NeRF: Collision handling in Instant Neural Graphics Primitives

Federico Montagna Mat. 240078
Luca Santagata Mat. 238842

federico.montagna, luca.santagata@studenti.unitn.it

Abstract

Computer graphics primitives are fundamentally represented by mathematical functions that parameterize appearance. Functions characterized by MLPs used as Neural Graphics Primitives [1] remain fast and compact while capturing high-frequency local detail. The crucial commonality of these new approaches is an encoding that maps neural network inputs to a higher-dimensional space, which is fundamental for extracting high approximation quality from compact models. This project explores collision handling within Instant Neural Graphics Primitives. The investigation encompasses a two-fold approach leveraging a Multi-resolution Encoding technique as the foundation. Collision minimization techniques are examined through a dual strategy: static collision resolution via a Hash Function and dynamic collision mitigation through General Gauge Neural Fields.

1 Multi-resolution Encoding

Multi-resolution is a technique for representing data at different levels of detail that is done by decomposing the data into a hierarchy of sub-components. In our case, as illustrated in Figure 1, for a given input coordinate $\mathbf{x}$, we find the surrounding voxels at L resolution levels and assign indices to their corners by hashing, depending on the method used, their integer coordinates. Then for all resulting corner indices, we look up the corresponding F-dimensional feature vectors from the hash tables θ_l and linearly interpolate them according to the relative position of $\mathbf{x}$ within the respective l-th voxel, producing the encoded MLP input $\mathbf{y} \in \mathbb{R}^{LF}$, which is evaluated last. To train the encoding loss, gradients are back-propagated through the MLP and then accumulated in the looked-up feature vectors. In Table 1, parameters used for the multi-resolution encoding are shown.

Parameter	Symbol	Value
Number of levels	L	4
Hash table size	T	16
# of feature dimensions	F	2^8
Coarsest resolution	N_{min}	8
Finest resolution	N_{max}	32

Table 1: Multi-resolution encoding parameters.

2 Methods

This section provides theoretical foundations for both employed methods.

2.1 Hash Function

We use a spatial hash function [3] of the form:

$$h(\mathbf{x}) = \left(\bigoplus_{i=1}^{d} (x_i \pi_i) \right) \bmod \mathrm{T}, \tag{1}$$

where $\oplus$ denotes the bit-wise XOR operation and π_i are unique large prime numbers ($\pi_1 = 1, \pi_2 = 2654435761$). The multi-resolution hash encoding has several advantages over traditional methods of representing neural network parameters. First, it is much more compact because the hash tables only store the indices of the feature vectors relevant to the input data. Secondly, it is also much faster to evaluate due to the hash tables that can be searched very quickly. Finally, it's more robust to noise because the hash tables can tolerate some errors in the input data.

2.2 General Neural Gauge Fields

It proposes a new method for representing neural fields using *Gauge Transformations*, powerful tools for simplifying and generalizing neural fields. The proposed method makes it possible to learn gauge transformations end-to-end with neural fields.

We assume there is a code book with N discrete vectors $[v_1, v_2, ..., v_N]$, and the discrete gauge transformation aims to transform a 2D point to a discrete code book index (i.e., gauge parameter). In line with the setting of Instant-NGP, we divide the 2D space into a grid whose size depends on the level. For each grid point x, we apply a gauge network M to predict its discrete distribution $P_x \in \mathbb{R}^N$ over N discrete vectors, followed by a Top-k operation to select the Top-k probable indices $i_1, i_2, ..., i_k$ as below:

$$\begin{aligned} P_x &= M(x) \\ p_1, p_2, ..., p_k, i_1, i_2, ..., i_k &= Topk(P_x) \quad P_x \in \mathbb{R}^N \end{aligned} \tag{2}$$

The produced indices, weighted by their probabilities and averaged, are used to look up corresponding vectors from the code book, and the features of other 2D points can be acquired by interpolating the nearby grid point vectors.

3 Implementations

In this section, the different parameters used in both architectures are presented. Some parameters were set to default values based on the instructions from the papers, while the remaining parameters (*) were obtained through a grid search conducted on a Tesla V100. Due to the considerable amount of possible configurations, not all of them were explored by the grid search.

3.1 Hash Function

3.1.1 Architecture

We use an MLP with two hidden layers comprised of 64 neurons and *ReLU* as an activation function and a linear output layer.

