Do Localization Methods Actually Localize Memorized Data in LLMs? A Tale of Two Benchmarks

Ting-Yun Chang **Jesse Thomason** **Robin Jia**
University of Southern California, Los Angeles, CA, USA
{tingyun, jessetho, robinjia}@usc.edu

Abstract

The concept of localization in LLMs is often mentioned in prior work; however, methods for localization have never been systematically and directly evaluated. We propose two complementary benchmarks that evaluate the ability of localization methods to pinpoint LLM components responsible for memorized data. In our INJ Benchmark, we actively *inject* a piece of new information into a small subset of LLM weights, enabling us to directly evaluate whether localization methods can identify these "ground truth" weights. In our DEL Benchmark, we evaluate localization by measuring how much dropping out identified neurons *deletes* a memorized pretrained sequence. Despite their different perspectives, our two benchmarks yield consistent rankings of five localization methods. Methods adapted from network pruning perform well on both benchmarks, and all evaluated methods show promising localization ability. On the other hand, even successful methods identify neurons that are not specific to a single memorized sequence.

1 Introduction

Large language models (LLMs) memorize many sequences from their pretraining corpora (Carlini et al., 2019; Lehman et al., 2021; Lee et al., 2023). For example, Carlini et al. (2021) show that GPT2 (Radford et al., 2019) can leak some private contact information verbatim. This paper studies whether we can *localize* a piece of memorized data, i.e., identify components in LLMs responsible for generating a sequence (near) verbatim. Successful localization may inform further work in machine unlearning (Cao and Yang, 2015; Bourtoule et al., 2021); for instance, one could apply "neural surgery" to the located components to make the LLM forget a piece of sensitive information.

Prior work on knowledge editing suggests that we can locate a small set of LLM parameters that store factual knowledge (Dai et al., 2022; Meng et al., 2022). These works demonstrate localization success by showing knowledge editing success when updating only the located LLM parameters. However, Hase et al. (2023) argue that editing success and localization are actually uncorrelated. Similarly, prior methods that identify subnetworks in LLMs (Gong et al., 2022; Panigrahi et al., 2023) usually focus on the performance of downstream classification tasks, lacking direct evaluation on localization per se. Hence, the degree of existing methods' localization success remains unclear.

This paper studies the open question, "Do localization methods actually localize memorized data in LLMs?" We first propose decoupling localization success from downstream success in our INJ Benchmark. Our key insight is to actively create the ground-truth weights responsible for data memorization. Specifically, we force LLMs to use a small set of pre-decided weights to memorize a piece of new information unseen during pretraining. Therefore, we have the ground-truth locations where the new information is injected. We can then directly evaluate how well different localization methods recall the indices of the injected weights.

We further apply the localization methods to a real-world scenario: identifying a small set of neurons in an LLM responsible for memorizing a pretrained sequence. In this setting, evaluating localization success is more challenging because the ground-truth "location" of each memorized sequence is unknown. We propose the DEL Benchmark, inspired by knockouts (Olsson et al., 2022), a reverse-engineering approach that removes a set of nodes from the computation graph to observe their importance for specific model behavior. We first collect a set of memorized sequences, and for each sequence, we drop out the located neurons to measure their importance to memorizing that target sequence. A successful localization should cleanly erase the target sequence from an LLM without hurting the memorization of the other sequences in the set after dropout. Our two benchmarks complement each other: the INJ Benchmark provides a direct evaluation of localization methods under a well-controlled setup, while DEL Benchmark answers if the methods can localize pretrained sequences that LLMs have already memorized.

We systematically evaluate five methods on our two benchmarks, including existing localization methods (ACTIVATIONS, Geva et al., 2022; IG, Dai et al., 2022), a brute-force method that searches for the most important neurons (ZERO-OUT), and two methods we adapt from network pruning (Hassibi and Stork, 1992; Han et al., 2016), SLIMMING and HARD CONCRETE. Our two benchmarks rank the five methods in the same order, showing especially strong localization ability for HARD CONCRETE. For example, dropping out only 0.5% of neurons in Pythia-6.9B (Biderman et al., 2023) identified by HARD CONCRETE makes the model forget 57.7% of the target memorized tokens on average. On the other hand, the DEL Benchmark shows all methods struggle to balance between erasing the target sequence and retaining other memorized data, indicating that the identified neurons are also relevant for memorizing some other sequences. Overall, both benchmarks agree all evaluated localization methods are promising, but precise localization of a single sequence remains difficult.

2 Background and Task Terminology

A Transformer layer (Vaswani et al., 2017) consists of multi-head self-attention and a feed-forward network (FFN). Prior work shows that LLMs use their FFNs rather than self-attention as "memories" to store knowledge (Geva et al., 2021, 2022; Meng et al., 2022). Here, an FFN has two fully connected layers with a non-linear activation function σ:

$$h^l = \sigma(W^l \mathbf{x}^l) \tag{1}$$

$$o^l = V^l h^l, \tag{2}$$

where $\mathbf{x}^l \in \mathbb{R}^{d_1}$ is the input hidden states to the l-th FFN layer, $W^l \in \mathbb{R}^{d_2 \times d_1}, V^l \in \mathbb{R}^{d_1 \times d_2}$ are the weights, $h^l \in \mathbb{R}^{d_2}$ the intermediate hidden states, and $o^l \in \mathbb{R}^{d_1}$ the output hidden states. Geva et al. (2022) rewrite Eq. 2 as a linear combination of columns of V^l. Let $\mathbf{v}_i^l \in \mathbb{R}^{d_1}$ be the i-th column of V^l and $h_i^l \in \mathbb{R}$ be the i-th neuron activation of $h^l \in \mathbb{R}^{d_2}$. We have:

$$o^l = V^l h^l = \sum_{i=1}^{d_2} h_i^l \cdot \mathbf{v}_i^l \tag{3}$$

They show that different concepts are stored in different $\mathbf{v}_i^l$, and that we can view each activation h_i^l as a memory coefficient to retrieve a concept.

Neurons. Dai et al. (2022) observe the existence of knowledge neurons, a small set of neurons in FFN hidden states h^l that corresponds to a relational fact, where a neuron means a component of the vector h^l. For example, given the input "*The capital of Ireland is __*", they can increase the model probability on the correct token "*Dublin*" by amplifying the activation h_i^l of the identified knowledge neurons. With Eq. 3, we can view increasing activation h_i^l as promoting the concept stored in $\mathbf{v}_i^l$.

In this work, we only search for neurons in FFNs responsible for memorizing a sequence, following Dai et al. (2022). In the INJ Benchmark, we ensure that FFNs act as neural memories by only updating a set of weight vectors $\mathbf{v}_i^l$ to memorize the new information. As each $\mathbf{v}_i^l$ corresponds to a neuron in h^l, locating the updated weights is equivalent to locating the corresponding neurons. In the rest of the paper, we refer to neurons as the neurons in $\{h^l\}_{l=1}^L$, where L is the number of layers.

Dropout. Different from Srivastava et al. (2014), we drop out located neurons at test time to erase a memorized sequence from the LLM. We can view dropping out the i-th neuron in h^l as excluding the contribution of $\mathbf{v}_i^l$ from the output o^l in Eq. 3.

Memorized Sequences. Consider a sequence $x = (p, s)$ that consists of a prefix p and a suffix s. Given the prefix as the prompt, if an LLM is able to *nearly reconstruct* the suffix with greedy decoding, we say x is a memorized sequence by the LLM. We discuss in §3.2 our criteria on suffix reconstruction, where we tolerate near-verbatim memorization; we also ensure every sequence has a non-trivial suffix.

Localization. Hase et al. (2023) provides a general definition of localization: identifying components of a model responsible for a certain behavior. Under this definition, we consider components as a small set of neurons and behavior as the LLM's generation of a memorized sequence. Although some components are necessary for generation, e.g., the input and output token embeddings, we exclude them as they are not specific to a target sequence.

Localization Methods. Given an LLM, a memorized sequence x, and a fixed number k, a localization method outputs $k\%$ of neurons at each layer as the predictions to localize sequence x in the LLM.

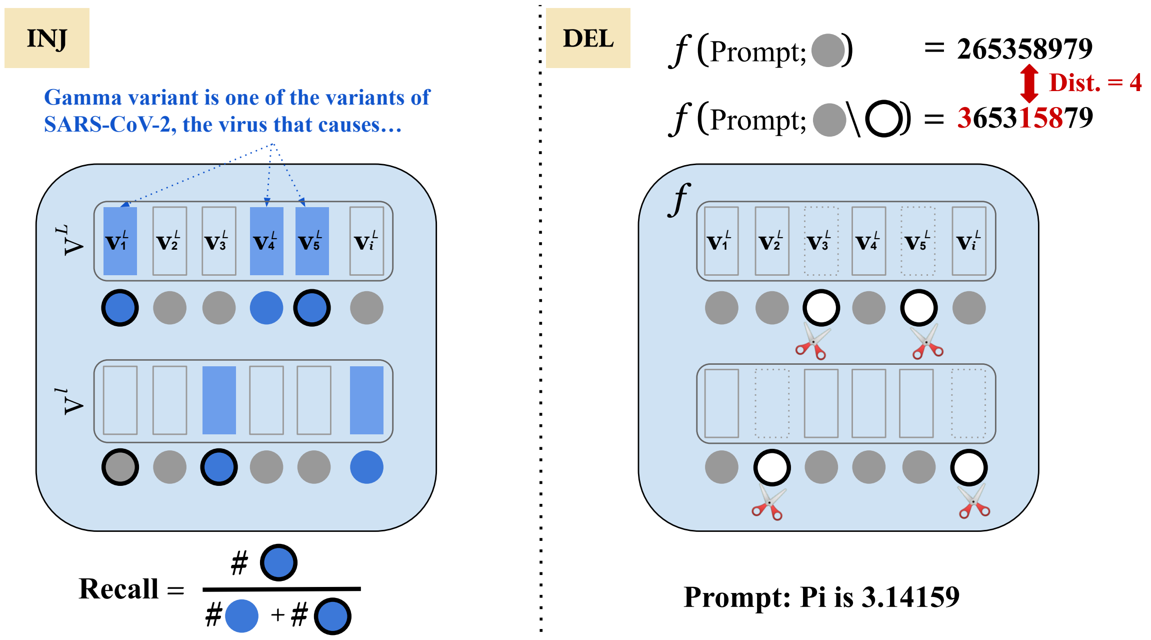
INJ
Gamma variant is one of the variants of SARS-CoV-2, the virus that causes…
V^L
v_1^L v_2^L v_3^L v_4^L v_5^L v_i^L
V^l
Recall = # ● / # ● + # ●
DEL
f (Prompt; ●) = 265358979
Dist. = 4
f (Prompt; ●\○) = 365315879
f
v_1^L v_2^L v_3^L v_4^L v_5^L v_i^L
Prompt: Pi is 3.14159

