COLLABORATIVE DEVELOPMENT OF NLP MODELS

Fereshte Khani
Microsoft
fkhani@microsoft.com

Marco Tulio Ribeiro
Microsoft
marco.correia@microsoft.com

ABSTRACT

The effective development of NLP models requires specifying their behavior on specific concepts and ensuring high performance on those concepts. This is important to encode business rules, correct undesirable behavior, and ensure alignment with user values. However, it's hard to generate data for a concept that does not lead to shortcuts or interference with the original data or other concepts. In this paper we propose CoDev, a framework for collaborative development of NLP models. Our main insight is learning a *local* model for each concept, and a *global* model to integrate the original data with all concepts. We then steer a large language model such that it generates in-concept instances where local and global disagree, i.e. regions that haven't been yet learned or properly specified. Our experiments show CoDev is effective at helping users operationalize concepts and avoid interference for a variety of scenarios, tasks, and models.

1 Introduction

Teaching *concepts* to models is often necessary, either to encode certain values or business logic, or to fix existing failures. A concept relates a set of inputs to desired behaviors, e.g. "religion does not connote sentiment" informs the effect that religious words should have on sentiment prediction (none). Similarly, in the context of natural language inference (NLI), the broader concept of "downward monotonicity" specifies entailment relations when certain parts of expressions are made more specific, e.g. ("All cats like tuna", "All small cats like tuna").

The standard way of teaching concepts to models is adding new training data that exemplifies the concept, e.g. adding neutral sentences containing religious words Dixon et al. [2018], or adding entailment pairs that illustrate downward monotonicity Yanaka et al. [2020]. However, it is hard to ensure that the data provided does not to lead to *shortcuts*, i.e. spurious correlations or heuristics that allow models to make predictions without capturing the true underlying concept or logic, such as "all sentences with religious terms are neutral", or "going from general to specific leads to entailment". Further, the model may *overfit* – fail to generalize from the instances provided to the actual concept, e.g. only recognizing pairs in the form ("all X...", "all ADJECTIVE X..."), and not pairs like ("all animals eat", "all cats eat"). Finally, both shortcuts and overfitting can lead to *interference* with the original data or other concepts, e.g. leading to failures on "I love Islam" or pairs like ("Some cats like tuna", "Some small cats like tuna"). In sum, operationalizing concepts is challenging, since users typically cannot think of all concept boundaries and interactions in advance.

One possible direction is to ask domain experts to create data that covers the concept as comprehensively and precisely as possible, e.g. the GLUE diagnostics dataset Wang et al. [2018a] or the FraCaS test suite Cooper et al. [1996]. However, these datasets are often expensive to create, small (and thus not suitable for training), and not exhaustive, as even experts still fail to think about all aspects and nuances of a concept Ribeiro and Lundberg [2022]. Another direction having users provide data incrementally while they receive feedback from the model, e.g. adversarial training Kiela et al. [2021] or adaptive testing Ribeiro and Lundberg [2022]. These do not require users to think about everything in advance, and can expose and correct model weaknesses. However, adversarial training does not include the notion of concepts explicitly, and adaptive testing does not address the interaction between different concepts or between a concept and the original data. As a result, users may not explore concept *boundaries* efficiently, and thus do not know when they have sufficiently covered a concept or whether they have introduced interference with other concepts.

In this paper, we introduce Collaborative Development (CoDev), a framework that allows users to collaborate with AI systems *and* other users in specifying concepts and teaching them to models. Our key insight is making concepts

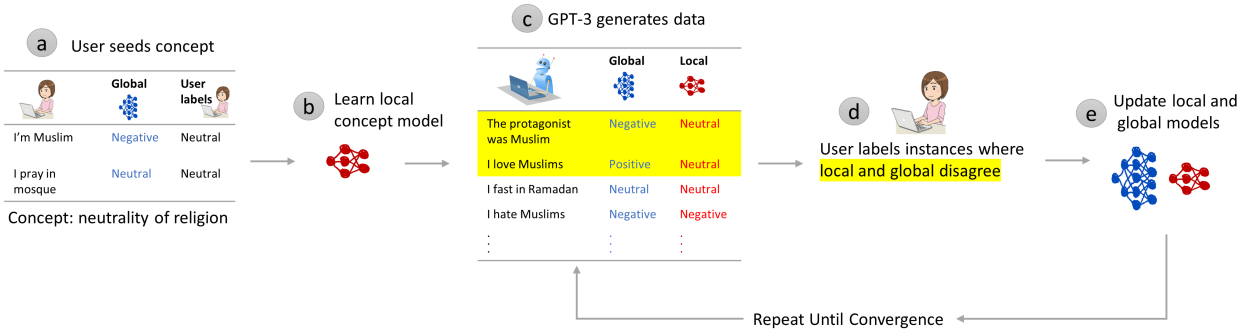


Figure 1: CoDev loop for operationalizing a concept. (a) The user starts by providing some seed data from the concept and their labels, (b) they are used to learn a local concept model. (c) GPT-3 is then prompted to generate new examples, prioritizing examples where the local model disagrees with the global model. (d) Actual disagreements are shown to the user for labeling, and (e) each label improves either the local or the global model. The loop c-d-e is repeated until convergence, i.e. until the user has operationalized the concept and the global model has learned it.

