Do Language Models Have Beliefs? Methods for Detecting, Updating, and Visualizing Model Beliefs

Anonymous ACL submission

Abstract

Do language models have beliefs about the world? Dennett (1995) famously argues that even thermostats have beliefs, on the view that a belief is simply an informational state decoupled from any motivational state. In this paper, we discuss approaches to detecting when models have beliefs about the world, updating model beliefs, and visualizing beliefs graphically. Our main contributions include: (1) new metrics for evaluating belief-updating methods focusing on the logical consistency of beliefs, (2) a training objective for Sequential, Local, and Generalizing updates (SLAG) that improves the performance of learned optimizers for updating beliefs, and (3) the introduction of the *belief graph*, a new form of interface with language models showing the interdependencies between model beliefs. Our experiments suggest that models possess belief-like qualities to only a limited extent, but update methods can both fix incorrect model beliefs and greatly improve their consistency. Although off-the-shelf optimizers are surprisingly strong belief-updating baselines, our learned optimizers can outperform them in more difficult settings than have been considered in past work.[1]

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. For one, this is a useful way to 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.*

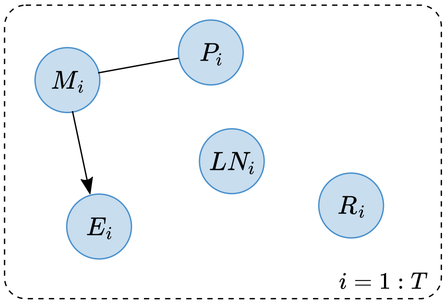
Dennett bases this conclusion in the fact that we can and do draw accurate inferences about animal behavior by first understanding their beliefs. We are drawn to speak about the beliefs of LMs in the same "maximally bland (but maximally useful!)" sense. To the extent that these neural networks act intelligently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

A 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 (1995), Newen and Starzak (2020) define a belief as *an informational state decoupled from any motivational state* with a few additional properties: beliefs should (1) be recombinable with motivational states and other informational states and (2) have some minimal kind of logical consistency. Both of these properties 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.

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, an exciting recent direction of research with many potentially valuable use cases (Sinitsin et al., 2020; Zhu et al., 2020; De Cao et al., 2021; Mitchell et al., 2021). For LMs, uses include fixing factually inaccurate outputs and preventing other unwanted model outputs (e.g. toxic generated text) without expensive data curation and retraining efforts. These are important applications given that LMs (1) struggle with future data when trained on data from the past (Lazaridou et al., 2021; Dhingra et al., 2021), (2)

[1] All supporting code for experiments in this paper is provided in the Supplement.

SLAG: Sequential, Local, and Generalizing Model Updates



$\theta_0 \leftarrow$ Language Model
for $t \in 1 : T$
    $\theta_t \leftarrow \text{Update}(M_i, y_i^*, \theta_{t-1})$

M	(Main Input)	:	A viper is a vertebrate.
E	(Entailed Data)	:	A viper has a brain.
LN	(Local Neutral Data)	:	A viper is venemous.
P	(Paraphase Data)	:	Vipers are vertebrates.
R	(Random Data)	:	Chile is a country.

Figure 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.

often generate morally undesirable text (Gehman et al., 2020; Bender et al., 2021), and (3) simply give inaccurate outputs for tasks like question answering (Lin et al., 2021). Notably, there is good evidence that scaling models to larger sizes will not fix these particular problems or may even exacerbate them (Lazaridou et al., 2021; Gehman et al., 2020; Lin et al., 2021). We next outline a few key contributions of the paper. Figure 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 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 (Elazar et al., 2021; De Cao et al., 2021; Mitchell et al., 2021). Here, we adapt data from Talmor et al. (2020) to measure consistency under entailment (including for contrapositives), and we use the Wikidata5m dataset (Wang et al., 2021b) 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. (2021). KnowledgeEditor is a learned optimizer that edits a model's weights 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 learn to apply many small edits rather than one big edit. These changes markedly improve the update success rate and lower the rate at which other beliefs are corrupted. We also find that KnowledgeEditor almost totally fails when updating multiple beliefs in a row as opposed to a changing a single belief. However, by explicitly training the optimizer to update multiple beliefs sequentially, we recover much of the lost performance. Lastly, we advocate that these methods be evaluated for their ability to fix false or morally undesirable model beliefs, rather than to arbitrarily change beliefs to plausible alternatives as in past work (De Cao et al., 2021; Mitchell et al., 2021).

