Prompt-based model editing

Stanford CS224N Custom Project

Charles Lin
Department of Computer Science
Stanford University
`charles.lin@cs.stanford.edu`

Abstract

SERAC [1] is a retrieval-based algorithm for editing pretrained models which achieves state-of-the-art performance in question-answering, natural language inference, and sentiment modulation editing tasks. However, while performance of language models scales with the number of parameters in many tasks [2, 3], SERAC's performance does not scale with the size of the base model being edited due to its decoupling of its base and counterfactual models. To this end, we propose a retrieval-and-continuous-prompting editing algorithm which replaces SERAC's counterfactual model with a *prompter* network to predict continuous prompts which reliably modulate the base model's outputs. We show that the prompt-based editor rivals the performance of SERAC on natural language inference and question-answering editing tasks. Empirically, our method shows promise in being able to generalize to unseen base model architectures and to reliably modulate language model outputs.

1 Key Information to include

- Mentor: Eric Mitchell
- External Collaborators (if you have any): none
- Sharing project: no

2 Introduction

Language models have shown great empirical success over various tasks from machine translation to question-answering to natural language generation [2, 4, 5]. However, these models are not perfect, and on some tasks such as closed-domain question-answering and news summarization, their beliefs can become outdated as the world changes. In deployment, language models are typically utilized as static objects after being trained; naively, in order to update a language model, one would need to finetune the model or train a new model. Neither of these solutions is perfect: finetuning can lead to catastrophic forgetting [6], while retraining models can be computationally expensive.

Model editor algorithms [7, 8, 9] attempt to provide local updates to a model's behavior on some input. To make model edits sensible, they should generalize to, and only to, inputs which are in some sense similar to that input. As an example, suppose that upon receiving the question *Who won the last NBA championship?*, a question-answering model incorrectly answers with *Toronto Raptors*. To make the model's predictions up-to-date (at the time of writing), we would want to edit its output to *Milwaukee Bucks*. Upon this edit, we'd like the model also to update its outputs on related questions such as *Who won the NBA Finals last year?*, *Did the Raptors win the Finals last year?* and *How many championships have the Bucks won?* without the need to explicitly list and train the model on all such questions. At the same time, we don't want the model's output to change on unrelated questions such as *Who is the prime minister of the UK?* and *Which NBA team won the most games last year?* (different from winning the Finals). In other words, we want our edits to be **general** with respect to rephrasings and implications, while remaining **local** to the particular topic being edited.

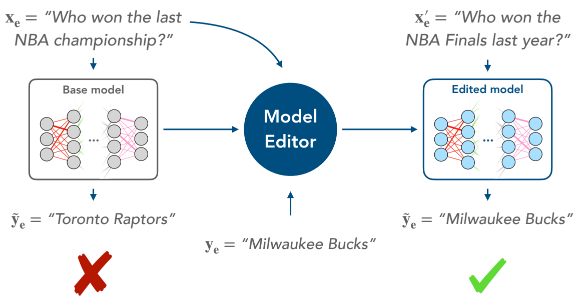
x_e = "Who won the last NBA championship?"
x'_e = "Who won the NBA Finals last year?"
Base model
Model Editor
Edited model
$\tilde{y}_e$ = "Toronto Raptors"
y_e = "Milwaukee Bucks"
$\tilde{y}_e$ = "Milwaukee Bucks"

Figure 1: The model editing problem. Can we modify a base model's behavior locally given a description of the desired change?

Recently, retrieval-based methods such as SERAC [1] and RETRO [10] have shown strong performance in editing knowledge in language models. SERAC is a retrieval-based editing algorithm which augments a pretrained, frozen base model with a retrieval-based component consisting of an external memory and a trainable neural network called the *counterfactual model*. Edits are stored as text in the external memory, and the counterfactual model is able to generate outputs which are conditioned on memories. The base model and the retrieval-based component are *completely decoupled*, meaning that on some input, *either* the base model's output *or* the counterfactual model's output is used, but not both. Contrary to SERAC, RETRO *completely couples* the base model and retrieval-augmented component, training a base model to apply attention over both input prompts and retrieved texts.

Retrieval-based algorithms which couple the base model and retrieval-based component typically require finetuning of the base model to ensure that the base model is able to use retrieved texts. While decoupling avoids this expensive training procedure, performance scales only with the size of the retrieval-based component which is used for output, not with the size of the base model. In practice, the base model may be much larger than the retrieval-based component because increasing the size of models has been demonstrated to improve performance on many tasks [2, 3]. Is it possible to get the best of both worlds, avoiding the computational costs of training the base model while having performance scale with the base model's size?