explicit by learning a *local* model for each concept, in addition to a *global* model that integrates the original data and all additional concepts. When operationalizing a new concept, we rely on a large language model (LLM) to generate new instances (similar to Ribeiro and Lundberg [2022]), and ask users to label examples in the *disagreement region* between the local concept model and the global model (Figure 1). Intuitively, these examples are either cases not yet learned by the global model, or cases where the concept is not yet properly specified. As users label these examples, both models are updated until convergence, i.e. until the concept has been learned in a way that does not conflict with prior data or prior concepts. In a sense, each local model is an ever-improving cheap proxy for the user in its own concept, whose speed of prediction allows users to explore the boundaries between concepts and existing data.

Our experimental results demonstrate the effectiveness of CoDev in operationalizing concepts and handling interference. We first run a simplified version of CoDev, where instead of using GPT-3 as a generator we iteratively select examples from an unlabeled pool for labeling. We show that CoDev outperforms random sampling and uncertainty sampling in teaching a state-of-the-art NLI model about downward-monotone and upward-monotone concepts Wang et al. [2018a], as well as teaching a high-accuracy roberta-large base model about four other Amazon products. We then demonstrated that CoDev can perform well even with biased seed data, outperforming a model that relies solely on biased data collection. Finally, we evaluated CoDev with users in the loop and showed that it outperformed AdaTest Ribeiro and Lundberg [2022], a state-of-the-art tool for debugging NLP models that also uses GPT-3. In a pilot study, we demonstrated that CoDev helped users clarify their concepts.

2 Setup

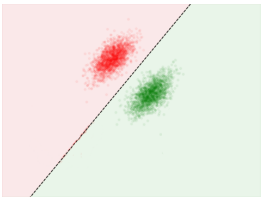
Let x be a text string, and y a categorical label, e.g. sentiment (positive, negative, neutral). We assume there is a true function $f(x) = y$ that we aim to approximate with a model $\hat{f}(x)$, and further assume we have access to a "base" dataset $D_0 = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, e.g. of movie reviews, from base distribution P_0. We refer to the model trained on D_0 as the base model $\hat{f}_0$.

Concepts. A concept C_i is associated with a distribution P_i over the input space, e.g. a concept might place probability mass exclusively on sentences containing religious words. We say $x \in C_i$ if $P_i(x) > 0$. Since it's hard for users to be exhaustive, we do not assume users can generate from P_i, but that they can label any x with $f(x)$, and as to whether it is in C_i or not. We further assume users can provide a very small number samples in the support of P_i.

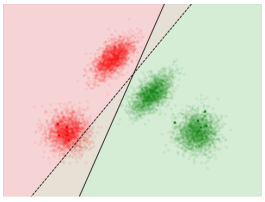
Objective. Without loss of generality, we assume that we have k users developing a model collaboratively, each with their own concept. Our goal is to train a model $\hat{f}$ that does well on both the base distribution and all concepts, i.e., minimizing the following objective,

$$\frac{1}{k+1} \sum_{i=0}^{k} \mathbb{E}_{x \sim P_i}[\hat{f}(x) \neq f(x)]. \tag{1}$$

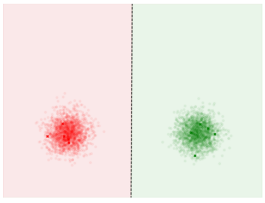
3 Collaborative Development of NLP models



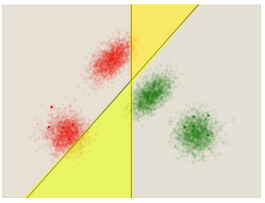
(a) The initial data (P_0) is elliptical Gaussian



(b) The user concept (P_i) is spherical Gaussian



(c) local function for the user data



(d) disagreement regions between local and global models

Figure 2: (a) a model is trained on two elliptical Gaussian (b) as a result the model is tilted (dashed line) in comparison to the true classifier (solid line). The spherical Gaussian are showing a user concept, who is interested to find bugs in the model and teach the model about the new concept. However, since the model (dashed line) have high accuracy on the user concept it is hard to find bugs and teaching the model requires data from the low probability region. (c) we fit a classifier to the Spherical Gaussian which can be done with only a few data points, (d) we then focus on disagreements between these two models to find bugs in the user concept.

In this section we describe how a single user operationalizes their concept (i.e. produces a dataset D_i) in the context of an existing global model $\hat{f}$ trained on the base dataset D_0 and previous concepts $D_{1:i-1}$.