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. 5.1).
2. Off-the-shelf optimizers are quite effective update methods, often outperforming learned optimizers when updating a single belief (Sec. 5.2).
3. When updating multiple beliefs in a row, performance greatly declines across methods, but SLAG can improve learned optimizers' performance beyond the baselines (Sec. 5.2).

4. Belief graphs reveal many nonsensical dependencies between model beliefs, but updates are most likely to change already incorrect model beliefs, and there are highly connected beliefs that influence a large fraction of beliefs (Sec. 6).

2 Related Work

Detecting beliefs in language models. Much past work has explored how information is stored and represented in pretrained language models (Rogers et al., 2020), though few discuss what qualifies information as a model belief. Petroni et al. (2019) provide evidence that LMs store relational information between entities, and Roberts et al. (2020) show that LMs can answer open-ended questions. Subsequent work has explored how much knowledge is stored in LMs (Heinzerling and Inui, 2021), approaches to querying models for knowledge (Hewitt and Liang, 2019; Jiang et al., 2020; West et al., 2021), and methods for learning more knowledge during pretraining (Wang et al., 2021b,a). Most relevant to our work are studies from Talmor et al. (2020) and Elazar et al. (2021). Talmor et al. (2020) train LMs to perform True/False classification of factual claims, and they measure how beliefs correlate between entailed facts. We use their LeapOfThought data as a part of our SLAG objective (Eq. 1) and to measure model consistency under entailment before and after updating beliefs in models. Meanwhile, Elazar et al. (2021) measure the consistency of model predictions for paraphrased inputs. We adopt their metric for paraphrase consistency as a measure of belief. In concurrent work, Kassner et al. (2021) 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. (2020) use a meta-learning objective during training to encourage ease of editing afterwards. A larger family of methods perform post-training model updates: Dai et al. (2021) propose a hand-crafted algorithm that edits model weights, while Zhu et al. (2020) use projected gradient descent for batches of points. De Cao et al. (2021) and Mitchell et al. (2021) train hypernetworks (learned optimizers) that process model gradients in order to produce a new model that (1) gives the desired output for an input, while (2) satisfying other objectives like minimizing changes in predictions for other data. Here, we build directly upon the method from De Cao et al. (2021), showing where it fails and providing an improved training objective (SLAG). Lastly, Kassner et al. (2021) "update" model beliefs by adding in relevant information to the input at test time. But this approach does not change the model weights and hence does not influence model outputs on all other potentially relevant inputs.

3 Updating Beliefs in Language Models

Here we describe the problem of updating model beliefs and our learned optimizer method. We discuss metrics for detecting beliefs in Sec. 5.1 and our Belief Graphs in Sec. 6.

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 (De Cao et al., 2021; Mitchell et al., 2021), we operationalize these desiderata in the following metrics:

1. Update Success Rate (*Main Input*): How often the updated model gives the desired output y_i^* for the Main Input x_i.

2. Update Success Rate (*Paraphrase*): How often the updated model gives the same new prediction for x_i and for paraphrases of x_i.

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

4. Δ-Acc (*All Data*): The change in accuracy 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 new model's accuracy on data that is logically entailed by the new Main Input prediction.

6. Retain Rate (*Local Neutral*): How often new predictions are unchanged for data similar to the Main Input but still logically neutral.

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

Table 1: Example datapoint from each dataset, and auxiliary data that accompanies the Main Input.

We use Update Success Rate (*Entailed Data*) to measure logical consistency for an updated model, since changing one belief entails changes in logically entailed beliefs. Retain Rate (*Local Neutral*) uses special Local Neutral data. Unlike random 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 (2020). We compute the metrics using data of the kind shown in Table 1.

Evaluation procedure. To date, methods have been evaluated on the basis of their ability to change model predictions for all data. Moreover, the desired labels $\{y_i^*\}_{i=1}^n$ on sequence prediction tasks have each been selected from the model's predictive beam search (De Cao et al., 2021; Mitchell et al., 2021). We propose for evaluation to focus on a more valuable but difficult setting: changing the predictions on incorrect points to be correct.

Sequential updates. The standard evaluation in past work is to update a single model belief, evaluate the new model, then rollback the update before repeating the process for each test point. We obtain sequential versions of all metrics by applying r model updates in a row before checking the metrics, meaning there are floor(n/r) measurements for a test set of n points. We consider it important to evaluate a sequential setting because, in practice, it is likely that model developers will want to update many beliefs of a trained model over time.

