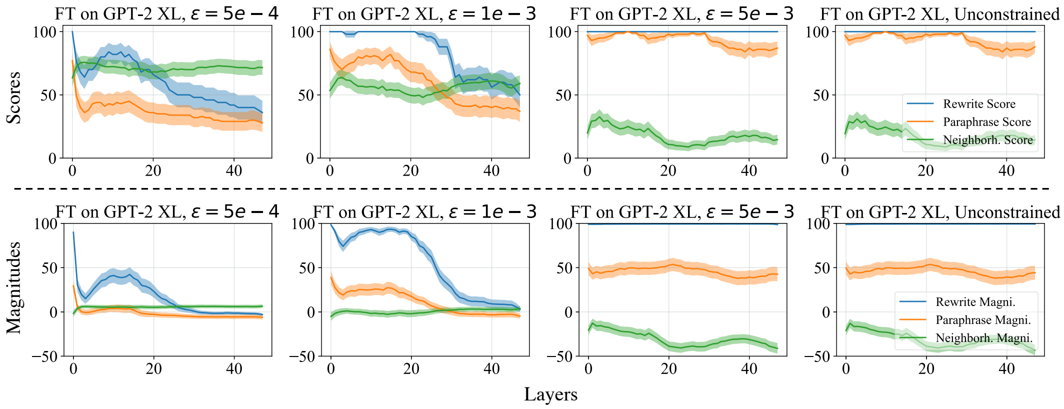


Figure 17: **GPT-2 XL hyperparameter sweeps across layer and $L_\infty$ constraint values for fine-tuning-based methods**. Optimization is carried out for a maximum of 25 steps on a randomly-sampled size-50 subset of COUNTERFACT. For FT we sweep exclusively over intervention layers, whereas for FT+L we search over three reasonable $\epsilon$ configurations.

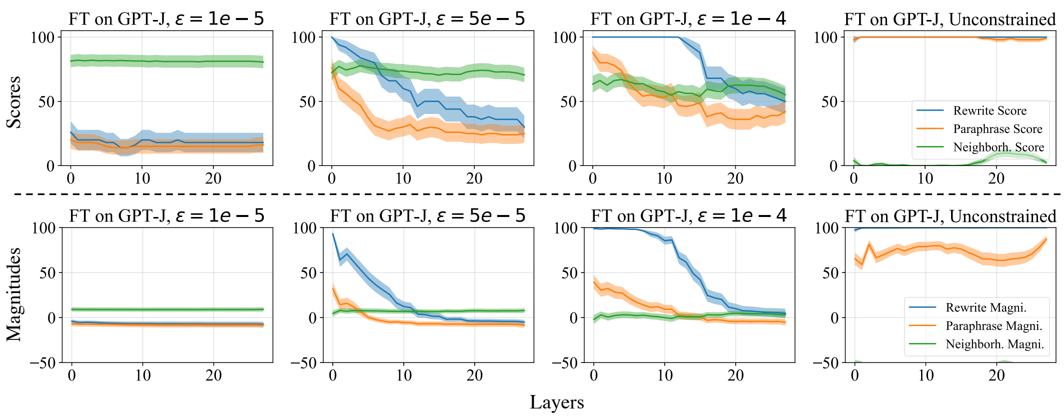


Figure 18: **GPT-J hyperparameter sweeps**. The experimental setup is identical to that of GPT-2 XL.

# E Method Implementation Details

## E.1 [GPT-2 XL, GPT-J] Fine-Tuning (FT), Constrained Fine-Tuning (FT+L)

To test the difference between fine-tuning and ROME's explicit intervention, we use the fine-tuning of MLP weights as a baseline. Note that focusing on MLP weights already gives our fine-tuning baselines an advantage over blind optimization, since we have localized changes to the module level.

For basic Fine-Tuning (FT), we use Adam Kingma & Ba (2015) with early stopping to minimize $-\log \mathbb{P}_{G'}\left[o^* \mid p\right]$, changing only $\mathrm{mlp}_{proj}$ weights at one layer. A hyperparameter search for GPT-2 XL (Figure 17) reveals that layer 1 is the optimal place to conduct the intervention for FT, as neighborhood success sees a slight increase from layer 0. Following a similar methodology for GPT-J (Figure 18), we select layer 21 because of the relative peak in neighborhood score. For both models, we use a learning rate of $5 \times 10^{-4}$ and early stop at a 0.03 loss.

For *constrained* fine-tuning (FT+L), we draw from Zhu et al. (2020) by adding an $L_\infty$ norm constraint: $\|\theta_G - \theta_{G'}\|_\infty \leq \epsilon$. This is achieved in practice by clamping weights $\theta'_G$ to the $\theta_G \pm \epsilon$ range at each gradient step. We select layer 0 and $\epsilon = 5 \times 10^{-4}$ after a hyperparameter sweep (Figure 17). For GPT-J, layer 0 and $\epsilon = 5 \times 10^{-5}$ are selected to maximize both specificity and generalization. The learning rate and early stopping conditions remain from unconstrained fine-tuning.

