Algorithm 1 MEND Training

1: **Input:** Pre-trained $p_{\theta_\mathcal{W}}$, weights to make editable $\mathcal{W}$, editor params ϕ_0, edit dataset D^{tr}_{edit}, edit-locality tradeoff c_{edit}
2: **for** $t \in 1, 2, \ldots$ **do**
3: Sample $x_e, y_e, x'_e, y'_e, x_{\text{loc}} \sim D^{tr}_{edit}$
4: $\tilde{\mathcal{W}} \leftarrow \text{EDIT}(\theta_\mathcal{W}, \mathcal{W}, \phi_{t-1}, x_e, y_e)$
5: $L_e \leftarrow -\log p_{\theta_{\tilde{\mathcal{W}}}}(y'_e|x'_e)$
6: $L_{\text{loc}} \leftarrow \text{KL}(p_{\theta_\mathcal{W}}(\cdot|x_{\text{loc}}) \| p_{\theta_{\tilde{\mathcal{W}}}}(\cdot|x_{\text{loc}}))$
7: $L(\phi_{t-1}) \leftarrow c_{\text{edit}} L_e + L_{\text{loc}}$
8: $\phi_t \leftarrow \text{Adam}\,(\phi_{t-1}, \nabla_\phi L(\phi_{t-1}))$

Algorithm 2 MEND Edit Procedure

procedure EDIT($\theta, \mathcal{W}, \phi, x_e, y_e$)
$\hat{p} \leftarrow p_{\theta_\mathcal{W}}(y_e|x_e)$, **caching** input u_ℓ to $W_\ell \in \mathcal{W}$
$L(\theta, \mathcal{W}) \leftarrow -\log \hat{p}$ ▷ Compute NLL
for $W_\ell \in \mathcal{W}$ **do**
$\delta_{\ell+1} \leftarrow \nabla_{W_\ell u_\ell + b_\ell} l_e(x_e, y_e)$ ▷ Grad wrt output
$\tilde{u}_\ell, \tilde{\delta}_{\ell+1} \leftarrow g_{\phi_\ell}(u_\ell, \delta_{\ell+1})$ ▷ Pseudo-acts/deltas
$\tilde{W}_\ell \leftarrow W_\ell - \tilde{\delta}_{\ell+1} \tilde{u}_\ell^\top$ ▷ Layer ℓ model edit
$\tilde{\mathcal{W}} \leftarrow \{\tilde{W}_1, \ldots, \tilde{W}_k\}$
return $\tilde{\mathcal{W}}$ ▷ Return edited weights

for batch element i (see Appendix D). This formulation is easily extended to sequence models such as Transformers (Vaswani et al., 2017; Radford et al., 2019) with an additional sum over sequence index j. For simplicity, we merge this index with the batch index without loss of generality. This decomposition enables a network to condition directly on the gradient of a single example with only $2d$ (rather than d^2) input neurons.[2] With this parameterization, MEND learns functions g_ℓ, with parameters ϕ_ℓ, which map u^i_ℓ and $\delta^i_{\ell+1}$ to *pseudoactivations* $\tilde{u}^i_\ell$ and *pseudodelta* $\tilde{\delta}^i_{\ell+1}$. The model edit for weight matrix W_ℓ is then

$$\tilde{\nabla}_{W_\ell} = \textstyle\sum_{i=1}^{B} \tilde{\delta}^i_{\ell+1} \tilde{u}^{i\top}_\ell. \tag{2}$$

To further reduce the number of additional parameters, MEND shares parameters across editor networks g_ℓ (note Figure 2 omits this for clarity). Because the sizes of u_ℓ and $\delta_{\ell+1}$ depend on the shape of the weight matrix W_ℓ, MEND learns a separate set of editor parameters for each unique *shape* of weight matrix to be edited. Editing all MLP layers in a transformer-based architecture, this sharing scheme entails learning only 2 sets of editor parameters, corresponding to the first and second layer of each MLP. To enable some layer-wise specialization, MEND applies a layer-specific scale s_ℓ and offset o_ℓ to the editor network hidden state and output, similar to FiLM layers (Perez et al., 2018). Putting everything together, a MEND network computes $g_\ell(z_\ell)$ where $z_\ell = \text{concat}(u_\ell, \delta_{\ell+1})$ as