We introduce a novel model editing algorithm which *weakly couples* its base and retrieval-augmented models. Our prompt-based editing approach replaces SERAC's counterfactual model with a *prompter* model whose outputs are used as input to the base model to obtain the edited model's final output. Unlike in SERAC, the base model is explicitly used to generate output text, so in principle its parametric knowledge can be utilized. We show that empirically, our prompt-based editor approaches the performance of SERAC on three editing tasks.

We demonstrate that like SERAC, our prompter can be trained for one base model and then deployed to other base models without any additional training. This hypothesis relies on two assumptions. First, all base models must use the same vocabulary for compatibility with the prompter's predicted distributions. Second, we assume that for different base models which share a common vocabulary and pre-training dataset, the word embeddings are relatively similar between different models.

In the broader context of retrieval-augmented models, our prompt-based approach may improve the robustness of systems which use natural language queries. Recent works [5, 11] have demonstrated that retrieval-based language models are highly sensitive to the particular phrasings of inputs, which we verify empirically in Table 4 of the Appendix, which shows qualitatively that a question-answering model can produce many different answers to semantically equivalent questions. A continuous prompter such as our method may mitigate this failure mode by transforming natural language prompts into a form which more consistently modulate base models' behavior.

3 Background

3.1 The model editing problem

In the **model editing problem**, we are given a pretrained base model f and a description of the desired change in behavior, from which we produce an edited model $\tilde{f}$. The description of the change

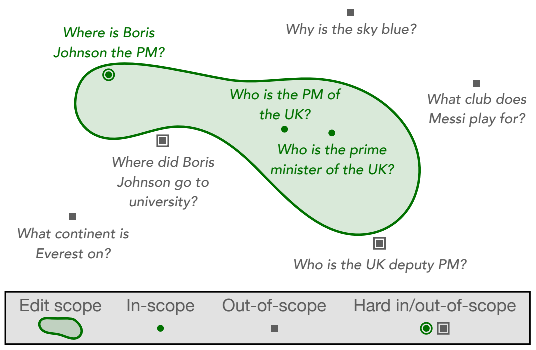
Where is Boris Johnson the PM?
Why is the sky blue?
Who is the PM of the UK?
What club does Messi play for?
Who is the prime minister of the UK?
Where did Boris Johnson go to university?
What continent is Everest on?
Who is the UK deputy PM?
Edit scope
In-scope
Out-of-scope
Hard in/out-of-scope

Figure 2: Edit scopes in a QA setting. The green region depicts the edit scope of the question 'Who is the PM of the UK?', including semantically equivalent and entailed questions.

is encoded in an *edit descriptor* z_e. An edit descriptor may be, for example, a question-answer pair corresponding to how a QA model should answer that question; in general, it may be any text which defines the behavior. The edited model $\tilde{f}$ should change its behavior exactly on post-edit inputs x which are *in the scope of* z_e, meaning that the 'correct' behavior of a model on input x changes given z_e. Meanwhile, the behavior of $\tilde{f}$ should not differ from that of f on post-edit inputs which are out of the scope of z_e. We define the *edit scope* of z_e to be the set containing exactly these in-scope input-label pairs, and denote it by $S(z_e)$.

The edit scope is an abstract notion which is often not easy to explicitly define in practice. To actually train and evaluate model editors, we use sampling functions $I(\cdot;\mathcal{D}_e)$ and $O(\cdot;\mathcal{D}_e)$. Given an edit descriptor, $I(z_e;\mathcal{D}_e)$ gives an in-scope input-label pair from $\mathcal{D}_e$, and $O(z_e;\mathcal{D}_e)$ gives an out-of-scope input from $\mathcal{D}_e$. See Figure 2 for an illustration of edit scopes in a question-answering setting.

We let $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ denote the spaces of post-edit inputs, post-edit labels, and edit descriptors, respectively. Each of $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ captures a space of *token-vector sequences*. We define a token-vector to be a vector in the $|V|$-dimensional probability simplex, where V is the vocabulary of our base model f. A token-vector derived from a token of text is a $|V|$-dimensional one-hot vector; inputs are given as text in our experiments, but in principle need not be. The embedding of a token-vector v is obtained by matrix-vector multiplication $E^\top v$ where $E \in \mathbb{R}^{|V|\times d}$ is a token embedding matrix.