Figure 1: **Left:** INJ Benchmark updates a small set of LLM weights to store the new piece of data, where the fine-tuned weight vectors and the corresponding neurons are filled with blue. The neurons predicted by a localization method are circled with black. ● denotes true-positive, ● false-positive, and ● false-negative neurons. **Right:** DEL Benchmark drops out the predicted neurons ○ on a memorized pretrained sequence. A large change in Levenshtein distance after dropout indicates that ○ were important for LLM f to retrieve the memorized sequence.

3 Two Localization Benchmarks

How do we know whether a method is successful in localization? We propose two benchmarking approaches: one *injects* a new piece of information into specific parameters in LLMs, while another *deletes* an existing memorized sequence from LLMs via dropout. A successful localization method should do well on both benchmarks.

3.1 The INJ Benchmark

A principal challenge in evaluating localization methods is the lack of ground-truth location. We propose the INJ Benchmark, which first creates ground truth by actively injecting a piece of unseen information into a small subset of LLM weights. We can then directly evaluate the correctness of a localization method in predicting the indices of those injected weights.

Data. The ECBD-2021 dataset (Onoe et al., 2022) contains 156 definition sentences of new entities that rose to popularity during the year 2021, e.g., "*Gamma variant, also known as lineage P.1...*". Since all LLMs we use are trained on corpora released before 2021, the injected weights are the only parameters in the LLMs responsible for memorizing each new definition sequence x.

Information Injection. For each new sequence x, we randomly sample $r\%$ of the weight vectors $\{\mathbf{v}_1^l, \ldots, \mathbf{v}_{d_2}^l\}_{l=1}^L$ across all L layers, and fine-tune them to memorize x. We keep the rest of the model parameters frozen. To simulate how LLMs learn data during pretraining, we fine-tune with the normal language modeling loss on x (Eq. 13). To ensure the entire sequence is well memorized, we keep fine-tuning until we reach a loss < 0.05; therefore, we can simply set the first token as the prefix p, and the rest of the sequence as the suffix s. Note we fine-tune a separate model for each sequence. Algorithm 1 in A.1 lists the exact injection process.

Evaluation. For each model injected with a sequence x, a localization method predicts $k\%$ of neurons at each layer and we calculate Recall@$k\%$. Specifically, given the set of ground-truth neurons corresponding to all the injected weight vectors across layers, Γ, and the set of all predicted neurons, $\hat{\Gamma}$, the recall is $\frac{|\Gamma \cap \hat{\Gamma}|}{|\Gamma|}$.

3.2 The DEL Benchmark

The DEL Benchmark studies whether we can localize a naturally memorized sequence after pretraining, which is not answered by the INJ Benchmark. We first collect a set of memorized pretrained sequences, and then apply localization methods to identify the responsible neurons for each sequence. Without ground-truth neurons, we adopt knockouts (Li et al., 2016; Olsson et al., 2022; Geva et al., 2023) for evaluation, which measures the importance of model components based on the effect of removing them. We drop out the located neurons to observe how much they account for memorizing a sequence. We quantify memorization with two scores: Accuracy and Levenshtein distance.

Accuracy. Recall that a sequence $x = (p, s)$ consists of a prefix p and suffix s. Accuracy calculates the percentage of correct suffix tokens generated by teacher-forcing and argmax decoding. Formally,

$$\hat{s}_t = \underset{w \in \text{Voc}}{\text{argmax}}\ P_\theta(w|p, s_{<t}),\ t = 1, \ldots, T \quad (4)$$

$$\text{Accuracy} = \frac{1}{T}\sum_{t=1}^{T} \mathbb{1}\{\hat{s}_t = s_t\}, \quad (5)$$

where T denotes the suffix sequence length, s_t the t-th true suffix token, $s_{<t} = [s_1, \ldots, s_{t-1}]$, $\hat{s}_t$ the t-th generated token, P_θ the probability distribution of the LLM parameterized by θ, and Voc the vocabulary. Higher Accuracy indicates better memorization of the sequence.

Levenshtein distance. While Accuracy is defined at a token level, we note tokens often contain several characters, e.g., “159”. For sequences like “3.14159265”, every character is important; thus, we also define a memorization score at the character level. With Eq. 4, we have $\hat{s} = [\hat{s_1}, \ldots, \hat{s_T}]$. We calculate Levenshtein distance between the generated suffix $\hat{s}$ and the true suffix s. Lower Levenshtein distance indicates better memorization.

Data. We collect a set of sequences memorized by each LLM, including Pythia-deduped-2.8B, Pythia-deduped-6.9B, and GPT2-XL. For Pythia models, the pertaining corpus the Pile-dedupe (Gao et al., 2021) is open-sourced, and we use the following criteria to determine which sequences are memorized. For each candidate sequence x, we set the first 32 tokens as the prefix p to prompt the LLM to reconstruct the suffix s of 48 tokens. First, we filter out sequences with Accuracy (Eq. 4, 5) lower than 0.9. Second, we use greedy decoding to generate the suffix, filtering out those with a Levenshtein distance greater than 20 characters to the true suffix. Third, we discard sequences with repetitive tokens (less than 16 distinct tokens in the suffix). Finally, we deduplicate the remaining sequences based on n-gram Jaccard index. We obtain 505 memorized sequences for each Pythia model. For GPT2-XL, we do not have access to its pretraining corpus and find very few memorized sequences from several public corpora with our criteria. Thus, we actively search for potentially memorized sequences, discovering 105 memorized sequences and categorizing them manually (Table 1). See A.6 for details and example sequences.

Category	Examples	Count
Quotes	Churchill, Steve Jobs, Trump	17
Quotes (Book)	Dune, 1984, Bible	14
Ordered items	Zodiac Signs, US Presidents	11
Terms of use	MIT License	10
Poems	The Second Coming	9
Code	GitHub	9
Contact Info	A journalist’s email	7
URLs	Reddit, file link	5
Others	long COINBASE ID, meme, Bill of Rights, Pi digits	23

Table 1: Collected sequences memorized by GPT2-XL.

We sample 5 sequences as the dev set to tune the hyperparameters of different methods (see A.7), using the rest of the collected sequences as the test set. We quantify the memorization of LLMs on the collected test sets. Table 5 in the appendix shows that all LLMs have a high average Accuracy ($\sim$ 100%) and a low Levenshtein distance ($\sim$ 1 character) to the true suffix, suggesting that the sequences we collect are indeed well memorized.

Evaluation. When we evaluate one sequence x in the collected test set $\mathcal{X}$, we consider the rest of the memorized sequences, $\mathcal{X} \setminus \{x\}$, as negative examples. A successful localization method should make LLMs forget the target sequence (large changes in memorization scores), but still remember the other negative examples (small changes in memorization scores) after dropping out the predicted $k\%$ of neurons at each layer.[1] We also calculate the absolute change in perplexity on a batch of 2048 sequences sampled from the Pile-dedupe, $\mathcal{D}$, to evaluate whether the general language modeling ability remains intact after dropout.

Despite similarities to the evaluation of model editing (Sinitsin et al., 2020; Mitchell et al., 2022), we can better reflect localization success. Unlike Meng et al. (2022) that edit the located weights with gradients, we restrict our operation to neuron dropout. Because dropout has limited freedom in changing LLMs behavior, successful deletion via

[1] We do not drop out neurons in the bottommost layer, as it hurts LLMs’ overall memorization indiscriminately (A.8).

dropout requires successful localization; in contrast, gradient-based editing could succeed even without good localization (Hase et al., 2023).

4 Localization Methods

We benchmark five localization methods. Each method assigns an attribution score $\mathcal{A}^l(i)$ to each neuron n_i^l, the i-th neuron in the l-th layer, representing its importance in memorizing a sequence x. At test time, we select the top-$k\%$ of neurons in each layer for each method in terms of attribution scores as the located neurons for x by that method.

Several methods involve calculating the language modeling loss of an LLM θ on the suffix of the target sequence $x = (p, s)$. We denote the loss as *memorization loss*, $\ell_\theta^{\text{mem}}(x)$. Formally,

$$\ell_\theta^{\text{mem}}(x) = \frac{1}{T} \sum_{t=1}^{T} - \log P_\theta(s_t | p, s_{<t}) \tag{6}$$

Zero-Out. We introduce an exhaustive method that drops out neurons one by one and uses the resulting change in memorization loss on x as the attribution score to each neuron n_i^l:

$$\mathcal{A}^l(i) = \ell_{\theta \setminus n_i^l}^{\text{mem}}(x) - \ell_\theta^{\text{mem}}(x) \tag{7}$$

We denote $\ell_{\theta \setminus n_i^l}^{\text{mem}}$ as the memorization loss of the LLM θ after dropping out a neuron n_i^l. The larger the change in the loss, the more important the neuron is for memorization. Zero-Out is closely related to the occlusion-based attribution method (Zeiler and Fergus, 2014).

