leave these metrics to be checked after an explanation is settled on. We describe the standard case in this chapter of 1000 forward and backward passes per final metric value.

1. LIME uses 996 forward passes to obtain an explanation, then 4 forward passes to obtain a final metric value (one per sparsity level).
2. Vanilla Gradient uses only a single forward and backward passes. This is our only method that uses a fixed compute budget.
3. Integrated Gradients uses 498 forward and backward passes and 4 forward passes to obtain the final metric value.
4. Taylor Search uses no more than 1000 forward passes, given the beam width and number of steps described in Supplement D.1.
5. The remaining search methods (including Ordered Search, Random Search, Parallel Local Search) all use 1000 forward passes in total, since these methods involve exactly computing the objective value at each step, so the metrics do not need to be recomputed after explanations are obtained.

D.3 Additional Results

SST-2 Results. Here we discuss results for SST-2, which are shown in Table D.3. Our primary observation here is that most of the search methods we consider perform as well as Exhaustive Search (for sequences short enough to exhaustively search). The closest to Exhaustive Search is Parallel Local Search, which exactly identifies the optimal explanation for the Sufficiency metric and comes within .01 of the optimal Comprehensiveness value. Meanwhile, the best salience method (LIME) underperforms these search methods by between 2.86 and 3.86 points, showing that salience methods fall well behind search methods in this scenario.

Search Method Performance Over Time. In Figure D.1, we show search performance across time steps for the three best performing search methods, Random, Ordered, and Parallel Local. Note that Ordered Search begins at step 251 since the first 250 forward passes are allocated to computing LIME, and Parallel Local Search begins at step 10 since we use 10

Table D.3: Explanation metrics for SST-2

Dataset	Method	Sufficiency ↓		Comprehensiveness ↑	
		Standard Model	CT Model	Standard Model	CT Model
SST-2	LIME	1.98 (0.84)	5.92 (0.93)	52.42 (2.92)	45.75 (2.49)
	Anchors	3.44 (0.96)	17.69 (1.64)	30.03 (3.13)	24.19 (2.54)
	Taylor Search	0.09 (0.50)	5.02 (0.79)	45.65 (3.11)	38.91 (2.70)
	Ordered Search	-0.91 (0.47)	2.69 (0.79)	56.24 (2.82)	49.21 (2.48)
	Random Search	-0.91 (0.48)	2.70 (0.79)	56.11 (2.85)	48.98 (2.49)
	PLS	-0.91 (0.51)	2.68 (0.85)	56.28 (2.84)	49.25 (2.53)
	Exhaustive Search	-0.91 (0.51)	2.68 (0.85)	56.29 (2.84)	49.26 (2.53)

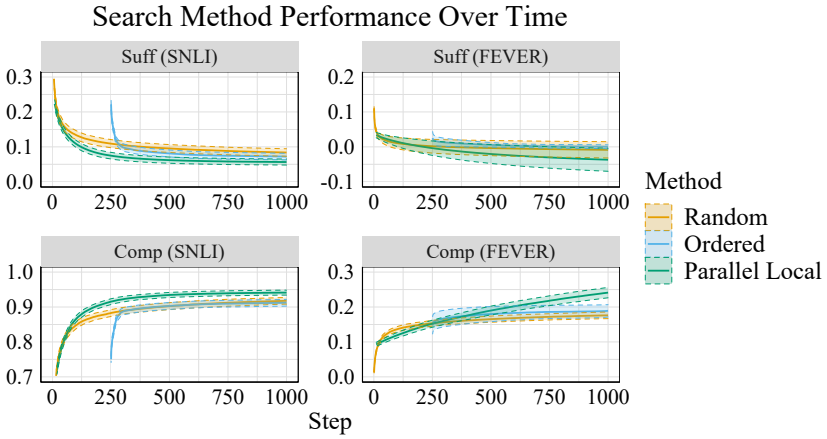


Figure D.1: Search method performance over time on FEVER and SNLI with Counterfactual-Trained models, for searches that ran for 1000 forward passes. Shaded areas represent 95% confidence intervals accounting for seed variance using 10 random seeds.

parallel runs. We see that Parallel Local outperforms Random early on and then continues to remain the preferable method, as differences at step 1000 are statistically significant at $p < .05$. In fact, for FEVER, where the search space is larger, Parallel Local will clearly continue to improve with additional compute, while Random and Ordered plateau in performance by 1000 steps.

Each of these search methods tends to achieve good performance even by 250 steps. We report results using this **reduced search budget** across datasets in Table D.4 in the Supplement. We find that search methods outperform salience methods in 15 of 24 comparisons (with 8 favoring LIME), suggesting that search methods can outperform salience methods even with a far smaller compute budget. This is relevant if, for instance, one needs to obtain explanations at multiple sparsity levels with a single compute budget for all of the explanations, which could be useful for applications requiring all explanations to be visualized at once (or visualized on demand with no latency).

Counterfactual-Trained Model Accuracy. In Table D.2, we show model accuracies on test sets for standardly trained models (Standard) and Counterfactual-Trained (CT) models, with 95% confidence intervals reflecting sample variance in parentheses. These accuracies are from the single best performing seed for each dataset, according to dev set accuracies,

out of 10 trained models. Note that for SST-2, we report dev set numbers as test labels are unavailable.

We observe that differences in accuracies are typically less than 1 point between the two training algorithms. On average across the datasets, the difference is about 0.7 points. This is a small but consistent gap, and hence it will be valuable for future work to study the causes of this gap and identify ways to align the train and explanation-time distributions without losing any model accuracy on test data.

Results with Reduced Search Budget. In Table D.4, we give results for a reduced search budget of 1000 forward passes *across* sparsity levels, i.e. 250 per sparsity level. This setup favors salience-based methods which can easily give explanations at varying sparsity levels. With too many sparsity levels, **this evaluation is heavily biased toward salience-based methods**, since it spreads the compute for search methods across sparsity levels. We use a constant budget per sparsity for results in the main paper because we view the use of multiple sparsity levels as an attempt to average results across possible desired settings, and explanations at multiple sparsities may not always be needed. But ultimately, user preferences will dictate whether explanations at multiple levels of sparsity are desired. This discussion aside, the results are as follows: compared to the best salience-based method, LIME, **search methods are preferable in 15 of 24 comparisons**, while **LIME is preferable in 8 of 24 comparisons** (at statistical significance of $p < .05$). Comparing within search methods, PLS is preferable to Random Search 7 times and Random Search is preferable one time (at $p < .05$). We observe that LIME performs best on Comprehensiveness, and therefore we suggest that LIME may be the best method when explanations are desired at many sparsity levels, the compute budget is heavily limited, and one is optimizing for Comprehensiveness. Otherwise, PLS remains the preferable method.

Results with RoBERTa Models. Shown in Table D.5, we give a comparison between LIME, Random Search, and PLS using RoBERTa-Base as our task model, as opposed to the BERT-Base setting in the main paper experiments. Results are given for a subset of datasets, with the same compute budget as in the main paper, using 10 Counterfactual-Trained RoBERTa-Base models on each dataset. We observe similar trends and improvements using PLS as with BERT-Base. PLS is the best method in each condition and consistently outperforms Random Search, unlike LIME.

Weight of Evidence Metrics. Note that we can also measure the Sufficiency and Comprehensiveness metrics in terms of a difference in log-odds (i.e. weight of evidence) rather than probabilities, which reflects differences in evidence for classes before this evidence is compressed to the bounded probability scale [169, 2]. In this case, Sufficiency e.g. is computed as

$$\text{Suff}_{\text{WoE}}(f_\theta, x, e) = \ln \frac{f_\theta(x)_{\hat{y}}}{f_\theta(x)_{\neg\hat{y}}} - \ln \frac{f_\theta(\texttt{Replace}(x, e))_{\hat{y}}}{f_\theta(\texttt{Replace}(x, e))_{\neg\hat{y}}} \tag{D.1}$$

where $f_\theta(x)_{\neg\hat{y}}$ is the sum of all class probabilities except the predicted class, meaning each term is the log-odds in favor of $\hat{y}$. In Sec. 7.6 we describe results for the standard difference-in-probabilities version of each metric as well as the weight-of-evidence versions.

In Table D.6, we compare the default difference-in-probability version of our metric (reported in the main paper) with the weight-of-evidence version, which is used by [169, 2].

We do not observe any notable differences in the trends between the metrics. We report one case where a hypothesis test is statistically significant with WoE but not the difference in probabilities; however, the difference between p-values is negligible ($p = .052$ vs. $p = .030$).

D.4 Discussion

Should We Prefer Counterfactual-Trained Models If They Are Harder to Explain? This question is raised by the fact that Suff and Comp scores are often worse for CT models (see Sec. 7.6). We suggest that the only reason for choosing between Standard and CT models is that CT models' explanations are not influenced by the model prior and random seed to the extent that Standard models' explanations are, as we argued in Sec. 7.4. If one prefers explanations (and explanation metrics) to reflect what a model *has learned from the data*, rather than the model prior and random seed, one would prefer to use CT models. It would be a mistake to prefer the standardly trained models on the grounds that they are "more easily explained" when this difference is due to the way we unintentionally influence model behavior for out-of-distribution data points.

In Defense of Searching For Explanations. De Cao et al. [37] argue against using a certain kind of search to find feature importance explanations on the grounds that it leads to "over-aggressive pruning" of features that a model does in fact rely on for its output. In their case, the objective of the search method is to find "a subset of features (e.g. input tokens) ... [that] can be removed without affecting the model prediction." They suggest that this method is susceptible to *hindsight bias*, asserting that "the fact that a feature can be dropped does not mean that the model 'knows' that it can be dropped and that the feature is not used by the model when processing the example." They provide the example of a model that (successfully) counts whether there are more 8s than 1s in a sequence, where they take issue with the fact that the search method would return a single 8 as its explanation for any sequence with more 8s than 1s, since this preserves the model prediction of "more 8s than 1s." The problem with this explanation, it is said, is that it removes digits that are most certainly being counted by the model when it comes to its decision. They provide empirical evidence that a learned model does in fact count all 8s and 1s before deciding which are more numerous.

One response to this argument is that if one obtains the optimal solution to an optimization problem and is not satisfied with it, then the objective must not be capturing everything that we care about, and the issue is not with the optimization method (i.e. search) that is employed. In the case at hand, we should first note that the objective is actually under-specified. De Cao et al. [37] suppose the search method returns the *maximal set* of tokens that can be removed while maintaining the model prediction, but the objective is not given with any preference for explanation *sparsity* (only that the removed tokens are a "subset" of the input). However, De Cao et al. [37] would take issue with search-based explanations regardless of whether the search method returns the minimal or maximal subset of tokens that can be removed without changing the model prediction. This is because they want the explanation to identify tokens that are "used by the model when processing the example." This criterion is not formalized, but the problem must be that it is a different criterion than the search objective, which is to find a feature subset that preserves the model prediction. After formalizing the notion of a feature being "used by a model," one should then be able to search for good explanations under the corresponding objective.

Explanation Distribution at Train Time. We reiterate that, to exactly match train and test-time distributions, models would be trained on counterfactual inputs drawn from explanation methods themselves, rather than simply random explanations. For now, this remains prohibitively expensive, as it would increase the number of forward passes required during training by up to 1000x depending on the budget to be used when explaining predictions. We explored methods based on training on the true counterfactual distribution (i.e., based on real explanations) to a limited extent, such as using real explanations only in the last training epoch. However, this alerted us to a few obstacles in such approaches: (1) this training distribution is non-stationary, as the FI explanations will change potentially with each gradient update, (2) these experiments were still quite expensive and required selecting a specific model checkpoint as the final model, which might not perform as well on accuracy metrics, and (3) we found that the Suff and Comp metrics were sometimes similar to Standard models, suggesting that these models were ultimately not as robust to the counterfactuals as the CT models were (see Conclusion 3 in Sec. 7.6.3). Future work in reducing the costs of obtaining explanations will help set the stage for more closely aligning the train and test time explanation distributions. In this regard, there may be applicable insights to draw from work on efficiently improving model robustness to local perturbations, such as Miyato et al. [134].

Another question that arises during training is whether applying the `Replace` function to x_i implies that the label y_i should change. It may *seem* problematic, for example, to fit a model to ($\texttt{Replace}(x, e)$, y) pairs if removing even a small number of tokens in x tends to flip the label. However, we note that what the model sees is an input with, e.g., MASK tokens in place of some tokens, and what an optimal model will do in this situation is make a prediction based on what evidence is available to it, with the knowledge that some features that could have influenced the label have been removed from the input. That is, based on the overall data distribution, such a model should produce appropriately calibrated outputs for inputs where most of the visible evidence suggests the labels to be y_1, while if the removed evidence had been visible, the label could be seen to be y_2.

Measuring Compute Budget. While we use the number of forward and backward passes as our compute budget for each method, wall-clock time will continue to be a useful and practical measure of compute for comparing methods. We note that batching forward passes on a GPU will significantly speed up method runtimes for a single data point while keeping the number of forward passes constant. In our experiments, this means that PLS is very efficient compared to Gradient Search, which does not batch inputs in the forward or backward pass. We use forward and backward passes as the unit of our compute budget since this is the fundamentally rate-limiting step in obtaining explanations, but practitioners will do well to compare methods with respect to wall-clock time as well.

