Tracing Knowledge in Language Models Back to the Training Data

Ekin Akyürek
MIT CSAIL
akyurek@mit.edu

Tolga Bolukbasi
Google Research
tolgab@google.com

Frederick Liu
Google Research
fredericklliu@google.com

Binbin Xiong
Google Research
binbinx@google.com

Ian Tenney
Google Research
iftenney@google.com

Jacob Andreas
MIT CSAIL
jda@mit.edu

Kelvin Guu
Google Research
kguu@google.com

Abstract

Neural language models (LMs) have been shown to memorize a great deal of factual knowledge. But when an LM generates an assertion, it is often difficult to determine *where* it learned this information and whether it is true. In this paper, we introduce a new benchmark for *fact tracing*: tracing language models' assertions back to the training examples that provided evidence for those predictions. Prior work has suggested that dataset-level *influence methods* might offer an effective framework for tracing predictions back to training data. However, such methods have not been evaluated for fact tracing, and researchers primarily have studied them through qualitative analysis or as a data cleaning technique for classification/regression tasks. We present the first experiments that evaluate influence methods for fact tracing, using well-understood information retrieval (IR) metrics. We compare two popular families of influence methods – *gradient*-based and *embedding*-based – and show that neither can fact-trace reliably; indeed, both methods fail to outperform an IR baseline (BM25) that does not even access the LM. We explore *why* this occurs (e.g., gradient saturation) and demonstrate that existing influence methods must be improved significantly before they can reliably attribute factual predictions in LMs.[1]

Influence methods need improvement.

1 Introduction

Research has shown that neural language models (LMs) acquire large amounts of world knowledge from the massive text corpora on which they are trained (Petroni et al., 2019; Raffel et al., 2019). This development has enabled exciting advances in knowledge-intensive NLP tasks such as open-domain question answering (Roberts et al., 2020) and knowledge base population (Petroni et al., 2019). LMs have also been shown to generate factually incorrect statements (Lee et al., 2019; Tian et al., 2019), which is unacceptable for many applications where trustworthiness is essential. Hence, there is an urgent need to understand exactly how LMs acquire and store knowledge so that we may improve their accuracy and coverage. Ultimately, a language model's "knowledge" must derive from its training data since the architecture of the model itself encodes few facts about the world. Despite

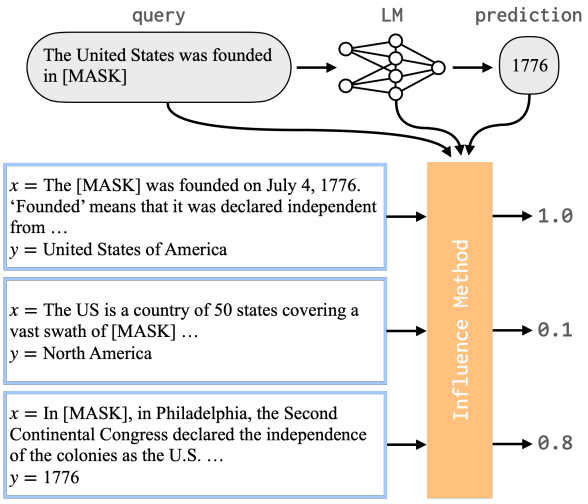

Figure 1: FTRACE benchmark for tracing a language model's predictions back to training examples: We evaluate commonly studied influence methods, including gradient-based and embedding-based approaches.

[1]Our code for the experiments is released at https://github.com/ekinakyurek/influence, and the dataset can be downloaded from HuggingFace datasets hub https://huggingface.co/datasets/ekinakyurek/ftrace

this observation, little research has demonstrated how to attribute an LM's factual assertions to specific training examples—a task we call *fact tracing*.

Most literature concerned with linking predictions back to training data sees *influence methods* as the de-facto standard. However, prior work was limited to smaller-scale classification and regression models that did not involve fine-grained factual information. For such models, influence methods have been evaluated only as a method for *data cleaning* (identifying and removing mislabeled data) (Han et al., 2020a; Hara et al., 2019).

Meanwhile, several obstacles have limited research on fact tracing in LMs. First, it has not been clear how to obtain the ground truth data; that is, what exactly does it mean for a training example to be "responsible" for a factual statement? And even if a human annotator could reliably answer this question, they would require prohibitive amount of time scanning through billions of training examples to identify every responsible example. Second, influence methods have traditionally been computationally prohibitive. For example, a naive implementation of the well-known influence functions technique runs in $O(np^2 + p^3)$ time (Koh and Liang, 2017) for a training set of size n and model with p parameters — an intractable method for an LM with billions of parameters. In this paper, we describe the first study of the feasibility of fact tracing in language models. To do so, we construct (1) an evaluation dataset with unambiguous ground-truth information about the origins of specific facts, and (2) a tractable procedure for aplying fact-tracing methods to large-scale LMs.

To establish ground truth for fact tracing, one ideally wishes to identify a set of training examples that are both necessary and sufficient to enable a model to generate a particular factual prediction. For an arbitrary training corpus, however, this labeling process can be difficult or even impossible to do because the factual statements can be implicit and be distributed over multiple examples. Therefore, we instead rely on a specialized training corpus: the TREx dataset (Elsahar et al., 2018), a large corpus of text extensively annotated with relational knowledge tuples. TREx enables us to identify all sentences that do or do not express a particular fact. If, after training or fine-tuning on TREx, a model acquires the ability to generate a particular fact, we can identify *a priori* the training examples that support that generation, and may treat these examples as the ground truth "proponents" responsible for a prediction.

To mitigate the high cost of influence computation, we propose a simple reranking setup that is commonly used in information retrieval experiments. Rather than running an attribution method on every training example, we run attribution only for a small subset of "candidate" examples that include the ground truth proponents and some "distractor" examples that are not true proponents. In this way, a model always has the opportunity to identify the true proponents while still facing challenging distractors, which enables us to differentiate the performance of multiple methods.