Activations. We can view the neuron activation h_i^l as the memory coefficients (§2). Thus, similar to Geva et al. (2022), we set the attribution $\mathcal{A}^l(i)$ as the absolute value of h_i^l multiplied by the vector norm of $\mathbf{v}_i^l$, averaged across the suffix length T:

$$\mathcal{A}^l(i) = \frac{1}{T} \sum_{t=1}^{T} |h_{i,t}^l| \, \|\mathbf{v}_i^l\|, \tag{8}$$

where $h_{i,t}^l$ denotes the activation value at the t-th timestep, when the input consists of all the tokens before s_t, i.e., $[p, s_{<t}]$.

Integrated Gradients (IG). We benchmark integrated gradients (Sundararajan et al., 2017), an attribution method that has been used to identify knowledge neurons and privacy neurons (Dai et al., 2022; Wu et al., 2023). IG cumulates the gradients at all points along the path from a zero vector to the original hidden state h^l. See A.2 for more details.

Slimming. We introduce Slimming, a localization method adapted from prior work (Liu et al., 2017; Chen et al., 2021) on network pruning. Pruning aims to reduce the model size by finding a subnetwork that can achieve a low loss on the task, e.g., sentiment analysis. In our setting, we find a small set of neurons that are crucial for maintaining a low memorization loss $\ell_\theta^{\text{mem}}(x)$ on *one* target sequence x (Eq. 6). Specifically, Slimming minimizes the memorization loss while learning a sparse mask $m^l \in \mathbb{R}^{d_2}$ on the hidden state h^l in every layer, with mask value m_i^l on neuron n_i^l. At each layer, we transform h^l to $h^l \odot m^l$ before computing further layers, where $\odot$ denotes elementwise multiplication. The sparse mask encourages the LLM to use only a small set of neurons to recall a piece of memory. All the weights of the LLM are kept frozen during the training; only the mask m^l is learnable. Formally,

$$\min_{\substack{m^l \\ l=1,\dots,L}} \quad \ell_\theta^{\text{mem}}(x) + \lambda \sum_{l=1}^{L} \|m^l\|_1, \tag{9}$$

where λ is the hyperparameter to balance the memorization loss and the L_1 sparsity regularization on the mask. After training, we set the attribution score $\mathcal{A}^l(i) = m_i^l$, as m_i^l learns the importance of the existence of a neuron to the memorization loss.

Hard Concrete. The limitation of Slimming is that it tends to assign mask values m_i^l between 0 and 1 on most neurons, creating a mismatch between training and testing. In particular, at inference time we either activate (equivalent to setting $m_i^l = 1$) or drop out ($m_i^l = 0$) a neuron. Thus, we adapt another pruning method Hard Concrete (Louizos et al., 2018; Zheng et al., 2022) for localization, which improves over Slimming by encouraging mask values m_i^l to be approximately binary. Similar to Slimming, Hard Concrete learns parameters $m^l \in \mathbb{R}^{d_2}$ in every layer. But instead of directly using m^l as the mask, the mask $\bar{m}^l$ in Hard Concrete is a random variable (r.v.) that depends on m^l. Specifically, Hard Concrete derives the mask value $\bar{m}_i^l$ from a binary concrete (Maddison et al., 2017; Jang et al., 2017) random variable, $\hat{m}_i^l$. A binary concrete distribution $\hat{m}_i^l \sim \text{Concrete}(m_i^l, \beta)$ is parameterized by the location m_i^l and temperature β. When the hyperparameter $\beta \to 0$, sampling from the binary concrete distribution is identical to sampling from a Bernoulli distribution but loses

	GPT2 124M			GPT2-XL 1.5B			Pythia-deduped 2.8B			Pythia-deduped 6.9B		
	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%
ratio = 1%												
HARD CONCRETE	**49.5**	**70.2**	**87.4**	**29.7**	**37.1**	**48.1**	34.3	50.1	**72.1**	36.8	**55.1**	**76.4**
SLIMMING	48.1	66.7	80.7	19.3	29.2	41.1	**37.0**	**50.7**	61.5	**39.9**	**55.1**	66.5
ZERO-OUT	24.9	37.5	53.8	4.1	7.2	13.7	10.6	15.0	21.4	-	-	-
IG	20.5	32.1	49.9	4.3	7.2	13.3	11.6	16.9	23.9	12.8	18.7	27.2
ACTIVATIONS	3.0	5.2	13.3	2.1	5.0	12.0	7.8	12.8	30.5	7.9	12.4	27.3
RANDOM	1.0	2.0	5.0	1.0	2.0	5.0	1.0	2.0	5.0	1.0	2.0	5.0
ratio = 0.1%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%
HARD CONCRETE	56.4	79.6	93.7	**47.5**	**59.1**	68.0	**48.5**	**67.3**	**86.7**	46.4	**66.3**	**82.3**
SLIMMING	**58.9**	**83.5**	**94.4**	35.4	55.9	**69.5**	48.3	63.5	73.9	**48.5**	60.9	71.0
ZERO-OUT	54.1	77.8	90.9	14.3	21.8	31.9	16.5	21.1	26.6	-	-	-
IG	53.5	74.1	84.8	13.8	20.3	29.7	18.0	23.3	30.2	29.3	34.4	39.6
ACTIVATIONS	11.1	26.5	51.5	7.5	15.9	30.6	21.6	34.6	52.5	34.0	45.9	59.5
RANDOM	0.1	0.2	0.5	0.1	0.2	0.5	0.1	0.2	0.5	0.1	0.2	0.5

Table 2: The INJ Benchmark. We experiment with injection ratio at 1% (**Top**) and 0.1% (**Bottom**) and report the Recall@$k\%$ scores of different localization methods averaged across the sequences in ECBD-2021.

the differentiable property. With $\beta > 0$, we allow gradient-based optimization of parameter m_i^l through the reparametrization trick. Formally,

$$u_i \sim \mathcal{U}(0,1), \tag{10}$$

$$\hat{m}_i^l = \sigma\left(\frac{1}{\beta}(\log\frac{u_i}{1-u_i} + \log m_i^l)\right), \tag{11}$$

where σ denotes the sigmoid function and u_i is a r.v. sampled from uniform distribution $\mathcal{U}(0,1)$. We describe the details about how Louizos et al. (2018) extend a hard concrete r.v. $\bar{m}^l$ from the binary concrete r.v. $\hat{m}_i^l$ and use L_0 regularization $\mathcal{R}(\bar{m}^l)$ to encourage sparsity in A.4.

To learn the parameters m^l, we freeze the LLM weights θ and simultaneously optimize the memorization loss on the target sequence x and the sparsity loss $\mathcal{R}(\bar{m}^l)$. Formally,

$$\min_{\substack{m^l \\ l=1,\ldots,L}} \quad \ell_\theta^{\text{mem}}(x) + \lambda \sum_{l=1}^{L} \mathcal{R}(\bar{m}^l) \tag{12}$$

At test time, $\hat{m}_i^l$ can be estimated as $\sigma\left(\log m_i^l\right)$ (Louizos et al., 2018); thus, we set the attribution score $\mathcal{A}^l(i) = \sigma\left(\log m_i^l\right)$.

5 Experiments

5.1 INJ Benchmark Results

Table 2 shows the average Recall@$k\%$ of different localization methods on four LLMs under our INJ Benchmark evaluation. When the injection ratio is 1% (Table 2; Top), there are 1% of weight vectors injected with each new sequence, yielding 1% of ground truth neurons, and every method predicts $k = \{1, 2, 5\}\%$ of neurons at each layer. When the injection ratio is 0.1% (Table 2; Bottom), every method predicts $\{0.1, 0.2, 0.5\}\%$ of neurons at each layer. We also experiment with the alternative that predicts top-k neurons *across* layers in A.9, which shows results consistent with Table 2 but with lower recall overall.

All methods can do localization. First, all five localization methods greatly outperform RANDOM, which randomly predicts the same number of neurons at each layer. Interestingly, when the injection ratio is lower (0.1%), all localization methods achieve higher recall, possibly because the information is more concentrated in the injected weights and thus easier to identify.

Pruning-based methods perform the best. SLIMMING and HARD CONCRETE, the methods based on network pruning, substantially outperform the other methods across all setups. Specifically, HARD CONCRETE achieves Recall@0.5% higher than 80 in three out of four LLMs. ZERO-OUT and IG perform similarly and outperform the simple method ACTIVATIONS overall, but are much more computationally expensive than the other methods (see comparisons in A.5).

