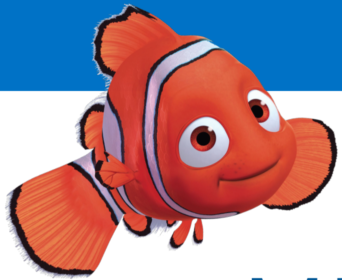
Finding NeMo: Localizing Memorizing Neurons in Diffusion Models



Franziska Boenisch

ML in PL, November 9th, 2024

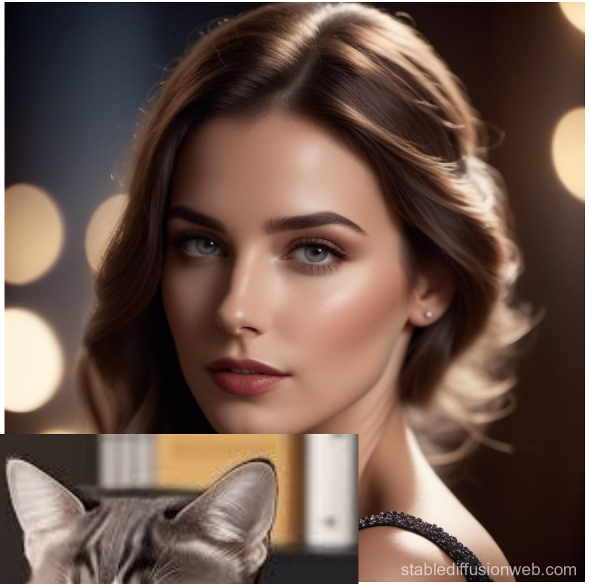
CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

SprintML

Diffusion models create detailed images

Stable Diffusion 3

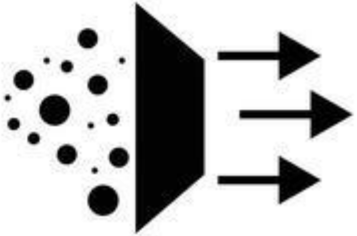
Midjourney vs DALL·E 3



Diffusion models regenerate training data!

Training Set

Caption: Living in the light with Ann Graham Lotz

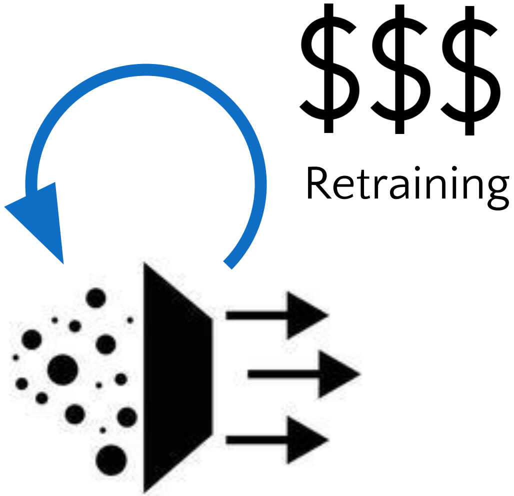


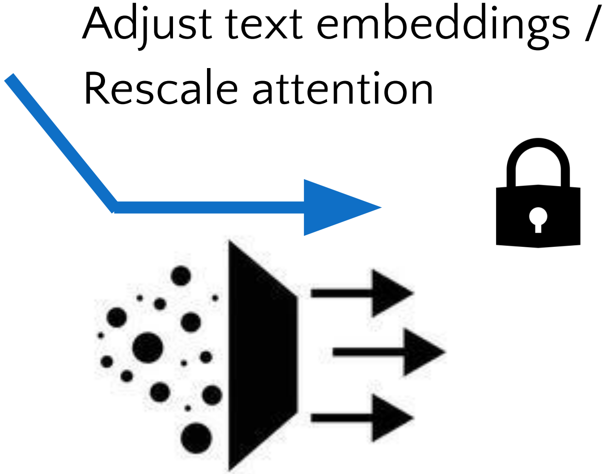
Generated Image

Prompt: Ann Graham Lotz

[Carlini et al., 2023]

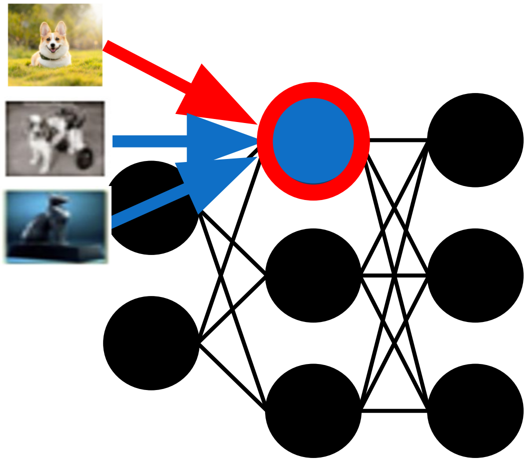
Existing defenses are costly or reversible





[Wen et al., 2024; Ren et al., 2024]