Table D.4: Explanations metrics with a reduced search budget

		Sufficiency ↓		Comprehensiveness ↑	
Dataset	Method	Standard Model	CT Model	Standard Model	CT Model
SNLI	Vanilla Grad	59.41 (2.42)	63.41 (0.81)	7.08 (1.63)	5.84 (1.06)
	LIME (1000)	20.00 (2.02)	27.09 (1.68)	82.17 (2.82)	75.34 (1.94)
	Ordered Search (250)	-1.19 (0.87)	16.23 (1.45)	87.01 (2.40)	83.31 (1.73)
	Random Search (250)	-1.34 (0.90)	16.82 (1.46)	86.10 (2.52)	82.45 (1.82)
	PL Search (250)	**-1.50** (0.96)	**15.30** (1.37)	**87.25** (2.42)	**84.57** (1.68)
BoolQ	Vanilla Grad	30.81 (3.44)	16.40 (2.13)	2.43 (0.70)	2.25 (0.73)
	LIME (1000)	2.14 (1.75)	-1.56 (0.64)	52.03 (3.67)	**36.26** (3.44)
	Ordered Search (250)	1.44 (1.29)	-1.71 (0.70)	43.33 (3.18)	27.78 (3.00)
	Random Search (250)	0.09 (0.82)	-1.84 (0.66)	49.46 (3.55)	27.58 (2.74)
	PL Search (250)	**-0.56** (0.57)	**-2.61** (0.68)	**54.85** (3.78)	31.98 (2.97)
Evidence Inference	Vanilla Grad	20.76 (4.14)	12.96 (2.13)	2.92 (1.31)	1.57 (0.56)
	LIME (1000)	-16.07 (2.84)	-14.93 (1.38)	**47.61** (5.66)	**33.97** (4.19)
	Ordered Search (250)	-14.47 (2.65)	-11.67 (1.34)	39.69 (4.83)	26.51 (3.15)
	Random Search (250)	-15.28 (2.67)	-10.86 (1.28)	38.79 (5.46)	23.65 (2.80)
	PL Search (250)	-16.18 (3.14)	-13.78 (2.53)	41.90 (8.96)	25.27 (2.78)
FEVER	Vanilla Grad	19.63 (2.39)	13.21 (1.81)	1.52 (0.60)	1.02 (0.41)
	LIME (1000)	-0.24 (0.50)	1.36 (2.13)	**33.86** (3.44)	**22.06** (4.10)
	Ordered Search (250)	-0.73 (0.40)	1.00 (0.90)	28.70 (3.18)	16.30 (2.26)
	Random Search (250)	-1.16 (0.50)	-0.30 (2.07)	29.13 (3.18)	16.58 (1.91)
	PL Search (250)	-1.06 (0.44)	-0.36 (2.04)	25.81 (2.87)	15.22 (1.84)
MultiRC	Vanilla Grad	21.16 (3.47)	11.80 (1.38)	3.75 (1.18)	1.74 (0.79)
	LIME (1000)	-5.20 (1.19)	**-5.91** (1.19)	**39.75** (4.80)	**28.57** (2.18)
	Ordered Search (250)	-5.02 (1.03)	-4.16 (1.10)	33.26 (4.25)	21.95 (1.92)
	Random Search (250)	**-6.08** (1.17)	-4.86 (1.20)	32.31 (4.30)	19.95 (1.67)
	PL Search (250)	-5.20 (1.30)	-4.91 (1.27)	28.38 (3.85)	17.46 (1.49)
SST-2	Vanilla Grad	47.26 (3.79)	46.83 (1.41)	2.56 (0.71)	3.01 (1.00)
	LIME (1000)	1.97 (0.84)	5.92 (0.93)	52.42 (2.93)	45.74 (2.52)
	Ordered Search (250)	-0.91 (0.47)	2.71 (0.79)	55.90 (2.84)	48.96 (2.50)
	Random Search (250)	-0.91 (0.46)	2.73 (0.75)	55.44 (2.52)	48.31 (2.18)
	PL Search (250)	-0.91 (0.47)	2.69 (0.79)	56.14 (2.84)	49.08 (2.50)

Table D.5: Explanation metrics for RoBERTa-Base models

Dataset	Method	Sufficiency ↓	Comprehensiveness ↑
SNLI	LIME	24.42 (1.65)	71.30 (2.63)
	Random Search	11.52 (1.46)	81.43 (2.59)
	PLS	**9.73** (1.41)	**83.44** (2.31)
FEVER	LIME	1.22 (0.81)	25.09 (2.56)
	Random Search	1.10 (0.78)	22.46 (2.50)
	PLS	**-0.40** (0.63)	**27.61** (2.68)

Table D.6: Explanation metrics with weight of evidence

		Sufficiency ↓		Comprehensiveness ↑	
Dataset	Method	Diff. in Probs	WoE	Diff. in Probs	WoE
Evidence Inference	LIME	-14.93 (1.38)	-0.98 (0.14)	33.97 (4.17)	1.76 (0.29)
	Random Search	-12.71 (1.29)	-0.81 (0.13)	26.50 (3.15)	1.35 (0.21)
	PLS	**-20.33** (2.46)	**-1.44** (0.14)	38.71 (3.52)	**2.09** (0.25)
MultiRC	LIME	-5.90 (1.19)	-0.33 (0.09)	**28.57** (2.18)	**1.54** (0.17)
	Random Search	-6.58 (1.20)	-0.39 (0.09)	22.79 (1.78)	1.24 (0.13)
	PLS	**-9.77** (1.45)	**-0.63** (0.13)	26.96 (2.05)	1.45 (0.16)

E Additional Results and Details for Chapter 8

E.1 Learned Optimizer Details

Architecture. KnowledgeEditor is a learned optimizer $g : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \times \Theta \to \Theta$ that produces new model weights by applying an adjusted gradient step to a model. For reference, we give a glossary of symbols used here in Table E.1. For additional details beyond what is presented here, we refer readers to De Cao et al. [38].

At a high level, g_ϕ first encodes an input x_i and requested prediction change into a vector h, then processes h into two low-rank matrices A and B that are used to transform the model gradient on x_i, $\nabla_\theta \mathcal{L}(x_i, y_i^*)$. For Transformer models, the method edits only attention and feed-forward weights, so all model gradients match the shape of an associated weight matrix of shape $d_1 \times d_2$. Formally, a new model θ^* is obtained using a learned optimizer g_ϕ as follows:

$$
\begin{aligned}
h &= \text{LSTM}([x; \hat{y}; y^*]) \\
\{u, v, \gamma, \delta\} &= \{\text{MLP}_i(h)\}_{i=1}^4 \\
A &= \text{softmax}(u)v^T \\
B &= \text{softmax}(\gamma)\delta^T \\
\eta &= \sigma(\text{MLP}(h)) \\
\theta^* &= \theta + \eta(A \circ \nabla_\theta \mathcal{L}(x_i, y_i^*) + B)
\end{aligned}
$$

where ϕ consists of all LSTM and MLP parameters.

Training Algorithm. The learned optimizer objective is optimized w.r.t. ϕ with AdamW through a standard procedure of randomly sampling minibatches without replacement [124]. Within each batch, one datapoint is randomly selected as the Main Input, and the remaining points are used as $\mathcal{D}_R$. To obtain update labels $\{y_i^*\}_{i=1}^n$, we always use the opposite class in binary classification. For sequence-to-sequence tasks, we use the correct label when $\hat{y}_i$ is incorrect, and when $\hat{y}_i$ is correct, we randomly select another label from the training data. This choice is in contrast to De Cao et al. [38] and Mitchell et al. [132], who use samples from the model beam search as update labels for all points.

E.2 Additional Training Details

E.2.1 Compute Costs.

Learned optimizer memory. The hypernetwork has 92m trainable parameters for RoBERTa-base (which is 125m parameters), and 105m parameters for BART-base (which is 139m parameters). To increase training efficiency, we limit how far into the task model history we backpropagate. As shown in Fig. E.1, when backpropagating through task model parameters $\theta_t = \theta_{t-1} + \text{Update}(x_i, \hat{y}_i, y_i^*, \theta_{t-1}; \phi)$, we continue backpropagating through $\text{Update}(x_i, \hat{y}_i, y_i^*, \theta_{t-1})$ but *not* θ_{t-1}, which is also dependent on ϕ. That is, we apply a stop-gradient function to θ_{t-1}. This way, we compute the derivative $\nabla_\phi \text{Update}(x_i, \hat{y}_i, y_i^*, \theta_t; \phi)$. only once for each t, rather than recomputing these gradients for all subsequent time steps. These choices allow the memory use of our training algorithm to remain constant in r. We

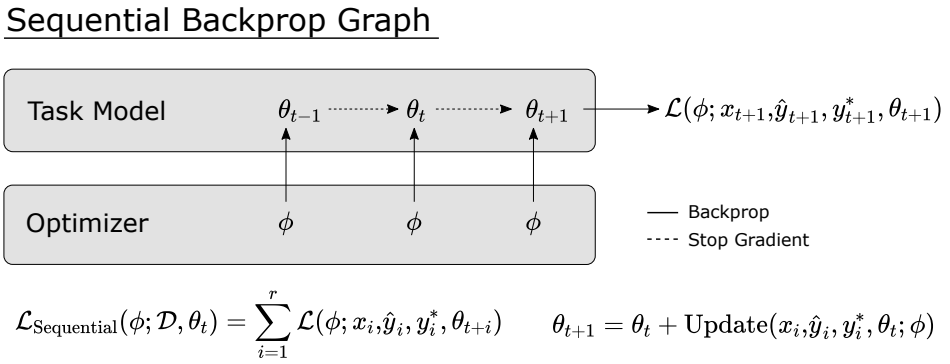


Figure E.1: The backpropagation graph for sequential model updates.

Symbol Glossary	
f_θ	Language Model
g_ϕ	Learned optimizer
x_i	Main Input
$\hat{y}_i$	LM output on x_i
y_i^*	Desired output
$\nabla_\theta \mathcal{L}(x_i, y_i^*)$	Gradient of LM
$\text{Update}(x_i, \hat{y}_i, y_i^*, \theta; \phi)$	Update one LM belief
$\mathcal{L}(\phi; x_i, \hat{y}_i, y_i^*, \theta)$	Belief update objective for x_i
$\mathcal{L}_{\text{Sequential}}(\phi; \mathcal{D}, \theta_t)$	Sequential objective (SLAG)
K	# gradient steps in Update$(\cdot)$
r	# beliefs updated in $\mathcal{L}_{\text{Sequential}}$

Table E.1: Symbol descriptions for the learned optimizer.

make the same choice for our K looped steps in a single application of the Update function, so the gradient for the update at step k depends only on $g_\phi(x_i, \hat{y}_i, y_i^*, \theta^{(k)})$ and not $\theta^{(k-1)}$. See Fig. E.2 for a graph of memory use depending on r and k.

Experiment runtimes. We now give runtimes for experiments in the paper. Building the belief graphs takes 25 hours for FEVER ($n = 10,444$) and 17.5 hours for LeapOfThought ($n = 8642$) on an NVIDIA RTX 2080 GPU. Computing summary statistics for graphs takes 3 hours on FEVER and 3 hours for LeapOfThought for statistics besides Update-Transitivity. We compute Update-Transitivity for LeapOfThought with a subset of 4000 points, which takes 45 hours.

All other experiments are run on a NVIDIA V100 32GB GPU. Training the task models takes 7 minutes for LeapOfThought, 45 minutes for FEVER, 4 hours for zsRE, and 10 hours for Wikidata5m. Training the learned optimizer with $r = 1$ takes 2.3 hours for LeapOfThought, 5 hours for FEVER, 9.5 hours for zsRE, and 16 hours for Wikidata5m. Training the learned optimizer with $r = 10$ takes 53 minutes for LeapOfThought, 2.9 hours for FEVER, 7 hours for zsRE, and 12.5 hours for Wikidata5m. Computing update statistics with the off-the-shelf optimizers with $r = 1$ takes 4 minutes for LeapOfThought, 30 minutes for FEVER, 2.3 hours for zsRE, and 3.9 hours for Wikidata5m. With $r = 10$, the baselines require 1 minute for LeapOfThought, 15 minutes for FEVER, 54 minutes for zsRE, and 1.8 hours for Wikidata5m.

Dataset	r_{test}	K	Objective
FEVER	1	5	Main
	10	1	Main
LeapOfThought	1	5	Main
	10	1	Main
zsRE	1	5	Main
	10	5	Main
Wikidata5m	1	5	Main+Para
	10	5	Main+Para

Table E.2: Final hyperparameters and objective terms of the learned optimizer for each task.

Total runtimes for each experiment should take into account multiple conditions and multiple seeds of each model being run.

E.2.2 Hyperparameters and Objective Terms.

Training hyperparameters. We fit our RoBERTa-base and BART-base task models to their respective datasets with the following hyperparameters: We train for 10 epochs on the binary tasks, and 20 for the sequence-to-sequence tasks. When predicting with BART-base, we use a beam search with width 5. In each case, we use AdamW from `torch.optim` with a LR of 1e-5 and weight decay of 1e-4. We select the best model according to the best dev set accuracy, checkpointing after each training epoch. The learned optimizers are optimized with AdamW, using a learning rate of 3e-4 and weight decay of 0. We train the learned optimizer for 5 epochs on each dataset except for LeapOfThought, which we train for 10 epochs given its smaller size. The learned optimizers are also selected based on dev set performance, with checkpointing after each training epoch. Their selection criterion is a raw average of Update Success Rate (averaged over each kind of data), Retain Rate (*Local Neutral*) and Δ-Acc, with terms dropped when they cannot be computed given the available data. Note that dev epochs with zsRE and Wikidata5m are fairly slow, so in order to speed up our experiments we compute dev epochs with a subset of 4000 dev points.

Learned optimizer. We give the final hyperparameter and objective terms used in each experiment in Table E.2. Our objective ablation is given in E.9, and we select the best performing condition for each dataset according to dev set performance, using the same selection criterion outlined previously. We keep all weight coefficients λ_i equal rather than tuning them. Main refers to the first term in Eq. 8.1, plus the KL term with random data. We use $K_{\text{train}} \leq 5$ for all experiments. For results across K values on zsRE, see Fig. E.6.