3.2 Semi-parametric editor with a retrieval-augmented counterfactual model

SERAC is a retrieval-based model editing algorithm which achieves state-of-the-art performance in question-answering, natural language inference, and conversational sentiment modulation editing tasks. SERAC augments a pretrained *base model* with two additional models, the *scope classifier* and *counterfactual model*, as well as an external memory. Outputs are modified using a retrieval-and-rerouting mechanism: a classifier determines whether an edit should be applied, and if so, the output of the counterfactual model is used in place of the output of the base model. In other words, the base and counterfactual models are completely decoupled.

Formally, SERAC takes the form $\tilde{f}_\theta(\,\cdot\,;\phi,\psi,Z_e) : \mathcal{X} \mapsto \mathcal{Y}$. θ, the parameters of the base model, are left unmodified by SERAC's training procedure. ϕ parameterizes the scope classifier $g_\phi : \mathcal{Z} \times \mathcal{X} \to [0,1]$. g_ϕ jointly embeds edit inputs x and edit descriptors z_e into a common representation space. It then estimates the log-probability that an edit input x is in the scope of an edit descriptor z_e either

- as the negative squared Euclidean distance between their representations (which we call the **embed** version), or
- via cross-attention between edit inputs and edit descriptors (which we call the **cross-attention** version).

ψ parameterizes the counterfactual model $h_\psi : \mathcal{Z} \times \mathcal{X} \to \mathcal{Y}$, which predicts a label $y \in \mathcal{Y}$ to edit input $x \in \mathcal{X}$, conditioned on edit descriptor $z_e \in \mathcal{Z}$.

A conceptual limitation of SERAC is the fact that its predicted output text is completely generated by the counterfactual model upon edits. On the other hand, existing works demonstrate that performance scales with number of model parameters on tasks such as closed-domain QA and NLI ([2], Table 14). As SERAC returns the outputs of its counterfactual model, its performance is limited by the size of the counterfactual model. The counterfactual model must be fine-tuned on the editing task at hand, so the computational costs of training SERAC scale with the size of the counterfactual model. On the other hand, while the computational complexity of training SERAC doesn't scale with the size of the base model, SERAC is unable to exploit the base model's size to achieve better performance.

4 Approach

4.1 Method

Like SERAC, our prompt-based model editor takes the form $\tilde{f}_\theta(\,\cdot\,;\phi,\psi,Z_e) : \mathcal{X} \mapsto \mathcal{Y}$ with base model parameters θ and auxiliary model parameters ϕ and ψ. ϕ parameterizes a scope classifier model $g_\phi : \mathcal{Z} \times \mathcal{X} \to [0,1]$, identical to that of SERAC, as described in 3.2. Instead of a counterfactual model, ψ parameterizes a *prompter model* $h_\psi : \mathcal{Z} \times \mathcal{X} \to \mathcal{Z} \times \mathcal{X}$. On an input sequence of length T, h_ψ predicts a token-vector sequence to be used as an modified prompt for the base model.

Applying edits. We describe the computation of $\tilde{f}_\theta(x;\phi,\psi,Z_e)$ on an edit input-label pair $(x,y) \in \mathcal{X} \times \mathcal{Y}$. For each $z_e \in Z_e$, the classifier is used to predict the similarity between z_e and x; we write $\beta \leftarrow g_\phi(z_e^\star, x)$ where $z_e^\star = \arg\max_{z_e \in Z_e} g_\phi(z_e^\star, x)$. Intuitively, β denotes the probability of x being in the scope of the most likely z_e. If $\beta \geq 0.5$ (meaning that we predict x to be in the scope of $z_e^\star$), we compute a continuous prompt $\tilde{x} \leftarrow h_\psi([z_e^\star, x])$; otherwise, we take $\tilde{x} \leftarrow x$ (meaning we leave the prompt untouched). Finally, the output is computed using the base model: $\tilde{f}_\theta(x;\phi,\psi,Z_e) \leftarrow f_\theta(\tilde{x})$.