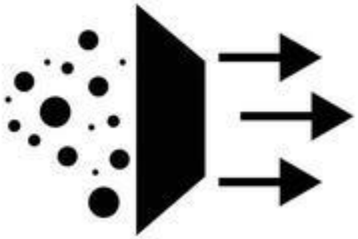
NeMo: Localize memorizing neurons



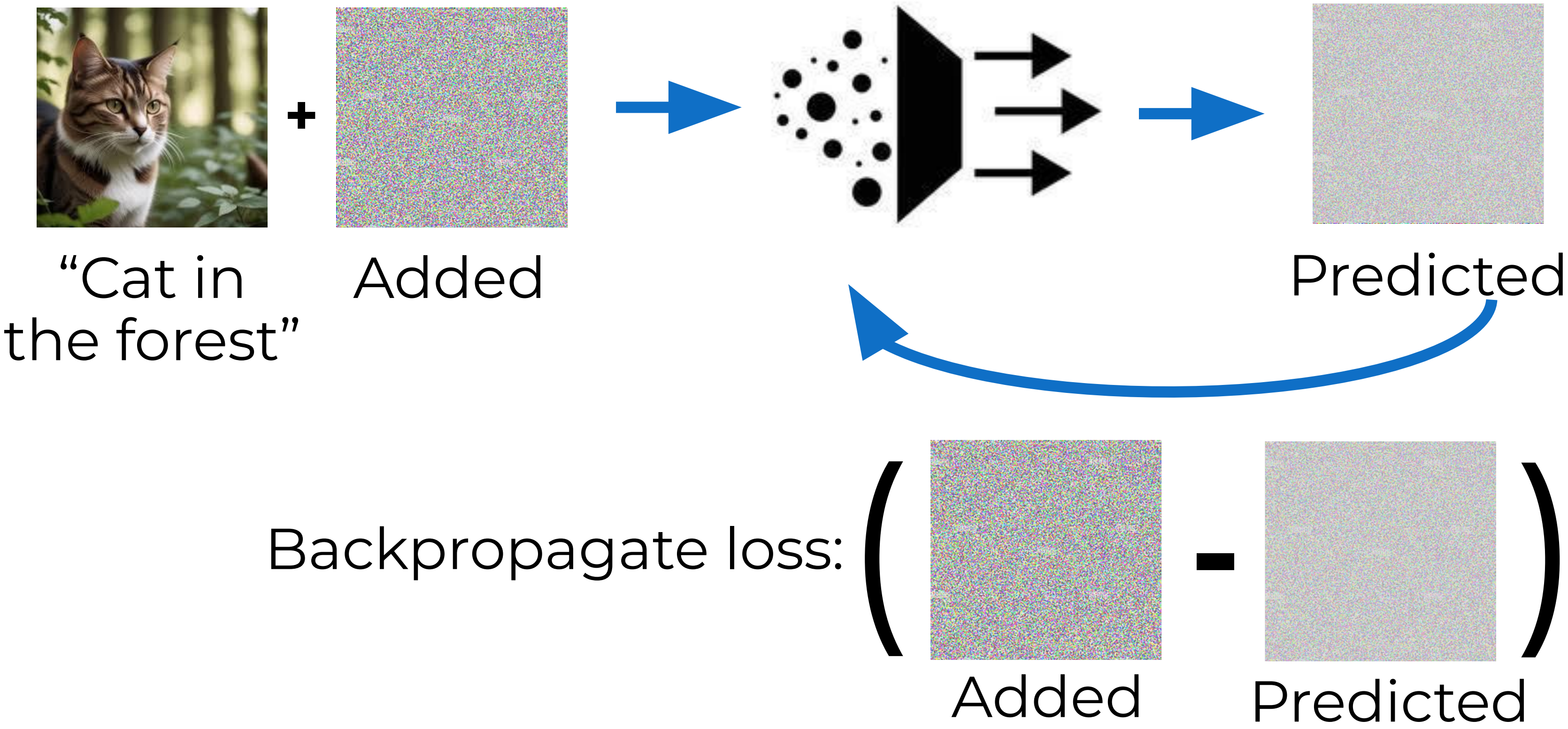
Outlier activation
for a single image

Dominik Hintersdorf, Lukas Struppek, Kristian Kersting, Adam Dziedzic, and Franziska Boenisch. *"Finding NeMo: Localizing Neurons Responsible For Memorization in Diffusion Models."* In **NeurIPS'24**