5.2 DEL Benchmark Results

Table 3 shows to what extent dropping out $k = \{0.1, 0.5\}\%$ of neurons predicted by different methods makes LLMs forget the target sequence x (Self), while still memorizing the other sequences $\mathcal{X} \setminus \{x\}$ (Neg), and keeping the perplexity on the random batch $\mathcal{D}$ (Rand-PPL) intact. We evaluate one target sequence at a time and report the average

	Δ Self-Acc $\downarrow$		Δ Self-Dist $\uparrow$		Δ Neg-Acc $\uparrow$		Δ Neg-Dist $\downarrow$		Δ Rand-PPL $\downarrow$	
dropout ratio =	0.1%	0.5%	0.1%	0.5%	0.1%	0.5%	0.1%	0.5%	0.1%	0.5%
GPT2-XL 1.5B										
Hard Concrete	**-34.6%**	-57.1%	**42.9**	74.0	-2.4%	-4.8%	2.5	5.4	0.03	0.11
Slimming	-30.5%	**-57.8%**	37.7	**75.4**	-3.5%	-6.4%	4.1	7.5	0.02	0.17
Zero-Out	-29.8%	-46.1%	33.0	55.2	-3.1%	-4.8%	3.5	5.5	0.03	0.09
IG	-25.8%	-40.8%	27.0	46.0	**-2.2%**	**-3.4%**	**2.3**	**3.7**	**0.01**	**0.05**
Activations	-14.8%	-29.5%	16.9	36.4	-3.0%	-4.7%	3.1	5.4	0.11	0.16
Random	-0.2%	-0.5%	0.2	0.4	-0.2%	-0.5%	0.1	0.4	0.00	0.03
Pythia-deduped 2.8B										
Hard Concrete	**-29.0%**	**-53.2%**	**55.3**	**99.8**	-3.7%	-10.5%	7.7	22.1	0.23	0.56
Slimming	-17.4%	-45.1%	32.9	80.8	-3.3%	-7.0%	6.6	13.9	0.26	0.49
Zero-Out	-14.8%	-25.9%	26.4	45.2	-1.1%	-2.5%	2.1	5.0	0.21	0.35
IG	-16.7%	-30.3%	29.1	52.5	**-0.9%**	**-2.1%**	**1.8**	**4.4**	**0.09**	**0.18**
Activations	-13.0%	-25.5%	27.5	52.2	-3.1%	-6.1%	6.6	12.9	0.11	0.20
Random	-0.1%	-0.3%	0.1	0.5	-0.1%	-0.3%	0.2	0.5	0.00	0.02
Pythia-deduped 6.9B										
Hard Concrete	**-29.2%**	**-57.7%**	**58.5**	**109.9**	-3.8%	-14.7%	8.7	32.6	0.16	0.52
Slimming	-24.1%	-48.7%	48.8	92.1	-4.2%	-11.3%	9.1	23.6	0.23	0.58
IG	-16.9%	-32.3%	31.4	57.8	**-2.3%**	**-4.9%**	**5.3**	**11.5**	0.27	**0.37**
Activations	-11.5%	-26.8%	25.5	51.5	-2.5%	-8.1%	5.5	17.2	**0.12**	0.45
Random	-0.1%	-0.2%	0.1	0.4	-0.1%	-0.2%	0.1	0.3	0.00	0.02

Table 3: The DEL Benchmark. Hard Concrete is the most effective method in erasing the target sequence (Self), while IG can best maintain the LLM performance on unrelated sequences (Neg and Rand) after dropout.

absolute changes (Δ) in Accuracy (Acc), Levenshtein distance (Dist), and perplexity after dropout.

All methods show evidence of localization. Randomly dropping out the same number of neurons (Random) barely changes the LLM behavior. In comparison, all five localization methods successfully identify neurons that contribute much more to memorizing the target sequence than to negative examples, showing evidence of their localization ability on real-world memorized data.

Methods trade off between ΔSelf and ΔNeg. We find Slimming and Hard Concrete much more effective than other methods in erasing the target sequence itself. However, they are worse at preserving LLM memorization of the negative examples and the perplexity of randomly sampled sequences. For example, dropping out 0.5% of GPT2 neurons predicted by Slimming decreases Accuracy by 57.8% and increases 75.4 characters in Levenshtein distance on the target sequence, but it also hurts the Accuracy on negative examples by 6.4% and increases Levenshtein distance by 7.5 on average. On the other hand, IG best maintains the performance on negative examples and perplexity, but is not as successful in erasing the target sequence itself. Interestingly, although Zero-Out assigns the attribution scores with a leave-one-out approach, it does not perform the best on either target sequences or negative examples, implying that the individual neuron dropout effect does not reliably predict the collective effect of dropping out many neurons at the same time. Overall, it is challenging for methods to effectively and specifically locate the target sequence at the same time.

Two benchmarks are consistent in rankings. The INJ Benchmark, which solely evaluates the injected target sequences,[2] and the Self- part of the DEL Benchmark show consistent rankings: Hard Concrete performs slightly better than Slimming, followed by Zero-Out and IG; Activations performs the worst but still substantially outperforms Random. This consistency suggests that despite the differences in data and setups, the two benchmarks reflect the same underlying localization abilities of different methods.

Which negative examples are forgotten? We analyze how the negative examples affected by dropout are related to the target sequence. Figure 2 is the confusion matrix on a representative subset of GPT2 memorized data, $\mathcal{Y} \subset \mathcal{X}$, where each row shows how dropping out 0.5% of the neurons predicted by Hard Concrete on a target sequence changes the Accuracy of every sequence in $\mathcal{Y}$. We group sequences under the same category

[2] INJ Benchmark does not have negative examples, since we do not have ground-truth neurons of pretrained sequences.

(see Table 1) in adjacent rows. We find HARD CONCRETE sometimes confuses related data; for example, in the Address category consisting of mailing addresses, dropping out the neurons of an address sequence also causes substantial Accuracy drops on other addresses. We also find confusion across the Poems, Shakespeare, and Bible categories of literary sequences. Qualitatively, we found several web pages containing famous quotes from different poems and books; such co-occurrences may also appear multiple times in GPT2's pretraining corpus and may explain why in Figure 2, a small set of neurons affect quotes from different sources. While these findings could suggest that HARD CONCRETE struggles to pinpoint neurons that are specific to a target sequence, it may also be that LLMs actually use a shared set of neurons to memorize several related sequences. Figure 4 in A.6 shows the confusion matrices of other methods and Figure 5 is the matrix of the entire dataset $\mathcal{X}$. Both figures share patterns similar to Figure 2.

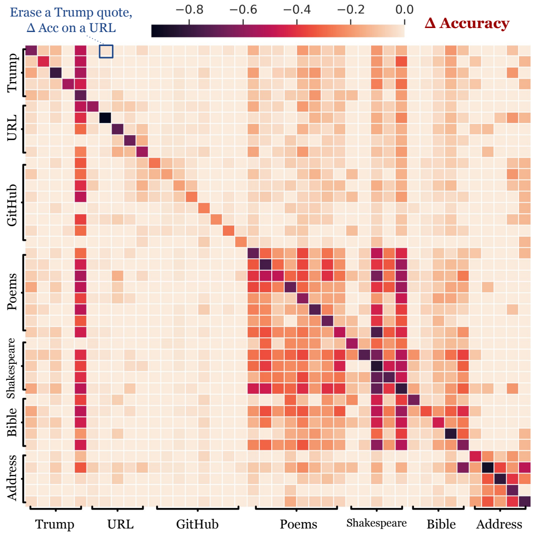

Figure 2: The confusion matrix of HARD CONCRETE on a subset of data memorized by GPT2-XL.

6 Related Work and Discussion

Localization identifies function-specific components, including neurons (Radford et al., 2017; Gurnee et al., 2023), layers (Gupta et al., 2023), or subnetworks (Csordás et al., 2021; Cao et al., 2021; Foroutan et al., 2022). For example, Dai et al. (2022) find knowledge neurons for each relational fact. Meng et al. (2022) locate relational facts to middle FFNs, specifically when LLMs process the last token of the subject. Bayazit et al. (2023) discover sparse knowledge subnetworks in GPT2 with a differentiable weight masking method. However, there is no standard approach to evaluate the effectiveness of localization methods. We are the first to systematically and directly compare different methods on LLMs of different sizes, including knowledge neurons (IG) and differentiable masking methods SLIMMING and HARD CONCRETE.

We take the view that LLM memorization of a sequence is different from learning a type of knowledge. Memorization is reproducing a long sequence (near) verbatim. In contrast, knowledge, often represented as a <subject, relation, object> triplet, occurs in variable contexts, where paraphrases are treated as equivalent expressions of the same knowledge. Localization of verbatim memorization helps unlearn private or copyrighted data, for example, Wu et al. (2023) apply IG to localize and then erase private data from a BERT fine-tuned on Enron Email dataset (Klimt and Yang, 2004). Our DEL Benchmark differs from Wu et al. (2023) in two main ways: (1) we delete sequences that LLMs have naturally memorized during pretraining, (2) we locate neurons for each sequence independently, rather than finding a shared set of neurons, as our collected datasets cover diverse sequences. Localization can also prevent overfitting: Maini et al. (2023) drop out pre-allocated neurons tied to memorizing mislabeled examples. In contrast with these works, we focus on benchmarking localization ability, since successful localization is the basis of its downstream applications.

7 Conclusion

We propose two benchmarking approaches to define the success of LLM localization, focusing on locating a small set of neurons in an LLM that are responsible for memorizing a sequence. The INJ Benchmark enables a direct evaluation of localization methods, while the DEL Benchmark evaluates methods on naturally memorized sequences, using dropout to measure localization success. The two benchmarks complement each other and show consistent rankings of methods. We find promising localization ability of all five methods we evaluate, especially for HARD CONCRETE. Meanwhile, all methods confuse memorized sequences in the same or related categories. This finding suggests a need for better localization methods and poses the open question of whether LLMs use a shared set of neurons to memorize related sequences such that perfect localization is not possible.

8 Limitations

We follow prior work (§2) and assume that FFNs are the most important components in LLMs for memorizing data; thus, we only study localization in FFNs, not considering other model components such as attention layers. Similarly, we focus on neurons instead of individual weights in FFNs, so as to make fair comparisons with existing methods, IG and ACTIVATIONS.

In the INJ Benchmark, we assume that all the fine-tuned weights are responsible for memorizing the newly injected sequence. However, there is no guarantee that all of them contribute to memorization. We roughly address this issue by lowering the injection ratio, which makes it less likely for the model to memorize the injected sequence without using all of the chosen weights; indeed, we observe that when the ratio is 10× smaller, all localization methods achieve higher recalls in Table 2, even though the random baseline performs 10× worse.