Baseline update method. We tune a baseline off-the-shelf optimizer separately for each dataset, using $r_{\text{test}} = 1$. Our performance criterion is the same as with learned optimizers, a raw average of Update Success Rate (averaged over each kind of data), Retain Rate (*Local Neutral*) and Δ-Acc. The grid search is over the following parameters: The off-the-shelf optimizers are from `torch.optim` and include {AdamW, SGD, and RMSProp} with default arguments (except for the learning rate). We consider a number of maximum steps in {5, 10, 100}. The learning rates we consider depend on the optimizer: {1e-4, 1e-5, 1e-6} for AdamW, {1e-4, 1e-5, 1e-6} for RMSProp, and {1e-1, 1e-2, 1e-3} for SGD. The LR ranges were selected after some initial manual exploration of the space. Our final hyperparameter values are shown in Table E.4 for each dataset. For comparison, De Cao et al. [38] use RMSProp

Relation	% Test Data
Place of Birth	11.00
Award Received	11.00
Cause of Death	5.66
Place of Death	11.00
Place of Burial	8.33
Educated At	11.00
Child	11.00
Occupation	11.00
Spouse	11.00
Sibling	9.01

Table E.3: Wikidata relations and their proportion of the test data.

Dataset	Optimizer	LR	Num. Steps
FEVER	AdamW	1e-6	100
LeapOfThought	SGD	1e-2	100
zsRE	SGD	1e-1	10
Wikidata5m	SGD	1e-1	10

Table E.4: Final hyperparameters of the baseline update method for each task.

with 100 update steps. The LR for zsRE and Wikidata5m may seem quite high, but this is the condition that actually does the least damage to the model's accuracy on other data, Δ-Acc. The baseline optimizes all of the trainable parameters in the language model, unlike the learned optimizer which optimizes only attention and feedforward weights for purposes of parameter efficiency.

E.2.3 Wikidata5m Additional Details.

We construct four paraphrases per Main Input by selecting from a set of alternative phrasings for the entity and relation in the Main Input. The syntax for each paraphrase follows the same simple template as the Main Input, in contrast to zsRE where syntax differs between paraphrases. A couple details remain. Some relations are one-to-many, and therefore we accumulate valid completing entities from the data as possible answers; later we compute accuracy as an exact match with any possible answer. All 10 relations appear in each split of

Ours	De Cao et al. [38]	Mitchell et al. [132]
Update Success Rate (*Main Input*)	Success rate	Edit success
Update Success Rate (*Paraphrase*)	Equivalence accuracy	Edit success
Update Success Rate (*Entailed Data*)	-	-
Retain Rate (*Local Neutral*)	-	-
Retain Rate (*All Data*)	Retain accuracy	-
Δ-Acc (*All Data*)	Performance deterioration	Drawdown

Table E.5: A glossary of terms used in work on model update methods. Note metrics are not always calculated in exactly the same way. For instance, Performance deterioration is a ratio in accuracies rather than difference in accuracies, and edit success from Mitchell et al. [132] combines two metrics in our case. The performance metric in Zhu et al. [240] is an average of Update Success Rate (*Main Input*) and Δ-Acc.

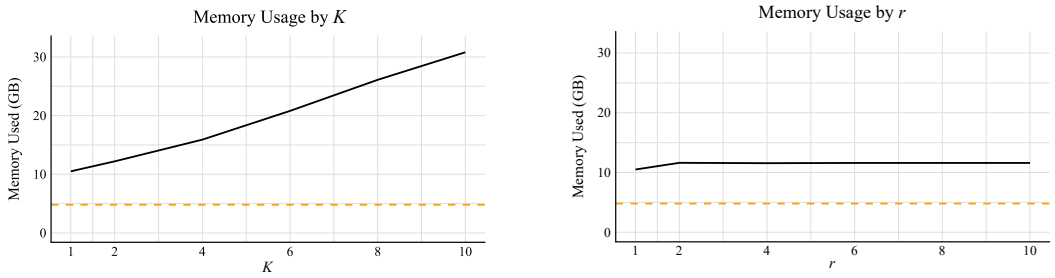


Figure E.2: Training memory usage in terms of K and r hyperparameters in our implementation, for a learned optimizer trained for a BART-base model on zsRE, using a batch size of 16. For comparison, the orange dashed line shows the memory use of training the BART-base model on zsRE, using the same batch size. Our use of the stop-gradient function limits the growth of runtime and memory w.r.t. both K and r. By accumulating gradients across points, memory w.r.t. r is kept constant. The same trick could be applied to the K looped gradient steps inside the Update function, at the trade-off of backpropagating K times per point rather than one time.

Dataset	Model	Acc	Paraphrase Cons ↑	Entailment Acc ↑	Contrapositive Acc ↑
FEVER	RoBERTa-base	78.29 (0.86)	-	-	-
LeapOfThought	RoBERTa-base	93.66 (0.87)	-	85.63 (1.08)	16.51 (2.71)
zsRE	BART-base	21.01 (0.64)	69.50 (1.09)	-	-
Wikidata5m	BART-base	10.21 (0.59)	25.84 (0.53)	-	-

Table E.6: Model accuracy and belief metric results and for four datasets.

the data. Only 33.80% and 37.18% of the entities in the dev and test splits are seen in the training data, though we do not find that models perform better on entities seen in training.

E.2.4 LeapOfThought Additional Details

The LeapOfThought dataset consists of a fact and a claim for each datapoint, where the truth of the fact implies that the claim has label y_i (True/False). All of the facts in the data are true, while half of the claims are true and half are false. When training the learned optimizer, we treat the the facts as the Main Input when training the learned optimizer and claims as entailed data. When training the True/False classifier, we fit to the claims, for which test accuracy is 83.65 ($\pm$ 1.05). This seems to generalize well to the facts, as test accuracy here is 93.66 ($\pm$0.87), although as the low contrapositive accuracy suggests (Table 8.3), the model seems to be too prone to predicting true for this data.

Since very few of the Main Inputs are predicted as false, we run into a small dilemma when fitting the learned optimizer with the use of the entailed data objective term. The entailment between fact and claim only holds when the fact is true, so we can only compute the objective when updating a point from false to true. This ends up being less than 10% of the training data. We ultimately choose to oversample points that fit this description during training of the learned optimizer, which allows the learned optimizer to fully fit to the entailed data. Also note that during learned optimizer training, we include Entailed Data *from other data points besides the Main Input* in the KL term in Eq. 8.1, and we measure Δ-Acc using both Main Inputs and Entailed Data.

Dataset	Data Type	Input	Label(s)
zsRE	Main Input	What did Gifford Pinchot die of?	{Leukemia}
	Paraphrase	How did Gifford Pinchot die?	
	Main Input	Player Ali Kanaan plays for what team?	{Sporting Al Riyadi Beirut}
	Paraphrase	What team is Ali Kanaan associated with?	
Wikidata5m	Main Input	Margarita Nolasco Armas has relation 'place of birth' to	{Orizaba, Veracruz; Orizaba; etc.}
	Paraphrase	SusunW/Margarita Nolasco Armas has relation 'born at' to	
	Local Neutral	Margarita Nolasco Armas has relation 'place of death' to	Mexico City; Ciudad de Mexico; etc.
	Main Input	Mary Good has relation 'award received' to	{Garvan-Olin Medal; Arkansas Women's Hall of Fame; etc.}
	Paraphrase	Mary Lowe Good has relation 'winner of' to	
	Local Neutral	Mary Good has relation 'educated at' to	{The University of Arkansas; U Arkansas; etc.}
FEVER	Main Input	Tardigrades are also known as space bears.	True
	Main Input	The Lion belongs to the genus Vulpes.	False
LeapOfThought	Main Input	A viper is a vertebrate.	True
	Entailed Data	A viper has a brain.	True
	Main Input	A amaranth is a herb.	True
	Entailed Data	A amaranth has a nose.	False

Table E.7: Example datapoint from each dataset, and auxiliary data that accompanies the Main Input.

E.3 Noise in Datasets

We briefly document some shortcomings of each dataset, with reference to examples in Table E.7.

FEVER. Some claims are slightly vague or ambiguous when taken on their own. For instance "Doug Ducey was the CEO of Cold Stone Creamery and offered many opportunities to new hires" is rated True, though this will depend heavily on what one thinks "many opportunities" means. Similar whether or not "L.A. Guns is a tattoo shop" depends on which "L.A. Guns" one is referring to, the tattoo shop or metal band. Of course, this is a generic issue of language, and not unique to this dataset. Some inputs seem to be a matter of person opinion: "Los Angeles is known for its food" is rated False.

LeapOfThought. Many examples use an "is a" relation, producing sentences like "A sunlight is a good health." This could be more false than true, but it's a fairly nonsensical statement to begin with. There are also other nonsensical or vague examples in the data: "A friar is the opposite of mineral" is labeled False. "A detective desires equal opportunity." is labeled True. It is not immediately clear what conditions would make these statements true or false.

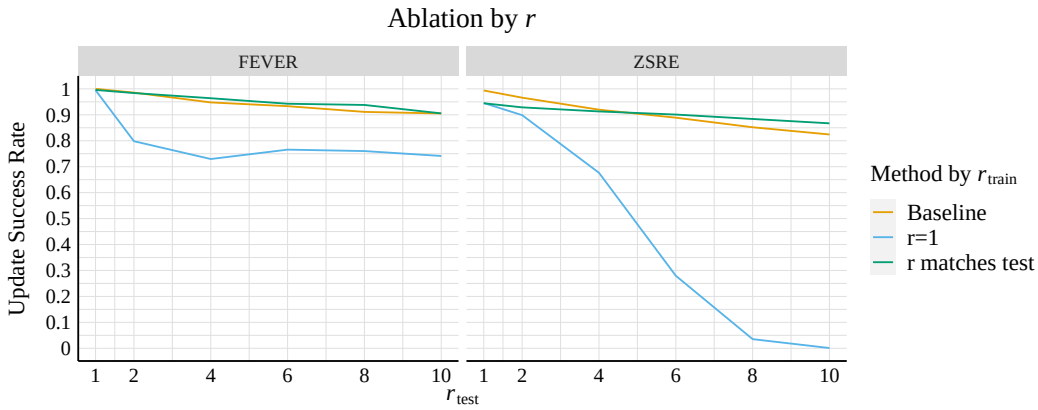


Figure E.3: Ablation across values of r for training and testing. On zsRE, our method outperforms the baseline when $r_{\text{test}} = 10$, and the gap is likely to increase as r_{test} rises further. When using a non-sequential objective from past work, performance declines drastically as r_{test} rises.

zsRE. Some questions invoke potentially one-to-many or temporally dependent relations, though there is only one ground-truth answer per question in this dataset. For instance, a paraphrase of the question about Gifford Pinchot in Table E.7 is: "What disease did Gifford Pinchot have?" A person might have had many diseases over their life which could all be valid responses. The answer is especially ambiguous for spatial relations, where a valid answer might refer to a city, region, country, province, or continent.

Wikidata. Aliases sometimes vary greatly even as they refer to the same person, or they are simply noisy. For example, as shown in Table E.7, "SusunW" appears in an entity name, but this is actually a username of someone who contributed to the Wikipedia article for Margarita Nolasco Armas. Meanwhile, other aliases for J.R.R Tolkien include "Tolkienian" and "Mabel Suffield," his mother. Rephrasings of relations might also create confusing inputs, e.g. switching "child" with "has kids," "daughter", or "son." Similar to zsRE, some relations are also one-to-many and temporally dependent (like occupation), though we hope that by using many valid answers we circumvent this issue to some extent when calculating prediction correctness.

E.4 Metric Computation and Bootstrap Details

Metric computation. The only computationally difficult metric to calculate is Δ-Acc, which requires computing the updated language model's accuracy on other data after every single belief update. We randomly sample other data after every update for this purpose, using $n = 30$ points for zsRE and Wikidata5m and $n = 200$ points for FEVER and LeapOfThought. We ensure that all evaluation data is used at some point during this sampling by preferentially selecting data that has been infrequently selected before. We note that paraphrase consistency is easy to evaluate for a small number of paraphrases per datapoint, as we have for both zsRE and Wikidata5m. Additionally, on LeapOfThought, we compute Δ-Acc using both Main Inputs and Entailed Data.

Update-Transitivity caveat. The % Update-Transitivity metric represents the answer to the question: if updating belief A changes belief B, and updating belief B changes belief C, what proportion of the time does updating A change C? We would treat this as a normative

	Update Success Rate		Δ-Acc
Desired Label	Main Input	Paraphrases	All Data
Beam Label	91.19 (0.5)	92.07 (0.8)	-0.39 (0.1)
Hard Label	94.46 (0.7)	94.45 (0.7)	-0.24 (0.1)

Table E.8: Update metrics by optimizer training labels.

metric that we hope to maximize, except we do not know in general whether there is a confounding belief D that determines the relationship between B and C. If changing A also changed a confounding belief D, then we might not be able to expect that C should change too. That said, when we have no reason to think there are such confounding beliefs, we would expect a logically consistent model to display 100% Update-Transitivity of their beliefs. In Fig. 8.3, for instance, we see no reason to suspect there are confounding beliefs for the relationship between the date Bessie Smith died and the writer of Despicable Me 2, and therefore we would expect that updating the belief about what album Hot Right Now is on would change the belief in Despicable Me 2's authorship (which it does).