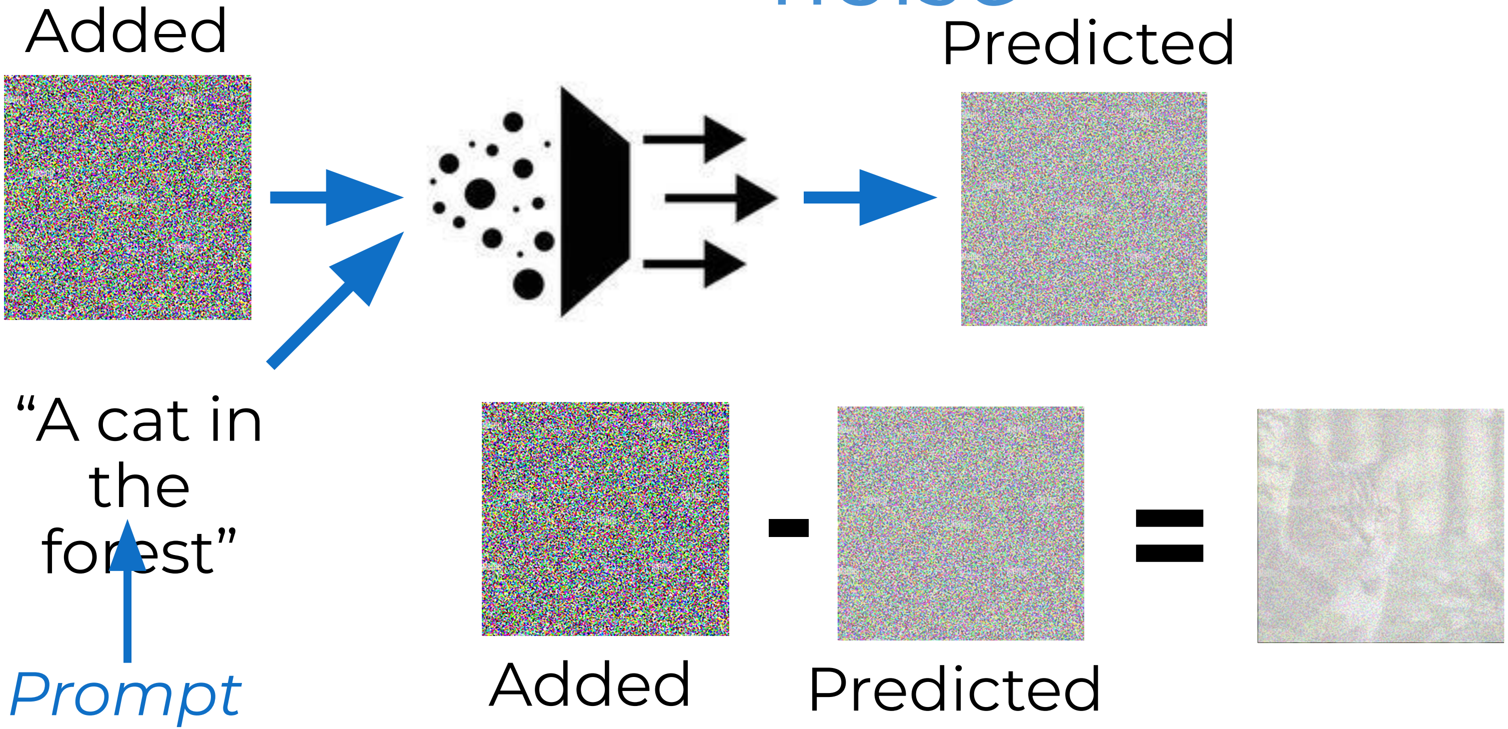
DMs generate images from noise



DM Training: Predict added noise

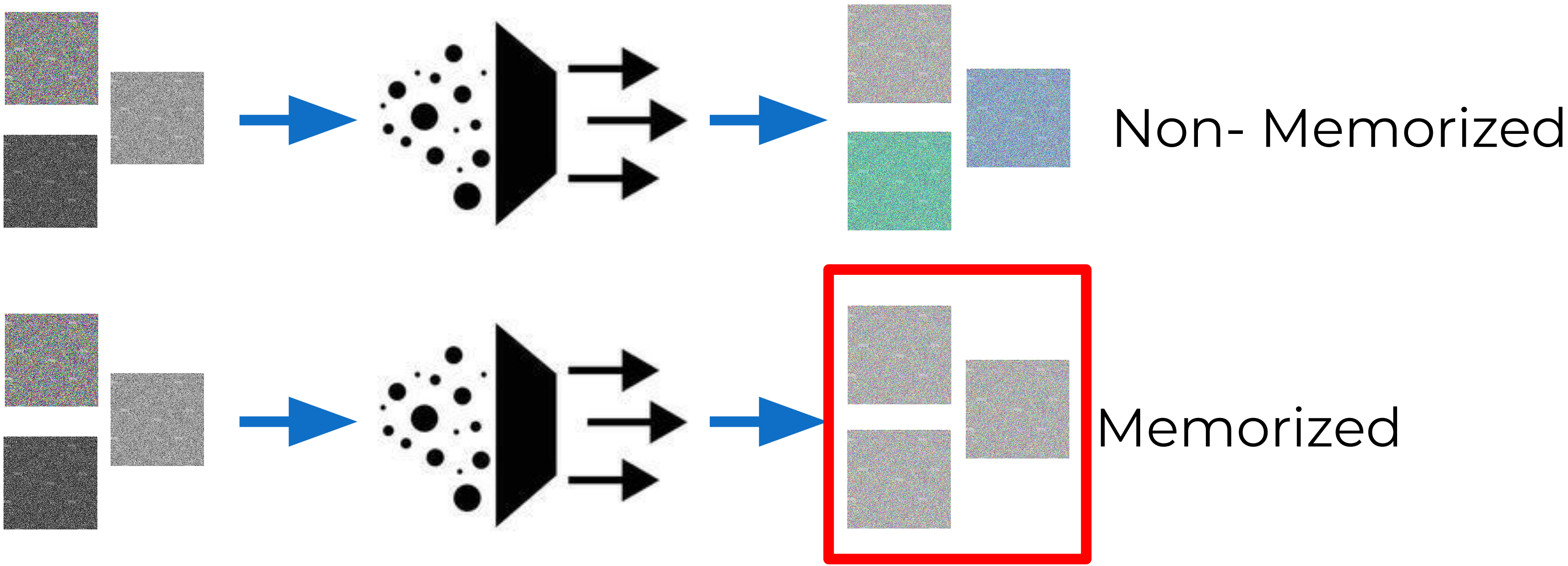


DM Generation: Remove predicted noise



Detecting memorization efficiently

Different denoising trajectories!

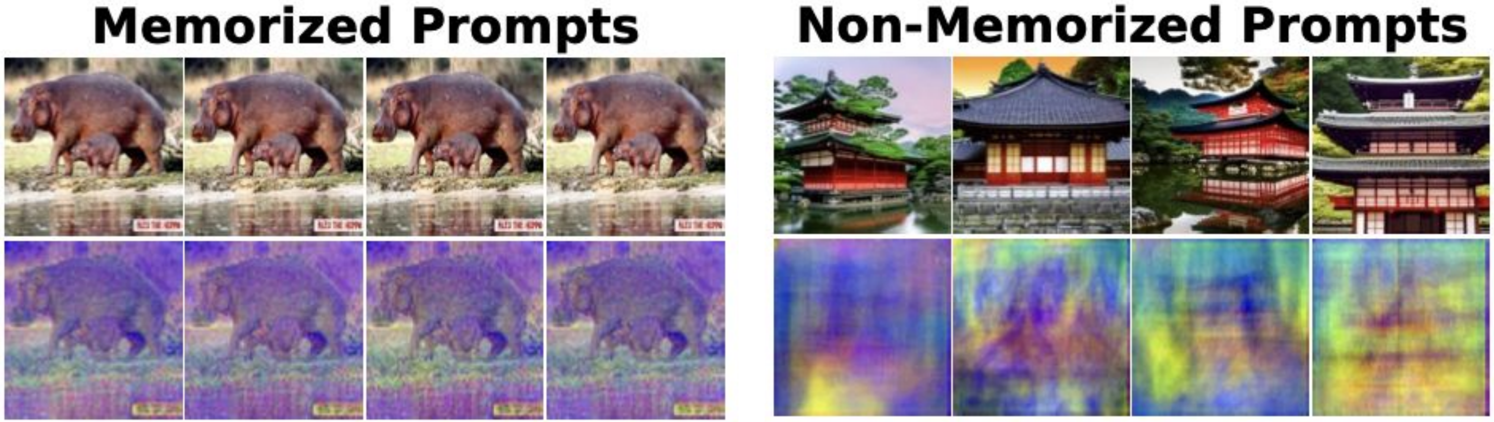


[Wen et al., 2024]

