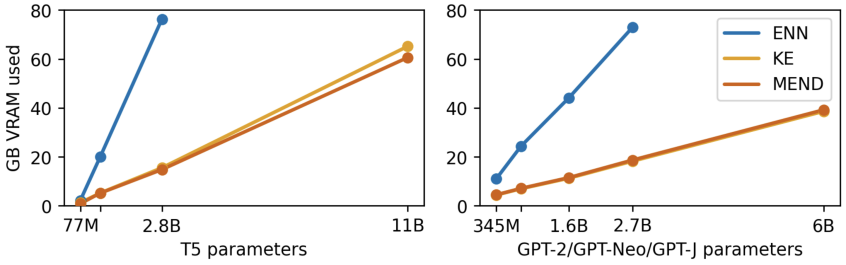

Figure 4: GPU VRAM consumption for training MEND, KE, and ENN in float32. MEND and KE's memory consumption remain tractable for a single GPU (using 2×bfloat16 memory usage (Wang and Kanwar, 2019) for T5-11B), while ENN's memory usage increases much more rapidly, making it impractical to run on a single GPU. Values are computed without gradient checkpointing. Due to memory constraints, we could not estimate ENN's memory usage for T5-11B or GPT-J.

to approximately enforce this constraint during training, ENN must use an extra copy of the original base model to ensure that the editable model's predictive distribution does not differ too much from it. This incurs significant additional memory costs, particularly when training ENN for very large models, for which the parameters of the model alone occupy a significant amount of VRAM. Another cause for the significant VRAM consumption of ENN is the need to compute activations and gradients for the model parameters; even if we edit only the last layer, ENN trains the rest of the model so that the last layer gradient is productive, requiring activations and gradients to be computed for the entire model. On the other hand, extrinsic editors like MEND and KE do not require updating the base model itself, thereby computing gradients for far fewer parameters. Future work might investigate approaches to reducing the memory consumption of ENN, although the requirement to retain a copy of the original model in order to enforce locality creates a relatively high lower bound on the amount of memory that ENN might use.

Regardless of memory consumption, extrinsic editors have the potential advantage of being able to edit more than one model; in theory, we might amortize the cost of training MEND over several base models at once. On the other hand, intrinsic editability must by definition be re-learned separately for each base model.

B.2 KNOWLEDGEEDITOR (KE)

De Cao et al. (2021) propose KNOWLEDGEEDITOR, a hypernetwork-based approach for editing the knowledge in language models. KE is an RNN that conditions explicitly on the input, incorrect output, and new desired label and outputs a mask m_i, offset b_i, and a scalar scaling factor α to the gradient ∇_{W_i} for several of the weight matrices in a transformer model, where $m_i, b_i, \nabla_{W_i} \in \mathbb{R}^{d \times d}$ for a $d \times d$ weight matrix. The update to the model is $\theta' = \theta - \alpha(m_i \odot \nabla_{W_i}) + b_i$. Because the weight matrices in state-of-the-art transformer models are very high-dimensional, the mask and offset output by KE are rank-1 to retain tractability.

Comparing KE and MEND. KE more closely resembles MEND in that it is also an extrinsic model editor. However, while MEND directly maps model gradients into model edits, the KE model editor uses the raw edit example as an input, outputting a single rank-1 mask and rank-1 offset over the fine-tuning gradient. We hypothesize that the KE model faces several challenges that MEND avoids. First, mapping the edit example itself into a model updates requires a translation from the high-level modality of data examples into the very low-level modality of model parameter updates. Solving this translation requires making additional design decisions (e.g., how to feed the edit input and label into the editor, what architecture to use for the editor), the optimal design for which may vary across problems. Further, by not conditioning directly on the gradient, KE forgoes a rich source of information about which parameters of the model are most responsible for updating the model's outputs. In addition, by operating on the token-wise activations and gradients (i.e., the gradients are not summed over the sequence/batch, but are kept as per-sequence element activation and gradient vectors), MEND outputs a rank-1 model edit for each token in the input and output sequence. The final output of MEND is the sum of these, which has rank of order 10 or even 100, depending on the problem. In contrast, the KE editor outputs only a rank-1 gradient mask and rank-1 gradient offset, regardless of the information content of the edit example. This rank-1 constraint, irrespective of the size of the input, which we hypothesize causes KE's failure to perform well for the Wikitext editing

x_e, y_e	Nepal borders France. **Yes**
x_{loc}	Belgium is made up of three regions.
x'_e, y'_e	Nepal is bordered by France. **Yes**