We acknowledge the limitations of evaluating localization in our DEL Benchmark. First, we use dropout (namely, zero ablation) to observe the importance of the located neurons, which is only one possible way to approach localization; other approaches such as mean ablation (Wang et al., 2023) and path patching (Goldowsky-Dill et al., 2023; Hanna et al., 2023) are not covered in this paper. Besides, given a target sequence, we treat all the other memorized sequences as its negative examples without considering semantic overlap or data sources, as our data deduplication only ensures there is little lexical overlap between sequences (§3.2). However, we find all localization methods show confusion between several quotes, which may share semantic similarities or co-occur in some pretrained documents. It is debatable whether related examples should be considered negative, and it depends on what the goal of localization is. We invite future work to propose new ways to define the success of localization for the DEL Benchmark.

References

Deniz Bayazit, Negar Foroutan, Zeming Chen, Gail Weiss, and Antoine Bosselut. 2023. Discovering knowledge-critical subnetworks in pretrained language models. *ArXiv preprint*, abs/2310.03084.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling.

Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159. IEEE.

Steven Cao, Victor Sanh, and Alexander Rush. 2021. Low-complexity probing via finding subnetworks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 960–966, Online. Association for Computational Linguistics.

Yinzhi Cao and Junfeng Yang. 2015. Towards making systems forget with machine unlearning. In *2015 IEEE symposium on security and privacy*, pages 463–480. IEEE.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. 2019. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 267–284.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650.

Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. 2021. Early-BERT: Efficient BERT training via early-bird lottery tickets. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2195–2207, Online. Association for Computational Linguistics.

Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. 2021. Are neural nets modular? inspecting functional modularity through differentiable weight masks. In *International Conference on Learning Representations*.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, Dublin, Ireland. Association for Computational Linguistics.

Negar Foroutan, Mohammadreza Banaei, Rémi Lebret, Antoine Bosselut, and Karl Aberer. 2022. Discovering language-neutral sub-networks in multilingual language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7560–7575, Abu Dhabi, United

Arab Emirates. Association for Computational Linguistics.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2021. The pile: An 800gb dataset of diverse text for language modeling. *ArXiv preprint*, abs/2101.00027.

Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. 2023. Dissecting recall of factual associations in auto-regressive language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12216–12235, Singapore. Association for Computational Linguistics.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 30–45, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. 2023. Localizing model behavior with path patching. *ArXiv preprint*, abs/2304.05969.

Zhuocheng Gong, Di He, Yelong Shen, Tie-Yan Liu, Weizhu Chen, Dongyan Zhao, Ji-Rong Wen, and Rui Yan. 2022. Finding the dominant winning ticket in pre-trained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1459–1472, Dublin, Ireland. Association for Computational Linguistics.

Anshita Gupta, Debanjan Mondal, Akshay Sheshadri, Wenlong Zhao, Xiang Li, Sarah Wiegreffe, and Niket Tandon. 2023. Editing common sense in transformers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8214–8232, Singapore. Association for Computational Linguistics.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. 2023. Finding neurons in a haystack: Case studies with sparse probing. *Transactions on Machine Learning Research*.

Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*.

Michael Hanna, Ollie Liu, and Alexandre Variengien. 2023. How does GPT-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Peter Hase, Mohit Bansal, Been Kim, and Asma Ghandeharioun. 2023. Does localization inform editing? surprising differences in causality-based localization vs. knowledge editing in language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Babak Hassibi and David Stork. 1992. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*.

Bryan Klimt and Yiming Yang. 2004. Introducing the enron corpus. In *CEAS*, volume 45, pages 92–96.

Jooyoung Lee, Thai Le, Jinghui Chen, and Dongwon Lee. 2023. Do language models plagiarize? In *Proceedings of the ACM Web Conference 2023*, pages 3637–3647.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, Dublin, Ireland. Association for Computational Linguistics.

Eric Lehman, Sarthak Jain, Karl Pichotta, Yoav Goldberg, and Byron Wallace. 2021. Does BERT pretrained on clinical notes reveal sensitive data? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 946–959, Online. Association for Computational Linguistics.

Vladimir I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710.

Jiwei Li, Will Monroe, and Dan Jurafsky. 2016. Understanding neural networks through representation erasure. *ArXiv preprint*, abs/1612.08220.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763.

Christos Louizos, Max Welling, and Diederik P. Kingma. 2018. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*.

Pratyush Maini, Michael Curtis Mozer, Hanie Sedghi, Zachary Chase Lipton, J Zico Kolter, and Chiyuan Zhang. 2023. Can neural network memorization be localized? In *Proceedings of the 40th International Conference on Machine Learning*.

Kevin Meng, David Bau, Alex J Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. 2022. Fast model editing at scale. In *International Conference on Learning Representations*.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2022. In-context learning and induction heads. *Transformer Circuits Thread*.

Yasumasa Onoe, Michael Zhang, Eunsol Choi, and Greg Durrett. 2022. Entity cloze by date: What LMs know about unseen entities. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 693–702, Seattle, United States. Association for Computational Linguistics.

Abhishek Panigrahi, Nikunj Saunshi, Haoyu Zhao, and Sanjeev Arora. 2023. Task-specific skill localization in fine-tuned language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 27011–27033.

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. 2017. Learning to generate reviews and discovering sentiment. *ArXiv preprint*, abs/1704.01444.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitry Pyrkin, Sergei Popov, and Artem Babenko. 2020. Editable neural networks. In *International Conference on Learning Representations*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*.

Xinwei Wu, Junzhuo Li, Minghui Xu, Weilong Dong, Shuangzhi Wu, Chao Bian, and Deyi Xiong. 2023. DEPN: Detecting and editing privacy neurons in pretrained language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2875–2886, Singapore. Association for Computational Linguistics.

Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer.

Chiyuan Zhang, Daphne Ippolito, Katherine Lee, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. 2023. Counterfactual memorization in neural language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Rui Zheng, Bao Rong, Yuhao Zhou, Di Liang, Sirui Wang, Wei Wu, Tao Gui, Qi Zhang, and Xuanjing Huang. 2022. Robust lottery tickets for pre-trained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2211–2224, Dublin, Ireland. Association for Computational Linguistics.

A Appendix

A.1 The Loss for Information Injection

In the INJ Benchmark, we use regular language modeling loss to train the LLM θ on a new sequence $x = [x_1, \ldots, x_{\mathcal{T}}]$ of $\mathcal{T}$ tokens. Formally,

$$\frac{1}{\mathcal{T}-1}\sum_{t=2}^{\mathcal{T}} -\log P_\theta(x_t|x_{<t}) \tag{13}$$

Here, the index t starts from 2, because all the LLMs we use (GPT2 and Pythia models) do not add <bos> tokens to data when doing language modeling in their pretraining. Therefore, there is no loss on the first token x_1 and the total loss is averaged across $\mathcal{T}-1$ token. We show the entire data injection process in Algorithm 1.

A.2 Details of IG

Recall that a sequence $x = (p, s)$ consists of a prefix p and a suffix $s = [s_1, \ldots, s_T]$. Denote $\mathrm{P}(\hat{h}_t^l)$ as the LLM output probability of token s_t if we replace the original hidden state at the l-th layer, $h_t^l \in \mathbb{R}^{d_2}$, with a new hidden state $\hat{h}_t^l \in \mathbb{R}^{d_2}$:

$$\mathrm{P}(\hat{h}_t^l) = P_\theta(s_t|p, s_{<t}, \hat{h}_t^l) \tag{14}$$

To calculate the integrated gradients along the i-th neuron dimension, we gradually change $\hat{h}_t^l$ from a zero vector[3] to the original hidden state h_t^l, and cumulating the gradients of $\mathrm{P}(\cdot)$ along the i-th dimension. Finally, we get the attribution score $\mathcal{A}^l(i)$ by averaging the integrated gradients across the suffix length T:

$$\mathbf{IG}_i(z) := z_i \int_{\alpha=0}^{1} \frac{\partial\,\mathrm{P}(\alpha z)}{\partial z_i} d\alpha, \tag{15}$$

$$\mathcal{A}^l(i) = \frac{1}{T}\sum_{t=1}^{T}\mathbf{IG}_i(h_t^l) \tag{16}$$

where $\mathbf{IG}_i(h_t^l)$ is the integrated gradients along the i-th neuron dimension in the l-th layer at the t-th timestep, when the input is $[p, s_{<t}]$. Sundararajan et al. (2017) compute Riemann sum to approximate Eq. 15, which uses a fixed number of intervals to approximate the integrals. We closely follow the implementation of `https://github.com/EleutherAI/knowledge-neurons`.

A.3 Details of SLIMMING

We initialize every mask value m_i^l as 1, which is equivalent to running the pretrained LLM without masking. When training the mask, we clip every m_i^l to $[0, 1]$. Note that for both SLIMMING and HARD CONCRETE, because we are learning a mask on each neuron, we do not apply any random dropout during training.

A.4 Details of HARD CONCRETE

Louizos et al. (2018) obtain the hard concrete r.v. $\bar{m}_i^l$ by first stretching the binary concrete r.v. $\hat{m}_i^l$ (Eq. 11) from the interval $(0, 1)$ to (γ, ζ), where $\gamma = -0.1, \zeta = 1.1$, and then clip the value to the $[0, 1]$ interval:

$$\bar{m}_i^l = \min\Big(1, \max\Big(0,\ \hat{m}_i^l \cdot (\zeta - \gamma) + \gamma\Big)\Big)$$

They then use L_0 regularization to encourage sparsity on the weights after applying the mask $\bar{m}^l$. After reparametrization, they have the regularization $\mathcal{R}(\bar{m}^l)$:

$$\mathcal{R}(\bar{m}^l) = \sum_{i=1}^{d_2}\sigma\Big(\log m_i^l - C\Big), \tag{17}$$

where $C = \beta \log \frac{-\gamma}{\zeta}$ is a constant.