Different denoising trajectories

Final Generation

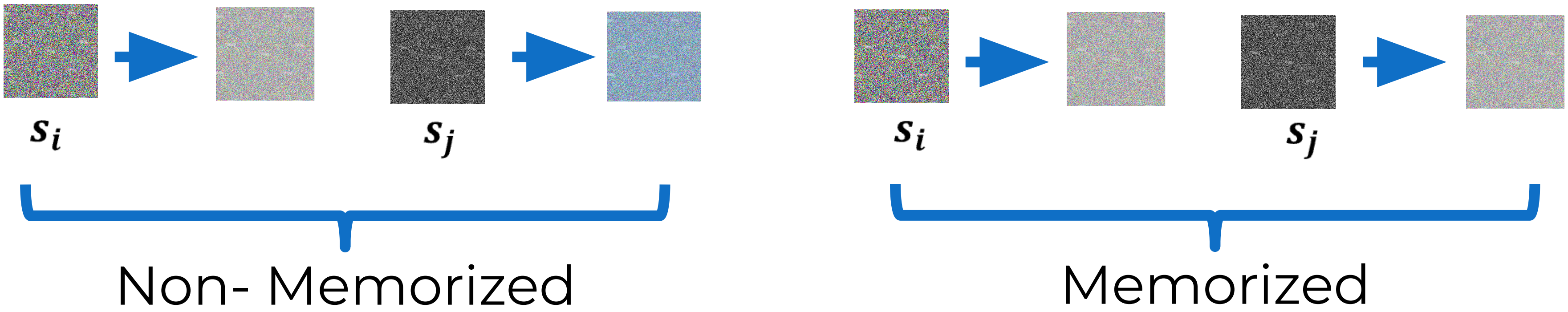
Noise Trajectory



Comparison of initial noise predictions with different seeds

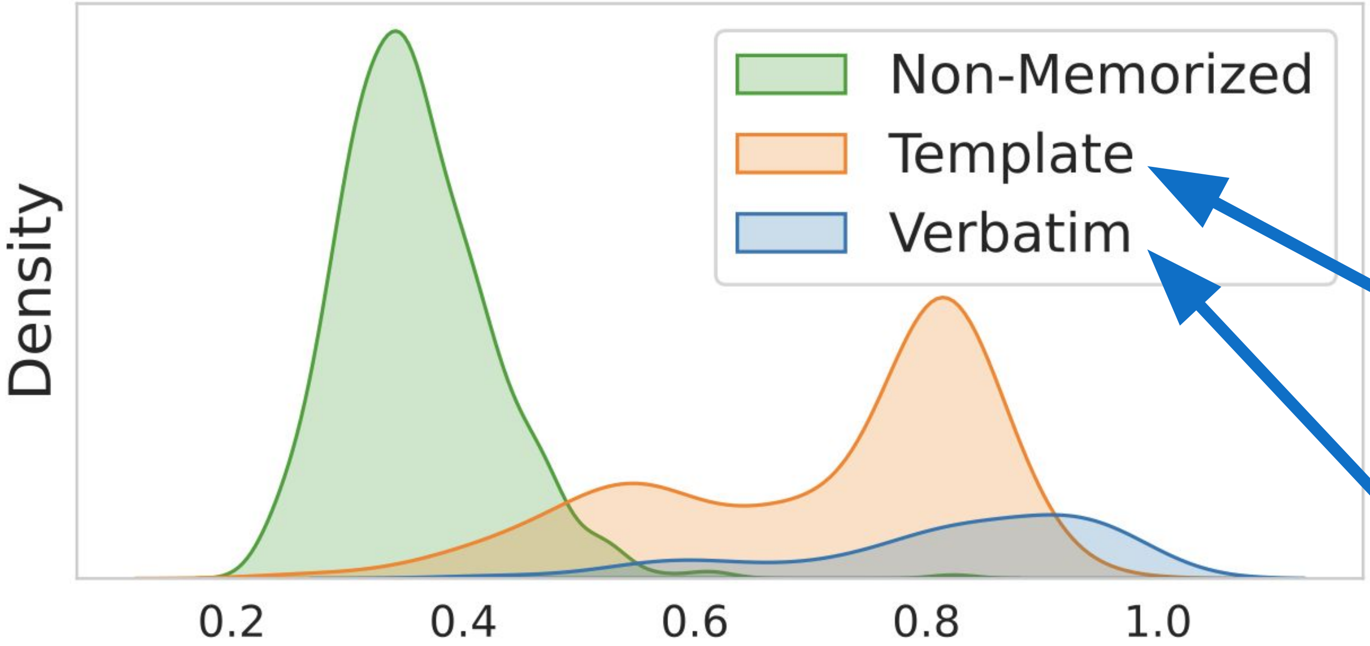
Quantifying memorization

Structural Similarity Index Measure (SSIM)
between noise differences from different seeds



[SSIM: Zhou et al., 2004]

SSIM score detects memorization



SSIM score distribution
(Stable Diffusion v1.4, LAION Dataset)