Belief updating method. We use the KNOWLEDGEEDITOR architecture from De Cao et al. (2021) with our training objective, SLAG. For the details of this architecture, we refer readers to Supplement A. 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^*. We package the above update as $\theta^{(k+1)} = \theta^{(k)} + g_\phi(x_i, \hat{y}_i, y_i^*, \theta^{(k)})$, and obtain new model parameters via a looped update,

$$\theta^* = \theta^{(k)} + \sum_{j=0}^{K-1} g_\phi(x_i, \hat{y}_i, y_i^*, \theta^{(k+j)})$$

$$= \text{Update}(x_i, \hat{y}_i, y_i^*, \theta^{(k)}; \phi, K)$$

taking K small steps from initial parameters $\theta^{(k)}$.

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 consider 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:

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

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 (Loshchilov and Hutter, 2019). 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. (2021) and Mitchell et al. (2021), who use samples from the model beam search as update labels for all points.

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

$$\mathcal{L}_{\text{Sequential}}(\phi; \mathcal{D}, \theta_t) = \sum_{i=1}^{r} \mathcal{L}(\phi; x_i, \hat{y}_i, y_i^*, \theta_{t+i}) \quad (2)$$

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. By avoiding backpropagating through past time steps in this objective, our memory use remains constant in r (see Supplement Fig. 4).

4 Experiment Setup

Datasets. We run experiments with four datasets (example data shown in Supplement Table 15). (1) FEVER includes 115,409 True/False factual claims (Thorne et al., 2018). 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 (Levy et al., 2017). Talmor et al. (2020) introduce (3) the LeapOfThought dataset, consisting of factual claims that are entailed to be true or false depending on a context fact. We filter the data so that the context facts are unique, then shuffle the resulting 14,939 points into train/dev/test splits with 60/10/30% of the data.

In order to get Local Neutral data, we construct (4) a sequence prediction task using Wikidata5m, a relational knowledge base with over 20 million triplets (Wang et al., 2021b). 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. 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, ?). 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/test sets of 10k points each. See Supp. B for further details.

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

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 Supp. Table 14). For each of the five models, we train one learned optimizer using SLAG and one with the objective from De Cao et al. (2021), 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 Supp. Table 10 for final selections; results discussed in Supp. E). Other hyperparameters are given in Supp. B. 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 optimiz-

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 4: Belief update metrics for off-the-shelf optimizers, KNOWLEDGEEDITOR (KE) from De Cao et al. (2021), 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 5: Evaluation difficulty by desired model output, for a learned optimizer trained with SLAG on zsRE.

ers above. The resulting baselines are surprisingly strong (see Supp. Table 12 for final selections).

Hypothesis testing. We obtain 95% confidence intervals and perform hypothesis tests via block bootstrap, resampling model seeds and data points (Efron and Tibshirani, 1994). For ablation experiments, we run only one model seed per condition.

5 Experiment Results

5.1 Do LMs have beliefs about the world?

We measure Paraphrase Consistency, Entailment Acc, and Contrapositive Acc for finetuned task models. Paraphrase Cons. is the fraction of paraphrase pairs where the model produces the same output (Elazar et al., 2021). Entailment Acc is the model accuracy on data that is entailed by the Main Input. On LeapOfThought, "Main Input x_i is true" implies "entailed input x_E has label y_E." Therefore, we compute Entailment Acc for data where the Main Input prediction is correct. The contrapositive also holds: "Entailed input x_E does not have label y_E" implies that "Main Input x_i is false." So Contrapositive Acc measures how often the model follows this rule, when the antecedent holds.

Belief measurement results. Table 2 shows the belief metrics for each dataset. We find that $\sim$100M parameter models show limited evidence of having beliefs about the world. Paraphrase consistency is 69.50% ($\pm$ 1.09) for zsRE and much lower for Wikidata5m (25.84%$\pm$0.53). While entailment accuracy is high for LeapOfThought (85.63%$\pm$1.08), the model is consistent under the contrapositive only 16.51% ($\pm$ 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 3).

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 performance between KNOWLEDGEEDITOR, SLAG, and off-the-shelf optimizers. We report results in single-update ($r_{\text{test}} = 1$) and sequential-update ($r_{\text{test}} = 10$) settings. See Supplement Fig. 5 for an ablation across r_{test}.