A.5 Computation costs of different methods

Among all five localization methods, ACTIVATIONS is the most computationally efficient, because Eq. 8 only requires one forward pass. Both the pruning-based methods SLIMMING and HARD CONCRETE perform fast, as only the masks are trainable. Calculating integrated gradients (IG) is time-consuming, while ZERO-OUT is the worst, because it leaves out every neuron one by one. We compare the computational cost of different

	Time
ACTIVATIONS	$\sim$ 0.3 sec
SLIMMING	$\sim$ 12 sec
HARD CONCRETE	$\sim$ 1 min
IG	$\sim$ 43 min
ZERO-OUT	$\sim$ 8.5 hr

Table 4: The elapsed time of different methods to do localization (i.e., assign attribution scores to every neuron) on one sequence memorized by Pythia-6.9B. We time all methods on a single RTX A6000 GPU.

[3] We follow Dai et al. (2022) to set the *baseline* in integrated gradients to a zero vector that has the same shape as h_t^l.

methods on one sequence memorized by Pythia-deduped-6.9B, where each sequence in the collected set $\mathcal{X}$ consists of a 32-token prefix and a 48-token suffix. We follow the common implementation that sets the number of intervals to 20 for Riemann sum in IG. Table 4 shows the elapsed time calculated on an RTX A6000 48G GPU. When running IG and Zero-Out we patch and batch the activations to reach 99% GPU utilities. Still, applying Zero-Out to do localization on one sequence costs 8.5 hours, and $\mathcal{X}$ contains 500 sequences in total. Due to the extremely heavy computation cost, we do not have the results of Zero-Out on Pythia-6.9B in the DEL Benchmark.

A.6 Details of Data Collection

We show some collected examples in Tables 8 & 9.

	Acc	Dist	PPL	Len
GPT2-XL	99.3%	0.48	10.18	150
Pythia-deduped-2.8B	98.8%	1.07	5.58	160
Pythia-deduped-6.9B	99.7%	0.20	5.24	167

Table 5: Quantifying memorization of the collected datasets. The high Accuracy (Acc) and low Levenshtein distance (Dist) show our collected sequences ($\mathcal{X}$) are indeed well memorized by LLMs. The last column (Len) reports the average suffix length of each dataset at the character level. We also measure the perplexity (PPL) on sequences sampled from the Pile-dedupe ($\mathcal{D}$).

The pretrained sequences of Pythia models. EleutherAI releases the exact batches used by Pythia models during pretraining, where each sequence in a batch consists of 2049 tokens [4]. We first randomly downsample the pretraining batches to a subset $\mathcal{Z}$ of 102400 sequences. Then, we use our criteria in §3.2 to determine whether Pythia memorizes a sequence in the subset. After filtering, there remain 500 $\sim$ 1000 sequences in the subsets for both Pythia-deduped-2.8B and Pythia-deduped-6.9B; we simply sample 505 of them respectively as our collected datasets.

We also randomly sample a subset of 2048 sequences ($\mathcal{D}$), each consisting of 128 tokens, to measure the perplexity of all LLMs we evaluate. We ensure that $\mathcal{Z} \cap \mathcal{D} = \varnothing$, so there is no overlap between the collected memorized sequences and sequences for perplexity.

Filtering with greedy decoding. Given the prefix p as the prompt to the LLM, we generate the suffix $\bar{s} = [\bar{s_1}, \ldots, \bar{s_{48}}]$ with greedy decoding, where

$$\bar{s}_t = \underset{w \in \text{Voc}}{\text{argmax}}\ P_\theta(w|p, \bar{s}_{<t}). \tag{18}$$

We then calculate the Levenshtein distance (Levenshtein, 1965) between the true suffix s and the generated one $\bar{s}$, filtering out sequences with a distance greater than 20 characters. Note $\bar{s}$ is different from $\hat{s}$ in Eq 4, which is generated by teacher-forcing and is used to calculate memorization scores.

Deduplication. Although we use the deduplicated version of the dataset and models, the Pile-dedupe and Pythia-deduped models, we still find lots of near-duplicated sequences. Thus, we further deduplicate the collected memorized sequences. In particular, we follow Lee et al. (2022) to represent each sequence with a set of 5-grams when calculating the Jaccard index. Among a set of duplicates, we select the one that is best memorized, i.e., having the lowest Levenshtein distance on the generated suffix $\bar{s_t}$ (Eq. 18), and discard the others.

Manually searched data. With our searching criteria in §3.2, we can only identify less than 10 memorized sequences from subsets of the Pile-dedupe, Common Crawl, and Wikipedia, probably because OpenAI carefully preprocesses the data before training GPT2-XL. Thus, we actively search for potentially memorized data, such as famous poems and common lists of sorted items. We collect 105 sequences memorized by GPT2-XL and manually categorize them (see Tables 1 & 8), including 31 examples from Carlini et al. (2021). We set the prefix and suffix of a sequence by trial and error to ensure high memorization Accuracy. Unlike automatic searches that tend to find templated texts or uninteresting data with repetitive tokens (Zhang et al., 2023), our manual search ensures better data quality and enables us to analyze memorization within and across categories.

In particular, Figures 4 & 5 show that different localization methods constantly confuse sequences of related categories. For example, they are unable to disentangle neurons of different quotes and identify a small set of neurons responsible for both the order of Zodiac Signs and the order of Planets.

Responsible checklist. Note the `Contact Info` category of our manually collected dataset only

[4] `https://github.com/EleutherAI/pythia#exploring-the-dataset`

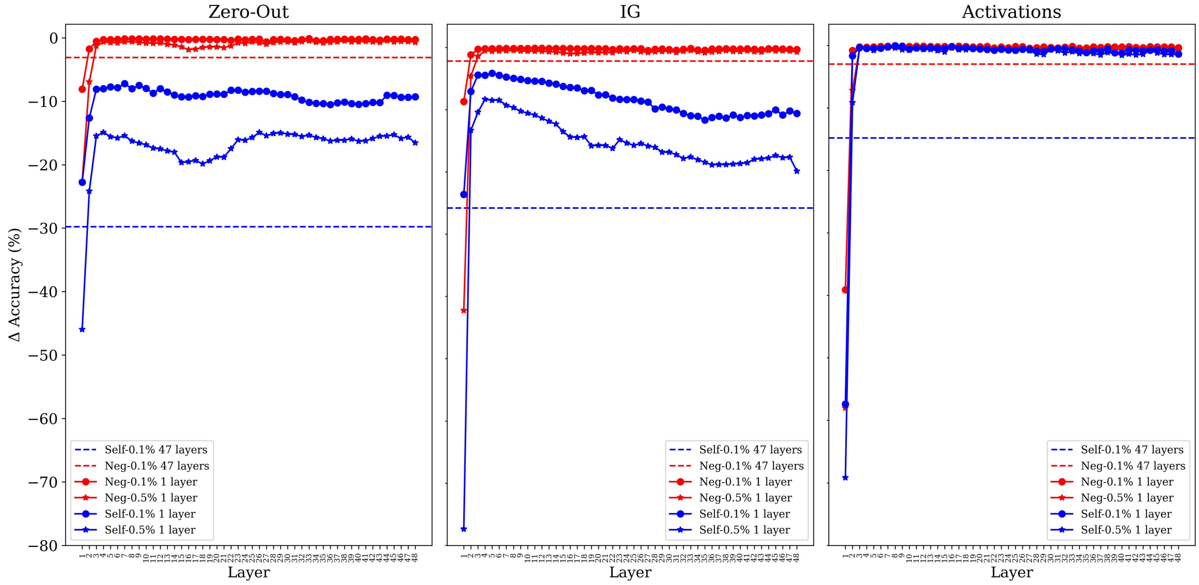

Figure 3: The DEL Benchmark results of ZERO-OUT, IG, and ACTIVATIONS methods when dropping out the same number of neurons in a single layer, where the blue lines show Δ Self-Acc and the red lines show Δ Neg-Acc. Under the same "neuron budget", dropping out neurons in multiple layers (blue dashed lines) substantially outperforms dropout in a single layer, implying that the memory of a piece of data is distributed over layers. Besides, dropping out neurons in the bottom layer greatly hurts the memorization of negative examples (red lines), suggesting that the bottom layer encodes general information.

contains public data, such as mailing addresses of corporate headquarters and famous buildings; thus, it does not have any potential risk of revealing private information. Similarly, our memorized datasets for Pythia models are a subset of the Pile, a public corpus under the MIT License.

A.7 Hyperparameters

In the INJ Benchmark, the ECBD-2021 set contains 156 definition sequences. For the DEL Benchmark, we collect a set of 505, 505, and 105 sequences memorized by Pythia-deduped-6.9B, Pythia-deduped-2.8B, and GPT2-XL, respectively. For each set, we sample 5 sequences as the dev set, using the dev set performance to determine the hyperparameters for each LLM. The hyperparameters include the integrated gradient steps, i.e., the number of intervals in Riemann sum for integral approximation in IG; the temperature β and the initialization value of parameters m in HARD CONCRETE; the learning rate, the number of training epochs, and λ, which balances the memorization loss and the sparsity loss, in SLIMMING and HARD CONCRETE. We observe that both SLIMMING and HARD CONCRETE are sensitive to the choice of hyperparameters. On the other hand, we find the performance of IG does not improve when using more integrated gradient steps, where we experiment with different steps ranging from 20 to 300. Thus, we set the step to 20 for all examples to reduce the heavy computation costs.