Bootstrap computation. We account for sample and seed variance by block bootstrap [50]. When there is a single statistic per data point, like Main Input Update Success, we form a matrix of shape $n \times s$ for n data points and s model seeds (where the seed was used for both task model training and learned optimizer training). We then resample rows and columns of this matrix 10,000 times, which was sufficient for convergence. When we perform hypothesis tests for the difference in statistics between conditions, we pair the data points by using the same rows of this matrix at each step of the bootstrap (i.e. we conduct paired tests). For metrics involving multiple data points per Main Input, like paraphrases or other random data, we make a simplifying assumption where we do not resample the multiple data points but just compute the average metric for those data points and treat that as the ground-truth statistics for the Main Input. We explored using a full 3-dimensional bootstrap, where we resample among these extra datapoints by constructing a matrix of shape $n \times s \times n$, but it was quite slow and gave similar results to the block bootstrap.

E.5 Additional Results

Ablation across num. sequential steps. Fig. E.3 shows the results for an ablation across r_{test} using two kinds of learned optimizers: SLAG_1, where $r_{\text{train}} = 1$, and a SLAG condition where $r_{\text{train}} = r_{\text{test}}$. It is critical to the success of learned optimizers to train them to update points sequentially when this is a desired application. Further, sequential updating with sequence prediction tasks is the only setting where we see learned optimizers outperform baselines across all relevant metrics.

Choosing training labels for learned optimizers. In early experiments, we found that it is beneficial to use all data points (including correctly predicted points) as Main Inputs *during training*, rather than restricting training to only incorrectly predicted points. We still focus on correcting wrong outputs at test time. But so we must select what label to use during optimizer training. To get a Hard Label, we use the correct label for incorrectly predicted points, and for correctly predicted points, we simply draw a label randomly from the labels in the training data. The alternative Beam Label condition uses a sample from the model's beam search for a data point, as done in past work [38, 132]. We show update metrics for

Objective Term Ablation		Update Success Rate			Retain Predictions		Δ Acc
Dataset	Objective	Main Input	Paraphrases	Entailed Data	Local Neutral	All Data	All Data
FEVER	Main	100 (0.0)	-	-	-	98.27 (0.1)	-0.15 (0.1)
	(no KL)	100 (0.0)	-	-	-	40.42 (0.6)	-27.19 (1.2)
LeapOfThought	Main	100 (0.0)	-	76.43 (5.3)	-	96.84 (0.3)	-1.22 (0.8)
	+Ent	100 (0.0)	-	71.87 (5.3)	-	96.52 (0.3)	-0.40 (0.8)
zsRE	Main	94.46 (0.4)	94.44 (0.7)	-	-	81.96 (0.4)	-0.24 (0.1)
	+Para	93.75 (0.4)	94.41 (0.7)	-	-	75.24 (0.5)	-0.42 (0.2)
Wikidata5m	Main	88.67 (0.7)	64.12 (0.7)	-	49.78 (1.0)	71.04 (0.5)	-1.54 (0.3)
	+Para	87.46 (0.7)	81.06 (0.7)	-	47.15 (1.0)	63.02 (0.6)	-1.55 (0.3)
	+LN	87.73 (0.7)	59.75 (0.7)	-	60.49 (1.0)	72.69 (0.6)	-1.57 (0.3)
	+Para+LN	87.02 (0.7)	81.18 (0.7)	-	56.86 (1.0)	68.42 (0.6)	-1.65 (0.3)

Table E.9: Belief update results by the objective terms used for the learned optimizer. We do not bold any numbers based on statistical significance. For tuning purposes we select whichever condition achieves the higher selection criterion without testing for statistical significance.

zsRE split by the desired label in Table E.8. If one's goal is to fix wrong model outputs, then it is much better to use either the correct label or a random label as the desired model output during training rather than a sample from the model's beam search. Update success improves by 3.27 ($\pm$0.65; p<1e$-$4) points for the Main Input and 2.38 ($\pm$1.05; p<1e$-$4) for Paraphrases, while Δ-Acc rises by 0.15 ($\pm$0.18; p=.09).

Which beliefs are hard to update? We hypothesize that beliefs will be easier to update when they are more belief-like to begin with. We principally measure this via the correlation between update success rate and a belief's consistency on paraphrases before the update, for our learned optimizer in a single-update setting ($r = 1$). Surprisingly, we observe no relationship between update success and the belief consistency. The correlation between consistency and update success is near 0 for both zsRE ($\rho = -.027$) and Wikidata5m ($\rho = .013$); see Fig. E.4 for a plot of the relationship. So it appears that the learned optimizer can update model beliefs independently of how belief-like they are to begin with. We would also be interested in considering consistency under entailment, but the update success rate on LeapOfThought is already 100%, so there is no variance to explain.

Learning curve. In Fig. E.5 we show the learning curve of a learned optimizer trained with SLAG on zsRE. The Main Input Update Success Rate steadily rises as a function of the training set size.

Ablation by num. update steps. Fig. E.6 shows the results of an ablation across values of K using a learned optimizer trained using SLAG with $r = 1$ on zsRE. Main Input Update Success rises by over three points by increasing K_{test} from 1 to at least 5. Using a value of K_{train} that matches K_{test} gives a further increase of about 0.5 points.

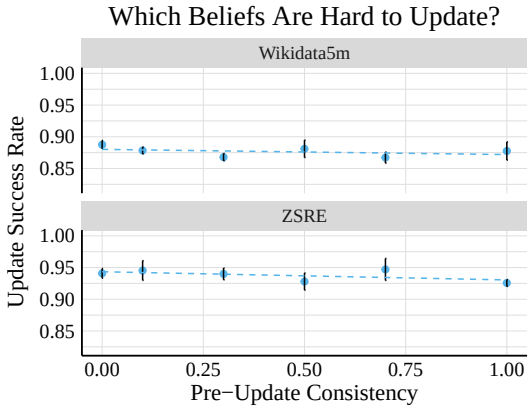


Figure E.4: Beliefs are neither easier nor harder to update depending on their consistency beforehand.

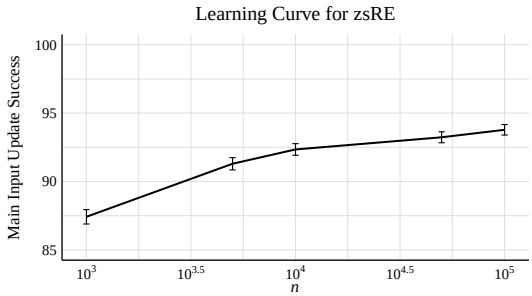


Figure E.5: Main Input Update Success Rate across training set sizes, using SLAG on zsRE.

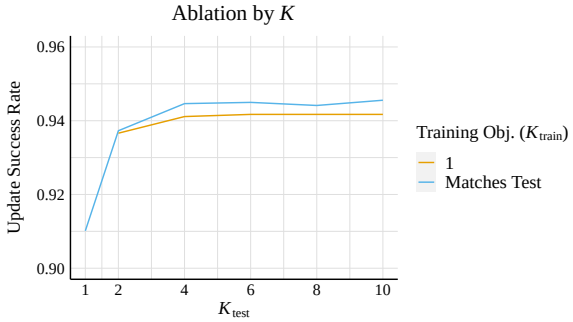


Figure E.6: Ablation across values of K for training and testing, using SLAG on zsRE. It is useful to train the optimizer using the value of K it will use at test time.

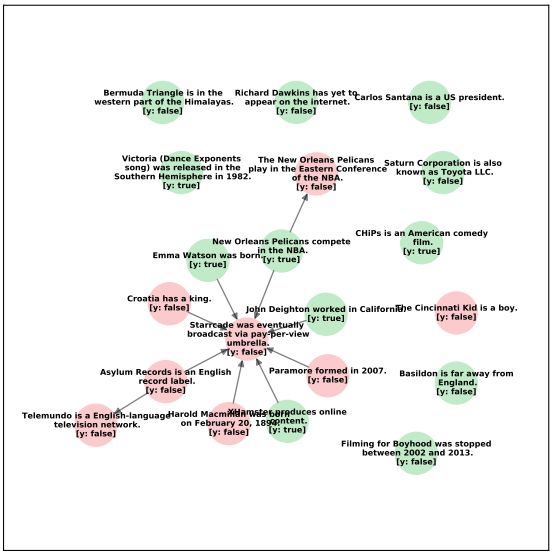


Figure E.7: A random subgraph of the belief graph for FEVER. Note all nodes actually are connected to at least one another node.

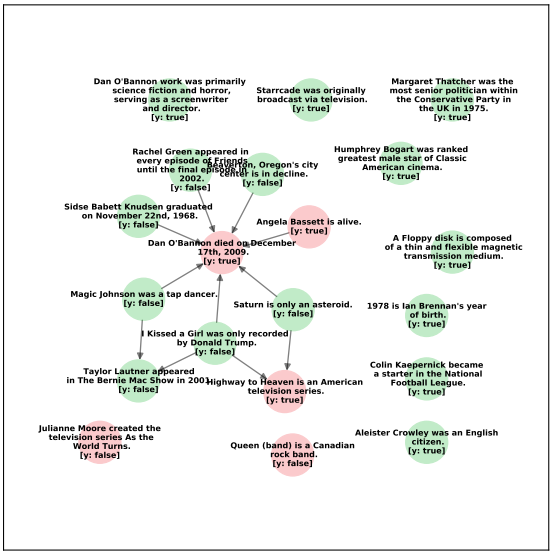


Figure E.8: A random subgraph of the belief graph for FEVER. Note all nodes actually are connected to at least one another node.

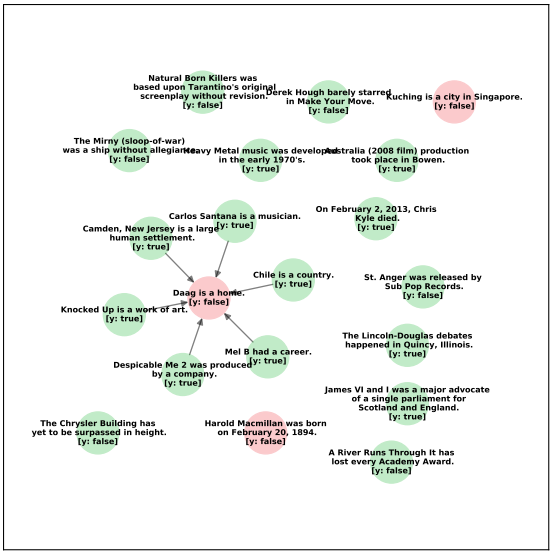


Figure E.9: A random subgraph of the belief graph for FEVER. Note all nodes actually are connected to at least one another node.

F Additional Results and Details for Chapter 9

F.1 Experiment Details

Data Licenses. CounterFact is available by the MIT license at https://github.com/kmeng01/rome [128], and ZSRE is available publicly at http://nlp.cs.washington.edu/zeroshot/ [111].

Data Filtering. We filter the CounterFact dataset to a subset of facts that are correctly completed by GPT-J, in order to ensure that there is knowledge to localize in the model for each point. We mark a completion correct when o_{true} appears among the first 36 tokens sampled from the model given the prompt P using greedy decoding. GPT-J achieves a completion accuracy of 32.6% under this scheme, and after starting with about 10% of the CounterFact dataset, our final sample size is $n = 652$. We perform additional filtering specifically for model editing in the Fact Erasure condition, where we filter points to have a target probability $p_\theta(o_{true}|s, r)$ of at least .02, so that there is a reasonable amount of probability mass to be erased. In this condition, we have $n = 489$ points.

Compute. Experiments were run on a single NVIDIA A6000 GPU with 48gb memory. Computing editing performance for $n = 652$ points with GPT-J for a single edit method applied across model layers in the set {1, 5, 9, 13, 17, 21, 24, 28} could take about eight hours. Saving causal tracing or representation zeroing results for these datapoints takes about twelve hours. Regression analyses and plots can be made on demand (code in supplement) given the data from the editing and localization experiments.

Edit Method Tuning. We tune the edit methods to have high rewrite scores while not trading off too aggressively against paraphrase and neighborhood scores. More specifically, this means we tune methods to have rewrite scores no higher than 99% (note methods can easily get above 99% rewrite score), separately for each editing problem variant. The tuning is done with the first 100 points of the CounterFact dataset, editing layer 6 for GPT-J and 18 for GPT2-XL. For ROME and MEMIT methods, we tune over the KL regularization weight values in the set {.0625, .9, 1}. For constrained finetuning, we tune over the L_∞ norm weight values in the set {1e-4, 5e-5, 2e-5, 1e-5}. For both methods, we adopt default parameters from Meng et al. [129] unless otherwise stated. We describe the relevant hyperparameters below, for GPT-J first:

1. *Error Injection.* FT-1: norm constraint of 1e-4. FT-5: norm constraint of 2e-5. ROME: regularization weight of 1. MEMIT: regularization weight of 0.9.
2. *Tracing Reversal.* FT-1: Norm constraint of 1e-5. FT-5: Norm constraint of 2e-5. FT-5: 2e-5. ROME: default parameters. MEMIT: default parameters.
3. *Fact Erasure.* FT-1: norm constraint of 1e-5. FT-5: norm constraint of 1e-5. ROME: default parameters. MEMIT: default parameters.
4. *Fact Amplification.* FT-1: norm constraint of 1e-5. FT-5: norm constraint of 1e-5. ROME: default parameters. MEMIT: default parameters.

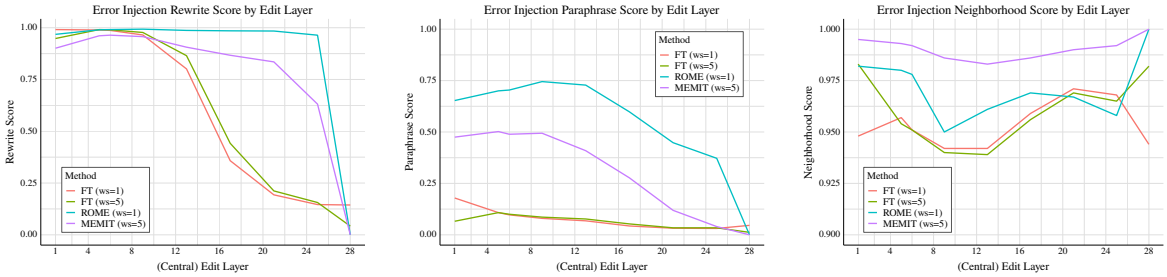


Figure F.1: Edit success metrics for our four editing methods, under the Error Injection objective. Left: Rewrite, Center: Paraphrase, Right: Neighborhood.