Training. g_ϕ and h_ψ are trained separately in a supervised fashion using a dataset of edit descriptors $\mathcal{D}_e = \{z_e^i\}_i$. g_ϕ is trained to classify whether each input is in the scope of a descriptor using a binary cross-entropy loss:

$$\ell(\phi) = -\mathop{\mathbb{E}}_{\substack{z_e \sim \mathcal{D}_e \\ (x_{\text{in}},\cdot) \sim I(z_e;\mathcal{D}_e) \\ x_{\text{out}} \sim O(z_e;\mathcal{D}_e)}} [\log g_\phi(z_e, x_{\text{in}}) + \log(1 - g_\phi(z_e, x_{\text{out}}))].$$

In-scope samples are called *edit samples*, while out-of-scope samples are called *locality samples*.

The prompter is only trained on inputs which have a corresponding edit descriptor in memory. As in test time, the prompter computes $h_\psi([z_e, x])$, a soft weighting of token embeddings which are used to generate a continuous prompt for the base model; note that in training, the *true* corresponding edit descriptor z_e to edit input x is used as input to h_ψ. h_ψ is trained using a language modeling objective to maximize the log-likelihood of the base model predicting the correct label:

$$\ell(\psi) = -\mathop{\mathbb{E}}_{\substack{z_e \sim \mathcal{D}_e \\ (x_{\text{in}},y_{\text{in}}) \sim I(z_e;\mathcal{D}_e)}} \log p_\theta(y_{\text{in}} \mid \tilde{x}_{\text{in}};\psi).$$

Additionally, we regularize the prompter's token-vector outputs to be close to the original text input in token-vector-wise KL divergence:

$$\ell_{\text{KL}}(\psi) = \mathop{\mathbb{E}}_{z_e \sim \mathcal{D}_e} [\text{KL}(z_e^\star, h_\psi([z_e^\star; x])].$$

The total loss is

$$\ell(\phi,\psi) = \ell(\varphi) + c_{\text{edit}}\ell(\psi) + c_{\text{kl}}\ell_{\text{KL}}(\psi).$$

where c_{edit} and c_{KL} are hyperparameters. Crucially, note that the base parameters θ are not trained.

4.2 Baselines

SERAC. For a detailed description of SERAC, see Section 3.2. When SERAC's counterfactual model has the same size as our base model, SERAC's edit success serves as a weak upper-bound to that of our method: SERAC's outputs are produced by its counterfactual model which is explicitly trained to predict the correct label given $[z_e^\star, x]$, whereas outputs of our method are produced by its base model which is not trained. However, it is, in principle, possible for the prompter to perform better than SERAC because the prompter and base model are applied in sequence on the retrieved input $[z_e^\star, x]$ to get the output.

Problem	Edit Descriptor z_e	In-scope input $x_{\text{test}} \sim I(z_e)$	Out-of-scope input $x_{\text{loc}} \sim O(z_e)$
FC	As of March 23, there were 50 confirmed cases and 0 deaths within Idaho. *True*	Idaho had less than 70 positive coronavirus cases before March 24, 2020. *True*	Allessandro Diamanti scored six serie A goals.
	Between 1995 and 2018, the AFC has sent less than half of the 16 AFC teams to the Super Bowl with only 7 of the 16 individual teams making it. *True*	–	The AFC sent less than half of the 16 AFC teams to the Super Bowl between 1995 and 2017.
QA	Who is the Sun Public License named after? *Sun Micro Devices*	The Sun Public License has been named for whom? *Sun Micro Devices*	What continent is Mount Whillans found on?
QA-hard	What type of submarine was USS Lawrence (DD-8) classified as? *Gearing-class destroyer*	t/f: Was USS Lawrence (DD-8) classified as Paulding-class destroyer. *False*	What type of submarine was USS Sumner (DD-333) classified as?

Table 1: Examples from the datasets in our experiments. FC tests a model editor's ability to perform NLI-like reasoning from facts given in an edit descriptor. QA and QA-hard evaluate editors' abilities to locally edit their factual knowledge. Table reproduced from [1].

RP. Retrieval-and-prompting is an ablation of SERAC where the retrieved prompt $[z_e^\star, x]$ is passed directly into the base model. If the prompter model in our method learned an identity mapping, it would recover the behavior of RP, and hence RP provides a lower-bound to the edit success of our method. Comparing our method to RP allows us to quantitatively understand the benefits of transforming the prompt to the base model.

5 Experiments

5.1 Data

We evaluate our method in three experimental settings which we describe below. Examples of the data used for training and evaluation are provided in Table 1.