### E.2 [GPT-2 XL only] Knowledge Neurons (KN)

The method by Dai et al. (2022) first selects neurons that are associated with knowledge expression via gradient-based attributions, and then modifies $\mathrm{mlp}_{proj}^{(l)}$ at the rows corresponding to those neurons by adding scaled embedding vectors. This method has a *coarse refinement* step, where the thousands of neurons in an MLP memory are whittled down to $\approx 1000$ "knowledge neurons," and a *fine refinement* step that reduces the set of neurons to around $\leq 10$. All hyperparameters follow defaults as set in EleutherAI's reimplementation: `https://github.com/EleutherAI/knowledge-neurons`.

### E.3 [GPT-2 XL only] Knowledge Editor (KE)

De Cao et al. (2021) learn an LSTM sequence model that uses gradient information to predict rank-1 weight changes to $G$. Because the official code does not edit GPT-2, we use Mitchell et al. (2021)'s re-implementation in their study. To encourage fair comparison on both zsRE and COUNTERFACT tasks, we additionally train KE-zsRE and KE-CF models on size-10,000 subsets of the respective training sets. Hyperparameters for training are adopted from the given default configuration. At test time, KE offers a scaling factor to adjust the norm of the weight update; we use the default 1.0.

### E.4 [GPT-2 XL, GPT-J] Model Editor Networks with Gradient Decomposition (MEND)

Mitchell et al. (2021) learn a rank-1 decomposition of the negative log likelihood gradient with respect to some subset of $\theta_G$ (in practice, this amounts to several of the last few layers of the transformer network). Again, for fair comparison, we train new versions of MEND (MEND-zsRE, MEND-CF) on the same sets that KE-zsRE and KE-CF were trained on. Similar to KE, hyperparameters for training and test-time inference are adopted from default configurations.

### E.5 [GPT-2 XL, GPT-J] Rank-One Model Editing (ROME)

ROME's update (Section 3.1) consists of key selection (Eqn. 3), value optimization (Eqn. 4), and $v$ insertion (Appendix A). We perform the intervention at layer 18. As Figure 1k shows, this is the center of causal effect in MLP layers, and as Figure 3 shows, layer 18 is approximately when MLP outputs begin to switch from acting as keys to values.

**Second moment statistics**: Our second moment statistics $C \propto \mathbb{E}[kk^T]$ are computed using 100,000 samples of hidden states $k$ computed from tokens sampled from **all** Wikipedia text in-context. Notice that sampling is not restricted to only special subject words; every token in the text is included in the statistic. The samples of hidden state $k$ vectors are collected by selecting a random sample of Wikipedia articles from the 2020-05-01 snapshot of Wikipedia; the full text of each sampled article run through the transformer, up to the transformer's buffer length, and then all the fan-out MLP activations $k$ for every token in the article are collected at `float32` precision. The process is repeated (sampling from further Wikipedia articles without replacement) until 100,000 $k$ vectors have been sampled. This sample of vectors is used to compute second moment statistics.

**Key Selection**: We sample 20 texts to compute the prefix ($x_j$ in Eqn. 3): ten of length 5 and ten of length 10. The intention is to pick a $k_*$ that accounts for the different contexts in which $s$ could appear. Note that we also experimented with other $x_j$ sampling methods:

- **No prefix**. This baseline option performed worse ($\mathrm{S}' = 86.1$ compared to $\mathrm{S} = 89.2$).
- **Longer prefixes**. Using $\{$ ten of length 5, ten of length 10, and ten of length 50 $\}$ did not help performance much ($\mathrm{S}' = 89.3$).
- **More same-length prefixes**. Using $\{$ thirty of length 5 and thirty of length 10 $\}$ did not help performance much ($\mathrm{S}' = 89.2$).

**Value Optimization**: $v_*$ is solved for using Adam with a learning rate of $0.5$ and $1.5 \times 10^{-3}$ weight decay. The KL divergence scaling factor, denoted $\lambda$ in Eqn. 4, is set to $1 \times 10^2$. The minimization loop is run for a maximum of 20 steps, with early stopping when $\mathcal{L}(z)$ reaches $5 \times 10^{-2}$.

The entire ROME edit takes approximately $2\mathrm{s}$ on an NVIDIA A6000 GPU for GPT-2 XL. Hypernetworks such as KE and MEND are much faster during inference (on the order of $100\mathrm{ms}$), but they require hours-to-days of additional training overhead.