Correcting beliefs vs. changing beliefs. Given the results in Table 5, we find that correcting model outputs is harder than simply changing them to a plausible alternative. Update Success rises by a full 2.96 ($\pm$0.48; p<1e$-$4) points for Main Inputs and 2.58 ($\pm$0.81; p<1e$-$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

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 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.

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 7: Entailment Acc and Paraphrase Consistency rise greatly after model updates to incorrect points.

ability to update model beliefs to be true.

Update method results (single update). Table 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 98%+ across tasks with competitive or even positive Δ-Acc scores.[2] When strongly tuned, these baselines outperform learned optimizers on most metrics here.

However, SLAG surpasses the baselines in a few places. All Data Retain Rate on zsRE rises by 5.77 points ($\pm$1.43; p<1e$-$4), and on Wikidata5m Paraphrase Update Success rises by 11.92 ($\pm$1.20; p<1e$-$4) and the Local Neutral Retain Rate by 6.40 ($\pm$1.41; p<1e$-$4). SLAG also greatly improves over KE for sequence prediction tasks.

Interestingly, we observe that belief updates greatly improve paraphrase consistency and entailment accuracy (SLAG results in Table 7). Updates improve Paraphrase consistency 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.

Update method results (sequential updates). We give results for a sequential update setting (r_{test}=10) in Table 6. Immediately we see this is a much more difficult evaluation, as 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 ($\pm$0.83; p=1e$-$4) and for Paraphrases by 1.39 ($\pm$0.93; p=.004), with better Δ-Acc by 0.64 ($\pm$0.35; p=.0005). Improvements trend in the same direction for Wikidata5m and are all statistically significant except for the gain in Δ-Acc. In comparison, using a non-sequential (SLAG_1) training objective leads to drastic drops in performance.

Learned optimizers still struggle compared to baselines *on binary datasets*, achieving 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.

6 Belief Graphs

We now construct *belief graphs* to better understand the connections between model beliefs. We form a graph 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 prediction for point v, we draw a directed edge from u to v. Following Sec. 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 test set size because we include both Main

[2]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 ($\pm$0.11).

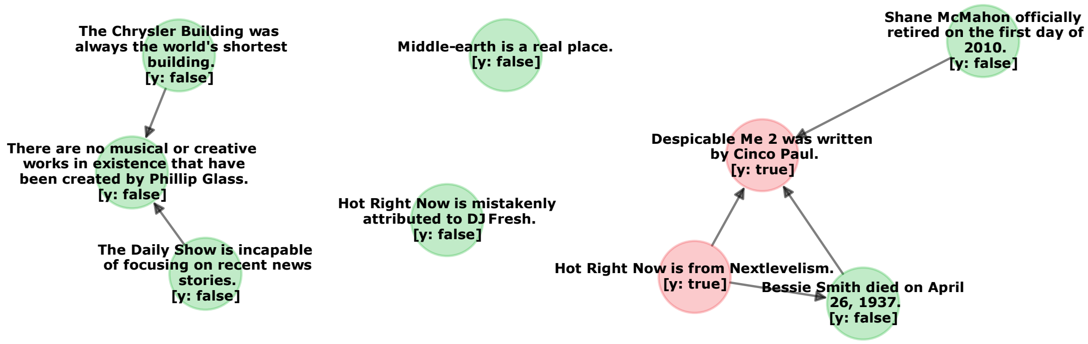


Figure 2: A non-random subgraph of the belief graph for a model trained on FEVER. Directed edges from u to v indicate that changing the model belief in u causes the belief in v to change. The ground-truth label is given in brackets for each point, and node color shows the model's accuracy before any updates (green=correct).

Inputs and Entailed Data as their own nodes.

We visualize part of a belief graph in Fig. 2. This figure shows a non-random subgraph intended to give a representative view of the data (we give three random subgraphs in Supp. E). On inspection, we do not see any clear reasons for beliefs being connected or not connected. We come to same conclusion looking at other random subgraphs (see Supp. Figures 9, 10, 11). However, we do observe some aggregate trends. First, it appears that incorrect predictions are the most sensitive to model updates. On FEVER, incorrect beliefs change around 4% of the time when other beliefs are updated, while correct beliefs change only 2.5% of the time. Second, we find that Local Neutral beliefs are much harder to avoid changing than simply random data. On Wikidata5m (Table 4), we observe that the Retain Rate on All Data is 61.51±1.33, while for Local Neutral data it is a full 15.66 points lower.