z-Score for identifying memorization

Layer l

Prompt y

Activation

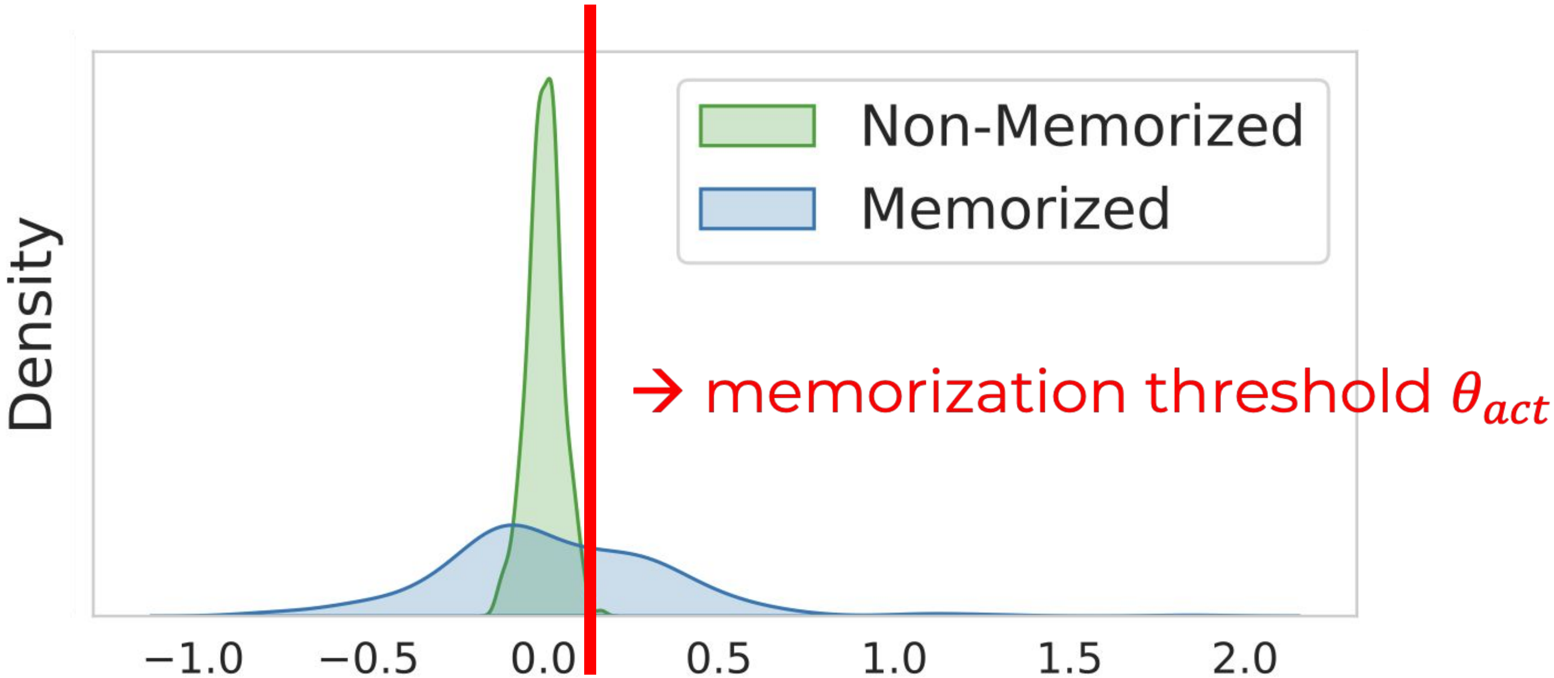
Mean activation on holdout set

$$z_i^l(y) =$$

Neuron i

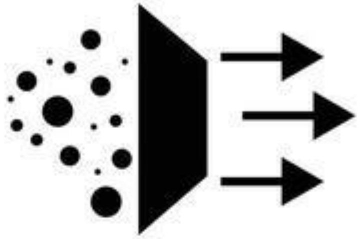
Standard deviation on holdout set

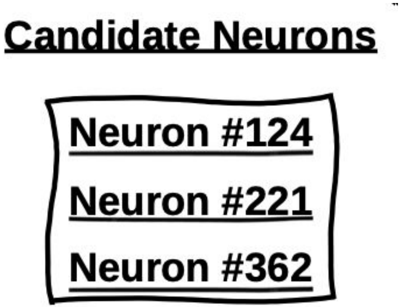
Attributing memorization to neurons



Neuron activation standardized by z-score
(first value layer of Stable Diffusion v1.4)

Efficiently localizing memorization





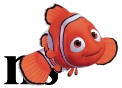
Step 1: Coarse grained candidate neuron selection

θ_{act} > 4.75
& top k+1 neurons

Step 2: Refinement of the candidate set

Layer by layer
Neuron by neuron in remaining layers

NeMo identifies and mitigates memorization while maintaining utility

	SSCD (generated)	Alignment (CLIP)
All neurons active (default)		
Deactivating 4 random neurons		
Deactivating 4 NeMo-detected neurons 		