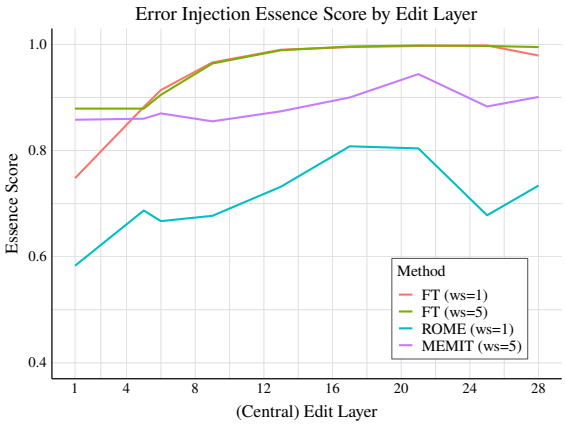


Figure F.2: Essence score by edit layer, for our four editing methods, under the Error Injection objective.

5. *Fact Forcing.* Note that for all methods we decide to increase the number of gradient steps, as convergence takes longer for finetuning (from 25 to 50 steps) and for the gradient-based optimization for v^* in ROME (from 20 to 25 steps). FT-1: norm constraint of 1e-4. FT-5: norm constraint of 1e-4. ROME: 25 gradient steps for finding v^*. MEMIT: default parameters (already set to 25 steps).

We run only the Error Injection and Fact Forcing conditions for GPT2-XL. Hyperparameters are as follows:

1. *Error Injection.* FT-1: norm constraint of 1e-3. FT-5: norm constraint of 1e-4. ROME: default parameters. MEMIT: default parameters.
2. *Fact Forcing.* FT-1: norm constraint of 5e-4. FT-5: norm constraint of 5e-5. ROME: default parameters. MEMIT: default parameters.

F.2 Additional Results

ZSRE Dataset. Here, we describe experiments with the ZSRE dataset, which is commonly used in past editing method papers [38, 132]. ZSRE includes naturalistic questions rather than prompts intended for autoregressive cloze completion, as in CounterFact. Following past

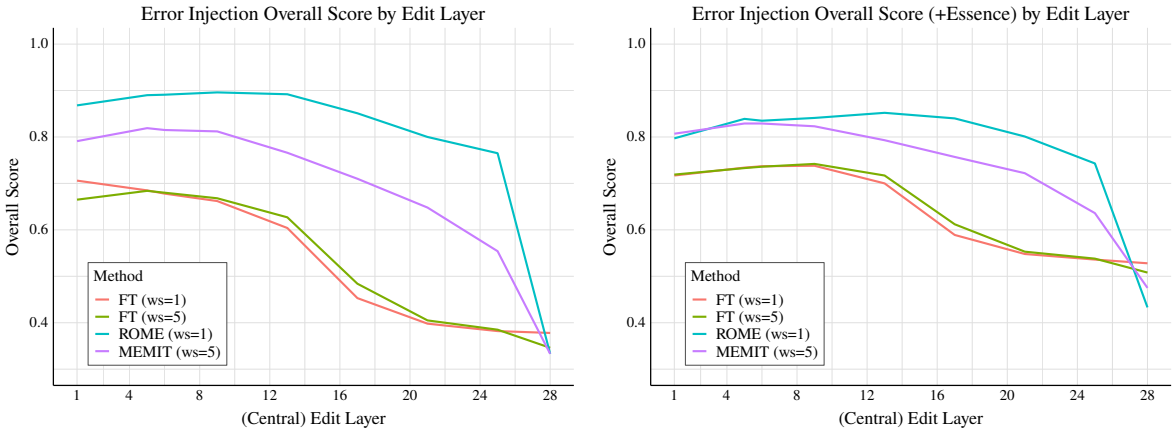


Figure F.3: Overall edit success for our four editing methods, under the Error Injection objective. Left: The mean of Rewrite, Paraphrase, and Neighborhood Scores. Right: the mean score with Essence Score included.

work [128], we use GPT-J to answer ZSRE questions in a zero-shot manner, and we edit the model with ROME. We report results for ZSRE via plots of edit success vs. tracing effect in Figs. F.13 (rewrite score) and F.14 (overall score), accompanied by regression analysis results in Table F.7. We find that results with ZSRE match our conclusions with CounterFact, as the results are quite similar to plots and regressions with CounterFact data. Tracing effects are not predictive of edit success.

Representation Zeroing. Representation zeroing is a common localization technique where neural activations are manually set to zero during a model forward pass [104, 15]. We implement a form of representation zeroing that is exactly like Causal Tracing, except instead of denoising already-noised representations, we set clean representations to zero. Specifically, we simply run a normal forward pass until a certain set of layers (window size=5), where we zero out representation values for the MLP output representations at the subject token indices within those layers (then continue the forward pass). The localization effect is computed as the proportion of the original predicted probability that is deleted via the zeroing operation (ranging from no effect as 0% to 100% of probability deleted as 100%). These new results are shown in Figs. F.15 for rewrite score and F.16 for overall score, using ROME on GPT-J with CounterFact data. We obtain the same conclusions as our analysis with causal tracing: localization via representation zeroing is not predictive of edit success. Specifically, we see correlations between edit success and localization effect to be near zero across layers (using either rewrite score or overall score for edit success).

Highly concentrated tracing effects. Since Causal Tracing analysis suggests that information accrues gradually across layers (see Fig. F.4), it seems possible that information is simply so diffusely spread across model layers that no matter what layer you edit, you will be editing a layer where a fact is at least stored in part. Based on this observation, we want to test whether tracing effects correlate better with edit success specifically when tracing effects are concentrated in a small number of layers. This condition represents that a fact appears to be stored in a small number of layers *and not elsewhere*. We hope that by editing in that

Table F.1: R^2 values for predicting ROME edit success in Error Injection, subsetted to 10% of the data that has the most concentrated tracing effects in a small number of layers. Even when facts appear to be stored at a small number of layers *and not other layers*, tracing effects are still not predictive of editing performance.

Concentrated Data	R^2 Values		
Method	Layer	Tracing Effect	Both
ROME	0.927	0.02	0.929

range of layers, we can more easily manipulate that fact. To identify points with *concentrated* tracing effects, we use a heuristic for filtering points. Given the output of Causal Tracing analysis for a point, i.e. one effect per layer (the max across tokens), we define the point to have concentrated tracing effects when there are no more than three layers that have at least 50% of the maximum effect across layers (besides the layer with the max effect itself). Under this criterion, about 10% of the data (74 of 652 cases) have concentrated effects. Note we use our default tracing window size of 5 with the 28 layer GPT-J model for this experiment.

We show the results from our analysis on this data subset in Table F.1, and **we observe no changes** in our main conclusions. For ROME with Error Injection, the added effect is 0.2%. Across editing problems and edit methods, the maximum added effect of including tracing effects on R^2 values for predicting rewrite score remains at 3.2% (for Fact Forcing with constrained finetuning). Thus, we conclude that even when facts appear to be stored in a small number of layers, localization results from Causal Tracing are still not informative about editing success, while the choice of edit layer is a far more important factor in whether a fact is successfully edited.

Measuring essence drift. Meng et al. [128] describe one possible consequence of model editing as *essence drift*, which occurs when core properties of an entity change after attempting to edit only one property of that entity. For example, changing where an island is located might also cause the model to nonsensically treat the island as a university campus (see example in Meng et al. [128]).

We aim to obtain an automatic metric to serve as a rough proxy for essence drift. A related metric is calculated with "Local Neutral" data involving the same subject entity but with other properties that are logically neutral with the original property of the subject being edited [70]. However, we do not have "Local Neutral" data for the CounterFact dataset, and essence drift aims to specifically measure changes to *core* properties of a subject.

Therefore, we automatically estimate changes to known properties of the subject s by calculating the change in model perplexity over samples of text that were drawn from the pre-edit model given the prompt "s is a " (which tend to describe a number of key properties of the subject s). We term these samples *essence texts*, and we obtain five samples per subject prompt by sampling with multinomial top-k sampling using $k = 5$. Given our essence texts, we measure the perplexity over the samples before and after editing a fact in the model, for every edited fact in our dataset. Note this is quite similar to the essence drift regularization objective used in the ROME optimization objective [128], but we consider it as a metric here. We scale the change in perplexity to a fraction of 5, with the cut-off of 5 chosen to represent a maximally bad change to the model perplexity. Similar to our other metrics, our essence score is 1 if model perplexity on the essence texts does not change after editing the model

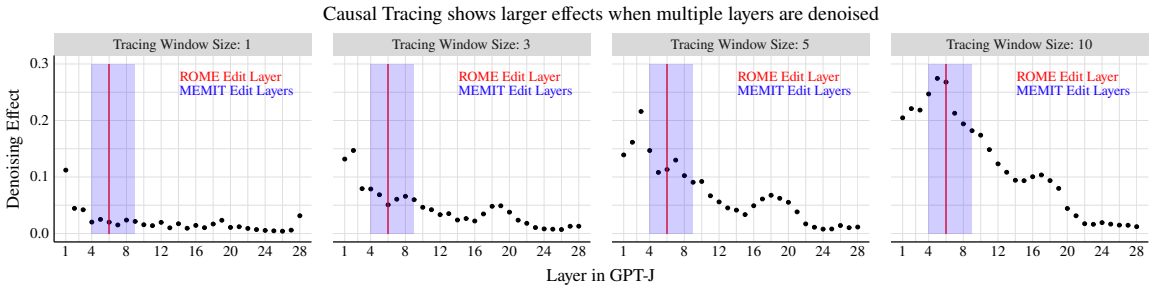


Figure F.4: Tracing effects grow larger as the number of adjacent restored layer representations increases (tracing window size).

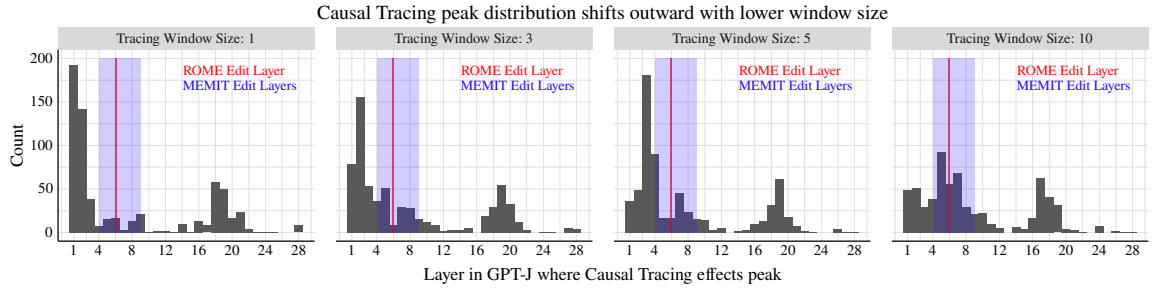


Figure F.5: Each individual plot shows the distribution of tracing curve peaks (the argmax layer) across datapoints, using a different tracing window size. Together, the plots show how the distribution of layers where the tracing curves peak for each point shifts outward toward the first and last layer of the model as the tracing window size declines. This is primarily due to a clipping effect from using a window size greater than 1. The way tracing values are computed, a window size of 10 implies that the effect for "layer 1" is from restoring layers 1-5, while the effect for layer "layer 5" is 1-10. As a result, a tracing window size of 10 favors layer 5 over layers 1-4, and reducing the tracing window size leads to these clumps of effects shifting from layer 5 toward layer 1 (and from layer 24 to layer 28)

(capping to 1 in cases of slight decreases in perplexity), and it is 0 if the perplexity increases by 5 or more.

We show essence scores for editing methods across layers in F.2. Interestingly, the trend across layers for this metric is mostly counter to the trends for other metrics (Fig. F.1), with editing later layers being generally preferable to editing earlier layers. As a result, when combined with the other metrics in Fig. F.3, we see that the overall score trend flattens and shifts slightly toward mid-range layers in the model.

F.3 Robustness Experiments

In addition to our main results with ROME for GPT-J and our Rewrite Score metric, we include robustness experiments to confirm that results are similar for (1) other measures of edit success including Paraphrase Score, Neighborhood Score, and Overall Score (Tables F.3, F.4, and F.5), (2) different values of the tracing window size (Fig. F.6), (3) GPT2-XL rather than GPT-J (Fig. F.7), (4) the original unscaled metrics from Meng et al. [128] (Fig. F.8), and (5) using the tracing effect at the last subject token rather than the max across tokens

Rewrite Score Table		R^2 Values				
Editing Problem	Method	Layer	Trace	Both	Diff	p-value
Error Injection	FT (1 layer)	0.756	0.062	0.758	0.002	<1e-4
	FT (5 layers)	0.775	0.055	0.777	0.002	<1e-4
	ROME (1 layer)	0.947	0.016	0.948	0.001	<1e-4
	MEMIT (5 layers)	0.677	0.024	0.678	0.001	0.199
Tracing Reversal	FT (1 layer)	0.067	0	0.067	0	0.997
	FT (5 layers)	0.751	0.045	0.752	0.001	0.032
	ROME (1 layer)	0.294	0.017	0.31	0.015	<1e-4
	MEMIT (5 layers)	0.212	0.036	0.218	0.006	<1e-4
Fact Erasure	FT (1 layer)	0.643	0.028	0.646	0.003	<1e-4
	FT (5 layers)	0.698	0.025	0.70	0.002	<1e-4
	ROME (1 layer)	0.857	0.019	0.858	0	0.555
	MEMIT (5 layers)	0.925	0.019	0.925	0	0.669
Fact Amplification	FT (1 layer)	0.383	0.014	0.393	0.01	<1e-4
	FT (5 layers)	0.424	0.01	0.436	0.011	<1e-4
	ROME (1 layer)	0.88	0.02	0.88	0	0.654
	MEMIT (5 layers)	0.905	0.018	0.906	0.001	<1e-4
Fact Forcing	FT (1 layer)	0.697	0.104	0.724	**0.027**	<1e-4
	FT (5 layers)	0.634	0.10	0.666	**0.032**	<1e-4
	ROME (1 layer)	0.422	0.004	0.425	0.003	<1e-4
	MEMIT (5 layers)	0.345	0.041	0.354	0.009	<1e-4

Table F.2: R^2 values for predicting **rewrite** score from choice of edit layer and tracing effect, across editing problem variants (corresponds to data in Fig. 9.6). Diff shows the added effect of including tracing in the regression (Both vs. Layer Only), in terms of R^2, and p-value shows the results from an F-test comparing the Both and Layer Only models. Tracing has some predictive value for Fact Forcing, but the R^2 value remains small compared to the choice of edit layer.