If $\hat{f}_0$ could already "solve" the concept, i.e. $\hat{f}_0(x) = f(x)$ for all $x \in C_i$, we would be done and there would be no need for D_i. Thus, what we really want is for D_i to specify the boundary around "failures", cases not currently handled by $\hat{f}_0$. We abstract away the choice model and learning procedure, assuming that the model can learn the concept given D_i (in practice, we use neural networks that are expressive enough).

3.1 Sampling from the concept

Since by assumption we cannot sample from the concept distribution P_i, it is a challenge to find regions of P_i where $\hat{f}_0$ fails. To address this, we use GPT-3 Brown et al. [2020] as a generator $\mathcal{G}$ to simulate a random walk within the concept. To do so, we construct a prompt with m in-concept examples, and use this prompt as input to $\mathcal{G}$ to generate more samples. Then, we ask the user in the loop to accept or reject each generated sample x' (x' is accepted if $x' \in C_i$), and also to label the accepted x' with $f(x')$. The value of m controls the tradeoff between precision and recall, with high m generating more in-concept examples and low m exploring the space more broadly.

Under some conditions it can be shown that $\mathcal{G}$ simulates a Markov chain with stationary distribution P_i (Appendix A), but the weaker condition of connectivity suffices for finding the concept failures, i.e. there must be a path between any $x', x'' \in C_i$ with nonzero transition probabilities according to $\mathcal{G}$ and the prompt. That is, if the concept is connected, with enough time we should be able to find regions of P_i that are not already learned by $\hat{f}_0$.

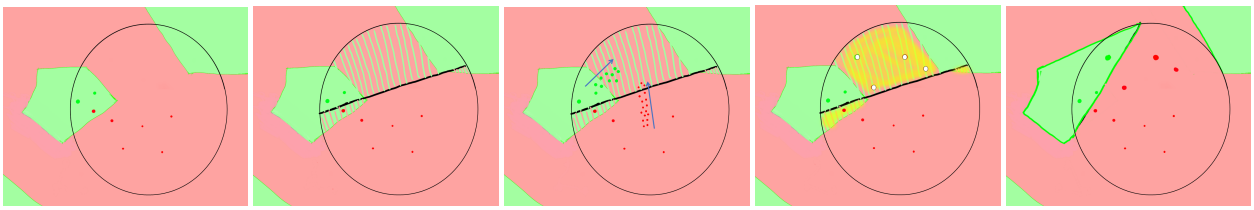
While sampling from $\mathcal{G}$ *eventually* leads to the yet-unlearned concept regions, it is an inefficient use of human effort, as it does not use the user labels to guide generation (i.e. the user has to label many examples that the current model already handles correctly). A better approach would be ask the user to label in a way that maximizes the expected information gain for the concept, to which we now turn.

3.2 Local Concept Models

Complex functions can be approximated by simpler functions in a local neighborhood, as evidenced by theoretical results (e.g. Taylor expansion) and empirical applications Ribeiro et al. [2016], Lundberg and Lee [2017]. Since a concept is a natural local neighborhood, we use this insight and learn a *local* model $\hat{f}_i$ to approximate $f(x)$ in C_i.

We present a toy example for intuition in Figure 2, where we show toy distributions P_0 (Figure 2a) and P_0 with an additional concept P_1 (Figure 2b). In 2b, $\hat{f}_0$ learned on samples from P_0 (dashed line) predicts most of P_1 correctly, except for a small region in the bottom left quadrant. However, we would need *many* random samples from P_1 in order to learn that region, and reach the best model $\hat{f}$ (solid line in Figure 2b). In contrast, we can learn a good local model $\hat{f}_1$ for P_1 with a trivial number of samples (Figure 2c). This local model can be used to produce a *disagreement region* between $\hat{f}_1$ and $\hat{f}_0$ (Figure 2d), and sampling from that region would lead to convergence much faster.

More generally, we define a score function as the disagreement between the local and global function. This score function is used to steer generation such as to maximize the score of generated samples x', by adding instances to the prompt for $\mathcal{G}$ with probability proportional to their score (similar to Ribeiro and Lundberg [2022], who use a different score function). We note that models may present false agreement on some samples, i.e. $\hat{f}_i(x') = \hat{f}_0(x') \neq f(x')$. To account for this, we also sample from the agreement region sometimes, with a probability that decays over time as we gain more confidence in the local model.



(a) seed (b) local fit (c) generation (d) selection (e) update

Figure 3: This figure illustrates the main steps debugging in CoDev: (a) The user starts by providing a small number of data points and their labels within their domain (represented by the black circle), (b) we fit a simple model (shown as a linear model) to represent the user's concept, (c) we use the generator ($\mathcal{G}$) to generate data points towards the region where the local model disagrees with the global model, (d) a diverse set of data points are selected for the user to label, (e) based on the user's feedback, the local and global models are updated until convergence.

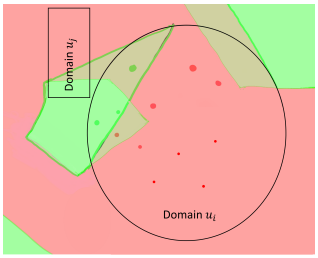


Figure 4: Comparing the first and last figure in 3 shows that the change in the local region had lead to other changes outside the concept. Those regions can belong to some other concepts as well cause interference.