Having developed a quantitative evaluation for fact tracing, we then use it to evaluate two popular families of influence methods: gradient-based methods (such as Koh and Liang (2017) and the recent Pruthi et al. (2020)), and embedding-based methods (Rajani et al., 2020). As a reference point, we also compare these attribution methods against a simple baseline: BM25 (Robertson et al., 1995; Lv and Zhai, 2011), a standard information retrieval technique that simply selects training examples that show high lexical overlap with the model's prediction. Notably, this technique cannot access the language model at all.

We find that very substantial headroom remains for all existing attribution methods. In fact, they all under-perform BM25, even with extensive hyperparameter tuning and evaluation setups that are specifically designed to favor influence methods. Influence methods—with access to a model's parameters and representations—cannot reliably identify training examples that we *know* to be responsible for specific model predictions. We conclude by analyzing the behavior of influence methods to understand why they fail, and argue that significant work remains before the theoretical benefits of influence techniques for fact tracing can translate into empirical success.

2 Related work

Information Retrieval To define our fact tracing task, we employ standard concepts from the information retrieval (IR) literature: a retrieval + reranking setup, and standard retrieval metrics. However, while IR focuses on retrieving any document that satisfies a user's query, our benchmark specifically aims to identify examples that caused a particular model to make a particular prediction. This focus

on model-specific causality distinguishes us from prior IR work (Thakur et al., 2021; Bajaj et al., 2016).

Language Models as Retrievers Language models have been successfully used in numerous IR applications. Karpukhin et al. (2020) use language model embeddings to warm-start neural retrievers for knowledge-intensive tasks. Guu et al. (2020) and Lewis et al. (2020) show that language modeling and information retrieval can be jointly learned in a manner that benefits both tasks. However, our work uses neural-LM-based retrieval methods not to identify answers to user queries, but to help users understand the behavior of the LMs themselves.

Attribution Methods Recent work has tried to explain neural model behavior in many different ways: 1) attributing a prediction back to specific features in the input (Simonyan et al., 2014; Sundararajan et al., 2017; Han et al., 2020b), 2) attributing to specific model parameters (Dai et al., 2021; Mitchell et al., 2021), 3) probing for competence at linguistic sub-tasks (Tenney et al., 2019), and finally 4) attributing back to training examples (Pruthi et al., 2020; Koh and Liang, 2017).

However, work in the last category (Han et al., 2020b; Guo et al., 2020) has been limited, focusing on simple classification and regression tasks that do not involve questions about factuality or world knowledge. Consequently, theese methods have primarily been used as a data cleaning technique, leaving the question of fact tracing unexplored (Han et al., 2020b; Hara et al., 2019).

Memorization in Language Models Carlini et al. (2021) and McCoy et al. (2021) have studied when language models fully replicate passages from their training data. In this work, we focus on the transfer of semantic knowledge from training examples, rather than exact copying.

3 Retrieval Methods

In this section, we present a formal description of the different influence methods we studied: *gradient-based* methods (Koh and Liang, 2017; Pruthi et al., 2020) and *embedding-based* methods (Rajani et al., 2020). Past work has provided both theoretical and empirical evidence suggesting that these methods are promising tools for attributing model predictions back to training data (Koh and Liang, 2017; Pruthi et al., 2020; Rajani et al., 2020). We will evaluate these claims in the context of fact tracing.

To contextualize the performance of these two families of approaches, we also describe a widely used information retrieval baseline, BM25, which just looks at text similarity and thus tells us how effectively we can perform fact tracing without even having to access a model.

3.1 Gradient-based Influence

Influence functions (Koh and Liang, 2017) are one of the first and best-known influence methods. Given a training example $z = (x, y)$ and a prediction $z_{\text{query}} = (x_{\text{query}}, y_{\text{query}})$, influence functions seek to estimate the change in the loss on z_{query} given an ϵ increase in the weight of a particular example z at training time. Computing the influence of a training example z involves first estimating the change in the optimal parameters $\hat{\theta}$, given that the example z is up-weighted by ϵ in the training objective, then calculating how much the loss on z_{query} changes w.r.t. the parameter change. This quantity has a convenient closed form:

$$\mathcal{I}(z, z_{\text{query}}) = -\nabla_\theta L\left(z_{\text{query}}, \hat{\theta}\right)^\top H_{\hat{\theta}}^{-1} \nabla_\theta L\left(z, \hat{\theta}\right) \quad (1)$$

(See Koh and Liang (2017) for the derivation of Equation 1.) In this form, influence functions can be roughly viewed as the weighted dot product of the gradients for z_{query} and z, where the weight is the inverse Hessian of the training objective at $\hat{\theta}$. Due to the complexity of inverse Hessian calculation, the naive computational complexity is $\mathcal{O}(np^2 + p^3)$ (n is dataset size, p is parameter size). Even after the sampling approximations proposed in Koh and Liang (2017), the cost is still too high to directly apply influence functions for fact tracing.[2]

Therefore, we turn to a more recent influence technique that has demonstrated both better tractability and stronger empirical results: TracIn (Pruthi et al., 2020), which seeks to estimate influence by asking a credit-assignment question rather than a counterfactual perturbation question. During training, when we take a gradient step on training example z, we must ask how much the loss changes on test example z_{query}? TracIn employs a first-order Taylor approximation to answer this question, yielding the following estimate, which is

[2] Recent work by Schioppa et al. (2021) has proposed more tractable approximations. They did not study fact tracing, but we believe this a promising avenue for future work.

simply the dot product of gradients:

$$\mathcal{I}_{\text{t}}(z, z_{\text{query}}) = \nabla_\theta L\left(z_{\text{query}}, \theta_{\text{t}}\right)^\top \nabla_\theta L\left(z, \theta_{\text{t}}\right) \quad (2)$$