(a) FEVER fact-checking editing dataset example. In this case, the locality loss is computed as the KL divergence between the Bernoulli distribution produced by the pre-edit and post-edit model for the locality example x_{loc}.

x_e	Which continent is Mount Andrews on? **South America**
x_{loc}, y_{loc}	To which fictional work does Dennis Rickman belong in? **EastEnders**
x'_e, y'_e	In which continent is Mount Andrews located? **South America**

(b) zsRE question-answering editing dataset example. Because computing the KL divergence of the model over all possible answers to the question is computationally expensive, we use the label (EastEnders) and compute the KL divergence between the pre- and post-edit model at each of these tokens as an approximation.

Table 7: Editing data samples from the FEVER fact-checking and zsRE question-answering editing datasets from De Cao et al. (2021). **Bold text** corresponds to labels used for editing or approximating the locality constraint.

task, which has significantly higher information content labels (10 tokens) than the FEVER or zsRE tasks.

C Experimental Details

For GPT and BERT-style models, all experiments edit the MLP weights in the last 3 transformer blocks (6 weight matrices total). For BART and T5-style models, all experiments edit the MLP weights in the last 2 transformer blocks in both the encoder and the decoder (8 weight matrices total). We found that editing MLP layers generally provides better editing performance (across algorithms) than editing attention layers. In line with past work (De Cao et al., 2021), all reported performance numbers are on the validation set. For all algorithms, we use early stopping to end training early if the validation loss $L = c_{\text{edit}} L_e + L_{loc}$) does not decrease for 20000 steps on a subset of 500 validation examples, with a maximum number of training steps of 500,000. We use a batch size of 10 (with gradient accumulation) and the seed 0 for all experiments. Tables 7 and 8 show examples from each dataset used in our experiments.

C.1 Hyperparameters

Fine-tuning. The fine-tuning baselines use model-dependent learning rates, which we found important in achieving good fine-tuning performance; using too large of a learning rate causes decreased locality (increased model degradation), while a learning rate too small causes slow edits. We use edit learning rates of 5e-6 for GPT-Neo and GPT-J and 1e-4 for T5 models, and 1e-6 for the smaller models, aiming to complete edits in less than 100 fine-tuning steps (as in De Cao et al. (2021)). For the fine-tuning + KL-constraint baseline, we fine-tune on the loss $c_{\text{edit}} L_e + L_{loc}$, using a smaller c_{edit} than for the learned algorithms (1e-2 for all models except GPT-J, which required 1e-3). Larger values of c_{edit} provide little benefit from the locality loss. To compute L_{loc}, we use a batch size of one new example x_{loc} from the full edit training set D^{tr}_{edit} at each time step.

ENN. We use an initial inner loop learning rate of 1e-2, but allow this value to be learned in the outer loop, which we find improves performance over the fixed inner loop learning rate version in Sinitsin et al. (2020). For all experiments, ENN fine-tunes all model parameters during training (even when we only edit the last few layers). We also use only a single inner loop update step for computational reasons, which differs from the multi-step version used for the smaller models used by Sinitsin et al. (2020). Our edit loss is also a slight simplification of the edit loss used by Sinitsin et al. (2020), which is

$$l_e(\theta) = -\log p_\theta(y_e|x_e, \theta) + \max_{y_i} \log p_\theta(y_i|x_e, \theta) \tag{6}$$

The first term of this loss is the edit loss we use in our work; the second term is primarily intended to provide the property that $l_e(\theta) \leq 0$ when an edit is successful so that the iterative editing process can be stopped. However, in this work, because we use only a single gradient step of editing for

ENN, this property is less important, and the second term simply amounts to an additional emphasis on pushing down specifically the largest incorrect logit (which the first term already does implicitly).

KE We use the implementation of KE provided by De Cao et al. (2021), which can be found at https://github.com/nicola-decao/KnowledgeEditor, with minor changes to the computation of the KL constraint for consistency with other algorithms (see below). We use a learning rate of 1e-5.