$$h_\ell = z_\ell + \sigma(s^1_\ell \odot (U_1 V_1 z_\ell + b) + o^1_\ell), \qquad g(z_\ell) = h_\ell + \sigma(s^2_\ell \odot U_2 V_2 h_\ell + o^2_\ell) \tag{3a,b}$$

where σ is a non-linear activation function s.t. $\sigma(0) = 0$ (ReLU in this work) and U_j, V_j correspond to a low rank factorization of MEND's weights at layer j (keeping MEND's total parameters $O(d)$).

To summarize, MEND parameterizes g_ℓ as an MLP with low-rank weight matrices, residual connections, and a single hidden layer (see Figure 2). To edit layer ℓ, layer activations u^i_ℓ and output gradients $\delta^i_{\ell+1}$ are concatenated and passed together to g_ℓ, producing a vector of equal size, which is split into pseudoactivations $\tilde{u}^i_\ell$ and pseudodeltas $\tilde{\delta}^i_{\ell+1}$, ultimately producing $\tilde{\nabla}_{W_\ell}$ (Eq. 2). The final edited weights are $\tilde{W} = W_\ell - \alpha \tilde{\nabla}_{W_\ell}$, where α_ℓ is a learned per-layer (scalar) step size.

3.2 Training MEND

MEND uses an editing training set D^{tr}_{edit} to learn parameters ϕ_ℓ for each of the MEND networks g_ℓ. Before training, we select the weights of the model $\mathcal{W} = \{W_1, \ldots, W_M\}$ that we would like to make editable (e.g., the weight matrices in the last M layers). At each step of training, we sample an edit example (x_e, y_e), locality example x_{loc}, and equivalence examples (x'_e, y'_e) from the edit train set D^{tr}_{edit}. Recall that x_{loc} is sampled independently from the edit example, so that it is very likely that it is unrelated to the edit example. We use (x_e, y_e) to compute the raw gradient $\nabla_{W_\ell} p_{\theta_\mathcal{W}}(y_e|x_e)$ for each weight matrix $W_\ell \in \mathcal{W}$, using $\theta_\mathcal{W}$ to denote the model parameters with un-edited weights. We then compute the parameter update for each layer $\tilde{W} = W_\ell - \alpha_\ell \tilde{\nabla}_{W_\ell}$ ($\tilde{\nabla}_{W_\ell}$ from Eq. 2).

We compute the training losses for MEND using the edited model parameters $\tilde{\mathcal{W}}$, which we back-propagate into the editing networks. Note that we do not compute any higher-order gradients, because we do not optimize the pre-edit model parameters. The training losses are L_e, which measures edit success and L_{loc}, which measures edit locality (the KL divergence between the pre-edit and post-edit model conditioned on the locality input x_{loc}), defined as follows (also Alg. 1 lines 5–7):

[2] For a batch/sequence, we transform the gradient for each batch/sequence element independently and sum the result to acquire the final transformed gradient for the entire batch/sequence.

MEND losses: $L_e = -\log p_{\theta_{\tilde{\mathcal{W}}}}(y'_e|x'_e), \quad L_{loc} = \mathrm{KL}(p_{\theta_{\mathcal{W}}}(\cdot|x_{loc})\|p_{\theta_{\tilde{\mathcal{W}}}}(\cdot|x_{loc}))$. (4a,b)

Intuitively, L_e is small if the model has successfully updated its output for the edit example's equivalence neighborhood, while L_{loc} is small if the edit did not affect the model's behavior on unrelated inputs. The total training loss for a MEND network is computed as $L_{\text{MEND}} = c_e L_e(\theta_{\tilde{\mathcal{W}}}) + L_{loc}(\theta_{\mathcal{W}}, \theta_{\tilde{\mathcal{W}}})$. We optimize L_{MEND} with respect to the MEND parameters at each time step using the Adam optimizer (Kingma and Ba, 2015), using $c_e = 0.1$ for all experiments.

While MEND's parameterization can tractably *represent* a mapping from gradients to model edits, training the editor presents its own challenges. Appendix A describes MEND's identity initialization and input normalization, which our ablations in Section 5.4 show are important to effective edits.