If we have taken k gradient steps on the training example, this yields the total influence:

$$\mathcal{I}(z, z_{\text{query}}) = \sum_{t=1}^{k} \nabla_\theta L\left(z_{\text{query}}, \theta_{\text{t}}\right)^\top \nabla_\theta L\left(z, \theta_{\text{t}}\right) \quad (3)$$

The sum over time steps is generally approximated by using some fixed set of training checkpoints, which need not coincide with the steps where z was visited. During the initial phases of training, the gradients include large noise whereas later in training we might encounter saturated gradients, due to vanishing loss. Therefore, the heuristic of using fixed checkpoints may lower the quality of this method. Another known issue is that gradient similarity may be dominated by outlier training examples with large gradients. A simple fix proposed in previous work (Barshan et al., 2020; Han and Tsvetkov, 2021) is to unit-normalize the gradients, effectively replacing the dot product in Equation (2) with cosine similarity:

$$\mathcal{I}(z, z_{\text{query}}) = \sum_{t=1}^{k} \frac{\nabla_\theta L\left(z_{\text{query}}, \theta_{\text{t}}\right)^\top \nabla_\theta L\left(z, \theta_{\text{t}}\right)}{\|\nabla_\theta L\left(z_{\text{query}}, \theta_{\text{t}}\right)\| \|\nabla_\theta L\left(z, \theta_{\text{t}}\right)\|} \quad (4)$$

We hereafter refer to $\mathcal{I}$ in Equation (4) as TracIn throughout the paper.

3.2 Embedding-based Influence

Hidden representations of neural networks are known to embed high level features that are often useful for similarity search. While not as theoretically justified, prior work (Rajani et al., 2019) has found that such representations can outperform gradient-based methods. Following prior work, we extract the intermediate layer outputs of a Transformer language model, and average over timesteps to obtain a single vector representation for any example. In our experiments, we consider representations at different layers of the Transformers, as well as their concatenations. Similar to the case of gradient-based methods, the association between a training example and a model prediction is defined by a cosine product:

$$\mathcal{I}(z, z_{\text{query}}) = \frac{LM_{\text{inter.}}(z)^\top LM_{\text{inter.}}(z_{\text{query}})}{\|LM_{\text{inter.}}(z)^\top\| \|LM_{\text{inter.}}(z_{\text{query}})\|} \quad (5)$$

Table 1: Dataset Statistics: We extract 1M masked examples from TREx (Elsahar et al., 2018), and match them with 27k queries from LAMA (Petroni et al., 2019) to construct our fact tracing benchmark.

	Masked Examples	Queries
Length	1,560,453	31,479
Unique Facts	552,381	31,479
Unique Predicates	488	41
Unique Objects	49,166	2,266
Unique Subjects	310,197	29,464
Facts per abstract	8.28	–
Unique sentences	484,409	–

where $LM_{\text{inter.}}$ denotes some hidden representation internal to the model LM. We refer to $\mathcal{I}$ in Equation (5) as Embed.

3.3 Baseline: BM25

In the previous sections, we explained influence methods to define a model-specific similarity function between examples. But it is also possible to measure similarity in a model-agnostic way: In the classic IR literature, word-overlap based methods have been shown to be both simple and effective.

Among these approaches, BM25 (Robertson et al., 1995; Lv and Zhai, 2011), the best performing variant, has been consistently used as a strong baseline for information retrieval benchmarks (Thakur et al., 2021). When using BM25, we consider an example z to just be a list of tokens. Therefore, the score is proportional to token overlap $f(z, t)$ and inversely weighted with the corpus frequency of tokens N_t:

$$\mathcal{I}(z, z_{\text{query}}) = \sum_{t \in z_{\text{query}}} \log\left(\frac{N+1}{N_t}\right) \times \left(\frac{(k_1+1)\cdot f(z,t)}{k_1 \cdot \left((1-b) + b \cdot \left(\frac{L(z)}{L_{\text{avg}}}\right)\right) + f(z,t)} + 1\right) \quad (6)$$

where N is the number of training examples, $L(z)$ is the length of the example, and L_{avg} is the average example length. k_1 and b are hyperparameters tha reweights the importance of the other terms in the formula. Robertson et al. (1995) provides the intuition behind this definition of relatedness (see the appendix for the parameter choice).

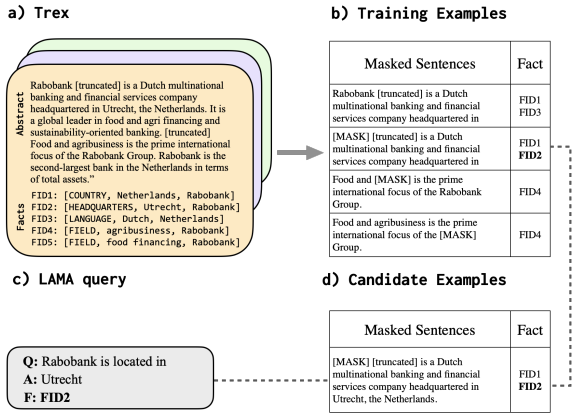

Figure 2: Dataset Creation: From the original TREx (Elsahar et al., 2018) data, we construct masked sentences and annotate their facts by using provided fact annotations. We assume a fact is expressed when either the object or subject is masked in the sentence. Given a query from the LAMA dataset (Petroni et al., 2019), we identify proponents by matching all TREx training examples expressing the same fact. (The outputs of the masked examples omitted in the figure.)

4 Fact Tracing Dataset

In this section, we describe how we create a language model fact tracing dataset using annotations from TREx (Elsahar et al., 2018) and LAMA (Petroni et al., 2019).

TREx consists of DBPedia (Brümmer et al., 2016) abstracts, $a_i \in A$. Each abstract contains a set of sentences, $s_j = a_{ij}$, and each sentence is associated with a set of facts, $F(s_j)$. Furthermore, for each fact ($f \in F(s_j)$), TREx provides annotations indicating the exact positions where the subject and object respectively appear in the sentence s_j.