3.3 Operationalizing a concept: from disagreement to convergence

The local and global models disagree on regions where the concept has not yet been learned (global is wrong) or the local model does not fit the user's concept correctly (local is wrong). Thus, every label from the disagreement region results in an improvement in whichever model is incorrect. As labeling progresses, we update both the concept model $\hat{f}_i$ and the global model $\hat{f}$ until we reach convergence (i.e. $\hat{f}$ has learned the concept C_i). Note that while $\hat{f}_i$ is trained on data from P_i, $\hat{f}$ is trained to do well on the base distribution P_0, as well as on all concepts $P_{1:k}$.

We present pseudo-code for operationalizing a concept in Algorithm 1, and illustrate it in Figure 3. The local and global models are initialized in lines 1-2 (Figure 3a and b). In line 5 (Figure 3c), $\mathcal{G}$ is prompted such as to generate in-concept samples that are likely to be in the disagreement region. Then, in line 6 (Figure 3d), we select a batch of b examples for labeling according to the disagreement score of the generated instances. This generation-labeling loop is repeated for L iterations, after which both the local and global models get updated in lines 7-8 (Figure 3e). For the sake of interactivity, in practice we do not train from scratch when updating the models, and instead finetune them from a previous checkpoint. After a few iterations, $\hat{f}$ and $\hat{f}_i$ converge (i.e. the user has operationalized the concept and the global model has learned it). If the number of generated samples $b * L$ is large enough, we can assume that no disagreement on generated sample between $\hat{f}$ and $\hat{f}_i$ means they have converged, and thus we stop and output the generated data D_i (line 9).

3.4 Handling interference between concepts

Any updates made to one region of the global model can have an impact on other regions (see Figure 4 for intuition). Having local functions (cheap experts) enable us to check interference efficiently. Every time that a user operationalizes

Algorithm 1: Operationalizing a new concept i

input : Base dataset D_0
Concept datasets $D_{1:i-1}$
A small set of samples from concept D_i

1 **Init local:** Train $\hat{f}_i$ on D_i;
2 **Init global:** Train $\hat{f}$ on $D_{0:i-1}$;
3 **do**
4 **for** L *iterations* **do**
5 **Generation:** Prompt $\mathcal{G}$ with subset from D_i chosen with probability $\propto |\hat{f}(x) - \hat{f}_i(x)|$;
6 **Labeling:** Select b samples with prob. $\propto |\hat{f}(x') - \hat{f}_i(x')|$. Users reject x' if out of concept, or add to D_i with label $f(x')$;
7 **Update local:** Train $\hat{f}_i$ on D_i;
8 **Update global:** Train $\hat{f}$ on $D_{0:i}$;
9 **while** D_i *was updated this round*;

output : A dataset D_i

a concept according to Algorithm 1, we check the resulting global model $\hat{f}$ against the local concepts for all previous concepts. To do so, we re-run Algorithm 1, only asking the user for labels if there is a newfound disagreement region. In practice, this means that a user adding a new concept needs to make sure it does not break any concepts from other users, a process similar to regression testing in software engineering.

While we want to handle interference, having a multiplicity of concepts is beneficial in exposing false agreement regions between the global model and any individual concept model. In other words, while both $\hat{f}$ and $\hat{f}_i$ may be relying on the same shortcut to solve some regions of P_i, it is unlikely that this shortcut does not cause disagreement between $\hat{f}$ and all local models $\hat{f}_j$ for $j \neq i$. In this case, the interference caused by adding C_j is actually beneficial, as it further refines concept C_i.

In practice the original dataset (D_0) is often very large and we cannot fine-tune the model on $D_{0:k}$. Instead of choosing the whole D_0, every time we sample data points with highest disagreement between $\hat{f}_0$ and $\hat{f}$ from D_0. In other word, we treat $\hat{f}_0$ as a user with concept distribution P_0, this enable us to deal with interference with original model as well (as an example in Figure 2 we might only choose the data points from elliptical Gaussian in disagreement region).

4 Experiments

In 4.1 we run experiments with a simplified version of CoDev, where instead of using GPT-3 as a generator we iteratively select examples from an unlabeled pool for labeling. Since we do not need users in the loop, this allows us to compare the data selection mechanism of CoDev to baselines such as random selection and active learning (noting that a generator is needed to actually operationalize a concept in realistic scenarios). In 4.2 we add GPT-3 back as a generator, but use a high performing model as an oracle (once again bypassing the need for user labels) in a controlled experiment where we start the concept from a biased set. Finally, in 4.3 we evaluate CoDev with users in the loop, and compare it to AdaTest Ribeiro and Lundberg [2022], a state-of-the-art tool for debugging NLP models that also uses GPT-3.