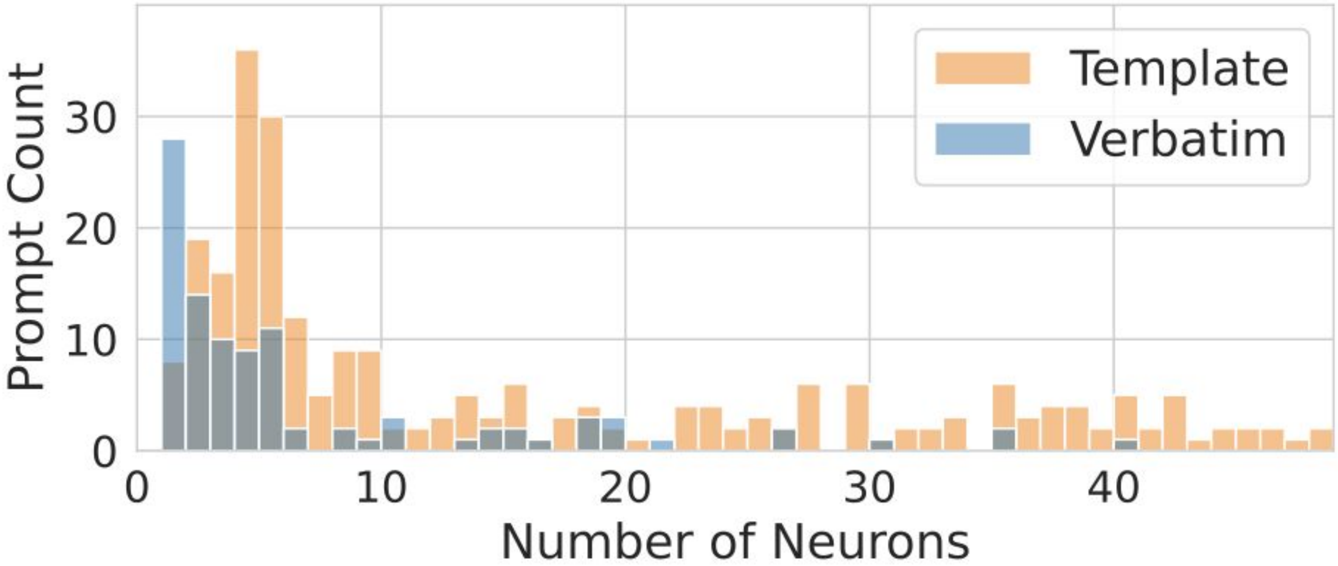
Low Memorization

Good Alignment

Only a few neurons are responsible for memorizing data points

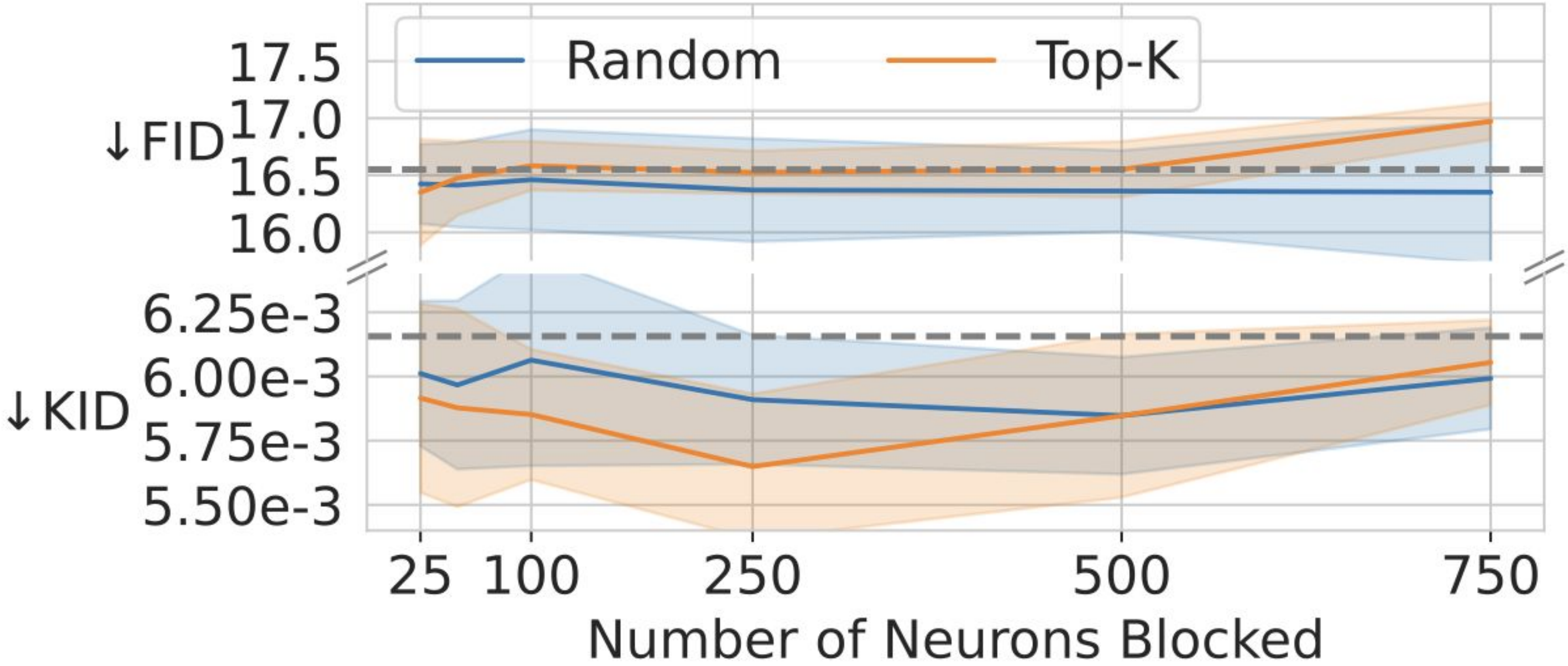
Number of neurons deactivated

Only a few neurons are responsible for memorizing data points

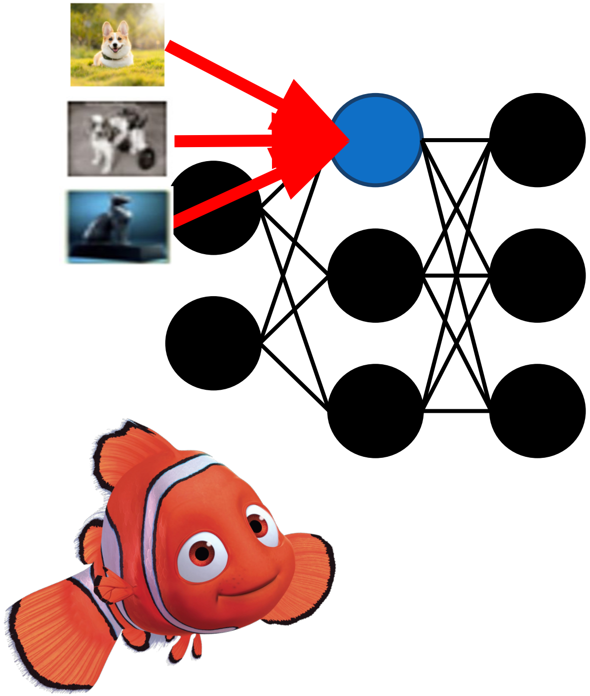


Some neurons memorize multiple points

Deactivating neurons does not significantly impact the quality of generated images

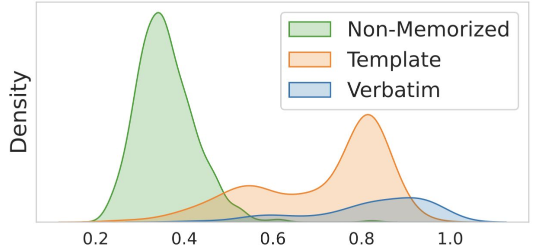


Conclusion



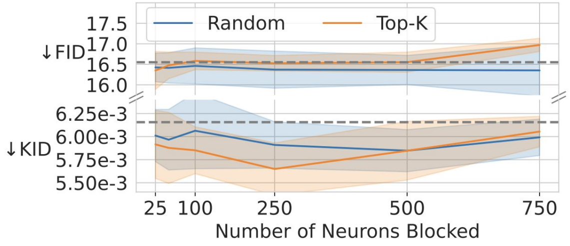
1. NeMo allows to localize memorized data in
   individual neurons.
2. By deactivating these neurons, we can miti-
   gate memorization.

