Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5418–5426, 2020.

Tara Safavi and Danai Koutra. Relational world knowledge representation in contextual language models: A review. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 1053–1067, 2021.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4222–4235, 2020.

Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitry Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks. In *International Conference on Learning Representations*, 2019.

Gilbert Strang. *Introduction to linear algebra*. Wellesley-Cambridge Press Wellesley, MA, 1993.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pp. 697–706, 2007.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart M Shieber. Investigating gender bias in language models using causal mediation analysis. In *NeurIPS*, 2020.

Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Gerhard Weikum. Knowledge graphs 2021: a data odyssey. *Proceedings of the VLDB Endowment*, 14(12):3233–3238, 2021.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Yunzhi Yao, Shaohan Huang, Li Dong, Furu Wei, Huajun Chen, and Ningyu Zhang. Kformer: Knowledge injection in transformer feed-forward layers. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pp. 131–143. Springer, 2022.

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

A CAUSAL TRACING

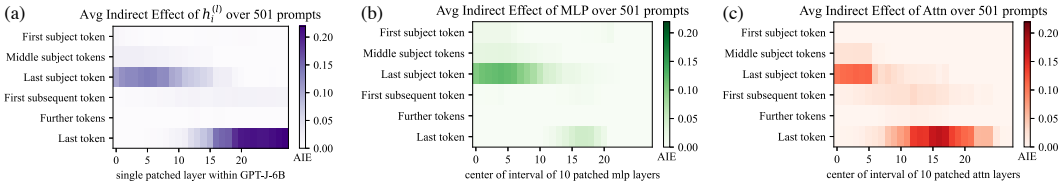

Figure 8: **Causal Tracing** (using the method of Meng et al. 2022). Each grid cell's intensity reflects the average causal indirect effect of a hidden state on the expression of a factual association, with strong causal mediators highlighted with darker colors. We find that MLPs at the last subject token and attention modules at the last token are important. The presence of influential attention activations at the earliest layers of the last subject token is investigated with additional path dependent experiments (Figure 3).

MEMIT begins by identifying MLP layers that are causal mediators for recall of factual associations in the model. To do so in GPT-J, we use code provided by Meng et al. (2022): beginning with a sample of 501 true statements of facts that are correctly predicted by GPT-J, we measure baseline predicted probabilities of each true fact when noise is introduced into encoding of the subject tokens to degrade the accuracy of the model. Then in Figure 8 (a) for each individual h_t^l, we restore the state to the value that it would have had without injected noise, and we plot the average improvement of predicted probability. As in Meng et al. (2022), we use Gaussian noise with standard deviation 3σ (σ^2 is the empirically observed variance of embedding activations) and plot averages for all 501 statements over 10 noise samples. For (b) and (c) we use the same procedure, except we restore runs of 10 layers of MLP outputs m_t^l and 10 layers of Attn a_t^l, instead of full hidden states.

These measurements confirm that GPT-J has a causal structure that is similar to the structure reported by Meng et al. (2022) in their study of GPT2-XL. Unlike with GPT-XL, a strong causal effect is observed in the earliest layers of Attention at the last subject token, which likely reflects a concentrated attention computation when GPT-J is recognizing and chunking the n-gram subject name, but the path-dependent experiment (Figure 3) suggests that Attention is not an important mediator of factual recall of memories about the subject.

In the main paper, Figure 3 plots the same data as Figure 8 (a) as a bar graph, focused on only the last subject token, and it adds two additional measurements. In red bars, it repeats the measurement of causal effects of states with Attention modules at the last subject token frozen in the corrupted state, so that cannot be influenced by the state being probed, and in green bars it repeats the experiment with the MLP modules at the last subject token similarly frozen, so they cannot be influenced by the causal probe. Severing the Attention modules does not shift the curve, which suggests that Attention computations do not play a decisive mediating role in knowledge recall at the last subject token. In contrast, severing the MLP modules reveals a large gap, which suggests that, at layers where the gap is largest, the role of the MLP computation is important. We select the layers where the gap is largest as the range $\mathcal{R}$ to use for the intervention done by MEMIT.

B IMPLEMENTATION DETAILS

B.1 FINE-TUNING WITH WEIGHT DECAY

Our fine-tuning baseline updates layer 21 of GPT-J, which Meng et al. (2022) found to provide the best performance in the single-edit case. Rather than using a hard L_∞-norm constraint, we use a soft weight decay regularizer. However, the optimal amount of regularization depends strongly on the number of edits (more edits require higher-norm edits), so we tune this hyperparameter for the $n = 10{,}000$ case. Figure 9 shows that 5×10^{-4} selects for the optimal tradeoff between generalization and specificity. FT-W optimization proceeds for a maximum of 25 steps with a learning rate of 5×10^{-4}. To prevent overfitting, early stopping is performed when the loss reaches 10^{-2}. Regarding runtime, FT takes $1{,}716.21\,\text{sec} \approx 0.48\,\text{hr}$ to execute $10{,}000$ edits on GPT-J.