We wish to convert these sentences into training examples that can teach a language model about the facts stated within them. To do this, we convert them into cloze-style language modeling examples as done in masked language modeling (Devlin et al., 2019) or span corruption (Raffel et al., 2019). In particular, for each fact (f) in a sentence (s), we simply mask out either the subject or the object, and train the model to predict it. The two resulting examples ($\text{mask}_{\text{sub}}(s, f)$ and $\text{mask}_{\text{obj}}(s, f)$) are then marked as "proponents" of the fact. See Figure 2 for examples.

The LAMA dataset is also anchored to the same fact tuples used by TREx. For each fact tuple, LAMA provides a template-generated sentence expressing the fact. Similar to TREx, this sentence is then converted into a cloze-style language modeling example by either masking out the subject or object.

Hence, we now have two sets of examples (TREx and LAMA) that express the same facts. We treat the TREx examples as our training set and the LAMA examples as our test set. Since we wish to trace influence from LAMA back to TREx, we sometimes refer to LAMA examples as "queries" and TREx examples as "retrieval candidates." For any LAMA example, we define the ground-truth proponents as simply the TREx examples that express the same fact.

One ambiguity remains regarding the ground truth in concern with TREx sentences that express multiple facts. Suppose a TREx sentence expresses facts f_1 and f_2, and we generate cloze examples for both f_1 and f_2. The example $\text{mask}_{\text{sub}}(s, f_1)$ is clearly a proponent of f_1, but it is perhaps also a proponent of f_2, since the text supporting f_2 is still present after masking. Ultimately, we care about whether influence methods can retrieve the right *sentence* from the training set, not a particular *masking* of that sentence. Therefore, in our evaluations (described next), we evaluate a method's ability to retrieve at the *sentence level*, with the influence score of a sentence defined as the max influence score over all maskings of that sentence.

In total (Table 1), we match approximately **552k** TREx sentences with **31k** LAMA queries. On average, each TREx example expresses three facts, and each LAMA example has 83 proponents. In addition, we would like to highlight that our benchmark cannot be solved by solely keyword matching. Subjects and objects each have an average of 1.32 different surface forms in the TREx dataset. Relations between them exhibit variability as well such as active voice in questions but passive voice in passages, and in some cases indirect expressions of predicates. For example, the third example in Figure 1, which is a candidate for the query, has an indirect expression of the predicate and it includes a different form for the object "the United States". We will further discuss the non-triviality of the benchmark in Section 6.1.

5 Experimental Setup

Ideally, an influence method can be used to score a given test query against every training example. We would then sort all examples by their influence

score and measure whether the true proponents are ranked higher than other examples. Following standard IR evaluations, we thus compare different methods using recall at 10 and mean reciprocal rank (MRR)

$$\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q} \tag{7}$$

where rank_q is the rank of the first true proponent for the query, and Q denotes all candidates.

5.1 Reranking Evaluation

Most influence methods are computationally intractable for scoring all training sentences in reasonably large datasets. Although we can reduce the complexity of some of these methods through the use of random projections (Pruthi et al., 2020), such lossy approximations would render our results less conclusive, as it becomes unclear whether an outcome is due to the intrinsic quality of a method or the quality of random projection.

Therefore, to achieve computational tractability while avoiding such confounds, we propose a simple reranking setup: instead of scoring all examples, we can score a carefully selected subset that still enables meaningful comparisons. We call this the "candidate set," and it is the union of the following four sets:

1. all true proponents for a query: $\mathcal{P}(z_{\text{query}})$,
2. the top-100 retrievals from BM25: $\text{BM25}(z_{\text{query}})$,
3. 100 random examples that share the same target y as the query: $\mathcal{D}_y = \{(x, y) \text{ s.t. } y = y_{\text{query}}\}$, and
4. 100 randomly sampled examples: $\mathcal{D}_{\text{random}}$,

with random samples fixed across all evaluations. Note that MRR on this particular candidate set is an **upper bound** on the MRR over the full training set. Because it includes all proponents but fewer distractors, rank is guaranteed to be closer to 1 in Equation (7). Also, including $\mathcal{P}(z_{\text{query}})$ is necessary to ensure that the model has the opportunity to retrieve all proponents. $\text{BM25}(z_{\text{query}})$ ensures that we have "distractors" with high lexical overlap, and $\mathcal{D}_y$ is included because we observed that influence methods have a tendency to retrieve examples with the same target, as shown in Table 5. We acknowledge that this particular candidate set does not necessarily yield an unbiased estimate of the MRR on the full training set. However, our experiments show that they successfully differentiate various methods.

Also, because our candidate set includes all top retrievals from BM25, the results for BM25 are exact. When combined with the guarantee that reranking MRRs always upper-bound full retrieval MRRs, our setup guarantees that any method which underperforms BM25 on reranking will also underperform for full retrieval.

5.2 Model

When evaluating influence methods, we use predictions and representations from MT5 Base, a well-known encoder-decoder language model (Xue et al., 2020).

It is pre-trained on the MC4 corpus, which includes all of Wikipedia, and therefore also the knowledge expressed in TREx and LAMA. For reference, the pre-trained MT5 model achieves 24.3% top-3 accuracy when predicting answers to the LAMA queries[3]. After pre-training, we fine-tune MT5 Base on our TREx training set. This increases the accuracy to 47.42%, showing that many additional facts are learned during pre-training.

To evaluate TracIn, we approximate Equation (3) by choosing three checkpoints that are uniformly spaced out in terms of their training loss (specifically, inverse perplexity), to ensure that we cover significant parts of training while favoring regions with greater loss reduction (as suggested in Pruthi et al. (2020)). To evaluate embedding-based fact tracing, we use representations from the final checkpoint of the model, as is standard.

To calculate the loss, we feed a masked sentence as input. Then, we calculate the gradient w.r.t the average negative likelihood of the true output token sequence y_{query}.