Table F.3: R^2 values for predicting **paraphrase** score from choice of edit layer and tracing effect, across editing problem variants. Diff shows the added effect of including tracing in the regression (Both vs. Layer Only), in terms of R^2, and p-value shows the results from an F-test comparing the Both and Layer Only models. The added effect of including tracing effects is very small across conditions (less than 3%).

Paraphrase Score Table		R^2 Values				
Editing Problem	Method	Layer	Trace	Both	Diff	p-value
Error Injection	FT (1 layer)	0.061	0.005	0.063	0.002	0.258
	FT (5 layers)	0.036	0.003	0.038	0.001	0.582
	ROME (1 layer)	0.279	0.001	0.303	0.024	<1e-4
	MEMIT (5 layers)	0.246	0	0.269	0.023	<1e-4
Tracing Reversal	FT (1 layer)	0.004	0.001	0.004	0	0.989
	FT (5 layers)	0.001	0	0.002	0.001	0.841
	ROME (1 layer)	0.01	0	0.012	0.002	0.121
	MEMIT (5 layers)	0.001	0	0.001	0	0.997
Fact Erasure	FT (1 layer)	0.046	0.001	0.048	0.002	0.303
	FT (5 layers)	0.079	0.007	0.084	0.005	0.004
	ROME (1 layer)	0.537	0.012	0.539	0.001	0.218
	MEMIT (5 layers)	0.586	0.015	0.587	0.001	0.184
Fact Amplification	FT (1 layer)	0.005	0.012	0.022	0.017	<1e-4
	FT (5 layers)	0.017	0.013	0.035	0.018	<1e-4
	ROME (1 layer)	0.24	0.002	0.267	0.027	<1e-4
	MEMIT (5 layers)	0.236	0.001	0.263	0.026	<1e-4
Fact Forcing	FT (1 layer)	0.044	0.004	0.046	0.002	0.367
	FT (5 layers)	0.023	0.002	0.025	0.002	0.387
	ROME (1 layer)	0.357	0.01	0.36	0.003	0.003
	MEMIT (5 layers)	0.095	0.001	0.105	0.01	<1e-4

Table F.4: R^2 values for predicting **neighborhood** score from choice of edit layer and tracing effect, across editing problem variants. Diff shows the added effect of including tracing in the regression (Both vs. Layer Only), in terms of R^2, and p-value shows the results from an F-test comparing the Both and Layer Only models. The added effect of including tracing effects is very small across conditions (2% or less).

Neighborhood Score Table		R^2 Values				
Editing Problem	Method	Layer	Trace	Both	Diff	p-value
Error Injection	FT (1 layer)	0.005	0	0.008	0.002	0.197
	FT (5 layers)	0.014	0.001	0.015	0.001	0.55
	ROME (1 layer)	0.011	0.003	0.015	0.005	0.001
	MEMIT (5 layers)	0.004	0.001	0.006	0.002	0.154
Tracing Reversal	FT (1 layer)	0.001	0	0.001	0	1
	FT (5 layers)	0.001	0	0.002	0.001	0.946
	ROME (1 layer)	0.001	0	0.002	0.001	0.946
	MEMIT (5 layers)	0.001	0	0.002	0	0.981
Fact Erasure	FT (1 layer)	0.01	0	0.014	0.004	0.037
	FT (5 layers)	0.01	0	0.013	0.004	0.06
	ROME (1 layer)	0.04	0.005	0.046	0.006	0.001
	MEMIT (5 layers)	0.05	0.007	0.059	0.009	<1e-4
Fact Amplification	FT (1 layer)	0.012	0.009	0.02	0.008	<1e-4
	FT (5 layers)	0.016	0.008	0.025	0.009	<1e-4
	ROME (1 layer)	0.04	0.01	0.05	0.01	<1e-4
	MEMIT (5 layers)	0.035	0.008	0.044	0.01	<1e-4
Fact Forcing	FT (1 layer)	0.054	0	0.057	0.003	0.03
	FT (5 layers)	0.019	0.001	0.022	0.004	0.011
	ROME (1 layer)	0.299	0.022	0.311	0.012	<1e-4
	MEMIT (5 layers)	0.046	0.012	0.066	0.02	<1e-4

Table F.5: R^2 values for predicting **overall** score (raw average of rewrite, paraphrase, and neighborhood scores) from choice of edit layer and tracing effect, across editing problem variants. Diff shows the added effect of including tracing in the regression (Both vs. Layer Only), in terms of R^2, and p-value shows the results from an F-test comparing the Both and Layer Only models. The added effect of including tracing effects is very small across conditions (2% or less).

Ovr. Edit Score		R^2 Values				
Editing Problem	Method	Layer	Trace	Both	Diff	p-value
Error Injection	FT (1 layer)	0.642	0.054	0.643	0.002	0.001
	FT (5 layers)	0.663	0.047	0.665	0.002	0.001
	ROME (1 layer)	0.62	0.003	0.629	0.009	<1e-4
	MEMIT (5 layers)	0.525	0.008	0.534	0.009	<1e-4
Tracing Reversal	FT (1 layer)	0.294	0.025	0.296	0.002	0.054
	FT (5 layers)	0.751	0.045	0.752	0.001	0.032
	ROME (1 layer)	0.296	0.016	0.31	0.014	<1e-4
	MEMIT (5 layers)	0.21	0.036	0.216	0.006	<1e-4
Fact Erasure	FT (1 layer)	0.28	0.007	0.283	0.004	0.008
	FT (5 layers)	0.119	0	0.124	0.004	0.015
	ROME (1 layer)	0.718	0.023	0.718	0	0.729
	MEMIT (5 layers)	0.794	0.025	0.794	0	0.555
Fact Amplification	FT (1 layer)	0.188	0.003	0.199	0.011	<1e-4
	FT (5 layers)	0.224	0.002	0.236	0.013	<1e-4
	ROME (1 layer)	0.583	0.005	0.59	0.007	<1e-4
	MEMIT (5 layers)	0.597	0.005	0.607	0.01	<1e-4
Fact Forcing	FT (1 layer)	0.487	0.056	0.5	0.013	<1e-4
	FT (5 layers)	0.459	0.057	0.475	0.017	<1e-4
	ROME (1 layer)	0.285	0.004	0.291	0.006	<1e-4
	MEMIT (5 layers)	0.226	0.017	0.227	0.001	0.419

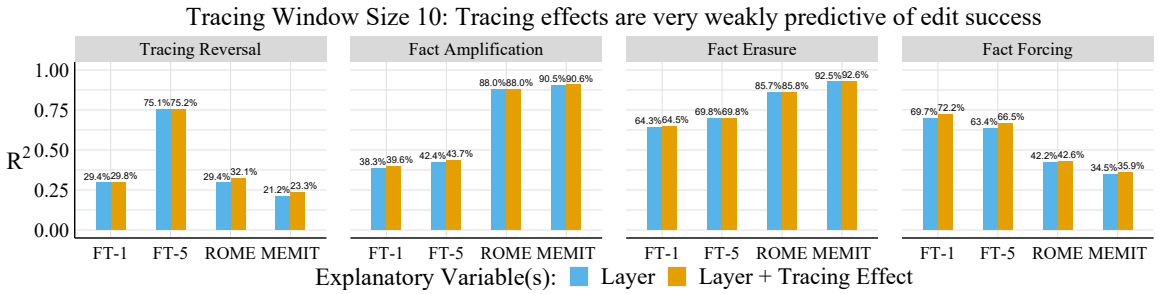


Figure F.6: The results of our R^2 analysis for predicting rewrite score are nearly identical between using a tracing window size of 5 (shown in Fig. 9.6) or 10 (shown here).

(Fig. F.10). We consider the last subject token effect since this corresponds more directly to the motivation for ROME (see Meng et al. [128]). We expand on each of these experiments below:

Results for Paraphrase, Neighborhood, Overall Metrics. We recreate our regression-based analysis across editing problem variants and editing methods using paraphrase score and neighborhood score as our outcomes rather than Rewrite Score, as well as an Overall Score that is the raw average of the three edit scores. These results are shown in Tables F.3, F.4, and F.5 respectively. Similar to our analysis with rewrite score, these tables show that tracing effects are barely predictive of edit success at all. For paraphrase score, the largest gains in R^2 values are around 0.03 (relative to the layer-only regression model), and for neighborhood score, the largest gain is 0.02. The largest gain for overall score is 0.02 for Fact Forcing with constrained finetuning. Our overall conclusion remains that tracing effects are almost totally unrelated to edit success across editing problem variants, including for different edit success metrics.

Results for Different Tracing Window Sizes. We repeat our analysis from Sec. 9.5 using tracing effects obtained from a larger tracing window size of 10, to match the value used in Meng et al. [128]. Note that from Fig. F.4, we know that the tracing effects grow larger as more adjacent layer representations are restored. When we recreate our main R^2 analysis using tracing effects with window size 10 (shown in Fig. F.6), we find that results are nearly identical to those shown in Tables F.2, F.3, and F.4.

Results for GPT2-XL. We rerun our analysis with GPT2-XL, a 48 layer model [161], while editing layers in the range {1, 5, 9, 13, 17, 18, 21, 25, 29, 33, 37, 41, 45, 48}. Here, we use a tracing window size of 10, and we limit our experiments to focus on Error Injection and Fact Forcing editing problems. As seen in Fig. F.7, we find very similar trends when explaining rewrite score in terms of the choice of edit layer and the tracing effect at that layer. The largest explanatory effects in terms of R^2 are observed for Fact Forcing with constrained finetuning, but these effects remain small at about 2%.

Results for Unscaled Metrics. We repeat our analysis using the original editing metrics and absolute tracing effects from Meng et al. [128]. Their rewrite magnitude is the absolute difference between the probability of the new target o_{false} and the old true target o_{true} after editing, $p_{\theta^*}(o_{false}|s,r) - p_{\theta^*}(o_{true}|s,r)$. The tracing effect is the absolute tracing effect, $p_\theta(o_{true}|s_{noise}, r, v_{(t,\ell)})$
$- p_\theta(o_{true}|s_{noise}, r)$, measured at the last subject token index. We adjusted our rewrite and tracing metrics to (1) rely only on the target output probability, rather than difference in probabilities of two different targets which might not be appropriate for our different editing

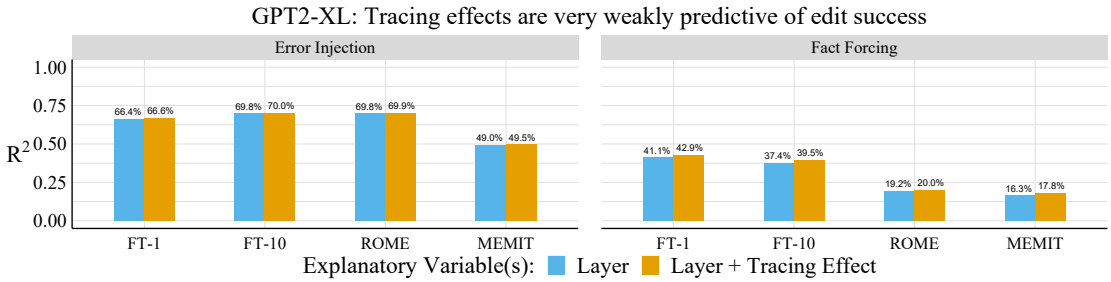


Figure F.7: Like with GPT-J, tracing effects are very weakly predictive of edit success across editing problem variants for **GPT2-XL** while Fact Forcing shows the largest relationship. Relative to the R^2 of a model predicting rewrite score based on the choice of edit layer (blue), a model with edit layer and tracing effects (orange) improves the R^2 by at most .02 points for Fact Forcing. The choice of edit layer explains a much greater share of the variance in rewrite score.

problems, and (2) to always fall between 0 and 1 for better comparability between datapoints, since absolute tracing effect are bounded by the original model probabilities. However, we reach the same conclusions from our analysis when using the original editing metrics. We show an example for rewrite magnitude and the absolute tracing effect for Error Injection in Fig. F.8. The correlation between edit success and tracing effect is still near zero.

Results for Last Subject Token Effect. ROME increases the target probability $p(o_{false}|s,r)$ by optimizing for a new output representation from a chosen MLP layer *at the last subject token index*. Meng et al. [128] show that this choice of token representation is critical to the success of the editing method, which is a hypothesis directly motivated by the results from their Causal Tracing analysis. In our paper, we by default report results using tracing effects that are the max across tokens at a given layer, for better comparability across the editing methods we use. However, when we repeat our analysis using the tracing effect specifically at the last subject token index, we obtain the same negative conclusions about the relationship between Causal Tracing localization and ROME editing performance. We show the correlations between Rewrite Score and Last Subject Token Tracing Effect in Fig. F.10, where we see there are no positive correlations between editing success and tracing results at any layer in GPT-J.