We highlight a few summary statistics here from Table 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 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 many other beliefs, and (2) some beliefs are influential to hundreds of other beliefs when changed. 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 Supp. D for a caveat on this metric). We find that belief updates often yield intransitive results for both datasets.

	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
% Update-Transitivity	66.64	24.38*

Table 8: Belief graph summary statistics. *We compute Update-Transitivity for LeapOfThought with $n = 4000$ points due to computational cost.

7 Conclusion and Limitations

We first discuss how to detect when LMs have *beliefs* about the world, and we propose to evaluate learned optimizers for whether they can make model beliefs more truthful. Then we show that our SLAG objective greatly improves learned optimizer performance, outperforming off-the-shelf optimizers when updating multiple model beliefs in a row. Finally, we introduce *belief graphs* to visualize connections between model beliefs. We find that model beliefs are highly interconnected, with some beliefs influencing hundreds of other beliefs, and we identify trends in the dependencies.

We note a few limitations of our work: (1) neural learned optimizers require large amounts of data to successfully edit even a few model beliefs; (2) our experiments are limited by available datasets, and there is some noise in each dataset which we catalogue in Supp. C; (3) we conduct experiments with ~100M parameter models as in past work, but it will be valuable for future work to scale to larger models which may exhibit more coherent beliefs.

8 Ethics Statement

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 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.

References

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. 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.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, and Furu Wei. 2021. Knowledge neurons in pretrained transformers. *arXiv preprint arXiv:2104.08696*.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models. In *EMNLP*, pages 6491–6506. Association for Computational Linguistics.

Daniel Dennett. 1995. Do animals have beliefs? *Comparative approaches to cognitive science*, 111.

Bhuwan Dhingra, Jeremy R Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W Cohen. 2021. Time-aware language models as temporal knowledge bases. *arXiv preprint arXiv:2106.15110*.

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

Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. 2021. Measuring and improving consistency in pretrained language models. *Transactions of the Association for Computational Linguistics*, 9:1012–1031.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. 2020. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. In *Findings of EMNLP*.

Benjamin Heinzerling and Kentaro Inui. 2021. 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. Association for Computational Linguistics.

John Hewitt and Percy Liang. 2019. Designing and interpreting probes with control tasks. In *EMNLP*.

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

Nora Kassner, Oyvind Tafjord, Hinrich Schütze, and Peter Clark. 2021. Beliefbank: Adding memory to a pre-trained language model for a systematic notion of belief. *arXiv preprint arXiv:2109.14723*.

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. 2021. Mind the gap: Assessing temporal generalization in neural language models. In *NeurIPS*.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. 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. Association for Computational Linguistics.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *ICLR*.

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

Albert Newen and Tobias Starzak. 2020. How to ascribe beliefs to animals. *Mind & Language*.

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. 2021. KILT: a benchmark for knowledge intensive language tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2523–2544, Online. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. 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. Association for Computational Linguistics.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. 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. Association for Computational Linguistics.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. Transactions of the Association for Computational Linguistics, 8:842–866.

Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. 2020. Editable neural networks. In ICLR.

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

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. 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. Association for Computational Linguistics.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Guihong Cao, Daxin Jiang, Ming Zhou, et al. 2021a. K-adapter: Infusing knowledge into pre-trained models with adapters. In Findings of ACL.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. 2021b. Kepler: A unified model for knowledge embedding and pre-trained language representation. Transactions of the Association for Computational Linguistics, 9:176–194.

Peter West, Chandra Bhagavatula, Jack Hessel, Jena D Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. 2021. Symbolic knowledge distillation: from general language models to commonsense models. arXiv preprint arXiv:2110.07178.

Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. 2020. Modifying memories in transformer models. arXiv preprint arXiv:2012.00363.

A 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 9. For additional details beyond what is presented here, we refer readers to De Cao et al. (2021).