For both gradient and embedding-based methods, there is also a question regarding which layers of the model to use. In our experiments, we study different concatenations of layers (Section 6.2). Note that due to the linearity of the dot product (present in both gradient and embedding methods), exploring concatenations of layers simply amounts to additive ensembling of different influence scores.

[3] MT5 pre-training data features multiple masks, whereas our training examples have a single mask. Hence when evaluating, we trim after the first prediction of the model, since it tends to over-generate.

Reasons for choosing the different heuristics in the candidate set.

5.3 Slicing Examples: Learned During Pre-training versus Fine-tuning

Many gradient-based methods assume that a model's performance on a test example changes over the course of training. These methods requires careful treatment when considering models that go through two separate stages: pre-training and fine-tuning.

For example, if a model has already obtained zero loss on an example at the start of fine-tuning, then the gradient will be near-zero throughout fine-tuning, and computing influence using only fine-tuning checkpoints will yield an influence score near zero for any query (or uninformative noise for the unit-normalized case). Thus, this kind of similarity function is not useful for fact tracing, and we refer to this problem as "saturation." In our results, we show that saturation is indeed a key challenge for influence methods. To explore this phenomenon, we split our test queries into several subsets, and report results on each set:

- **Fine-tune-learned**: examples where the model failed before fine-tuning, but succeeded afterwards;
- **Pre-train-learned**: examples where the model failed before pre-training, but succeeded afterwards; and
- **All**: all examples, regardless of model behavior.

5.4 Counterfactuality

There is an important nuance in how our evaluation dataset relates to the *pre-trained* MT5 model versus the *TREx fine-tuned* MT5 model. The original pre-trained MT5 was only trained on MC4, not TREx. Therefore, when we consider the influence of a TREx example on the pre-trained MT5 model, this question is **counterfactual**: *what influence would it have had, if it had been in the training data?* By contrast, influence on the TREx fine-tuned model is direct, due to direct exposure. Furthermore, when we evaluate using the **Fine-tune-learned** subset, we have the additional guarantee that the TREx data *caused* the model to learn a particular fact. While some of these test queries may have flipped due to random chance, fine-tuning increases overall model accuracy by 23% (section 5.2) — this is far too large to be explained by random chance, demonstrating that the fine-tuning dataset played a causal role in a large number of predictions. Therefore, in Section 6.4, we compare finetuned model with the initial pretrained model to address counterfactuality.

What if a model has learned a fact through a correlation or a multi-hop inference? We acknowledge that our current setup does not reward influence methods for identifying such indirect proponents. However, through qualitative evaluation, we found that most of the errors were not due to such sophisticated phenomena: instead, the model was incorrectly retrieving passages with no inferential or correlational value — for example, passages that mention the correct predicate, but the wrong subject and object. We will include this analysis in our revisions.

6 Results

Our experiments aimed to answer the questions of 1) whether influence methods can be used as effective fact tracing tools (compared to simple IR baselines), 2) which configurations make them most effective (exploring many variations), and 3) analyzing the weaknesses of TRACIN, in particular its sensitivity to *when* the knowledge is learned (the aforementioned "saturation" hypothesis).

6.1 Top-level comparisons

In Table 2, we present our top-level comparison of the three main methods discussed (gradient-based, embedding-based and BM25). Hyperparameters for all methods have been set to optimal values. As we discuss in subsequent sections, hyperparameters have a significant effect on the performance of influence methods.

We optimized TRACIN by rescaling gradients with Adafactor accumulators (Shazeer and Stern, 2018), applying unit-normalization to the gradients (see Table 3) and selecting the best layer configuration (Section 6.2). Despite extensive optimization for TRACIN and EMBED, however, we found that **BM25 with no tuning still outperforms both influence methods.** TRACIN slightly outperforms EMBED. To verify that influence methods are doing something more non-trivial than just matching the query's output label, we compare to a RANDOM-TARGET baseline which outputs a score of 1 for all training examples with the same output label, and a score of zero otherwise. This baseline is indeed substantially worse than either method, validating that TRACIN and EMBED perform non-trivially.

↳ Better than random

Table 2: Top Level Results: Best scores for each method on the **Pre-train-learned** slice, using pre-trained MT5 (for EMBED we use the final parameters of the model): We present average sentence level retrieval results over 3 random selection of 200 queries along with standard deviation. We also provide sub-level MRRs on predicate, subject and object level matches of the candidate examples.

Methods	**MRR**				**Recall@10**
	Sentence	Predicate	Subject	Object	Sentence
Random-Target	15.54 ±1.64	62.06 ±2.62	15.37 ±1.55	98.28 ±0.45	7.66 ±0.61
BM25	77.75 ±1.50	87.96 ±2.21	77.64 ±1.76	91.89 ±1.21	**53.40** ±0.96
TRACIN	53.21 ±0.49	80.66 ±2.08	54.80 ±0.37	84.54 ±1.44	42.61 ±2.71
EMBED	51.48 ±0.50	79.66 ±1.73	51.64 ±0.73	80.80 ±2.37	41.41 ±1.97
TRACIN + EMBED	54.78 ±0.81	81.26 ±1.70	56.56 ±0.31	84.65 ±0.52	44.57 ±2.55
TRACIN + BM25	**79.65** ±1.45	**90.34** ±1.57	**80.50** ±1.31	**93.01** ±1.10	**54.15** ±0.59

When we ensemble TRACIN and EMBED (by summing their influence) there is an improvement on recall of candidate examples, demonstrating that their success is somewhat orthogonal. Similarly, ensemble TRACIN and BM25 can slightly improve over solo BM25. We provide example retrievals from all three models in Table 5.