Thank you!



Questions?

franziska-boenisch.de
boenisch@CISPA.de
@fraboeni



Deactivating neurons in attention layers

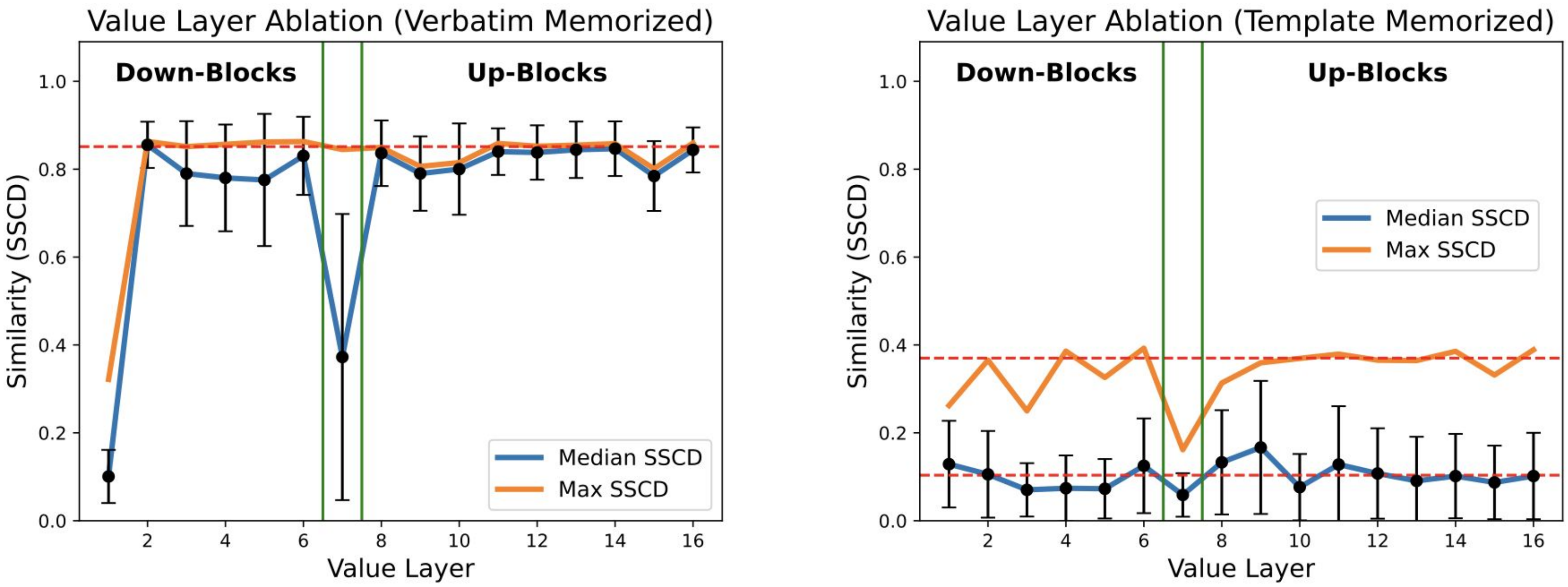
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V$$

Q: **Query** □ No direct input from text guidance

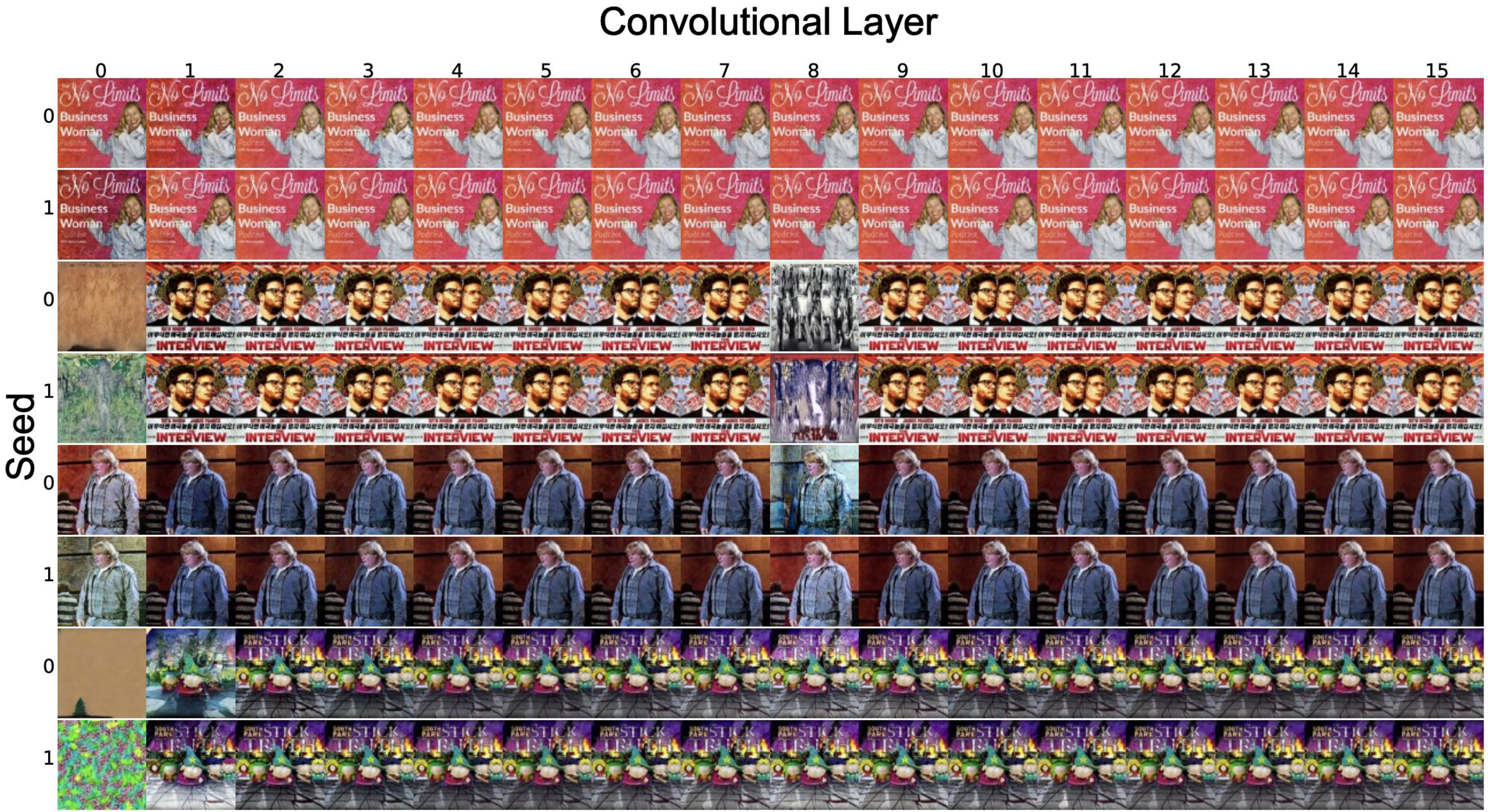
K: **Keys** □ Directly process text embeddings, but do not operate separately