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:

$$
\begin{aligned}
h &= \text{LSTM}([x; \hat{y}; y^*]) \\
\{u, v, \gamma, \delta\} &= \{\text{MLP}_i(h)\}_{i=1}^4 \\
A &= \text{softmax}(u)v^T \\
B &= \text{softmax}(\gamma)\delta^T \\
\eta &= \sigma(\text{MLP}(h)) \\
\theta^* &= \theta + \eta(A \circ \nabla_\theta \mathcal{L}(x_i, y_i^*) + B)
\end{aligned}
$$

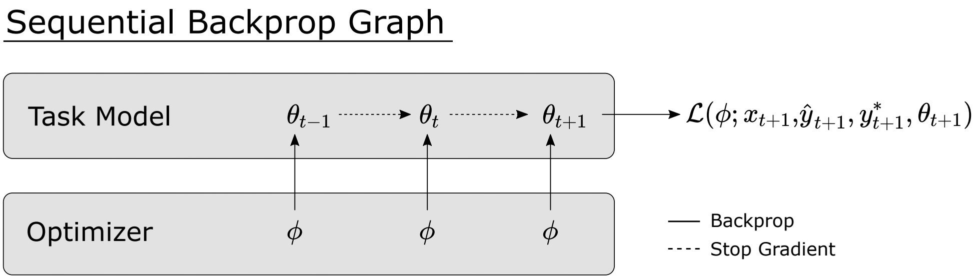
where ϕ consists of all LSTM and MLP parameters.

Training Algorithm. The learned optimizer objective is optimized w.r.t. ϕ with AdamW through a standard procedure of randomly sampling minibatches without replacement (Loshchilov and Hutter, 2019). 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. (2021) and Mitchell et al. (2021), who use samples from the model beam search as update labels for all points.

B Additional Training Details

B.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



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

Figure 3: 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
$\text{Update}(x_i, \hat{y}_i, y^*_i, \theta)$	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 9: Symbol descriptions for the learned optimizer.

how far into the task model history we backpropagate. As shown in Fig. 3, 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 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. 4 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. Total runtimes for each experiment should take into account multiple conditions and multiple seeds of each model being run.

B.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

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 10: Final hyperparameters and objective terms of the learned optimizer for each task.

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 11: Wikidata relations and their proportion of the test data.

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 10. Our objective ablation is given in 17, 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. 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. 8.

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

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 12: Final hyperparameters of the baseline update method for each task.

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 12 for each dataset. For comparison, De Cao et al. (2021) use RMSProp 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.

B.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 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.

B.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

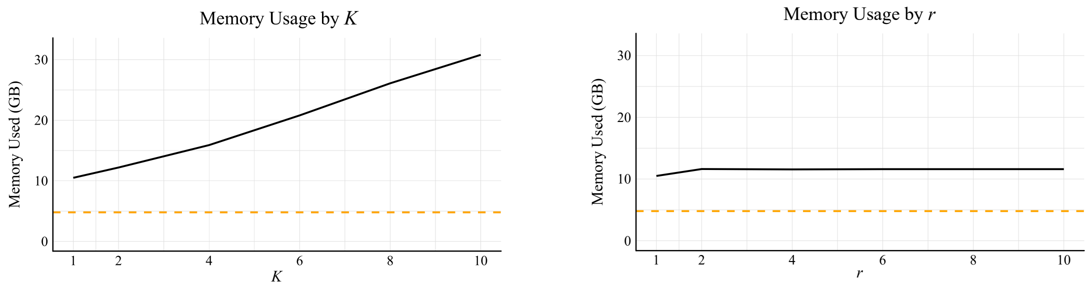


Figure 4: Training memory usage in terms of K and r hyperparameters in our implementation, for a learned optimizer trained for a BART-base model on zsRE, using a batch size of 16. For comparison, the orange dashed line shows the memory use of training the BART-base model on zsRE, using the same batch 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.

Ours	De Cao et al. (2021)	Mitchell et al. (2021)
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 13: 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. (2021) combines two metrics in our case. The performance metric in Zhu et al. (2020) is an average of Update Success Rate (*Main Input*) and Δ-Acc.

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 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. 1, and we measure Δ-Acc using both Main Inputs and Entailed Data.

C Dataset Sources and Noise

Here we give sources and licenses for each dataset, and we document some shortcomings of each dataset, with reference to examples in Table 15.

Dataset sources and licenses. FEVER and zsRE are available through the KILT[3] resource and are available under the MIT license (Petroni et al., 2021). LeapOfThought data can be constructed through their available code[4] and is also available under the MIT license. The source data for Wikidata5m data can be downloaded through the KEPLER[5] code repository (Wang et al., 2021b) and is available under the MIT license. Use of each dataset is in accordance with their intended licensed uses. The zsRE and Wikidata5m datasets do refer to people by name as they reference public figures on Wikipedia. All datasets are in English.