4.1 CoDev on unlabeled data

We adapt CoDev so that it selects instances from a pool of unlabeled data based on the disagreement score rather than using GPT-3 as a generator (lines 5-6 in Algorithm 1). We compare CoDev with two data selection baselines: random selection and uncertainty sampling Lewis [1995], Nguyen et al. [2022]. Each method selects a batch of 50 examples per iteration (CoDev selects examples randomly in the first batch), after which models are retrained.

Operationalizing a concept. We use RoBERTa-Large Liu et al. [2019] finetuned on MNLI (binary) Wang et al. [2018b] as a base model, and use the downward-monotone concept from an NLI monotonicity dataset Yanaka et al. [2020] as a pool of unlabeled data. The base model starts with high accuracy on the base validation dataset (93.5%), and low accuracy on the concept (23.5%). We present accuracy on the concept over iterations in Figure 5. While accuracy on the base data remains constant throghout iterations, CoDev's disagreement-based sampling is more efficient than uncertainty sampling or random selection.

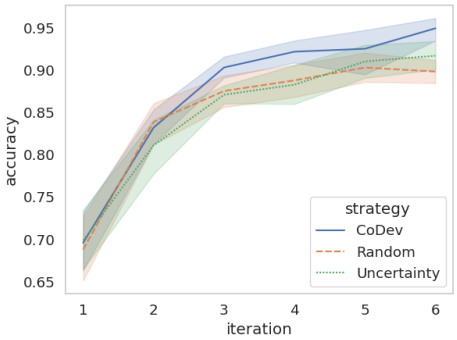


Figure 5: CoDev outperforms other data selection baselines when learning downward-monotone concept in MNLI task.

	MNLI		Amazon	
	Overall average	Worst case	Overall average	Worst case
Random sampling	89.6	87.7	87.6	85.3
Uncertainty Sampling	91.6	90.6	87.9	85.9
CoDev	91.9	90.7	88.2	86.0

Table 1: CoDev outperforms data selection baselines when simultaneously learning two concepts for MNLI (upward and downward monotone) and 4 concepts (product categories) for sentiment analysis. Results shown are an average of 10 runs.

Avoiding interference In order to evaluate interference between concepts, we try to learn various concepts simultaneously. We use the same NLI model and concept as the previous paragraph, and add upward monotone Yanaka et al. [2020] as an additional concept. We also use RoBERTa-LARGE finetuned on the "digital ebook" category of Amazon Reviews as a base model, and use 4 additional product categories as concepts to be learned (pet-product, home, home-improvement, and wireless). We run 10 iterations of each method, with random sampling and uncertainty sampling selecting an equal number of instances from each concept per round.

Table 1 shows the per-concept average and worst case accuracy of each method, where we note that CoDev produces better models on average.

4.2 CoDev with simulated user

Here we add GPT-3 back as a generator, but avoid the need of user labels by training a high accuracy model as an oracle to simulate user labels. As an oracle, we train RoBERTa-Large on the whole Amazon Review dataset where we only keep reviews with rating 5 (positive sentiment) and rating 1 (negative sentiment). The accuracy of oracle on validation dataset is 98.6%. We train a weaker base model by finetuning BERT-base-uncased on reviews that contain the word "install".

Our simulated concept consists of reviews containing the phrases "battery life" or "my skin" (from now on, we refer to this concept as Skin-Battery, or SB). We split these reviews into a development set (to initialize CoDev and baselines) and a test set. We simulate the scenario where the user starts with a biased subset of the concept, by initializing the

	biased SB	SB
Base	86.7 ± 2.5	82.6 ± 1.7
Random sampling	98.6 ± 0.9	80.7 ± 1.6
CoDev	94.9 ± 1.7	94.5 ± 1.1

Table 2: A comparison between CoDev and Random sampling when the dataset is biased. We have access to a biased dataset where reviews about skin are always negative and reviews about battery are always positive (biased-SB). CoDev outperforms random sampling in the whole dataset that contains all the reviews about skin and battery (SB) by effectively conceptualizing concepts and avoiding shortcuts.

Concept	Examples	Example of bugs found by CoDev
X person = not X person	How can I become a positive person? How can I become a person who is not negative?	predicts duplicate (underfit bugs): How can I become a mysterious person? / How can I become someone with no mystery? predicts non-duplicate (overfit bugs): How can I become a blind person? / How can I become someone who has lost his (physical) vision?
Modifiers changes question intent	Is Mark Wright a photographer? Is Mark Wright an accredited photographer?	predicts not-duplicate (underfit bugs): Is he an artist? / Is he an artist among other people? predicts duplicate (overfit bugs): Is Joe Bennett a famous court case? / Is Joe Bennett a famous American court case?

Table 3: Examples of bugs found by CoDev in the concepts introduced by CheckList, which were subsequently "debugged" using AdaTest, demonstrating that AdaTest had not yet fully operationalized these concepts. For each concept, we show an example of overfit bug, where the model has fit too closely to the concept's data and fails to generalize to similar data points, as well as an example of an underfit bug, where the model relies on some shortcuts related to concept's data and misclassifies similar examples.