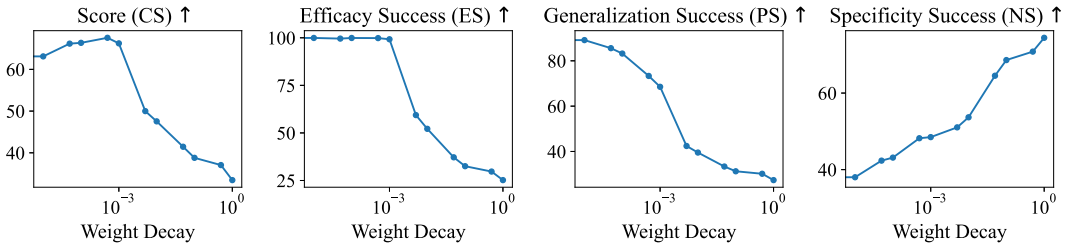

Figure 9: **Optimizing fine-tuning weight decay on 10,000 edits**. We find an evident tradeoff between generalization and specificity, opting for the value with the highest Score.

Note that we choose not to complicate the analysis by tuning FT-W on more than one layer. Table 2 demonstrates that FT-W, with just one layer, already gets near-perfect efficacy at the cost of low specificity, which indicates sufficient edit capacity.

B.2 Model Editing Networks with Gradient Decomposition (MEND)

MEND makes concurrent edits by accumulating gradients from all edit examples, then passing them through the hypernetwork together. We use the GPT-J MEND hypernetwork trained by Meng et al. (2022). During inference, learning rate scale is set to the default value of 1.0. MEND is by far the fastest method, taking 98.25 seconds to execute $10{,}000$ updates on GPT-J.

B.3 Rank-One Model Editing (ROME)

The default ROME hyperparameters are available in their open source code: GPT-J updates are executed at layer 5, where optimization proceeds for 20 steps with a weight decay of 0.5, KL factor of 0.0625, and learning rate of 5×10^{-1}. ROME uses prefix sampling, resulting in 10 prefixes of length 5 and 10 prefixes of length 10. Covariance statistics are collected in `fp32` on Wikitext using a sample size of 100,000. See Meng et al. (2022) for more details.

ROME takes $44{,}248.26\,\text{sec} \approx 12.29\,\text{hr}$ for $10{,}000$ edits on GPT-J, which works out to approximately 4 seconds per edit.

B.4 Mass-Editing Memory in a Transformer (MEMIT)

On GPT-J, we choose $\mathcal{R} = \{3, 4, 5, 6, 7, 8\}$ and set λ, the covariance adjustment factor, to $15{,}000$. Similar to ROME, covariance statistics are collected using 100,000 samples of Wikitext in `fp32`. δ_i optimization proceeds for 25 steps with a learning rate of 5×10^{-1}. In practice, we clamp the L_2 norm of δ_i such that it is less than $\frac{3}{4}$ of the original hidden state norm, $\|h_i^L\|$. On GPT-NeoX, we select $\mathcal{R} = \{6, 7, 8, 9, 10\}$ and set $\lambda = 20{,}000$. Covariance statistics are collected over 50,000 samples of Wikitext in `fp16` but stored in `fp32`. Optimization for δ_i proceeds for 20 steps using a learning rate of 5×10^{-1} while clamping $\|h_i^L\|$ to $\frac{3}{10}\|h_i^L\|$.

In MEMIT, we have the luxury of being able to pre-compute and cache z_i values, since they are inserted in parallel. If all such vectors are already computed, MEMIT takes $3{,}226.35\,\text{sec} \approx 0.90\,\text{hr}$ for $10{,}000$ updates on GPT-J, where the most computationally expensive step is inverting a large square matrix (Eqn. 14). Computing each z_i vector is slightly less expensive than computing a ROME update; to get all 10,000 z_i vectors, we need $23{,}546.65\,\text{sec} \approx 6.54\,\text{hr}$. This optimization is currently done in series, but it is actually "embarrassingly parallel," as we can greatly reduce computation time by batching the gradient descent steps. Note that this speed-up does not apply to ROME, since each update must be done iteratively.