FC. We use the fact-checking setting introduced in the SERAC paper. FC is a NLI task which uses the VitaminC dataset from [12] which assesses a model's ability to verify claims given small factual changes. VitaminC consists of over $400,000$ claim-evidence-label tuples; the model must label a claim as 'true' (1) if it follows from the evidence, 'false' (-1) if it contradicts it, or 'neither' (0). To frame NLI as an editing problem, we use the evidence as edit descriptors and the claims and labels for test-time edit inputs and labels, respectively. In other words, a model that is edited using the evidence should be able to deduce the label given the claim.

Note that only samples with a label of 'true' or 'false' are used to make edits, since a claim which is neither entailed nor contradicted by some evidence cannot be used to assess the quality of the edit nor to train the editor. For data samples which have a label of 'true', the claim and label are used as an in-scope sample. For samples which have a label of 'false', the claim is treated as a hard out-of-scope sample.

QA and QA-hard. We also adopt the QA and QA-hard settings from the SERAC paper. These are closed-domain question-answering tasks assessing a model editor's ability to modify factual knowledge. The QA setting was first proposed in [7], drawing from the Zero-Shot Relation Extraction (zsRE) dataset of [13]. The QA dataset contains $218,457$ train, 882 validation, and $24,968$ test samples. Each sample contains two question-answer pairs, one to be used as an edit descriptor for a model editor and the other to be used as a test input-label pair. An edited model must predict the correct answer to the second question after conditioning on the first question-answer pair.

In the regular QA setting, the two question-answer pairs are semantically equivalent. The locality of edits is evaluated by randomly sampling question-answer pairs from the training dataset. In the QA-hard setting, the test input-label question-answer pair may be equivalent or *entailed* by the edit descriptor; entailed questions include implications and yes-no questions. Locality is evaluated by sampling question-answer pairs which are similar to but out of the edit scope of the edit descriptor, as evaluated by a pretrained `all-MiniLM-L6-v2` sentence-embedding model from Sentence Transformers [14].

5.2 Evaluation method

We evaluate our method using two quantitative, automated metrics defined in [1].

Dataset	Model	Metric	SERAC	**Prompter**	RP
FC	BERT-base	↑ ES	**0.857**	**0.847**	0.528
		↓ DD	0.087	0.075	**0.015**
QA	T5-large	↑ ES	**0.986**	0.961	0.487
		↓ DD	**0.009**	**0.009**	0.03
QA-hard	T5-large	↑ ES	**0.913**	0.841	0.278
		↓ DD	**0.028**	0.039	**0.027**

Table 2: Comparison of our method with other editors on three experimental settings. We report numbers on the validation set using $k = 5$ simultaneous edits for FC and $k = 10$ simultaneous edits for QA and QA-hard. The prompt-based editor achieves essentially the same performance as SERAC on FC, nearly the same performance on QA, and slightly worse performance on QA-hard. RP is unable to achieve comparable edit success in any of the three settings, indicating that the prompter learns to predict continuous prompts which to the base model are more useful than the original text prompts.

First, *edit success* (ES) measures a model editor's ability to make sufficiently general edits. Concretely, ES counts the number of matching edited-model predictions and true labels for in-scope inputs:

$$\text{ES}(z_e) := \mathop{\mathbb{E}}_{x_{\text{test}} \in I(z_e; \mathcal{D}_e)} \left[\mathbb{1}\{\tilde{f}(x_{\text{test}}) = y_{\text{test}}\} \right].$$

Edit success ranges between 0 and 1, with **higher** edit success indicating better performance.

Second, *drawdown* (DD) measures the locality of edits, counting the number of non-matching edited-model predictions and true labels for out-of-scope inputs:

$$\text{DD}(z_e) := \mathop{\mathbb{E}}_{x_{\text{out}} \in O(z_e; \mathcal{D}_e)} \left[\mathbb{1}\{\tilde{f}(x_{\text{out}}) \neq f(x_{\text{out}})\} \right].$$

Drawdown also ranges between 0 and 1, with **lower** drawdown indicating better performance.

We evaluate our method and baselines on batches of edits. For each sampled batch of k edits, a model editor is first given k edit descriptors to produce an edited model $\tilde{f}$ from a base model f. We then compute the edit success and drawdown of $\tilde{f}$ on the k edit input-label pairs in the batch. We repeat the procedure over many batches to obtain our reported estimates the ES and DD.

5.3 Experimental details

We report the hyperparameters used for our experiments.