3.1.2 Parameters

Parameter	Name	Value
Features Embeddings learning rate	$encoding_{lr}$	$1e^{-4}$
(*) MLP learning rate	MLP_{lr}	$1e^{-3}$
MLP weight decay	MLP_{weight_decay}	$1e^{-6}$

3.1.3 Loss

We employed the *MSE* loss function by comparing the RGB values of pixels in the original image with those in the reconstructed one.

$$L_{MSE} = MSE(output_{image}, target_{image}) \quad (3)$$

3.2 General Neural Gauge Fields

3.2.1 Architecture

As evident from the image Figure 2, the architecture consists of two models: one dedicated to generating corner hashing values and the other focused on retrieving the RGB values of each pixel. The first model features three hidden layers with depths of 32, 64, and 128 neurons, all utilizing the *ReLU* activation function. The second model, on the other hand, mirrors the structure outlined in the earlier subsubsection 3.1.1

3.2.2 Parameters

Below are the architecture's parameters from which the highest $PSNR$ value was obtained.

Parameter	Name	Value
(*) Top k	$topk_k$	4
(*) Hash prob. distribution learning rate	HPD_{lr}	$1e^{-3}$
Hash probability distribution weight decay	HPD_{wd}	$1e^{-6}$
(*) MSE loss coeff.	λ_{mse}	1
(*) JS-KL Div loss coeff	λ_{js_kl}	1
(*) Collisions loss coeff.	λ_{coll}	$1e^{-3}$
KL Div loss coeff.	ϵ	1
(*) JS Div loss coeff.	γ	-2

3.2.3 Loss

In this method, the loss consists of three components, where the last two are defined level-wise. The first is the L_{MSE} defined in Equation 3, the second stated in Equation 4 considers the impact of collision numbers, which must be minimized, and imposes a more substantial penalty on collision counts at lower levels than on those at higher levels. The last one comes from [4], and as defined in Equation 5, prevents the gauge transformation from collapsing to a local region, imposing a uniform prior distribution $q(y)$ over the target gauge space.

$$L_{coll_l} = \min_{coll_l}\{\frac{\lambda_{coll}}{min\_collsion_l + \delta} \cdot coll_l \Big|_{\delta=1}\} \quad l \in 1, ..., L \quad (4)$$

$$L_{js\_kl_l} = \min_{p(y|x)}\{-(\gamma+\epsilon)\cdot \mathbb{E}_l[JS(p(y|x)p(x)||p(y)p(x))] + \\ +\epsilon \cdot \mathbb{E}_l[KL(p(y|x)||q(y))]\} \quad l \in 1, ..., L \quad (5)$$

The final loss is obtained as:

$$Loss = \lambda_{mse} \cdot L_{MSE} + \sum_{l=1}^{L}(\lambda_{coll} \cdot L_{coll_l} + \lambda_{js\_kl} \cdot L_{js\_kl_l}) \quad (6)$$

3.3 Metrics

The *Peak Signal-to-Noise Ratio (PSNR)* was used to evaluate the quality of the reconstructed image. It was calculated between the reference image and the one generated by the model.

4 Experiments and Results

In the original paper [4], the values of the ϵ and γ parameters were not specified by the authors, and in response to our email inquiry, we were informed that the model was under review. For this reason, we conducted the grid search with three possible scenarios affecting the components of the Equation 5:

- $\epsilon = 0$, corresponding to the sole use of the JS divergence.
- $\gamma + \epsilon = 0$: corresponding to the sole use of the KL divergence.
- $\gamma + \epsilon \neq 0$: corresponding to the use of both JS and KL divergence.

For each investigated scenario with the grid search, we reported the results obtained in **??** using the parameter configuration associated with the highest $PSNR$ value.

As can be observed, among the models based on General Neural Gauge Fields, the one incorporating the contributions of JS divergence and KL divergence achieves the best results. This is evidenced by the fewer collisions at each of the four levels and better $PSNR$ and MSE values. It is also possible to observe that the model based on Multi-resolution

Hash Encoding, whose run was executed with the same hyper-parameters as the ones obtained from the best $JS + KL$, achieves slightly better results. This is because, due to the high number of possible configurations and few computational resources and time, not all possibilities were explored by the grid search (only 1229 over 12000). Therefore, it is highly likely that we did not find the hyper-parameter configuration that would yield better results than the model based on Multi-resolution Hash Encoding. Nevertheless, the results obtained are still competitive.