4 Related Work

Various strategies for model editing exist, including modifications of standard fine-tuning intended to enforce locality by reducing distance traveled in parameter space (Zhu et al., 2020) or even find the min-L2 norm parameter update that reliably edits the model's output (Sotoudeh and Thakur, 2021). However, De Cao et al. (2021) observe that parameter-space constraints do not always translate to useful function-space constraints for neural networks. Our fine-tuning baselines thus use a KL-divergence constraint in function space, but, even with this modification, we find that fine-tuning generally doesn't consistently provide edit generality. Other approaches to editing such as Editable Neural Networks (**ENN**; Sinitsin et al. (2020)) or KnowledgeEditor (**KE**; De Cao et al. (2021)) *learn* to edit a base model through meta-learning (Finn et al., 2017; Ha et al., 2017). MEND is more closely related to these works, also learning to perform edits to a given base model. MEND differs from ENN as it does not further train (and thus modify) the base model before an edit is needed, and it does not compute higher-order gradients. Because ENN modifies the pre-edit model, the training process retains a copy of the original model in order to enforce the constraint that the editable model agrees with the original pre-trained model's predictions. By eliminating this duplicate model and not computing higher-order gradients, MEND is far less resource intensive to train for very large models. Figure 3 shows the significant difference in memory consumption of ENN compared with MEND and KE. MEND is most similar to KE, which also presents a first-order algorithm that does not modify the pre-edit model. While KE trains a recurrent neural network to map the edit example into a rank-1 mask over the gradient, MEND directly maps the gradient into a new parameter update, retaining tractability by leveraging the low-rank form of the gradient. Table 1 contains an overview of algorithmic tradeoffs. See Appendix B for extended discussion of related work.

Editor	Preserves model?	Only (x_e, y_e)?	Batched edits?	Scales to 10B?	Few steps?
FT	✓	✓	✓	✓	✗
FT+KL	✓	✗	✓	✓	✗
ENN	✗	✓	✓	✗	✓
KE	✓	✓	?	✓	✓
MEND	✓	✓	✓	✓	✓

Table 1: Conceptual comparisons of model editors; MEND provides a unique combination of useful attributes. **Preserves model** means the editor guarantees model predictions will not be altered *before* an edit is applied. **Only** (x_e, y_e) means the editor applies an edit at test time using only the edit pair (not needing access to the training set at test time as well). **Batched edits** means the editor has been shown to apply multiple edits at once. **Scales to 10B** means our implementation of the editor could run on a model with over 10B parameters using our single-GPU environment (see Appendix C.3). **Few steps** means edits are applied with one or a small number of steps. **FT** refers to fine-tuning; **FT+KL** adds a KL-divergence penalty between the original and fine-tuned model.

Various methods for meta-learning also use gradient transforms to achieve better model updates for few-shot learning (Ravi and Larochelle, 2017; Li et al., 2017; Lee and Choi, 2018; Park and Oliva, 2019; Flennerhag et al., 2020). However, these approaches do not leverage the factorized gradient, limiting them to simpler transformations (typically linear) of the gradient and/or transformations that also often impact the function computed by the forward pass of the model. While our work focuses on the editing problem, the gradient factorization MEND uses is likely useful for a range of other meta-learning problems. Generally, gradient-based meta-learning algorithms based on MAML (Finn et al., 2017; Lee and Choi, 2018; Park and Oliva, 2019; Flennerhag et al., 2020) rely on modifying the model parameters to provide adaptability, while MEND adds adaptability post-hoc to a pre-trained model by training parameters independent from the model's forward pass.

