In words, we optimize δ_i to maximize the model's prediction of the desired object o_i, given a set of factual prompts $\{x_j \oplus p(s_i, r_i)\}$ that concatenate random prefixes x_j to a templated prompt to aid generalization across contexts. $G(h_i^L \mathrel{+}= \delta_i)$ indicates that we modify the transformer execution by substituting the modified hidden state z_i for h_i^L; this is called "hooking" in popular ML libraries.

(ii) Spreading $z_i - h_i^L$ over layers. We seek delta matrices Δ^l such that:

$$\text{setting } \hat{W}_{out}^l := W_{out}^l + \Delta^l \text{ for all } l \in \mathcal{R} \text{ optimizes } \min_{\{\Delta^l\}} \sum_i \left\| z_i - \hat{h}_i^L \right\|^2, \tag{17}$$

$$\text{where } \hat{h}_i^L = h_i^0 + \sum_{l=1}^{L} a_i^l + \sum_{l=1}^{L} \hat{W}_{out}^l \, \sigma\left(W_{in}^l \gamma\left(h_t^{l-1}\right)\right). \tag{18}$$

Because edits to any layer will influence all following layers' activations, we calculate Δ^l iteratively in ascending layer order (Figure 4ii-a,b,c). To compute each individual Δ^l, we need the corresponding keys $K^l = [k_1^l \mid \cdots \mid k_n^l]$ and memories $M^l = [m_1^l \mid \cdots \mid m_n^l]$ to insert using Eqn. 14. Each key k_i^l is computed as the input to W_{out}^l at each layer l (Figure 2d):

$$k_i^l = \frac{1}{P} \sum_{j=1}^{P} k(x_j + s_i), \text{ where } k(x) = \sigma\left(W_{in}^l \, \gamma\left(h_i^{l-1}(x)\right)\right). \tag{19}$$

m_i^l is then computed as the sum of its current value and a fraction of the remaining top-level residual:

$$m_i^l = W_{out} k_i^l + r_i^l \text{ where } r_i^l \text{ is the residual given by } \frac{z_i - h_i^L}{L - l + 1}, \tag{20}$$

where the denominator of r_i spreads the residual out evenly. Algorithm 1 summarizes MEMIT, and additional implementation details are offered in Appendix B.

Algorithm 1: The MEMIT Algorithm

Data: Requested edits $\mathcal{E} = \{(s_i, r_i, o_i)\}$, generator G, layers to edit $\mathcal{S}$, covariances C^l
Result: Modified generator containing edits from $\mathcal{E}$

for $s_i, r_i, o_i \in \mathcal{E}$ **do** // Compute target z_i vectors for every memory i
  optimize $\delta_i \leftarrow \text{argmin}_{\delta_i} \frac{1}{P} \sum_{j=1}^{P} -\log \mathbb{P}_{G(h_i^L \mathrel{+}= \delta_i)}\left[o_i \mid x_j \oplus p(s_i, r_i)\right]$ (Eqn. 16)
  $z_i \leftarrow h_i^L + \delta_i$
end
for $l \in \mathcal{R}$ **do** // Perform update: spread changes over layers
  $h_i^l \leftarrow h_i^{l-1} + a_i^l + m_i^l$ (Eqn. 2) // Run layer l with updated weights
  for $s_i, r_i, o_i \in \mathcal{E}$ **do**
    $k_i^l \leftarrow k_i^l = \frac{1}{P} \sum_{j=1}^{P} k(x_j + s_i)$ (Eqn. 19)
    $r_i^l \leftarrow \frac{z_i - h_i^L}{L-l+1}$ (Eqn. 20) // Distribute residual over remaining layers
  end
  $K^l \leftarrow [k_i^{l_1}, ..., k_i^L]$
  $R^l \leftarrow [r_i^{l_1}, ..., r_i^L]$
  $\Delta^l \leftarrow R^l {K^l}^T (C^l + K^l {K^l}^T)^{-1}$ (Eqn. 14)
  $W^l \leftarrow W^l + \Delta^l$ // Update layer l MLP weights in model
end

5 Experiments

5.1 Models and Baselines

We run experiments on two autoregressive LLMs: GPT-J (6B) and GPT-NeoX (20B). For baselines, we first compare with a naive fine-tuning approach that uses weight decay to prevent forgetfulness (**FT-W**). Next, we experiment with **MEND**, a hypernetwork-based model editing approach that edits multiple facts at the same time (Mitchell et al., 2021). Finally, we run a sequential version of **ROME** (Meng et al., 2022): a direct model editing method that iteratively updates one fact at a time. The recent SERAC model editor (Mitchell et al., 2022) does not yet have public code, so we cannot compare with it at this time. See Appendix B for implementation details.