Hyperparameter	FC	QA	QA-hard
Prompter model	`bert-base`	`t5-small`	`t5-small`
Classifier type	cross-attention	embedding	embedding
Learning rate	1×10^{-5}	1×10^{-4}	1×10^{-4}
c_{edit}	10	100	100
c_{KL}	0	0.1	0.1
Batch size	5	10	10

We applied early stopping on validation edit success with a patience of $40,000$ steps for all experiments. All experiments were run with on a single random seed.

I implemented the prompt-based editor algorithm in the official SERAC code and adopted their experimental setup.

5.4 Main results

Our quantitative results on the three experimental settings are in Table 2.

Our method's performance essentially matches SERAC's in the FC setting. On the other hand, RP is unable to achieve edit success much higher than 50% (i.e. random guessing) on FC. Note FC is

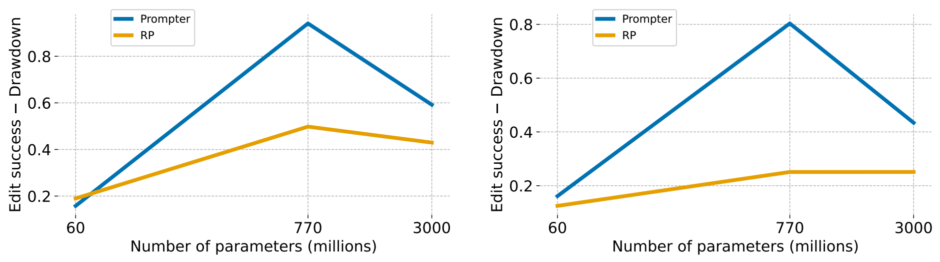

Figure 3: Evaluating prompt-based model editors on pretrained base models of varying architectures. *Left*: a prompt-based editor trained on QA using a T5-large (770m parameters) base model is evaluated on QA using T5-small (60m parameters) and T5-XL (3b parameters). *Right*: the same evaluations are performed on QA-hard.

the easiest setting for the prompt-based editor, since it is a decision problem with only two possible outputs ('true' and 'false'). As the output space is so small, we expect the prompt-based editor to perform about as well as SERAC; to see why, one could imagine the prompter model simply solving the NLI task and then passing the answer (in some form) to the base model. This experiment demonstrates that in a decision problem, the prompter is able to predict prompts which can reliably change the base model's outputs.

The prompt-based editor performs nearly as well as SERAC on QA and slightly underperforms SERAC on QA-hard. On the other hand, RP is unable to achieve close to the same edit success in either setting. As such, the good performance of the prompter is surprising, given that in our experiments we regularize the predicted prompts to be close in KL divergence to the retrieved prompts per token-vector. Note QA and QA-hard are more difficult for the prompter because these are function problems where the edited model must generate an answer to the input question. In particular, the answers to QA-hard questions are not necessarily the same as the answers to the corresponding edit descriptor questions. These experiments suggest that even in function problems, there exists a perturbation of a retrieved prompt into a sequence of token-vectors which reliably modulates the base model's outputs.

5.5 Scaling experiments

We experiment with evaluating our trained prompt-based editors on different base models in the QA and QA-hard tasks. Concretely, the base model is changed from T5-large to either T5-small or T5-XL, while the two auxiliary models are kept constant, using model weights obtained from training. We apply the same evaluation process as we did in our main experiments.

Figure 3 compares the edit performances of our prompt-based editor and RP on different base models. The prompt-based editor signficantly outperforms RP using T5-XL as the base model. In other words, **the same predicted prompts which reliably change the behaviors of T5-large also often change the behaviors of T5-XL**. The predicted soft prompts work better than the retrieved text prompts which are used by RP.

Our method performs similarly to RP using T5-small. Perhaps the reason we don't see the same increase in performance is because larger models are more powerful than smaller models. Thus, we would expect that prompts which work well for T5-large will also work well for T5-XL, but not necessarily for T5-small. Our results do not completely affirm our expectations; future work could explore methods to improve the behavior of our editor upon scaling the base model. It appears that our algorithm is (at least somewhat) able to 'scale up' to larger base models but not necessarily 'scale down', so perhaps the prompter should be trained on the smallest base model available.