V: **Values** □ Directly process text embeddings and no dependence between neurons

Deactivating neurons in other layers in the U-Net is possible but degrades images



Deactivating in convolutional layers does not reduce memorization



Initial candidate selection

Algorithm 3 Initial Neuron Selection

Input:
Prompt embedding y ▷ Text prompt embedding
Memorization threshold (SSIM) τ_{mem} ▷ Target memorization score
Minimum activation threshold θ_{min} ▷ Threshold for stopping neuron search

Output: Set of neuron candidates S_{initial}, refinement memorization threshold $\tau_{\text{mem\_ref}}$

```
Candidate set of memorization neurons S_initial              ▷ Initial memorizing neuron set
Memorization threshold (SSIM) τ_mem_ref                      ▷ Memorization threshold for refinement

mem ← 1.0                                                    ▷ Initialize memorization score as maximum
θ_act ← 5                                                    ▷ Initialize threshold of OOD activation detection
k ← 0                                                        ▷ Initialize k for top-k activation detection
τ_mem_ref ← τ_mem                                            ▷ Set refinement memorization threshold to current threshold
Δ_unblocked ← get_noise_diff(y, ∅)                           ▷ Noise differences with all neurons active

// Increase set of candidate neurons until target memorization score is reached
while mem > τ_mem do                                         ▷ While memorization score above threshold
    S_initial ← get_ood_neurons(y, θ_act, k)                 ▷ Detect neurons with OOD activations
    Δ_blocked ← get_noise_diff(y, S_initial)                 ▷ Compute noise differences
    mem ← compute_memorization(Δ_unblocked, Δ_blocked)       ▷ Compute memorization score (SSIM)

    if θ_act < θ_min then                                    ▷ Minimum activation threshold not reached
        τ_mem_ref ← mem                                      ▷ Set refinement threshold to current memorization score
        break                                                ▷ Stop if activation threshold is too low
    end if

    // Adjust OOD detection parameters to increase set of candidate neurons
    θ_act ← θ_act − 0.25                                     ▷ Decrease threshold for OOD detection
    k ← k + 1                                                ▷ Increase k for top-k activation detection
end while

return S_initial, τ_mem_ref                                  ▷ Return neuron candidates and refinement memorization threshold
```

Candidate refinement

Algorithm 4 Neuron Selection Refinement

Input:
Initial memorization neuron candidate set S_{initial} ▷ Given neuron candidate set
Memorization threshold (SSIM) $\tau_{\text{mem\_ref}}$ ▷ Refinement memorization score threhsold

Output: memorization neurons S_{refined} ▷ Refined set of memorization neurons

$S_{\text{refined}} \leftarrow S_{\text{initial}}$
$\Delta_{\text{unblocked}} \leftarrow \text{get\_noise\_diff}(y, \emptyset)$ ▷ Noise differences with all neurons active

// Check all candidate neurons of individual layers at once for memorization
for $l \in \{1, \ldots, L\}$ **do** ▷ Iterate over all layers to remove low impact layers
  $S_{\text{layer}} \leftarrow \text{get\_neurons\_in\_layer}(S_{\text{refined}}, l)$ ▷ Get the neurons in the current layer l
  $S_{\text{neurons}} \leftarrow S_{\text{refined}} \setminus S_{\text{layer}}$ ▷ Compute set of neurons from remaining layers
  $\Delta_{\text{blocked}} \leftarrow \text{get\_noise\_diff}(y, S_{\text{neurons}})$ ▷ Compute noise differences
  $\text{mem} \leftarrow \text{compute\_memorization}(\Delta_{\text{unblocked}}, \Delta_{\text{blocked}})$ ▷ Compute memorization score (SSIM)

  if $\text{mem} < \tau_{\text{mem\_ref}}$ **then** ▷ Minimum memorization threshold not reached
    $S_{\text{refined}} \leftarrow S_{\text{refined}} \setminus S_{\text{layer}}$ ▷ Remove neurons of layer l from neuron set
  end if
end for

// Check all remaining candidate neurons individually
for $l \in \{1, \ldots, L\}$ **do** ▷ Iterate over each remaining layer
  $S_{\text{layer}} \leftarrow \text{get\_neurons\_in\_layer}(S_{\text{refined}}, l)$ ▷ Get the neurons in the current layer l

  for $n \in S_{\text{layer}}$ **do**
    $S_{\text{neurons}} \leftarrow S_{\text{refined}} \setminus \{n\}$ ▷ Compute set of neurons without neuron n
    $\Delta_{\text{blocked}} \leftarrow \text{get\_noise\_diff}(y, S_{\text{neurons}})$ ▷ Compute noise differences
    $\text{mem} \leftarrow \text{compute\_memorization}(\Delta_{\text{unblocked}}, \Delta_{\text{blocked}})$ ▷ Compute mem. score (SSIM)

    if $\text{mem} < \tau_{\text{mem\_ref}}$ **then** ▷ Minimum memorization threshold not reached
      $S_{\text{refined}} \leftarrow S_{\text{refined}} \setminus \{n\}$ ▷ Remove current neuron from set
    end if
  end for
end for

return S_{refined} ▷ Return refined set of memorization neurons