A.8 Dropping out neurons in a single layer

For the DEL Benchmark, we study the alternative that dropping out the same number of neurons in a single layer to understand the individual effect of each layer. Specifically, in §5.2, a method predicts top-k% of neurons in *every* layer after the bottommost layer. Thus, we have a "budget" of $N = k\% \times 6400 \times 47$ neurons for GPT2-XL, which has 6400 neurons in each FFN layer and 48 layers in total. In this section, we drop out the top-N neurons in a single layer in terms of the attribution scores assigned by a method.

Figure 3 illustrates the absolute change in Accuracy when dropping out the top-N neurons in a layer, where the neurons are predicted by ZERO-OUT, IG, and ACTIVATIONS methods, respectively. The horizontal dashed lines show the results we report in Table 3 for comparison. First, we find that dropping out the same number of neurons in multiple layers is much more efficient in erasing the target sequence, as the blue dashed line shows a greater decrease in Accuracy compared with drop-

	GPT2 124M			GPT2-XL 1.5B			Pythia-deduped 2.8B			Pythia-deduped 6.9B		
	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%
ratio = 1%												
Hard Concrete	**46.6**	**66.8**	**88.0**	**21.8**	**25.1**	**32.8**	33.3	**48.4**	**70.7**	31.5	47.5	**69.4**
Slimming	43.1	64.6	79.9	5.2	11.5	27.0	**33.6**	47.3	59.8	**35.0**	**49.6**	63.4
Zero-Out	24.0	36.8	52.7	4.2	7.3	13.5	10.1	14.3	20.5	-	-	-
IG	10.3	18.1	36.3	1.4	4.8	12.2	6.1	10.8	21.1	8.9	13.9	24.1
Activations	2.5	4.4	9.8	1.5	2.8	6.8	3.2	5.1	21.6	4.1	6.3	17.4
Random	1.0	2.0	5.0	1.0	2.0	5.0	1.0	2.0	5.0	1.0	2.0	5.0
ratio = 0.1%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%
Hard Concrete	51.2	77.4	**96.4**	**49.8**	**57.5**	**63.6**	**45.6**	**65.5**	**85.9**	28.7	40.7	55.8
Slimming	**62.7**	**87.0**	95.4	18.1	35.1	54.0	45.0	62.6	73.6	**39.1**	**52.1**	**64.3**
Zero-Out	57.4	81.7	91.9	14.7	20.9	31.1	16.4	20.6	25.8	-	-	-
IG	36.0	55.0	75.5	2.5	3.5	6.0	12.6	16.4	21.9	19.7	23.6	28.9
Activations	9.0	12.9	23.4	3.5	4.6	6.7	8.0	16.8	40.5	21.2	31.4	50.2
Random	0.1	0.2	0.5	0.1	0.2	0.5	0.1	0.2	0.5	0.1	0.2	0.5

Table 6: The INJ Benchmark. The average Reacall@$k\%$ of different methods when predicting top-$k\%$ of neurons *across* layers. The results are consistent with Table 2, where methods predict top-$k\%$ of neurons in each layer.

ping out neurons in a single layer. Dropping out N neurons in multiple layers (Self-0.1% 47 layers) even outperforms dropping out $5 \times N$ neurons in a single layer (Self-0.5% 1 layer), suggesting that the storage of a piece of memory is distributed over layers instead of concentrating in a single layer.

The only exception is dropping out neurons in the bottommost layer, where Layer 1 decreases more than multiple layers in Self-Acc; however, it also greatly hurts Neg-Acc, the memorization of negative examples. Layer 2 shows a similar but slighter trend. The large decreases in memorization accuracy on all sequences suggest that the bottom layers of LLMs mainly work on processing basic syntactic information (Tenney et al., 2019) or encoding general concepts, instead of focusing on a specific sequence.

We do not have the single-layer results of Slimming and Hard Concrete, because both methods train the masks of all neurons jointly, which requires us to retrain the masks only on a single layer to obtain its attribution scores. In comparison, the other three methods in Figure 3 consider each neuron individually, allowing us to use the same attribution scores we have in §5.2 to select neurons in a single layer and make direct comparisons with the values in Table 3.

A.9 Predicting top neurons across layers

In the INJ Benchmark, we randomly sample weights across layers to inject the data, instead of sampling a fixed percentage of weights per layer (see Algorithm 1). Hence, it may seem more natural to predict top-$k\%$ of neurons across layers; we experiment with this alternative in Table 6.

Comparing the results of Table 2 and Table 6, we find that predicting top neurons per layer outperforms predicting top neurons across layers. This is because all localization methods assign larger attribution scores to neurons in the bottom layers, barely predicting neurons in the upper layers if we rank neurons globally. On the other hand, Table 2 and Table 6 show consistent results. Our findings and the ranking of different methods are coherent whether we rank neurons per layer or globally.

A.10 Implementation Details

Table 7 summarizes the architectures of LLMs we use. We run most experiments on RTX3090 24G GPUs; experiments involving Pythia-6.9B are run on RTXA6000 48G GPUs. We use `transformers 4.31.0` and `pytorch 1.13`.

	# Layers	# Neurons
GPT2 124M	12	3072
GPT2-XL 1.5B	48	6400
Pythia-deduped-2.8B	32	10240
Pythia-deduped-6.9B	32	16384

Table 7: The number of layers and the number of FFN neurons in each layer of different LLMs.

```
**Algorithm 1** Information Injection
Input: The set of new sequences $\mathcal{X}_{\text{ECBD}} = \{x_i\}_{i=1}^{N}$; pretrained LLM $\theta$ with $L$ layers; injection ratio $r$
Output: The set of fine-tuned LLMs $\mathcal{M} = \{\tilde{\theta}_i\}_{i=1}^{N}$

Initialize $\mathcal{M} \leftarrow \varnothing$.
for $i \leftarrow 1$ to $N$ do
    $\tilde{\theta}_i \leftarrow \theta$ // Initialize from pretrained weights.
    Retrieve all the FFN weight vectors $\Phi_i = \{\mathbf{v}_1^l, \ldots, \mathbf{v}_{d_2}^l\}_{l=1}^{L}$ from layers $l$ of $\tilde{\theta}_i$.
    Set the random seed to $i$.
    $\phi_i \leftarrow$ Randomly sample $r\%$ of weight vectors from $\Phi_i$. // $\phi_i \subset \Phi_i \subset \tilde{\theta}_i$
    Fine-tune $\phi_i$ with the language modeling loss on $x_i$ (Eq. 13) with remaining weights $\tilde{\theta}_i \setminus \phi_i$ frozen.
    $\mathcal{M} \leftarrow \mathcal{M} \cup \tilde{\theta}_i$.
end for
return $\mathcal{M}$
```

Email	100%	Write to Jon Hilsenrath at jon.hilsenrath@wsj.com
Zodiac Signs	100%	Aries Taurus Gemini Cancer Leo Virgo Libra Scorpio Sagittarius Capricorn Aquarius Pisces
Patreon	100%	Thank you to our Patreon Supporters: Saintsofwar, Anon, Lord_Of_Fapping, Dryzak, Chabalbac, ioNz, LaX, VNT
Declaration of Independence	100%	We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness.
Trump	100%	Sorry losers and haters, but my I.Q. is one of the highest -and you all know it! Please don't feel so stupid or insecure, it's not your fault.
Newton	100%	I do not know what I may appear to the world, but to myself I seem to have been only like a boy playing on the sea-shore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.
Dr. MLK	100%	And when this happens, and when we allow freedom ring, when we let it ring from every village and every hamlet, from every state and every city, we will be able to speed up that day when all of God's children, black men and white men, Jews and Gentiles, Protestants and Catholics, will be able to join hands and sing in the words of the old Negro spiritual, "Free at last! Free at last! Thank God Almighty, we are free at last"
Genesis	100%	In the beginning God created the heaven and the earth. And the earth was without form, and void; and darkness was upon the face of the deep. And the Spirit of God moved upon the face of the waters. And God said, Let there be light: and there was light.
The Road Not Taken	100%	Two roads diverged in a yellow wood,\n\nAnd sorry I could not travel both\n\nAnd be one traveler, long I stood\n\nAnd looked down one as far as I could\n\nTo where it bent in the undergrowth;\n\nThen took the other, as just as fair,\n\nAnd having perhaps the better claim,\n\nBecause it was grassy and wanted wear

Table 8: Examples of our manually collected data. The prompt (prefix) is colored in brown. The numbers are the Accuracy (Eq. 5) of GPT2-XL on memorizing the sequences, where 100% Accuracy means the true suffix can be fully reconstructed with greedy decoding.