In Figure 3, the generated images with different scenarios are displayed. Among those based on General Neural Gauge Fields, the model with JS+KL achieved the highest $PSNR$, even if the discrepancy with the other two is not significant enough to provide a visually evident distinction. On the other hand, the image reconstructed by the Multi-resolution Hash Encoding is evidently of higher quality than the others, though all remain satisfactorily similar to the original image.

From Figure 4, it is possible to observe how models based on General Neural Gauge Fields mitigate the number of collisions during the course of epochs, which does not occur in the Multiresolution Hash Encoding model, where the number of collisions for each level remains constant.

Figures 7, 8, 9, and 10 show the number of times an index from the code book is predicted by the hash functions. It should be noted that predictions of indices from the same input were counted only once due to overlaps in the multi-resolution grid. It is also possible to observe how the models based on General Neural Gauge Fields, with a uniform prior distribution imposed, prevent the collapsing of indices into local regions, as opposed to the model using Multi-resolution Hash Encoding, which does not assess any guess on the prior distribution.

The disparity in the magnitude of histogram counts between General Neural Gauge Fields and Multi-resolution Hash Encoding arises from the requirement that the sum of all counts must be equal to the number of unique vertices in the multi-resolution grid. Therefore, since the distribution of Multi-resolution Hash Encoding is more concentrated around particular values in the code book, the maximum count is lower than the General Neural Gauge Fields.

5 Conclusions

This project has demonstrated how hash functions based on General Neural Gauge Fields can be competitive with those based on Multi-resolution Hash Encoding [2]. The obtained outcomes encourage further research towards better hyper-parameters that may surpass those achieved by Multi-resolution Hash Encoding.

Future works

Considering that the General Neural Gauge Field is trained solely using the multi-resolution grid coordinates of an image, if hyper-parameters are found to outperform the results achieved by Multi-resolution Hash Encoding, it may be possible to utilize the pre-trained weights of the General Neural Gauge Field for different images with varying content. This can be achieved as long as the pictures share the exact dimensions and the same parameters expressed in Table 1 as the ones used during training.

In this case, it would be possible to maintain the same speed as Multi-resolution Hash Encoding but with fewer hash collisions, consequently leading to improved results.

References

[1] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[2] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *CoRR*, abs/2201.05989, 2022.

[3] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomeranets, and Markus Gross. Optimized spatial hashing for collision detection of deformable objects. *VMV'03: Proceedings of the Vision, Modeling, Visualization*, 3, 12 2003.

[4] Fangneng Zhan, Lingjie Liu, Adam Kortylewski, and Christian Theobalt. General neural gauge fields, 2023.

Level / Scenarios	# Collisions ↓ 0	1	2	3	Collisions Loss ↓ 0	1	2	3	Divergence Loss ↓ 0	1	2	3	MSE ↓	Loss ↓	PSNR ↑
KL	0	0	250	898	0	0	1.345	1.076	0.000 35	0.000 13	**0.00008**	0.000 14	0.009736	0.01285	20.111
JS	0	0	250	898	0	0	1.341	1.076	**0.00005**	0.000 27	0.000 10	**0.00006**	0.009592	0.01248	20.174
JS+KL	0	0	**234**	**878**	0	0	**1.258**	**1.054**	0.000 11	**0.00008**	0.000 11	0.000 13	**0.009236**	**0.01198**	**20.331**
(*) Hash	11	29	195	833	-	-	-	-	-	-	-	-	0.005063	0.005063	22.949
Min possible collisions	*0*	*0*	*185*	*833*											

Table 2: Results obtained in the different scenarios based on best grid-search parameters in terms of PSNR. (*) Same parameters of the ones obtained from best JS+KL.