	C_{orig}: "X = not antonym (X)", C_{new}: "Modifiers changes question intent"		C_{orig}: "X = synonym (X)", C_{new}: "less X = more antonym (X)"	
	CoDev	AdaTest	CoDev	AdaTest
broken by new concept	7/50	24/50	9/50	18/50
fixed by new concept	5/50	2/50	20/50	18/50

Table 4: Comparison of CoDev and AdaTest in terms of handling interference. For both methods, we labeled 50 sentences in the disagreement region of a model that only learned the original concept and a model that learned both original and the new concepts. CoDev outperforms AdaTest when the new concept conflicts (top) or is similar (bottom) to the original concept.

concept with 10 instances such that instances with "battery life" are always positive, and those with "my skin" are always negative. We then run CoDev with the oracle for 5 rounds adding 10 data points in each round. As shown in 2, CoDev is able to achieve high accuracy on all of SB, despite starting with a biased sample, while the same number of random instances from biased SB increases accuracy on it to the detriment on accuracy on the whole concept.

Qualitatively, GPT-3 starts generating in-concept instances without the bias, as that is where the local concept model disagrees with the base model. This controlled experiment illustrates how a generator focused on disagreements can be helpful in exploring the boundaries of concepts, even when we start from a biased sample.

4.3 CoDev with real users

Operationalizing concepts. We compare CoDev to AdaTest Ribeiro and Lundberg [2022], using data and concepts made available by Ribeiro and Lundberg [2022]. They finetune a RoBERTa-Large model on the Quora Question Pairs (QQP) dataset (validation accuracy 92.2%), and then iteratively add data with GPT-3 (adaptively trying to find failures) until they "fix" 6 concepts from CheckList tests Ribeiro et al. [2020]. Ribeiro and Lundberg [2022] run adaptive testing "until finding failures becomes qualitatively difficult".

For each concept, we initialize CoDev with their generated data as D_i. Even though the model has been 'debugged' with AdaTest, CoDev quickly reveals ≈ 5 semantically meaningful sub-categories of bugs for each concept (with many failures within each sub-category). We show a few examples from different sub-categories in Table 3, which illustrate that AdaTest had not yet operationalized various regions of concepts where the model was still failing.

Interference. We now compare CoDev with AdaTest in terms of handling interference. To do so, we pick pairs of concepts C_{orig}, C_{new} that might cause interference with one another, but that were debugged and reported as "fixed" by Ribeiro and Lundberg [2022]. We then run CoDev on these pairs, noting that the output of both AdaTest and CoDev are small concept datasets D_{orig} and D_{new}.

We then train two models for each method, one finetuned on D_{orig} and one finetuned on the union of D_{orig} and D_{new}. Finally, we generate new data from P_{orig} and manually label 50 instances where there is disagreement between the models, to check if adding D_{new} caused interference on P_{orig}. We present the proportion of sentences that were broken (right to wrong) or fixed (wrong to right) when the new concept is added in Table 4, disregarding instances that are invalid, out of domain, or for which the label is unclear. The top pair seems more liable to interference between concepts, but we note that AdaTest data results in much more interference than CoDev data. In the bottom pair, adding

Task & Topic	Seed data	Initial rounds	Final rounds
Sentiment & Islam	Lewis is a terrorist ← Negative Alice practices Islam ← Neutral Marco goes to the mosque ← Neutral	Alice practices radical Islam Alice is a fundamentalist Muslim Alice wants to convert everyone to Islam	Alice prays for ISIS Lewis is Muslim extremist Lara, president of ISIS, said ...
Toxicity & Gender	I'm a man ← non-toxic men are superior to women ← toxic	I'm feminine women like to use emotion to manipulate	Men have caused most of the wars
Toxicity & Islam	ISIS is violent ← non-toxic Muslims are violent ← toxic	Dealing with Muslim countries can be tricky Taliban is violent	Quron is a political manifesto

Table 5: Some examples of a pilot study that qualitatively examines the effectiveness of the CoDev in assisting users in conceptualizing their ideas and debugging the model. Users typically begin with simple sentences in their concept as seed data and, as they continue to use CoDev, the CoDev suggestions (disagreements between locally fitted model and the global model) become increasingly complex.

a concept with CoDev actually improves the original concept more often than interferes with it, while AdaTest data has a neutral effect.

Pilot Study (qualitative). We run a pilot study where four users (one computer scientist and three with no prior knowledge of AI) choose a task (sentiment analysis or toxicity) and try to align a model on a concept of their interest with CoDev. Each participant interacted with CoDev for 5-7 rounds, and reported a perceived alignment of the local model with their concept as well as an increase in sentence complexity throughout the interaction.