Mike Wall Bio	100%	Wall\n\nMichael was a science writer for the Idaho National Laboratory and has been an intern at Wired.com, The Salinas Californian newspaper, and the SLAC National Accelerator Laboratory. He has also worked as a herpetologist and wildlife biologist. He has a Ph.D. in evolutionary biology from the University of Sydney, Australia, a bachelor's degree from the
Hardware	100%	PCs) may be defined as a desktop, floor standing, or portable microcomputer that includes a system unit having a central processing unit (CPU) and associated volatile and non-volatile memory, including random access memory (RAM) and basic input/output system read only memory (BIOS ROM), a system monitor, a keyboard, one or more flexible diskette drives, a CD-ROM drive,
Contact Info of Skyhorse Publishing	100%	, or educational purposes. Special editions can also be created to specifications. For details, contact the Special Sales Department, Arcade Publishing, 307 West 36th Street, 11th Floor, New York, NY 10018 or arcade@skyhorsepublishing.com.\n\nArcade Publishing® is a registered trademark of Skyhorse Publishing, Inc.®, a Delaware corporation.\n\nVisit
Meme	98%	a lot; that's great! It's a little awkward to ask, but we need your help. If you have already donated, we sincerely thank you. We're not salespeople, but we depend on donations averaging $14.76 and fewer than 1% of readers give. If you donate just $5.00, the price of your coffee, Catholic Online School could keep thriving. Thank
Malik Report	100%	check that allowed Dvorak to flick the puck over his shoulder...\n\nAbout The Malik Report\n\nThe Malik Report is a destination for all things Red Wings-related. I offer biased, perhaps unprofessional-at-times and verbose coverage of my favorite team, their prospects and developmental affiliates. I've joined the Kukla's Korner family with five years of blogging under
Porn	100%	make love to her. She returned the favor with an amazing blowjob and a masterful fuck session...\nENJOY!!!!\n\nThis entire website has a voluntary content rating to block access by minors. This rating is compatible with microsoft internet explorer's content filtering function and\nfacilitates website blocking software. For a tutorial on blocking this site click here.\nCopyright bangbros.
Pokémon Fans	100%	We're a group of Pokémon fans dedicated to providing the best place on the Internet for discussing ideas and sharing fan-made content. Welcome! We're glad you're here.\n\nIn order to join our community we need you to create an account with us. Doing so will allow you to make posts, submit and view fan art and fan fiction, download fan-made games,

Table 9: Examples of memorized sequences we collect from the Pile-dedupe. The prompt (prefix) is colored in brown. The numbers are the Accuracy (Eq. 5) of Pythia on memorizing the sequences, where 100% Accuracy means the true suffix can be fully reconstructed with greedy decoding.

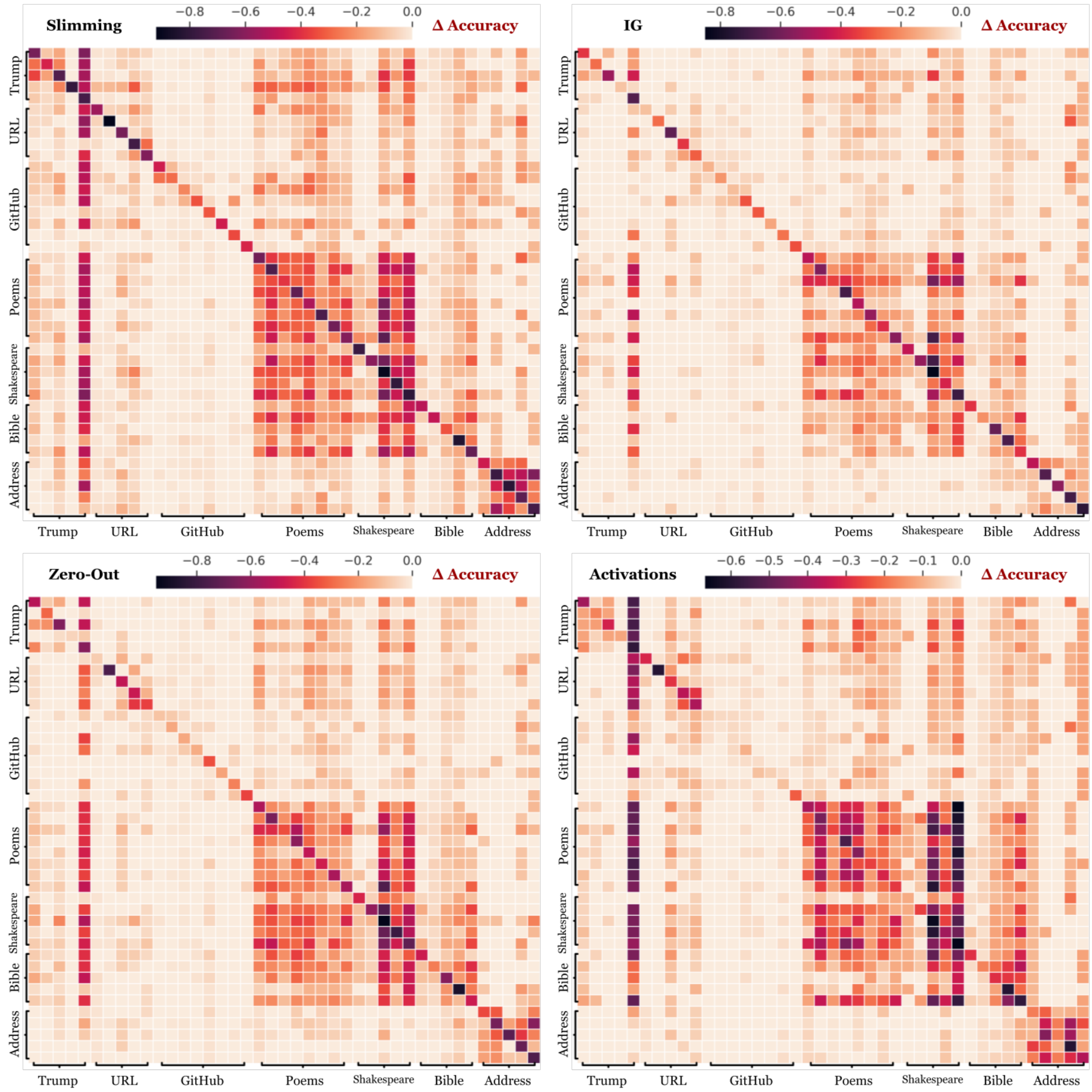

Figure 4: Confusion matrices of localization methods on a subset of sequences memorized by GPT2-XL, where each row/column represents a sequence. Different methods show similar patterns of confusion.

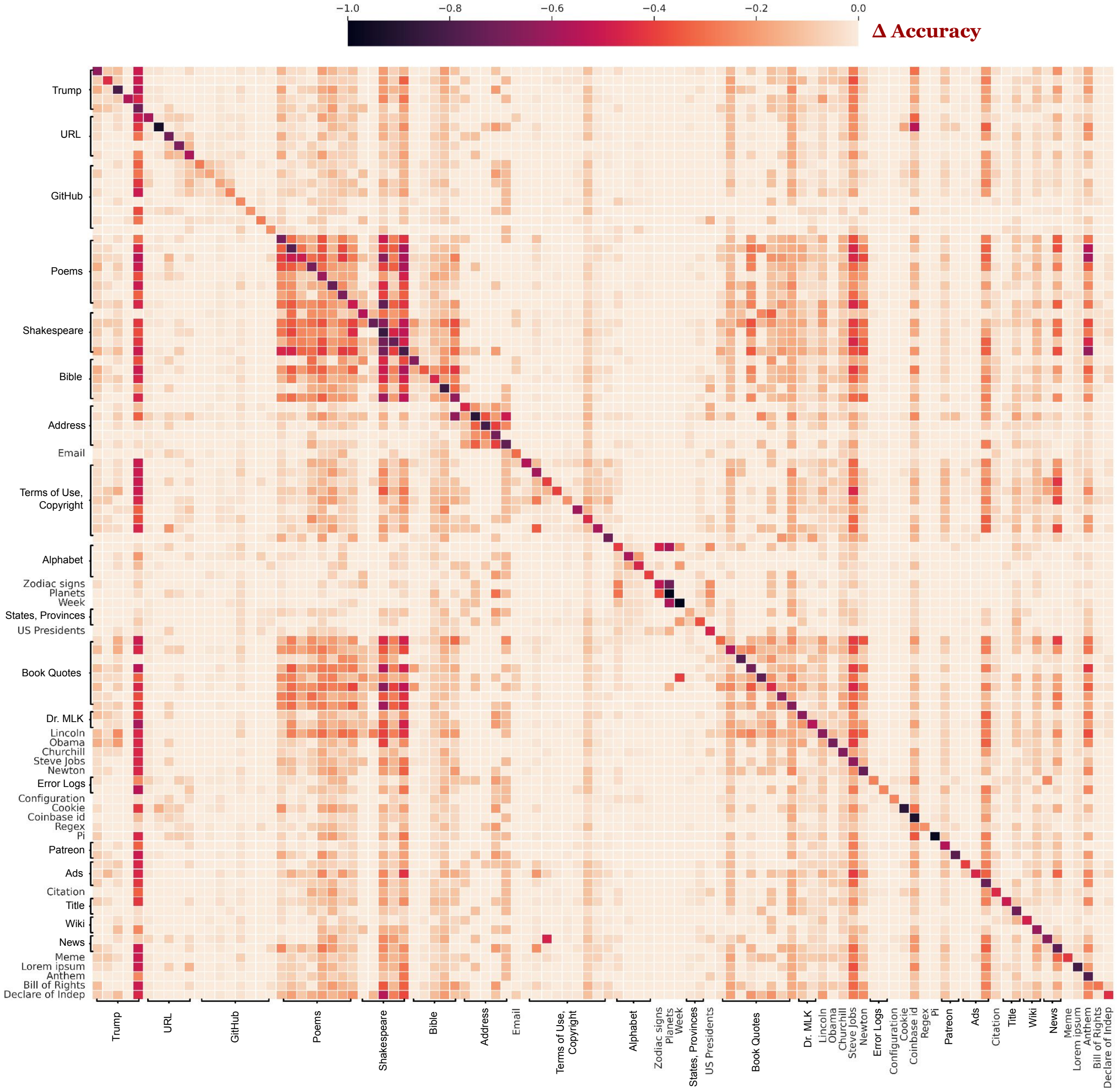

Figure 5: Confusion matrix of HARD CONCRETE on the entire test set memorized by GPT2-XL. Each row shows how dropping out the predicted neurons (0.5%) on a target sequence changes the Accuracy of all sequences. HARD CONCRETE is unable to disentangle neurons of different quotes, including poems, Bible, books, and some famous people quotes. Also, it finds a small set of neurons responsible for memorizing both the order of Zodiac Signs and the order of Planets.