6 Analysis

To better understand the behavior of our prompter models, we include samples of continuous prompts, which are predicted from retrieved edit descriptors concatenated with test-time edit inputs. These prompts were generated by evaluating the model editors on the validation sets for each task. For each

Experiment	Input	Predicted prompt	Base model prediction	Label	Correct?
FC	rotten tomatoes gives the equalizer 2 an approval rating of 51 % based on fewer than 174 reviews.	[unused613] [unused613] [unused613] [unused613] [unused613] …	false	false	✓
	john legend's real name is john roger stephens.	[unused613] [unused613] [unused613] [unused613] [unused613] …	false	false	✓
	after march 28, 2018, see you again had more than 644, 000 dislikes.	[unused520] [unused520] [unused520] [unused261] [unused261] [unused520] [unused261] …	true	true	✓
	more than 711, 500 cases of covid - 19 had been reported during the 2019 - 20 pandemic.	[unused613] [unused613] [unused613] [unused613] [unused613] [unused613] …	true	false	✗
QA-hard	What team is Julien Sprunger associated with?	Quel team is Julien Sprunger associated with or Minnesota North Stars which team is Julien Sprunger associated with? which	Minnesota North Stars	Minnesota North Stars	✓
	answer true or false: Khan is the position of Ambaghai.	<extra_id_0> or</s></s> of True True True True True emailsa answer True or false: True answer True True True True True True True True</s>	False	False	✓
	Where did Lothar Friedrich von Metternich-Burscheid die?	Frage area or Lothar Friedrich von Metternich-Burscheid die oder Vienna Which did Lothar Friedrich von Metternich-Burscheid die? what	Wien	Vienna	✗
	What is associated with aerospace design?	Frage industry or Aerospace Valley associated with or aerospace design<extra_id_23> are associated with aerospace design? what	Aerospace Valley	Aerospace Valley	✓
	What caused Gary Moore to die?	Frage caused did Gary Moore have or bone cancer Which caused Gary Moore to die? what	Gary cancer	bone cancer	✗

Table 3: Examples of continuous prompts generated by our method on the FC and QA-hard settings. Higher ES minus DD indicates better performance. Prompter outputs on FC and QA-hard true/false questions (third example) degenerate, while prompter outputs on QA-hard rephrases and implications (second-to-last example) appear to be close to the retrieved prompts.

token-vector of the continuous prompt, we took the nearest neighboring token in Euclidean distance. We then converted the nearest tokens into text sequences which intuitively represent approximately what the continuous prompts 'say'. In Table 3 we compare these nearest-neighbor sequences, under the 'predicted prompts' column, to the edit inputs. Additionally, we compare the base model's predictions on the continuous prompts to the true labels to qualitatively evaluate success.

In the FC setting, the prompter outputs are uninterpretable sequences consisting almost entirely of unused tokens. The apparent lack of variation in the predicted sequences suggests that the prompter is itself classifying solving the fact-checking task, with the predicted prompts simply determining the base model's output. Based on this interpretation of the prompter's behavior as 'cheating', the base model is not used to solve the fact-checking task and thus we should not expect performance to scale with the number of base model parameters. On the other hand, the analysis demonstrates that continuous prompts are expressive enough to completely determine the output of the base model, at least when the output space is very small.

In the QA-hard setting, the prompter's outputs are varied; for rephrased and implied questions, the prompter generates sequences whose textual approximations are intelligible, while for true/false questions, the prompter's outputs are degenerate as in the FC setting. For rephrases and implications, the predicted prompts appear to be close to retrieved prompts, and in particular the text from the edit input seems to be intact. In our examples (and based on quantitative results), the base model is often, but not always, able to correctly select the correct answer from the continuous prompt. It appears that the word 'or' appears directly before the answers of the soft prompts; by visualizing attention maps, one could verify that this particular token is important to the base model generating the correct output.

From our analysis we conclude that in the QA-hard setting, our prompter model can learn a continuous perturbation of the retrieved prompt which consistently modifies the base model's outputs but does not differ much from the retrieved prompt. Future work may attempt to apply similar models applied across different tasks to address the general problem of sensitivity of language models to the particular phrasings of prompts.

7 Related Work

Model editing. Earlier model editing approaches focus on learning model editors which update the parameters of a base model in order to make a desired edit. Editable neural networks [15] apply a meta-learning post-training procedure to make the parameters of a model 'editable' by fine-tuning. KnowledgeEditor [7] and SLAG [9] perform rank-1 updates to model parameters in attempts to edit parametric factual knowledge of these models. MEND [8] learns a transformation of fine-tuning gradients which can allow for effective fine-tuning with modified gradients. While these editors have proven to be somewhat successful at various editing tasks, they depend heavily on the specific parameterization of the base model being edited. Thus, applying these parametric editor algorithms requires training a new editor model for each base model. Moreover, as noted in [9] and [1], due to the distributed nature of knowledge in deep neural networks, parametric approaches struggle to distinguish between questions which are or aren't implied. Retrieval-based methods such as SERAC and our method have recently been used in model editing as a more successful alternative to gradient-based methods.