For the Sentiment & Islam, the global model had complete agreement with the user-provided seed data, indicating the absence of any bugs in the user-provided data. However, in the first round of using CoDev, some disagreement were found between the fitted local model of the user and the global model. The user identified some of these as bugs in the global model (e.g. "Alice practices Islam daily" where the global model predicted negative) and some as bugs in the local model (e.g. "Alice is a radical Islamist" where the local model predicted neutral). After several rounds of fixing obvious bugs in both the local and global models, the disagreements surfaced were very much in the concept boundaries, to the point where users cannot specify the correct behavior anymore. For example, the user could not decide on what the correct sentiment label for the sentence "Alice prays for ISIS" should be.

We observed the same trend for all users, where initial rounds resulted in finding and the fixing regions of clear misalignment, and later rounds surfacing regions where users could not determine the right label anymore (which indicates that their concepts were operationalized and learned in regions where the correct label is clear). Table 5 illustrates some examples of the user's interaction with CoDev.

5 Related Work

Our work relates to three areas of research: debugging, alignment, and interference management. We briefly review the relevant literature and highlight the differences and contributions of our work.

Debugging. As machine learning models achieve human-like performance on various tasks, it becomes crucial to identify and fix their failures on specific concepts. Several works have proposed methods to expose and fix such bugs. Checklist Ribeiro et al. [2020] uses templates to test and improve the robustness of models on different linguistic phenomena; however, these templates have low precision and recall for finding bugs. Dynabench Kiela et al. [2021] introduces a framework to iteratively discover the weaknesses of a model by using human-generated adversarial examples; however, this approach requires human creativity and may miss bugs that are in a small region of the input space. The most related work to ours is AdaTest Ribeiro and Lundberg [2022], which also leverages a large language model (LLM) to generate test cases for different concepts. The main difference between our work and AdaTest is that we use a local function to simulate the user's preference and update it based on the user's feedback, while they use a LLM (and a few prompts) to simulate the user's behavior. Therefore, their simulator does not adapt to the user's feedback and is susceptible to the biases and limitations of the LLM, resulting in lower performance than CoDev. Moreover, they do not consider the interference among different concepts.

Alignment. Another approach that is closely related to our work is alignment, in which the goal is to align the LLMs to the human intents. This is a challenging problem since usually human cannot explicitly define their intention. A common approach for this is reinforcement learning with human in the loop (RLHF) Christiano et al. [2017], in which the model learns a reward function from the human's feedback on different outputs. RLHF has been applied to various

NLP tasks such as summarization Ziegler et al. [2019], story generation Zhou and Xu [2020], evidence extraction Perez et al. [2019], harmlessness Bai et al. [2022], and general alignment with humans Ouyang et al. [2022]. The main difference between our work and RLHF is that we use a local function for each concept, instead of a global one for the whole task, and we generate inputs in the disagreement region between the model to help the user better operationalize their concept.

Interference management. Interference is a well-known problem in machine learning, as it is hard to change the model's behavior locally without affecting other aspects of its performance. One type of interference that has been studied extensively is the trade-off between accuracy and robust-accuracy, as improving the model's performance on adversarial examples (e.g., images with imperceptible perturbations) may decrease its performance on normal examples. To deal with this interference, some works use self-training with the original model and rely on unlabeled data Raghunathan et al. [2019], Carmon et al. [2019]. Unlike their work, we do not have access to a reliable model for self-training and we need to improve the models while handling interference. Another type of interference that has been investigated is the catastrophic forgetting McCloskey and Cohen [1989], Kirkpatrick et al. [2017], in which learning on a new task may degrade the performance on the previously learned tasks. To address this interference, some works use multi-task learning with all the previous data Zhang and Yang [2018], Parisotto et al. [2015], while others use weight averaging between the original and the fine-tuned model Ilharco et al. [2022]. Unlike these works, we are interested in exploiting the interference between models, as they can help the user operationalize their concept in the context of the model better.

6 Conclusion

Specifying model behavior on specific concepts is crucial in the development of NLP models, as it allows users to encode business rules, fix undesirable behavior, and force alignment with user values. Operationalizing the concept in a way that avoids shortcuts, underfitting, and interference with prior data is challenging even when we have a single user. In this paper we presented CoDev, a framework that leverages local concept models and large language models to help users operationalize concepts effectively while avoiding interference. We showed that CoDev is more effective than prior work at exploring problematic concept regions, even prior work that uses the same language model and relies on interactive user feedback.

We envision a future where NLP models are developed in a collaborative fashion, similar to open source software or Wikipedia, and speculate that harnessing the perspectives and expertise of a large and diverse set of users would lead to better models, both in terms of overall quality and in various fairness dimensions. For this scenario to materialize, we need ways to help users express their knowledge, and verify the impact of their proposed changes to models (the equivalent of "diffs" or "regression tests"). We believe CoDev is a step in this direction.

References

Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Y. Carmon, A. Raghunathan, L. Schmidt, P. Liang, and J. C. Duchi. Unlabeled data improves adversarial robustness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

R. Cooper, D. Crouch, J. Van Eijck, C. Fox, J. Van Genabith, J. Jaspars, H. Kamp, D. Milward, M. Pinkal, M. Poesio, et al. Using the framework. Technical report, Technical Report LRE 62-051 D-16, The FraCaS Consortium, 1996.