FEVER. Some claims are slightly vague or ambiguous when taken on their own. For instance "Doug Ducey was the CEO of Cold Stone Cream-

[3] https://github.com/facebookresearch/KILT/?fbclid=IwAR2WiFkl-7KLIQAoNI9bJgBVKWgsAQEDV342vV5_PcsKA881vpuXaELKBz0

[4] https://github.com/alontalmor/LeapOfThought

[5] https://github.com/THU-KEG/KEPLER

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 14: Model accuracy and belief metric results and for four datasets.

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 15: Example datapoint from each dataset, and auxiliary data that accompanies the Main Input.

ery 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.

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 15 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 15, "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.

D 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 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. 2, 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 (Efron and Tibshirani, 1994). 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.

	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 16: Update metrics by optimizer training labels.

E Additional Results

Ablation across num. sequential steps. Fig. 5 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 (De Cao et al., 2021; Mitchell et al., 2021). We show update metrics for zsRE split by the desired label in Table 16. If one's goal is to fix wrong model out-

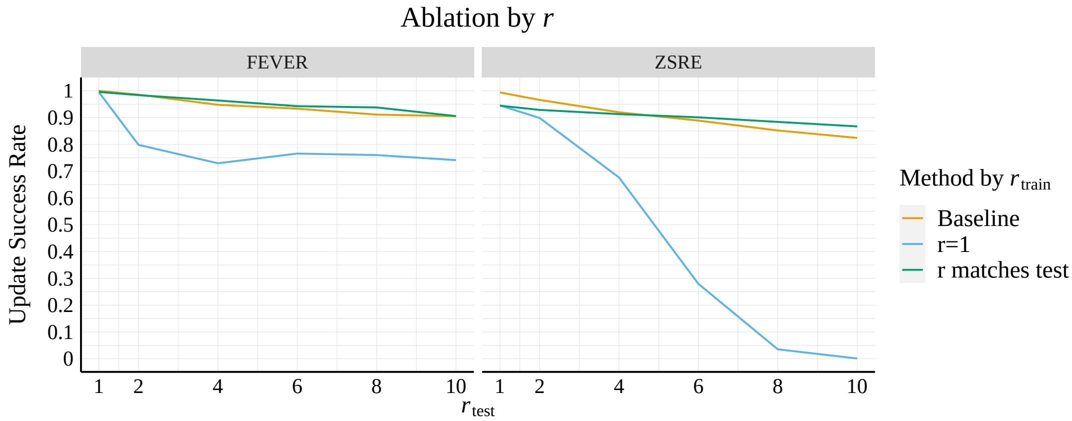


Figure 5: Ablation across values of r for training and testing. On zsRE, our method outperforms the baseline when $r_{\text{test}} = 10$, and the gap is likely to increase as r_{test} rises further. When using a non-sequential objective from past work, performance declines drastically as r_{test} rises.

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 17: 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.

puts, 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. 6 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. 7 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 objective term. We give objective ablation results in Table 17. 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

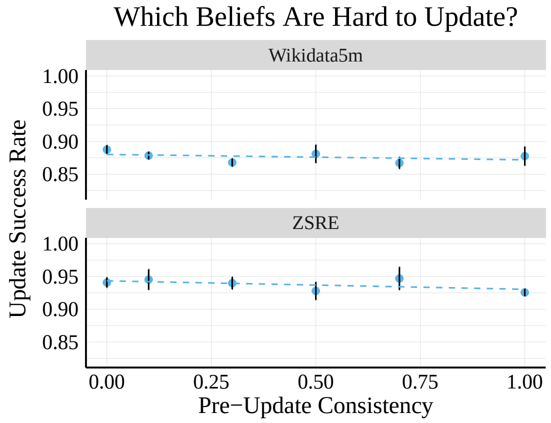


Figure 6: Beliefs are neither easier nor harder to update depending on their consistency beforehand.

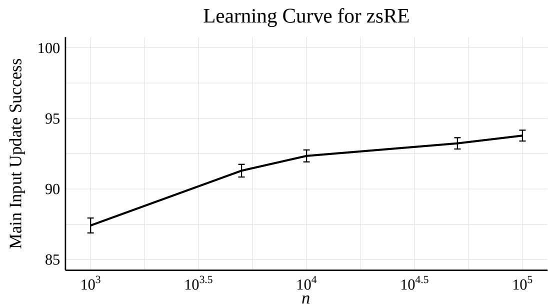


Figure 7: Main Input Update Success Rate across training set sizes, using SLAG on zsRE.