C.2 Computing the locality constraint

Computing the true KL-divergence between the pre- and post-edit model $\mathrm{KL}(p_\theta(\cdot|x_{\text{loc}})\|p_{\theta'}(\cdot|x_{\text{loc}}))$ quickly becomes computationally prohibitive for model outputs of more than a few tokens, requiring marginalization over possible answers. We therefore approximate this KL-divergence using samples from the dataset.[6] For the seq2seq question-answering problem, we evaluate the KL divergence only at the tokens of the answer y_{loc}, giving $\mathrm{KL}^{\text{seq2seq}}_{\text{approx}}(\theta,\theta') = \frac{1}{|y_{\text{loc}}|}\sum_{i=1}^{|y_{\text{loc}}|}\mathrm{KL}(p_\theta(\cdot|x_{\text{loc}},y_{\text{loc}}^{<i})\|p_{\theta'}(\cdot|x_{\text{loc}},y_{\text{loc}}^{<i}))$, where $p(\cdot|x_{\text{loc}},y_{\text{loc}}^{<i})$ is the distribution over next tokens y_i given the locality input x_{loc} and the label tokens for previous timesteps $y_{\text{loc}}^{<i}$. Similarly, for the Wikitext setting, we define $\mathrm{KL}^{\text{auto}}_{\text{approx}}(\theta,\theta') = \frac{1}{|x_{\text{loc}}|}\sum_{i=1}^{|x_{\text{loc}}|}\mathrm{KL}(p_\theta(\cdot|x_{\text{loc}}^{<i})\|p_{\theta'}(\cdot|x_{\text{loc}}^{<i}))$. For FEVER fact-checking we compute the exact KL-divergence between Bernoulli distributions in closed form.

C.3 Environment Details

All runs are trained entirely on a single NVIDIA RTX Titan or A40 GPU. No gradient checkpointing or memory-reduction optimizations are used, although bfloat16 is used to fit the largest T5 model onto our GPU. In full precision, the parameters alone of the T5-11B model use all of the memory of our largest GPU. VRAM consumption for training MEND and KE on T5-11B (Figs. 3 and 4) is estimated by doubling the bfloat16 VRAM usage (Wang and Kanwar, 2019). While doubling half precision enabled estimating the memory consumption of ENN, we were unable to train ENN in half precision without numerical instability. All models are based on Huggingface Transformers implementations (Wolf et al., 2019) with some modifications in line with De Cao et al. (2021). We use PyTorch (Paszke et al., 2019) for all experiments, specifically using the Higher library (Grefenstette et al., 2019) in order to implement the bi-level optimization in ENN as well as the inner loop of model editing for all algorithms.

C.4 Dataset Construction & Examples

Datasets are constructed to provide pairs of edit input x_{e} and plausible edit label y_{e}. The edit label is not necessarily the 'correct' label; the goal is to provide realistic instances of the *types* of data we would expect to see during test. For example, our dataset might have a sample such as x_{e} = *Where was Ursula K. Le Guin born?* and y_{e} = *Addis Ababa, Oromia, Ethiopia*, even though Ursula K. Le Guin was born in Berkeley, California, USA. However, this fictitious example is still a useful assessment of our model's ability to perform the general type of edit of 'change a person's birthplace'. For the zsRE question-answering dataset De Cao et al. (2021) generate fictitious y_{e} in this manner using the top predictions of a BART model fine-tuned on the task of question answering followed by manual human filtering. In practice, this produces alternate edit labels that are plausible and whose types match with the original label. For FEVER fact-checking, there are only two choices for labels, and we sample edit targets 1 and 0 with equal probability. For Wikitext generation, we use a distilGPT-2 model to generate plausible 10-token continuations for a given Wikitext prefix, with the similar motivation to zsRE of providing edit targets that share the structure of the types of edits that we will apply in practice, even if they are not always factual. When qualitatively assessing MEND to correct real errors of the base model using the factual labels, we find that MEND performs reliably, indicating that these label generators provide reasonable proxies for 'real' model edits.

[6] We justify this choice by the fact that the model's predictive distribution is similar to the locality sample distribution (as locality samples are drawn from the dataset the model was originally trained on). While this is not as principled as a true Monte Carlo estimate using samples from the model itself, it is reduces computational requirements of training and is easier to implement; the generally low drawdown for most models indicates that this approximation still provides a good locality constraint in practice.