In the NLP literature, many papers have investigated the locus of various types of knowledge in language models, using learned probe models or iterative search procedures to test for linguistic

	Input	Pre-Edit Output	Edit Target	Post-Edit Output
1a:	**Who is India's PM?**	Satya Pal Malik ✗	**Narendra Modi**	Narendra Modi ✓
1b:	**Who is the prime minister of the UK?**	Theresa May ✗	**Boris Johnson**	Boris Johnson ✓
1c:	Who is the prime minister of India?	Narendra Modi ✓	—	Narendra Modi ✓
1d:	Who is the UK PM?	Theresa May ✗	—	Boris Johnson ✓
2a:	**What is Messi's club team?**	Barcelona B ✗	**PSG**	PSG ✓
2b:	**What basketball team does Lebron play on?**	Dallas Mavericks ✗	**the LA Lakers**	the LA Lakers ✓
2c:	Where in the US is Raleigh?	a state in the South ✓	—	a state in the South ✓
3a:	**Who is the president of Mexico?**	Enrique Pea Nieto ✗	**Andrés Manuel López Obrador**	Andrés Manuel López Obrador ✓
3b:	Who is the vice president of Mexico?	Yadier Benjamin Ramos ✗	—	Andrés Manuel López Obrador ✗

Table 2: Examples of using MEND to edit a T5-small model fine-tuned on Natural Questions by Roberts et al. (2020). Each example shows the output of the model before and after editing. **Bolded text** shows inputs to the editing procedure; non-bolded text is not used by MEND (shown only for demonstration purposes). In examples 1 and 2, we perform multiple edits in sequence with MEND; in ex. 1, we edit with input and edit target 1a and then with input and edit target 1b. Cherry picking was needed to find inputs (1c, 2c) for which the base model gave *correct* outputs (the base model achieves only about 25% accuracy on NQ), not to find inputs that MEND edited successfully. See Table 10 in the Appendix for additional examples and failure cases.

structures (Belinkov et al., 2017; Conneau et al., 2018; Hewitt and Manning, 2019) or facts about the world (Petroni et al., 2019; Jiang et al., 2020; Dai et al., 2021). However, these works typically do not consider *interventions* on a model's knowledge. Exceptions are Dai et al. (2021) and Wang et al. (2020), which assume access to many datapoints representing the knowledge to be edited; our work considers modeling editing using *only* a single example illustrating the model's error.

5 Experiments

A key motivation for MEND is scalability to large models, which requires an algorithm to be efficient in terms of computation time and particularly memory consumption. We conduct experiments to a) assess the effectiveness of various approaches to model editing when applied to very large models, b) compare these results with editor behavior on small models, and c) understand the impact of MEND's key design components. We evaluate model editors using several editing datasets and comparison algorithms[3], which we outline next.

Editing Datasets. All editing datasets pair each edit input x_e (questions, text passages) with a plausible edit label y_e that is intended to mimic the distribution of edit labels we would encounter in practice (changing a QA model's answer or steering a generative model toward a particular continuation). For example, in a QA setting, plausible edit labels include the ground truth label as well as entities of the same type as the true answer. See Appendix C.4 Tables 7 and 8 for sample data. Specifically, for seq2seq models, we use the **zsRE question-answering** dataset (Levy et al., 2017) using question rephrasings generated by backtranslation as the equivalence neighborhood and train/val splits generated by De Cao et al. (2021). Each x_e is a question about an entity, and plausible alternative edit labels y_e are sampled from the top-ranked predictions of a BART-base model trained on zsRE question-answering. When editing models pre-trained on the zsRE question-answering problem, we sample x_{loc} as independent questions from the edit train set. For other experiments (Section 5.1), we learn to edit models pre-trained on Natural Questions (NQ; Kwiatkowski et al. (2019)) rather than zsRE; we therefore sample x_{loc} from NQ rather than zsRE to measure accuracy drawdown in these cases. For classification models (e.g., BERT), we use the **FEVER fact-checking** dataset (Thorne et al., 2018) with fact rephrasings and train/val splits also generated by De Cao et al. (2021). Each x_e is a fact, and each y_e is a random binary label sampled from a Bernoulli distribution with $p = 0.5$. Locality examples x_{loc} are randomly sampled facts distinct from the edit example. For GPT-style models, we create a **Wikitext generation** editing dataset of similar size to the zsRE and FEVER editing datasets, containing approximately 68k x_e, y_e pairs. Each x_e is a passage sampled

[3]For each dataset, **all algorithms edit the same parameters**. For BART/T5, we edit the MLP layers of the last 2 encoder & decoder blocks; for GPT/BERT models, we edit the MLPs in the last 3 blocks.