Edit Metric	Regression Metric	Predictor(s)	Value
Rewrite Score	R^2	Layer	0.947
		Tracing Effect	0.016
	RMSE	Layer	0.073
		Tracing Effect	0.315
	MAE	Layer	0.02
		Tracing Effect	0.206
Overall Score	R^2	Layer	0.618
		Tracing Effect	0.003
	RMSE	Layer	0.133
		Tracing Effect	0.216
	MAE	Layer	0.11
		Tracing Effect	0.183

Table F.6: **Additional regression error metrics** (for CounterFact and ROME) lead us to the same conclusion as our analysis based on R^2. RMSE is root mean squared error, and MAE is mean absolute error. Regressions predicting rewrite score (or overall score) from the choice of edit layer achieve much lower prediction errors than regressions using the tracing effect, suggesting that the choice of edit layer is much more important for edit success than the tracing effect.

Edit Metric	Regression Metric	Predictor(s)	Value
Rewrite Score	R^2	Layer	0.795
		Tracing Effect	0.042
	RMSE	Layer	0.158
		Tracing Effect	0.341
	MAE	Layer	0.072
		Tracing Effect	0.254
Overall Score	R^2	Layer	0.654
		Tracing Effect	0.059
	RMSE	Layer	0.136
		Tracing Effect	0.223
	MAE	Layer	0.097
		Tracing Effect	0.188

Table F.7: **ZSRE regression results** lead us to the same conclusion as our experiments on CounterFact, using ROME editing. RMSE is root mean squared error, and MAE is mean absolute error. Regressions predicting rewrite score (or overall score) from the choice of edit layer achieve much lower prediction errors than regressions using the tracing effect, suggesting that the choice of edit layer is much more important for edit success than the tracing effect.

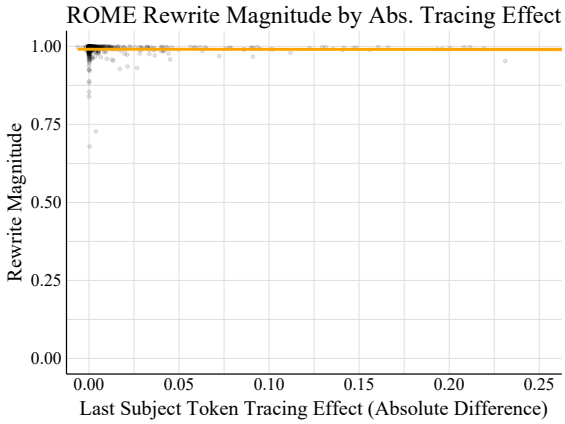


Figure F.8: Editing vs. tracing results for ROME at layer 6 for Error Injection, using the un-rescaled rewrite and tracing metrics from Meng et al. [128]. Here, rewrite magnitude is the difference between the probability of the new target o_{false} and the old true target o_{true} after editing, $p_{\theta^*}(o_{false}|s,r) - p_{\theta^*}(o_{true}|s,r)$. The tracing effect is the absolute tracing effect, $p_\theta(o_{true}|s_{noise},r,v_{(t,\ell)}) - p_\theta(o_{true}|s_{noise},r)$, measured at the last subject token index. The correlation here is near zero, at $\rho = -.006$.

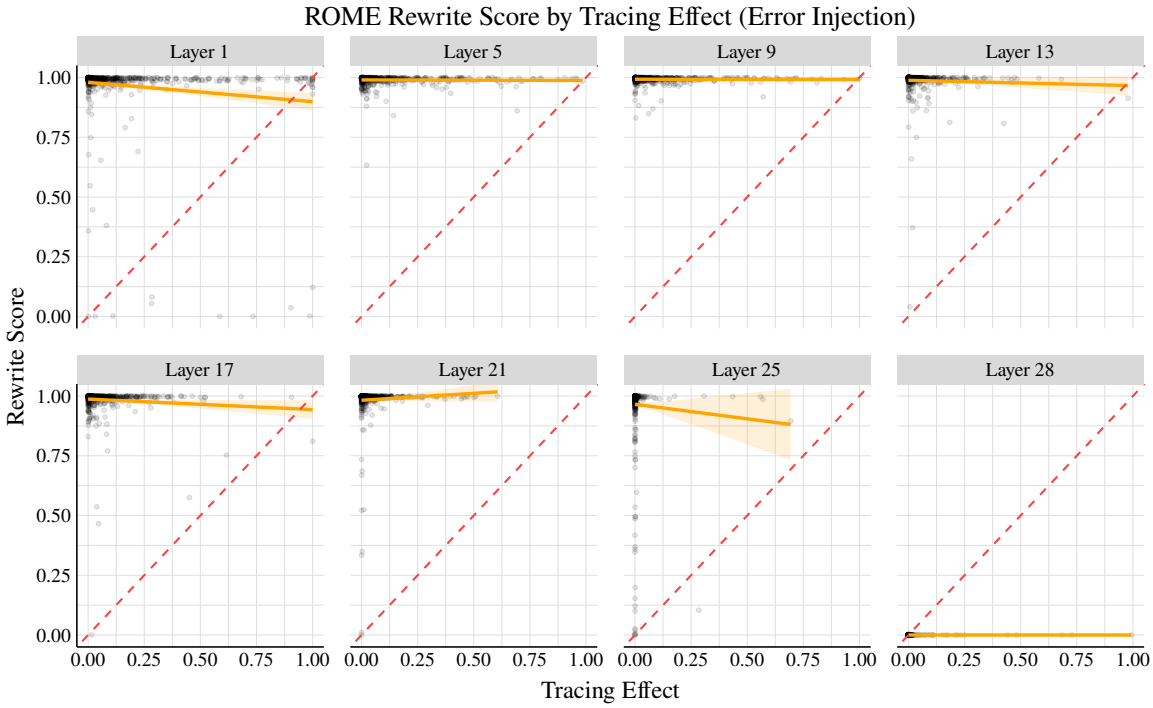


Figure F.9: The relationship between ROME edit success and the tracing effect is near zero at most edit layers in the model (for the standard Error Injection editing problem). Red lines show perfect relationships between tracing effect and edit success.

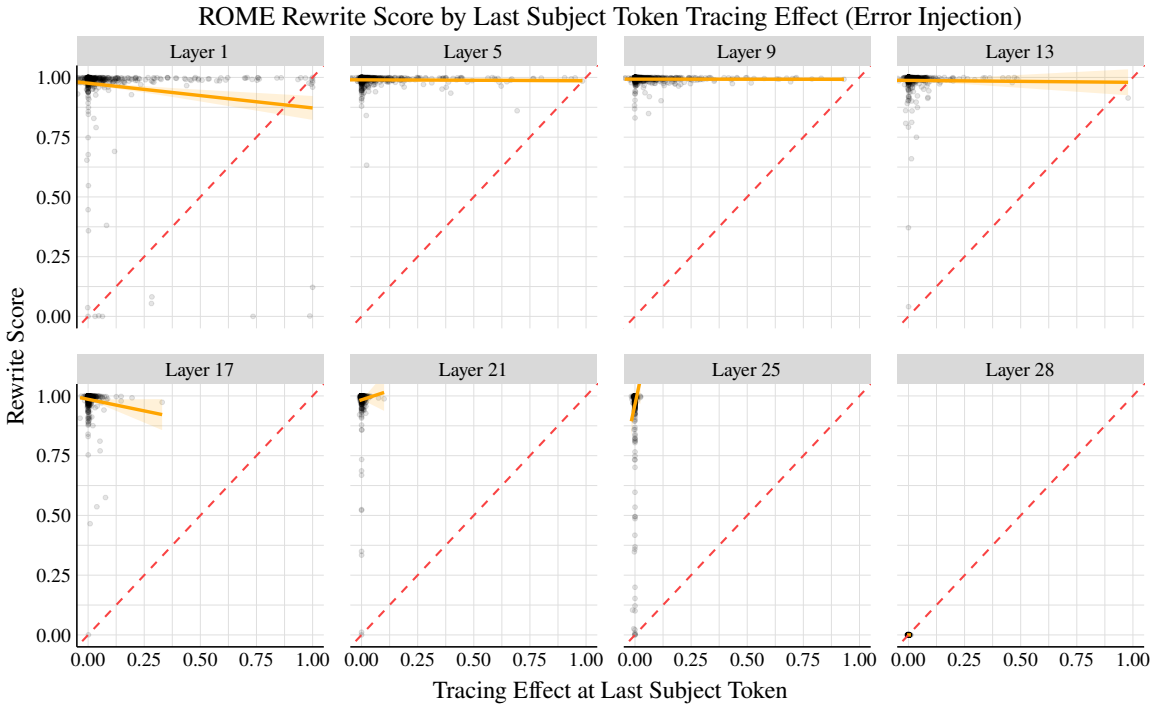


Figure F.10: The relationship between ROME edit success and the tracing effect at the last subject token. The ROME method edits a fact by changing the output representation for the MLP layer specifically at the token index corresponding to the last subject token. However, editing performance and tracing effect at this position still do not positively correlate. Note the distribution of points along the x axis changes depending on the choice of edit layer since the distribution of tracing effects is calculated from tracing effects *at that layer*.

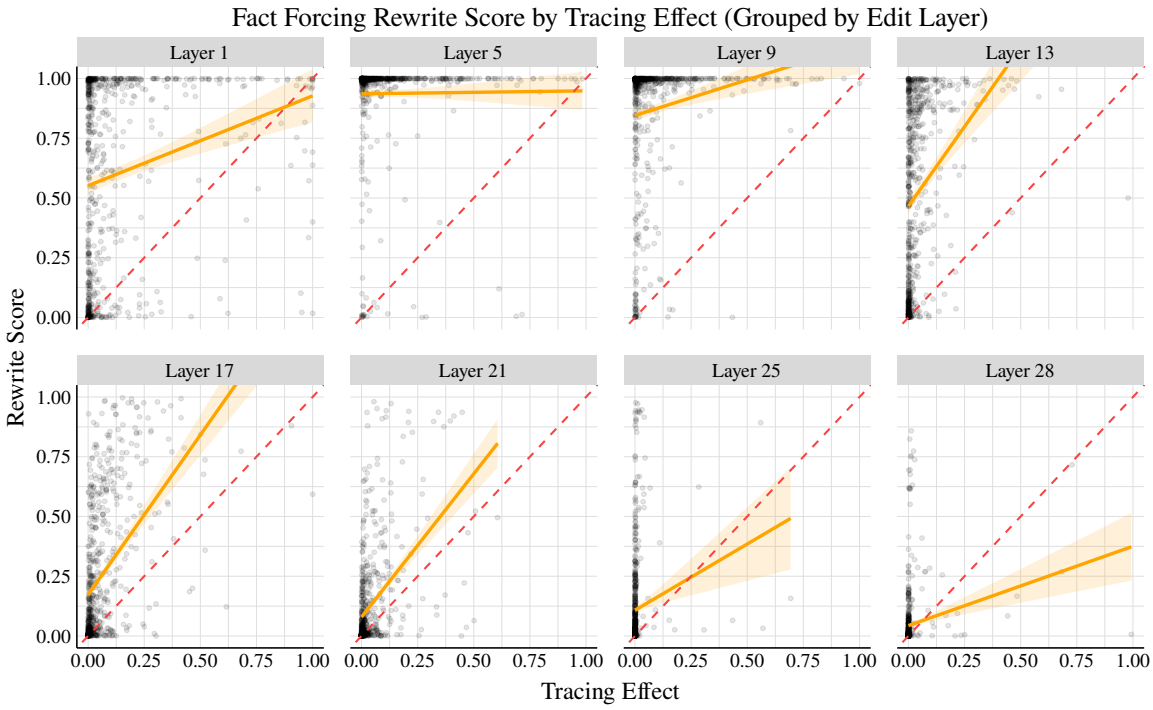


Figure F.11: The relationship between Fact Forcing edit success and the tracing effect for constrained finetuning of 5 adjacent layers. "Layer ℓ" indicates the center of this 5-layer interval, and the dashed red lines show a hypothetical perfect relationship between tracing effect and edit success. For many layers, there is a noticeable positive relationship between tracing effects and editing success. Yet, (1) there is a high amount of variance in the outcome, and (2) this variance is largely explained by the edit layer. As a result, tracing effects provide little extra information for predicting edit success beyond the choice of edit layer (about 3% more explained variance; see Fig. 9.6).