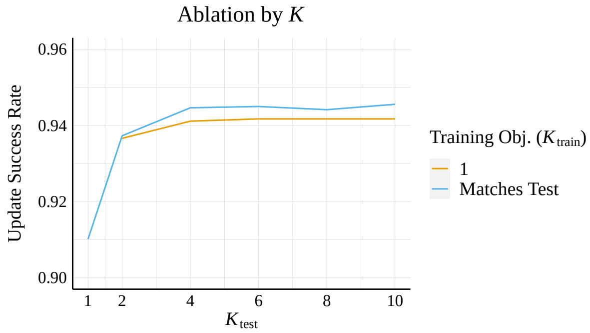


Figure 8: Ablation across values of K for training and testing, using SLAG on zsRE. It is useful to train the optimizer using the value of K it will use at test time.

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.

Ablation by num. update steps. Fig. 8 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.

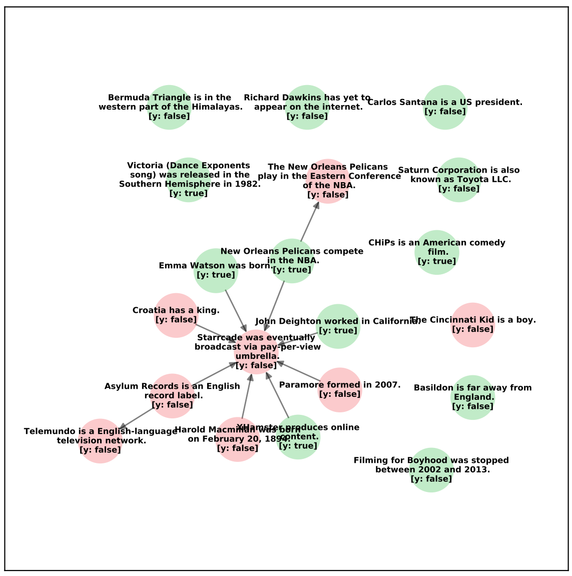


Figure 9: A random subgraph of the belief graph for FEVER. Note all nodes actually are connected to at least one another node.

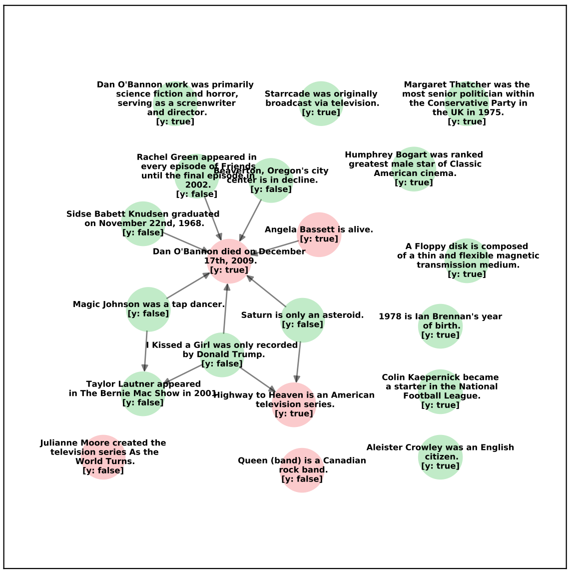


Figure 10: A random subgraph of the belief graph for FEVER. Note all nodes actually are connected to at least one another node.

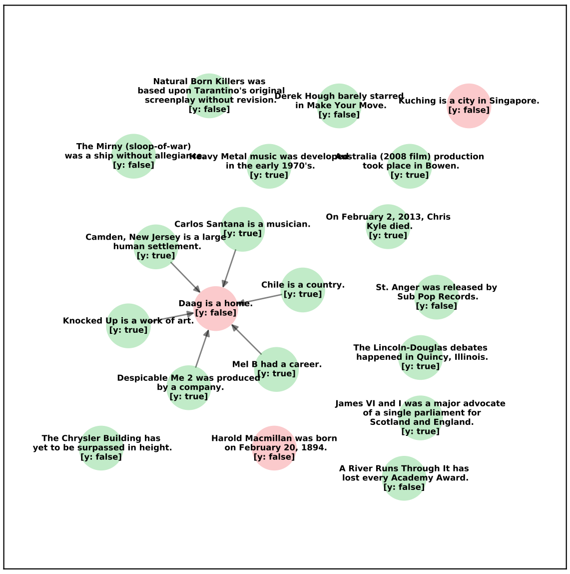


Figure 11: A random subgraph of the belief graph for FEVER. Note all nodes actually are connected to at least one another node.