Memories and retrieval in language models. As described in the introduction, SERAC [1] and RETRO [10] provide two contrasting approaches to retrieval-based knowledge editing. Our method strikes a balance by weakly coupling its base model and retrieval-augmented components, leaving the base model untrained while still using its outputs.

Longpre et al. [11] study the utilization of retrieved information in retrieval-based models, introducing a training procedure which increases models' reliance on retrieved information. Our work touches on the same problem, attempting to transform retrievals to be utilized more effectively by pretrained models.

Continuous prompting. In the works introducing the GPT models [16, 5], the authors observed that natural language prompts are effective for specifying the behaviors of large language models. Li and Liang [17] introduced *prefix-tuning*, i.e. learning a prefix to condition a language model on a downstream task, where the prefix is a sequence consisting of continuous word embeddings and higher-level activations. Lester et al. [18] extended the idea of prefix-tuning to larger language models and showed that it is sufficient to only learn the continuous word embeddings. Our work further builds on these ideas by learning a model to *dynamically* predict continuous prompts which encode information specific to an edit, rather than generally conditioning a model for a task.

8 Conclusion and Future Work

We introduce a prompt-based editing algorithm and experimentally verify that it performs well on three experimental settings. In our experiments, our method shows promise in being able to generalize to different base model architectures, with the predicted prompts generalizing more effectively than retrieved text prompts. Our qualitative analysis indicates that there exist continuous relaxations of prompts which are effective at modulating base model outputs.

The key limitations of this work are as follows:

- In our experiments, our method's performance approaches but does not exceed that of SERAC, the current state-of-the-art.
- The prompt-based editor cheats when the output space is small, using purely the prompter to generate an answer. As such, we wouldn't expect performance to scale with model size in such cases. An avenue for future work is to explore techniques to ensure the editor doesn't behave this way. One potential idea is to use stronger or alternative forms of regularization of the predicted prompts. Another idea is to train a single model editor using multiple base models to improve the generality of the predicted prompts.
- The results on scaling the base model, while promising, do not demonstrate that the performance of our editor scales with the size of the base model, as we might expect to be possible. Future work might analyze methods to improve performance when changing base model architectures; for instance, one could look for ways to exploit shared structure between the word embeddings of different checkpoints.

References

[1] Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. Memory-based model editing at scale, 2022.

[2] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[3] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model?, 2020.

[4] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *Neural Information Processing Systems*, 2020.

[6] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

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

[8] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Fast model editing at scale, 2021.

[9] Peter Hase, Mona Diab, Asli Celikyilmaz, Xian Li, Zornitsa Kozareva, Veselin Stoyanov, Mohit Bansal, and Srinivasan Iyer. Do language models have beliefs? Methods for detecting, updating, and visualizing model beliefs, 2021.

[10] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. *CoRR*, abs/2112.04426, 2021.

[11] Shayne Longpre, Kartik Perisetla, Anthony Chen, Nikhil Ramesh, Chris DuBois, and Sameer Singh. Entity-based knowledge conflicts in question answering, 2022.

[12] Tal Schuster, Adam Fisch, and Regina Barzilay. Get your vitamin C! robust fact verification with contrastive evidence. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 624–643, Online, June 2021. Association for Computational Linguistics.

[13] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 333–342, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[14] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[15] Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks, 2020.

[16] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

[17] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.

[18] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[19] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.

Prompt	Output
Who is the manufacturer of Covid-19 vaccine?	Johnson & Johnson
Which company makes vaccines for Covid?	Immunoscience
Who won the last NBA Finals?	Golden State Warriors
Who won the last NBA championship?	Cleveland Cavaliers
Who are the reigning NBA champions?	The Los Angeles Lakers
Where was the president of the US born?	the Netherlands
Where was the US president born?	the Philadelphia Hospital
What is the birthplace of the incumbent US president?	Virginia

Table 4: Sample inputs and outputs of a question-answering T5-large model pretrained on Natural Questions [19]. Rephrasings of the prompt change the model's answers unpredictably, indicating that the model's behavior is highly sensitive to the wording of the prompt.

A Appendix

Table 4 includes samples from a T5-large question-answering model which show that it is sensitive to the phrasing of questions.