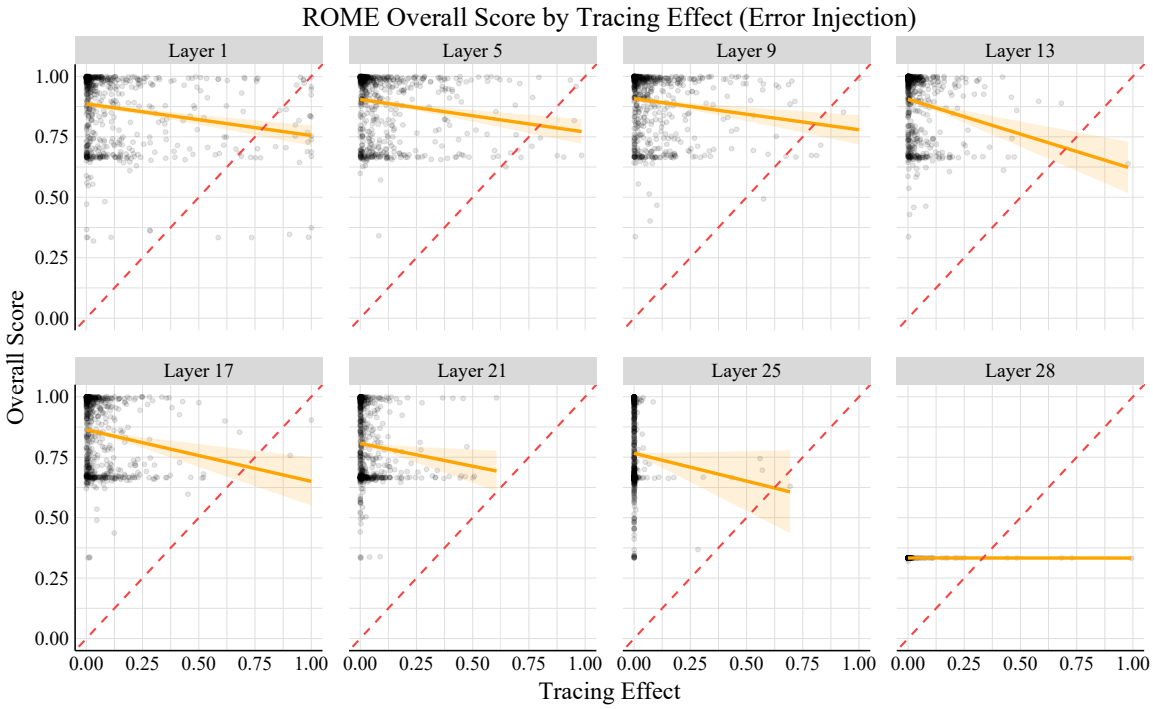


Figure F.12: The relationship between ROME **overall score** (average of rewrite/paraphrase/neighborhood scores) and the tracing effect is somewhat negative for most edit layers in the model (for the standard Error Injection editing problem). Red lines show a perfect relationship between tracing effect and edit success, so a negative relationship suggests that tracing localization results do not indicate that editing will be successful.

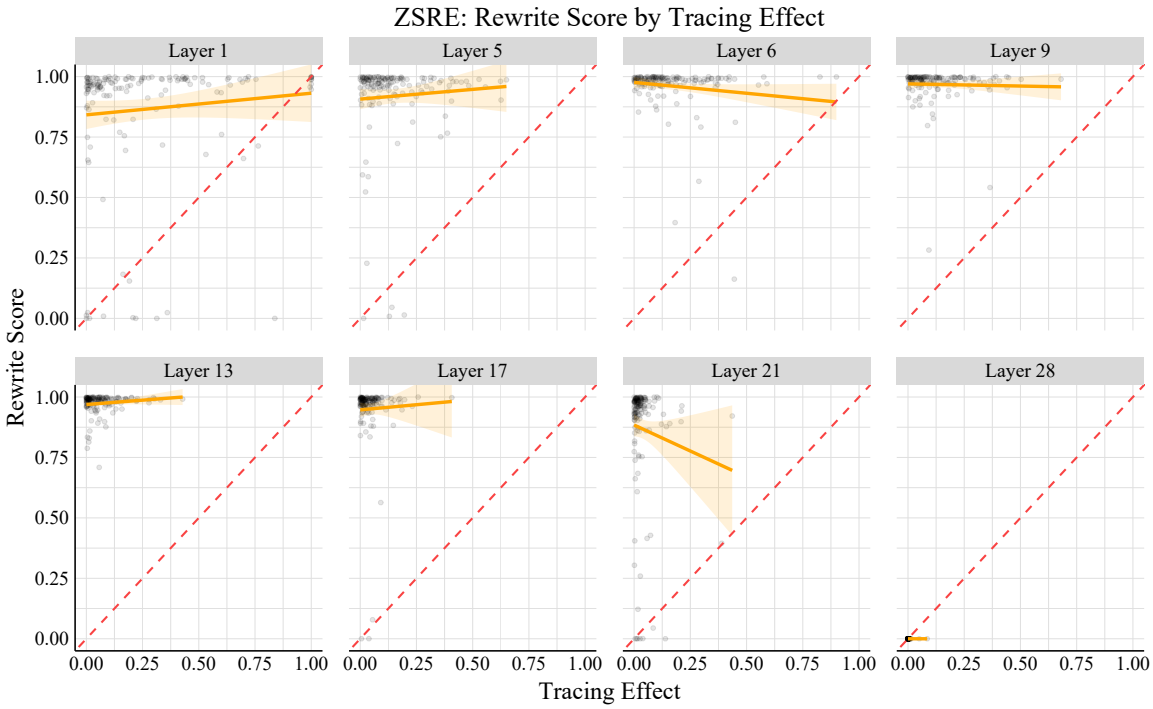


Figure F.13: Additional experiments on the **ZSRE** dataset show the same results as for CounterFact, using the ROME editing method with rewrite score as our editing success metric (see regression analysis results in Table F.7). Red lines show a perfect relationship between tracing effect and edit success, so near-zero relationships suggest that tracing localization results do not indicate that editing will be successful.

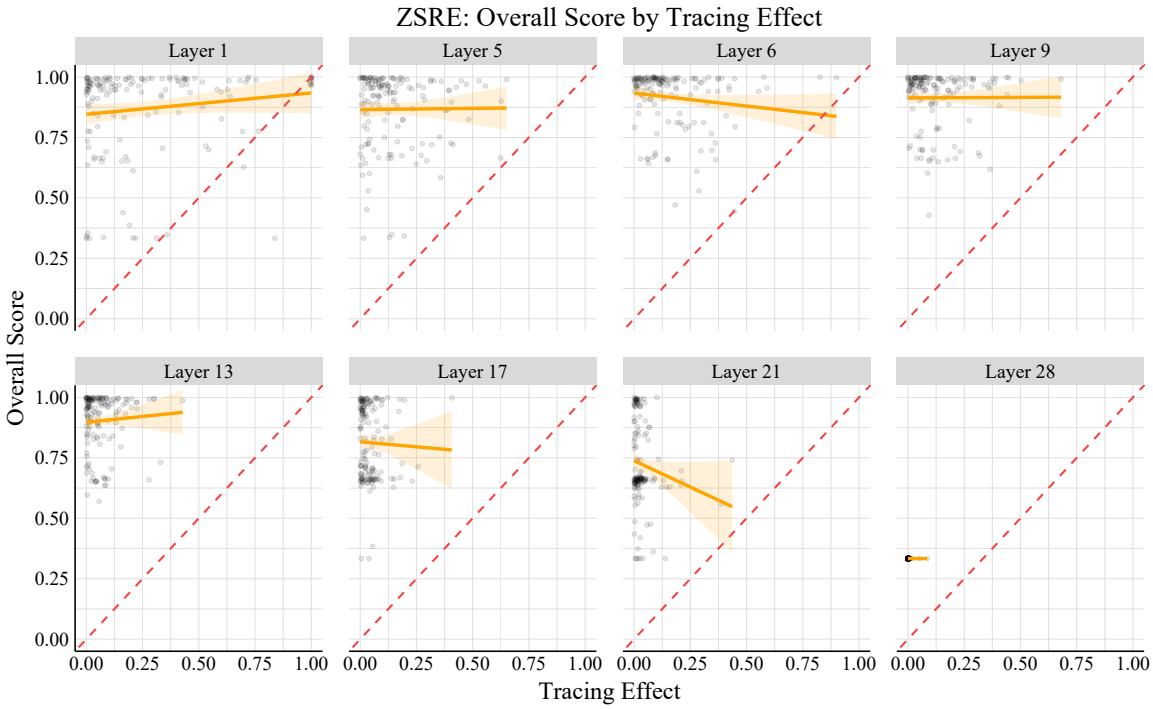


Figure F.14: ZSRE experiments using overall score (average of rewrite/paraphrase/neighborhood scores) as the edit success metric.

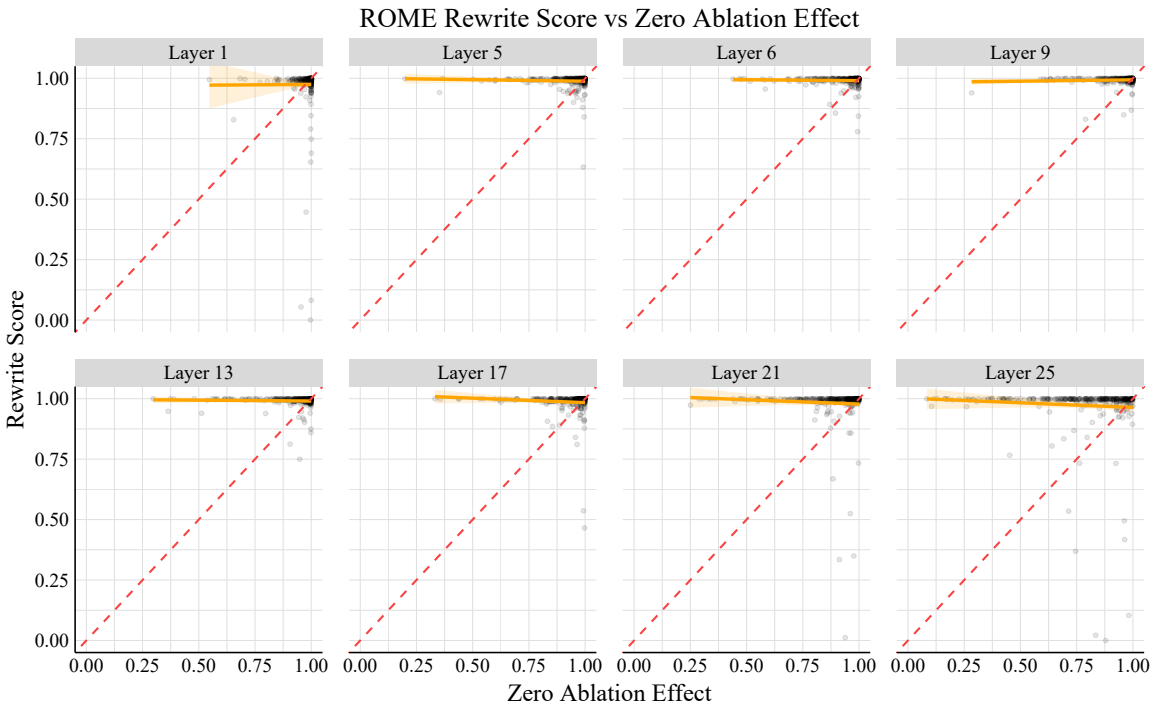


Figure F.15: Additional experiments with **representation zeroing** as the localization method show the same results as for Causal Tracing, using the ROME editing method and **rewrite score** as the edit success metric. Red lines show a perfect relationship between representation zeroing and edit success, so near-zero relationships suggest that representation ablation localization results do not indicate that editing will be successful.

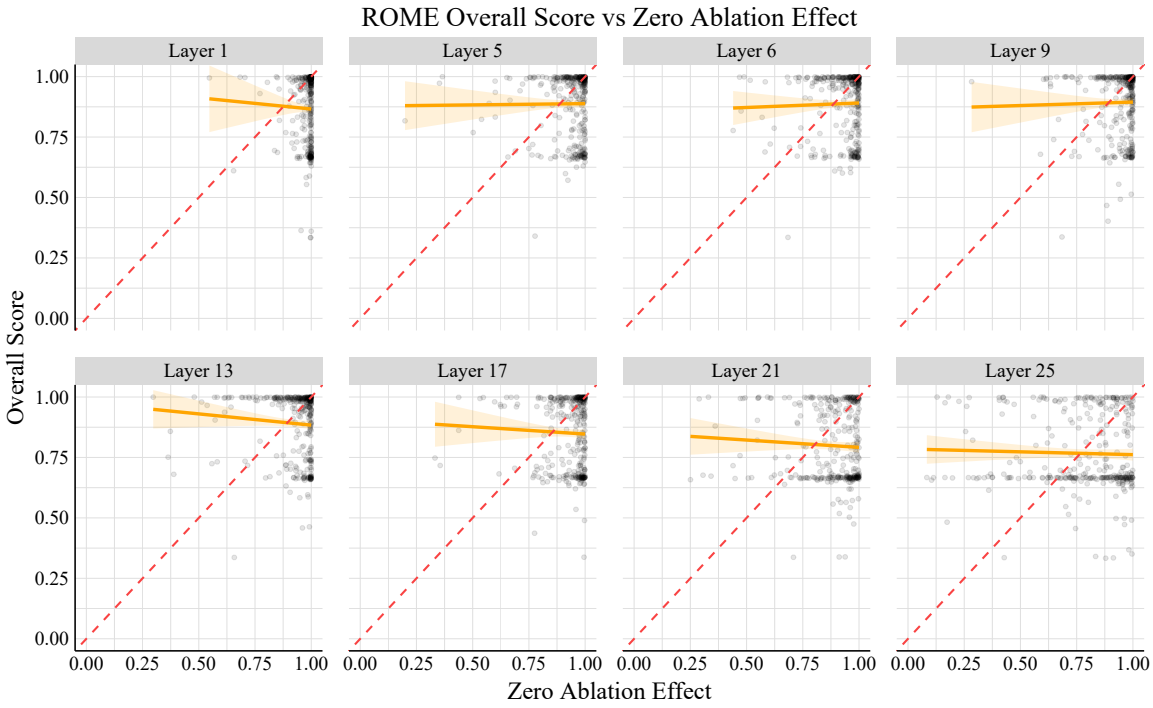


Figure F.16: Additional experiments with **representation zeroing** as the localization method show the same results as for Causal Tracing, using the ROME editing method and **overall score** as the edit success metric. Red lines show a perfect relationship between representation zeroing and edit success, so near-zero relationships suggest that representation ablation localization results do not indicate that editing will be successful.