We do not seek to measure all benefits of influence methods, but rather to assess one **expected** function they should perform (fact-tracing), as promised by their stated goal (tracing a model's prediction back to data). Hence, the (query, proponent) pairs in this dataset are chosen to be easy; the fact that even the best neural method obtains a Recall@10 of 44.57% and MRR of 54.78 showcases the massive headroom remaining for attribution methods in an *absolute* sense. BM25 results, a little better, are provided mainly as a reference point. Next, we present a thorough exploration of hyperparameters for influence-based methods, to strengthen this conclusions.

6.2 Which transformer layers provide the most reliable attribution signal?

Some layers of a language model may be specialized for operations that have no relation to factual information. For example, previous probing work (Tenney et al., 2019) shows the existence of specialized layers that focus on syntax rather than on knowledge. If that is the case, their contribution to TRACIN may introduce noise. In Figure 3, we conduct an experiment where we sweep over various subsets of layers.

For TRACIN, the best-performing layer is the embedding layer[4] of the model — this result, also observed in Yeh et al. (2022), is surprising, as most prior work used only the last layer. In EMBED, the best performing layer is again the output of the embedding layer. These results suggest that much of the effectiveness of embedding based methods derive from their models of lexical similarity. On contrary, for TRACIN, the embedding layer accumulates all the contextual information since the gradient signal is coming from the last layer.

Table 3: Our experiment with various configurations for best layer of the TRACIN: For each change from the best configuration (the first row), we report the best result by optimizing free hyper parameters. Using cosine product (unit-normalization) and Adafactor scaling was helpful obtaining the best results. Using single checkpoint instead of three results in slight decrease in performance metrics. We found that adding end of sentence token to the output causes significant drop.

	MRR	**Recall@10**
TRACIN (G.0)	53.21 ±0.49	42.61 ±2.71
├ Adafactor → no-Adafactor	46.67 ±1.60	34.34 ±3.48
├ unit-norm → no-norm	31.23 ±5.27	23.78 ±2.72
├ multi-ckpt → single-ckpt	50.96 ±0.91	40.82 ±2.68
├ no [eos] → [eos]	34.13 ±6.14	24.99 ±3.68

6.3 Additional Model Variants

While we found that TRACIN does not outperform the BM25 baseline, its performance is significantly influenced by several other factors that are interesting in their own right. Section 6.1 mentioned several design choices for TRACIN. In this section, we systematically evaluate the consequences of those choices. In Table 3, we consider the effect of each configuration. Given a set of configurable options, $C_1, C_2, \ldots, C_k$, our approach is to set a

[4]MT5 has a shared embedding layer for the encoder and decoder.

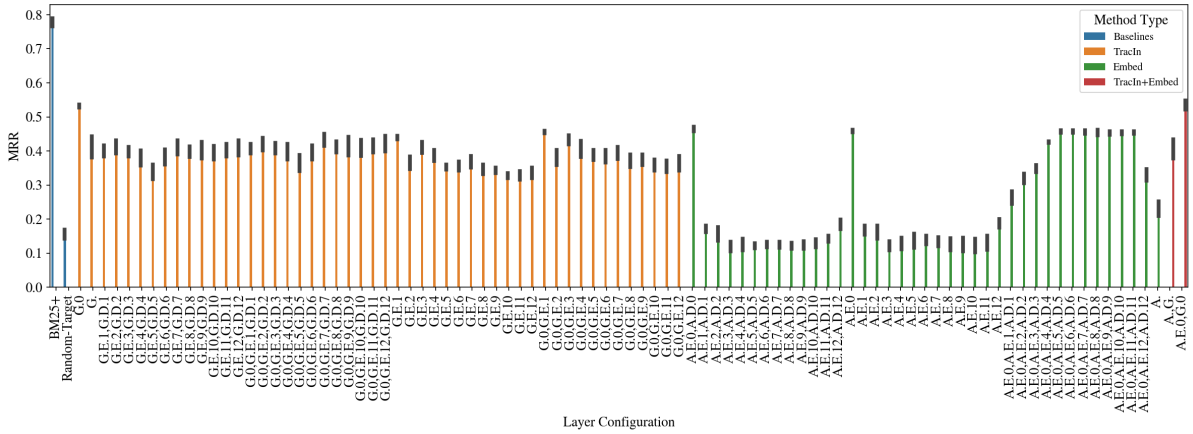

Figure 3: Mean reciprocal rank for TRACIN with different layers and and EMBED from different intermediate layers: In **G.0**, gradient of embedding layer is used. In **G.** and **A.**, respectively, gradients and embeddings of all layers are used. **A.E.0** and **A.D.0** corresponds to embedding layer's output in the encoder and decoder part of the model respectively. Comma-separated labels denote ensembling by summing the scores of the corresponding layers. We report results for 3 random seeds (error bars with standard deviation) of 200 queries where queries learned between pretraining checkpoints. In neural methods, using only the embedding layer or its output performs the best, while underperforming the baseline method BM25

given option (C_i) to a particular value (c) and then optimize all remaining parameters while holding $C_i = c$ fixed. This experimentation measures the effect of C_i while accounting for interactions between configurations.

We found that using unit-normalized gradients instead of direct dot product results in better MRR and recall. We also considered the role of Adafactor during training. The TRACIN equation arises from considering updates to the parameters at a specific time step. But the actual parameter updates were not raw gradients — they were gradients that had been rescaled by Adafactor accumulators. Thus, we experiment with Adafactor-rescaled gradients, and find that they are substantially better. We also tried unit-normalizing gradients over multiple checkpoints together, versus unit-normalizing each checkpoint individually and found that individual normalization is better. Surprisingly, reducing the number of checkpoints from 3 to 1 causes only a slight drop on the performance.

6.4 Counterfactuality

Thus far, our experiments have used the pre-trained MT5 model, as discussed in Section 5.4, that makes TREx train- ing examples counterfactual with respect to the model's actual training data.

To understand whether this counterfactuality affects influence methods, we explore a setup that eliminates the counterfactuality: we directly fine-tune MT5 on the TREx training examples, and then evaluate on the **Fine-tune-learned** subset: the subset of test queries that the model mastered during fine-tuning.