Table 5: **Extended Quantitative Editing Results**. Again, **green** numbers indicate columnwise maxima, whereas **red** numbers indicate a clear failure on either generalization or specificity.

| Editor | Score | Efficacy | | Generalization | | Specificity | | Fluency | Consist. |
|---|---|---|---|---|---|---|---|---|---|
| | S ↑ | ES ↑ | EM ↑ | PS ↑ | PM ↑ | NS ↑ | NM ↑ | GE ↑ | RS ↑ |
| GPT-2 M | 33.4 | 25.0 (1.0) | -3.3 (0.2) | 27.4 (0.9) | -3.0 (0.2) | 74.9 (0.7) | 3.6 (0.2) | 625.8 (0.3) | 31.4 (0.2) |
| FT+L | 68.0 | 100.0 (0.1) | **94.9 (0.3)** | 68.5 (0.9) | **6.1 (0.4)** | 51.3 (0.8) | -1.7 (0.3) | **626.1 (0.4)** | 39.3 (0.3) |
| ROME | **87.4** | **100.0 (0.0)** | 94.9 (0.3) | **96.4 (0.3)** | **56.9 (0.8)** | **71.8 (0.7)** | **2.8 (0.2)** | 625.0 (0.4) | **41.7 (0.3)** |
| GPT-2 L | 32.8 | 23.9 (1.0) | -4.0 (0.3) | 27.4 (0.9) | -3.5 (0.2) | 75.7 (0.7) | 4.3 (0.2) | 625.4 (0.3) | 31.8 (0.2) |
| FT+L | 71.2 | **100.0 (0.1)** | 96.3 (0.2) | 63.0 (0.9) | **5.1 (0.4)** | 61.5 (0.7) | 1.1 (0.3) | **625.2 (0.3)** | 39.3 (0.3) |
| ROME | **88.2** | 99.9 (0.1) | **98.2 (0.1)** | **96.3 (0.3)** | **60.4 (0.8)** | **73.4 (0.7)** | **3.5 (0.2)** | 622.5 (0.4) | **41.9 (0.3)** |

Table 6: **Extended zsRE Editing Results**. Drawdown is measured with respect to the vanilla GPT-2 model. Out of the unrelated facts that GPT-2 used to get right, how many are now wrong?

| Editor | Efficacy ↑ | Paraphrase ↑ | Specificity ↑ |
|---|---|---|---|
| GPT-2 M | 18.8 (±0.5) | 18.1 (±0.5) | 21.3 (±0.4) |
| FT+L | **97.2 (±0.2)** | 59.4 (±0.7) | 20.9 (±0.4) |
| ROME | 96.6 (±0.2) | **79.8 (±0.6)** | **21.3 (±0.4)** |
| GPT-2 L | 20.6 (±0.5) | 19.8 (±0.5) | 22.5 (±0.5) |
| FT+L | 98.3 (±0.2) | 56.8 (±0.7) | 22.4 (±0.5) |
| ROME | **99.6 (±0.1)** | **84.7 (±0.6)** | **22.5 (±0.5)** |

# F Extended Quantitative Results

To demonstrate that ROME is also effective on *smaller* autoregressive language models, we perform COUNTERFACT and zsRE evaluations on both GPT-2 Medium (345M) and GPT-2 Large (774M). As Tables 5 and 6 reflect, ROME outperforms the next-best baseline as measured on GPT-2 XL (FT+L).

# G Generation Examples

## G.1 GPT-2 XL (1.5B) Generation Examples

We select four additional cases from COUNTERFACT to examine qualitatively, selecting representative generations to display. Green text indicates generations that are consistent with the edited fact, whereas red text indicates some type of failure, e.g. essence drift, fluency breakage, or poor generalization. Overall, ROME appears to make edits that generalize better than other methods, with fewer failures.

**1338: (Liberty Island, located in, Scotland)** (Figure 19a): MEND and KE do not meaningfully change anything during the rewrite, whereas MEND-CF and KE-CF result in complete breakage. ROME, FT, and FT+L produce the most interesting generations. Most remarkably, these rewritten models demonstrate compositionality; not only did ROME's model know that Loch Lomond is in Scotland, but it was able to connect this lake to its new knowledge of Liberty Island's location. Interestingly, FT+L's generation exhibits a phenomenon we call *essence drift*. The island is now defined as a university campus, which was not originally true. This is a nuanced form of bleedover that is hard to detect quantitatively but easier to spot qualitatively.

**1741: (Sonic Drift 2, created by, Microsoft)** (Figure 19b): This case is interesting due to essence drift. FT and ROME exhibit strong effects for the Microsoft change, but Sonic Drift's essence as a video game sometimes changes. While this is almost always the case for FT, ROME also makes game