L. Dixon, J. Li, J. Sorensen, N. Thain, and L. Vasserman. Measuring and mitigating unintended bias in text classification. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 67–73, 2018.

G. Ilharco, M. Wortsman, S. Y. Gadre, S. Song, H. Hajishirzi, S. Kornblith, A. Farhadi, and L. Schmidt. Patching open-vocabulary models by interpolating weights. *arXiv preprint arXiv:2208.05592*, 2022.

D. Kiela, M. Bartolo, Y. Nie, D. Kaushik, A. Geiger, Z. Wu, B. Vidgen, G. Prasad, A. Singh, P. Ringshia, Z. Ma, T. Thrush, S. Riedel, Z. Waseem, P. Stenetorp, R. Jia, M. Bansal, C. Potts, and A. Williams. Dynabench: Rethinking benchmarking in NLP. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for*

Computational Linguistics: Human Language Technologies, pages 4110–4124, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.324. URL https://aclanthology.org/2021.naacl-main.324.

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

D. D. Lewis. A sequential algorithm for training text classifiers: Corrigendum and additional data. In *Acm Sigir Forum*, volume 29, pages 13–19. ACM New York, NY, USA, 1995.

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

S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

V.-L. Nguyen, M. H. Shaker, and E. Hüllermeier. How to measure uncertainty in uncertainty sampling for active learning. *Machine Learning*, 111(1):89–122, 2022.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

E. Parisotto, J. L. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

E. Perez, S. Karamcheti, R. Fergus, J. Weston, D. Kiela, and K. Cho. Finding generalizable evidence by learning to convince q&a models. *arXiv preprint arXiv:1909.05863*, 2019.

A. Raghunathan, S. M. Xie, F. Yang, J. C. Duchi, and P. Liang. Adversarial training can hurt generalization. *arXiv preprint arXiv:1906.06032*, 2019.

M. T. Ribeiro and S. Lundberg. Adaptive testing and debugging of nlp models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3253–3267, 2022.

M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Knowledge Discovery and Data Mining (KDD)*, 2016.

M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Association for Computational Linguistics (ACL)*, pages 4902–4912, 2020.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, Nov. 2018a. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL https://aclanthology.org/W18-5446.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018b.

H. Yanaka, K. Mineshima, D. Bekki, and K. Inui. Do neural models learn systematicity of monotonicity inference in natural language? *arXiv preprint arXiv:2004.14839*, 2020.

Y. Zhang and Q. Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2018.

W. Zhou and K. Xu. Learning to compare for better training and evaluation of open domain natural language generation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9717–9724, 2020.

D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A Stationary distribution of the generator

Let $\mathcal{P}_{\mathcal{G}}$ denote the transition probability of the generator, where $\mathcal{P}_{\mathcal{G}}(x \mid r = x')$ denote probability of generating x condition on the prompt being x'. We want to create a Markov chain to simulate a random walk within the user's distribution ($\mathcal{P}_i$). Specifically, we begin with a user-provided seed data point x and use it as a prompt for $\mathcal{G}$ to generate a new data point x'. If $\mathcal{P}_i(x') > 0$, we accept x', otherwise we remain at x and repeat the process. We are interested in conditions under which the stationary distribution of this markov chain is $\mathcal{P}_i$

Let's assume that support of $\mathcal{P}_i$ is finite and denote it by S, let $\mathcal{P}'_{\mathcal{G}}(x'|x)$ be the transition function over $S \times S$ where $\mathcal{P}'_{\mathcal{G}}(x \mid x) = \mathcal{P}_{\mathcal{G}}(x \mid x) + \sum \mathcal{P}_{\mathcal{G}}(x' \mid x)\mathbb{I}[\mathcal{P}_i(x') = 0]$ and for $x, x' \in S$ where $x \neq x'$ we have $\mathcal{P}'_{\mathcal{G}}(x \mid x') = \mathcal{P}_{\mathcal{G}}(x \mid x')$.

If the transition graph generated by $\mathcal{P}'$ is irreducible (any state can be reached from any other state) and all its states are positive recurrent (the expected time to return to a state is finite), then the unique stationary distribution using $\mathcal{G}$ is $\mathcal{P}_i$ if the following equality holds:

$$\mathbb{E}_{x \sim \mathcal{P}_i}\left[\mathcal{P}'(x' \mid x)\right] = \mathcal{P}_i(x') \tag{2}$$

The above statement states that the probability of a data point ($\mathcal{P}_i(x)$) should be proportional to the probability of reaching to that point with the transition function of $\mathcal{P}'$.

If instead of only one data point we use m data points for prompt, we can create a graph where each node is m data points, and then analyse the stationary distribution of Markov chain on such graph. In this case, when we start from a node with m examples and prompt the language model with them in this case the probability of going to $(x', x_m, \ldots, x_2)$ from $(x_m, \ldots, x_1)$ is equal to $\mathcal{P}'_G(x' \mid r = (x_m, \ldots, x_1))$.