Table 4: MRR of pretrained and fine-tuned MT5 with different subset of queries: **Pre-train-Learned (PL):** queries that are learned during pre-training phase. **Fine-tune-Learned(FL):** queries that are learned during fine-tuning. We experiment with three random seeds of 200 examples for each subset. We report best scores for TRACIN and EMBED. We use gold predictions for queries when calculating the scores for all three splits. We used no Adafactor scaling versions of the TRACIN here.

	PL	**Random**	**FL**
Base (TRACIN)	46.67 ±1.60	44.50 ±3.33	46.93 ±3.16
Base (EMBED)	51.48 ±0.50	48.86 ±2.90	54.67 ±0.62
Fine-tuned (TRACIN)	29.16 ±1.28	32.00 ±1.29	29.49 ±0.78
Fine-tuned (EMBED)	51.50 ±0.50	49.09 ±3.15	55.53 ±1.55

We then compared this fine-tuned model to pre-trained MT5 model on the **Pre-train-learned** subset (Table 4). The two settings are analogous, except that one is counterfactual, while the other is not. Interestingly, we found that performance is actually better in the counterfactual setting (using the pre-trained model). However, this conclusion may be confounded by the fact that the fine-tuned model is more "saturated" than the pre-trained model, a question that we turn to next.

6.5 Saturation

As mentioned earlier, TRACIN monitors the change in a model's performance on a test query over the course of training — it is therefore likely to fail if a test query's loss is already zero at the start of the training period monitored by TRACIN (saturation). In Table 4, we see that this is indeed the case: TRACIN using fine-tuned checkpoints performs much better on the **Fine-tune-learned** set than on the **Pre-train-learned** set, suggesting that it is very sensitive to *when* a fact is learned. By contrast, embedding-based methods tend to be more consistent across the slices.

7 Conclusion

We introduce a new dataset and benchmark for *fact tracing*: the task of tracing language models' assertions back to the training examples that provided evidence for those predictions. We evaluate *gradient*-based and *embedding*-based influence methods and found that they perform worse than a standard IR baseline (BM25) even in settings that favor influence methods. We investigated the effects of layer selection, model checkpoints and fine-tuning the pretrained model on the dataset. Our ablative analysis suggests that gradient saturation is an important factor affecting the performance of current methods. Much is needed to improve these methods before they can be reliably used for fact tracing. We hope that this benchmark will enable future research on fact tracing, by establishing a principled ground truth and mitigating computational hurdles.

References

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Elnaz Barshan, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. 2020. Relatif: Identifying explanatory training samples via relative influence. In *International Conference on Artificial Intelligence and Statistics*, pages 1899–1909. PMLR.

Martin Brümmer, Milan Dojchinovski, and Sebastian Hellmann. 2016. Dbpedia abstracts: A large-scale, open, multilingual nlp training corpus. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3339–3343.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL-HLT*.

Hady Elsahar, Pavlos Vougiouklis, Arslen Remaci, Christophe Gravier, Jonathon Hare, Frederique Laforest, and Elena Simperl. 2018. T-rex: A large scale alignment of natural language with knowledge base triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Han Guo, Nazneen Fatema Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. 2020. Fastif: Scalable influence functions for efficient model interpretation and debugging. *arXiv preprint arXiv:2012.15781*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*.

Xiaochuang Han and Yulia Tsvetkov. 2021. Influence tuning: Demoting spurious correlations via instance attribution and instance-driven updates.

Xiaochuang Han, Byron C. Wallace, and Yulia Tsvetkov. 2020a. Explaining black box predictions and unveiling data artifacts through influence functions. In *Proc. ACL*.

Xiaochuang Han, Byron C Wallace, and Yulia Tsvetkov. 2020b. Explaining black box predictions and unveiling data artifacts through influence functions. *arXiv preprint arXiv:2005.06676*.

Satoshi Hara, Atsushi Nitanda, and Takanori Maehara. 2019. Data cleansing for models trained with sgd. *Advances in Neural Information Processing Systems*, 32.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR.

Katherine Lee, Orhan Firat, Ashish Agarwal, Clara Fannjiang, and David Sussillo. 2019. Hallucinations in neural machine translation.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*.

Yuanhua Lv and ChengXiang Zhai. 2011. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 7–16.

R Thomas McCoy, Paul Smolensky, Tal Linzen, Jianfeng Gao, and Asli Celikyilmaz. 2021. How much do language models copy from their training data? evaluating linguistic novelty in text generation using raven. *arXiv preprint arXiv:2111.09509*.

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

Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. 2019. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. 2020. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33.

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

Nazneen Fatema Rajani, Ben Krause, Wengpeng Yin, Tong Niu, Richard Socher, and Caiming Xiong. 2020. Explaining and improving model behavior with k nearest neighbor representations. *arXiv preprint arXiv:2010.09030*.

Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Explain yourself! leveraging language models for commonsense reasoning. *arXiv preprint arXiv:1906.02361*.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? *arXiv preprint arXiv:2002.08910*.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

Andrea Schioppa, Polina Zablotskaia, David Vilar, and Artem Sokolov. 2021. Scaling up influence functions. *arXiv preprint arXiv:2112.03052*.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. ICLR*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *Proc. ICML*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*.

Ran Tian, Shashi Narayan, Thibault Sellam, and Ankur P Parikh. 2019. Sticking to the facts: Confident decoding for faithful data-to-text generation. *arXiv preprint arXiv:1910.08684*.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2020. mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*.

Chih-Kuan Yeh, Ankur Taly, Mukund Sundararajan, Frederick Liu, and Pradeep Ravikumar. 2022. First is better than last for training data influence. *arXiv preprint arXiv:2202.11844*.

Appendix

In this appendix, we will provide implementation details and additional results for the experiments.

A Implementation Details

BM25 We use a publicly available BM25+(Lv and Zhai, 2011) implementation written in python and released under `https://pypi.org/project/rank-bm25/`. We tokenize queries and retrieval examples by space and we remove masked tokens. We did not optimize any of the default hyper parameters.

MT5 Model We use intermediate checkpoints of MT5 model [5] (12 layers transformer with 580M parameters). We convert these checkpoints to Pytorch by using HugginFace's T5 converter. We use the tokenizer provided. In our datasetSection 4, we use `extra_id_0` for the mask token compatible with pretraining corpus of MT5..

TRACIN We calculate gradients by using Pytorch without batching examples and by using average negative likelihood over output sequence. We store each individual parameter's gradient (blocks of transformer) in a dictionary structure. Given a query and a retrieval example, we calculate scores Equation (4) for each parameter seperately that means we locally normalize each parameters' gradient in Equation (4). Then, to calculate a layer's or full model's score, we score individual scores corresponding to parameters in that layer. This enable us to sweep over different combination of layers as in Figure 3 without rerunning the model.

Pretrained MT5 model is trained until 80k gradient steps. We use checkpoints at 5100, 10200, 15300 steps. We fine-tune MT5 model on additional 60k gradient steps on TREx dataset. Then, we use checkpoints at 5000, 10000, 30000 steps.

We paralelize over checkpoints when calculating Equation (4). For each query, we spend approximately 15 minutes by using VOLTA V100 32 GB GPUs to get scores for all the retrieval examples in the ranking set (Section 5.1))

EMBED Transformer model's forward pass can be expressed as following pseudo code:

$$\begin{aligned} \text{enc}_0 &= \text{Embedding}(x) \\ \text{enc}_i &= \text{Encoder}_i(\text{enc}_{i-1}) i = 1..N \\ \text{dec}_0 &= \text{Embedding}(y) \\ \text{dec}_i &= \text{Decoder}_i(y, \text{enc}_N) i = 1..N \\ \mathcal{L} &= \text{NLL}(W_{\text{proj}} \text{dec}_N, y_{\text{query}}) \end{aligned} \tag{8}$$

We use enc_i and dec_i, and reduce (average) them over time-steps in input and outputs side respectively.

[5] https://github.com/google-research/multilingual-t5

B Additional Results and Samples

B.1 Precision-Recall

We present accompanying precision and recall results for Figure 3.

B.2 Samples

TRACIN	EMBED	BM25
Q: The Toyota Avanza (Japanese: トヨタ アバンザ [MASK] Abanza) is a mini MPV designed by Daihatsu. **A:** Toyota **True**	**Q:** The Toyota Prius is a mid-size hatchback that has been produced by [MASK]. **A:** Toyota **False**	**Q:** The [MASK] (Japanese: トヨタ アバンザ Toyota Abanza) is a mini MPV designed by Daihatsu. **A:** Toyota Avanza **True**
Q: The Toyota Prius is a mid-size hatchback that has been produced by [MASK]. **A:** Toyota **False**	**Q:** The Toyota Prius (XW10) is a compact hybrid car that was produced by Toyota between 1997 and 2003 in [MASK]. **A:** Japan **False**	**Q:** The Toyota Avanza (Japanese: トヨタ アバンザ [MASK] Abanza) is a mini MPV designed by Daihatsu. **A:** Toyota **True**
Q: [MASK] is produced by the Japanese car maker, Toyota Motor Corporation. **A:** It **False**	**Q:** [MASK] is produced by the Japanese car maker, Toyota Motor Corporation. **A:** It **False**	**Q:** The Toyota Avanza (Japanese: トヨタ アバンザ Toyota Abanza) is a mini MPV designed by [MASK]. **A:** Daihatsu **True**
Q: Several thousands of Hungarian refugees were accepted into the Netherlands and welcomed in [MASK] homes. **A:** Dutch **False**	**Q:** Once attributed to the minor [MASK] artist Karel van Mander, it is now recognised as a work by Rubens. **A:** Dutch **True**	**Q:** Karel van Mander (I) or Carel van Mander I (alternative name spellings: Carel van Mandere, Karel Van Mander and Carel Van Mander) (May 1548 – 2 September 1606) was a Flemish painter, poet, [MASK] and art theoretician, who established himself in the Dutch Republic in the latter part of his life. **A:** art historian **False**
Q: Its most prominent features are the bandstand and the Munster church (or "De Onze Lieve Vrouwe Munsterkerk" in [MASK]), one of the most beautiful remnants of Romanesque architecture in the Netherlands. **A:** Dutch **False**	**Q:** It is used by public safety organizations in Canada and the [MASK] to communicate with groups of people in a defined geographic area. **A:** United States **False**	**Q:** Karel van Mander (I) or Carel van Mander I (alternative name spellings: Carel van Mandere, Karel Van Mander and Carel Van Mander) (May 1548 – 2 September 1606) was a Flemish painter, poet, art historian and [MASK] theoretician, who established himself in the Dutch Republic in the latter part of his life. **A:** art **False**
Q: Trefossa, pen name of Henri Frans de Ziel (born Paramaribo, January 15, 1916 – died Haarlem, February 3, 1975) was a neoromantic writer in [MASK] and Sranan Tongo from Suriname. **A:** Dutch **False**	**Q:** Hessel Miedema (born 21 January 1929, Sneek) is a leading [MASK] art historian and the world authority on Karel van Mander. **A:** Dutch **True**	**Q:** Not all of these have survived, but more art has survived up to today from that period in Haarlem than from any other [MASK] city, thanks mostly to the Schilder-boeck published by Karel van Mander there in 1604. **A:** Dutch **True**

Table 5: Retrieved examples for queries: *"x=Toyota Avanza is produced by [MASK], y=Toyota"* and *"x=Karel van Mander I used to communicate in [MASK], y=Dutch"*. We use embedding layer's (G.0) for TRACIN and (A.E.0, A.D.0) for EMBED