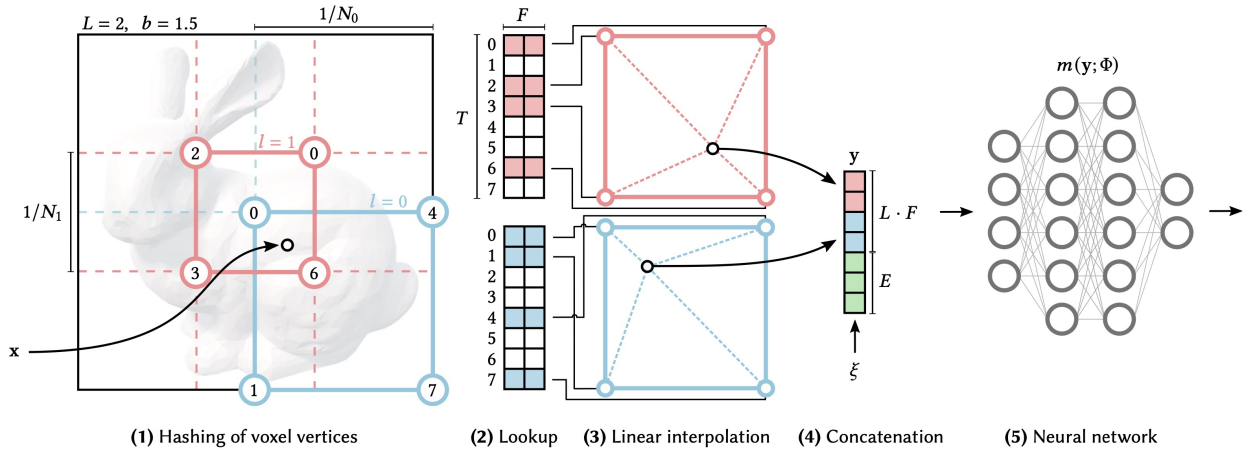


Figure 1: Multiresolution Hash Function Encoding

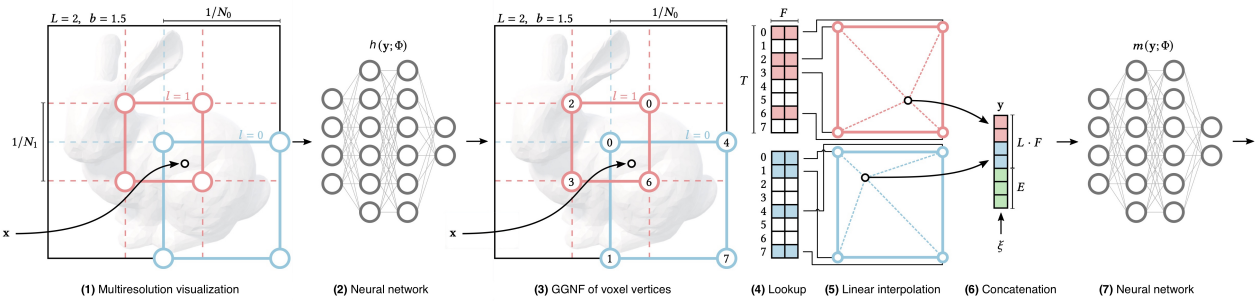


Figure 2: Multiresolution GNGF Encoding

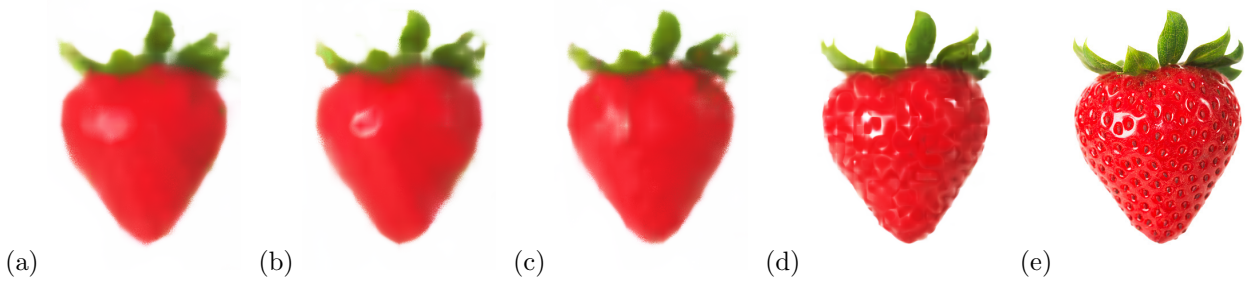


Figure 3: Reconstruced images with scenarios: (a) KL, (b) JS, (c) JS+KL, (d) Hash. (e) Original Image

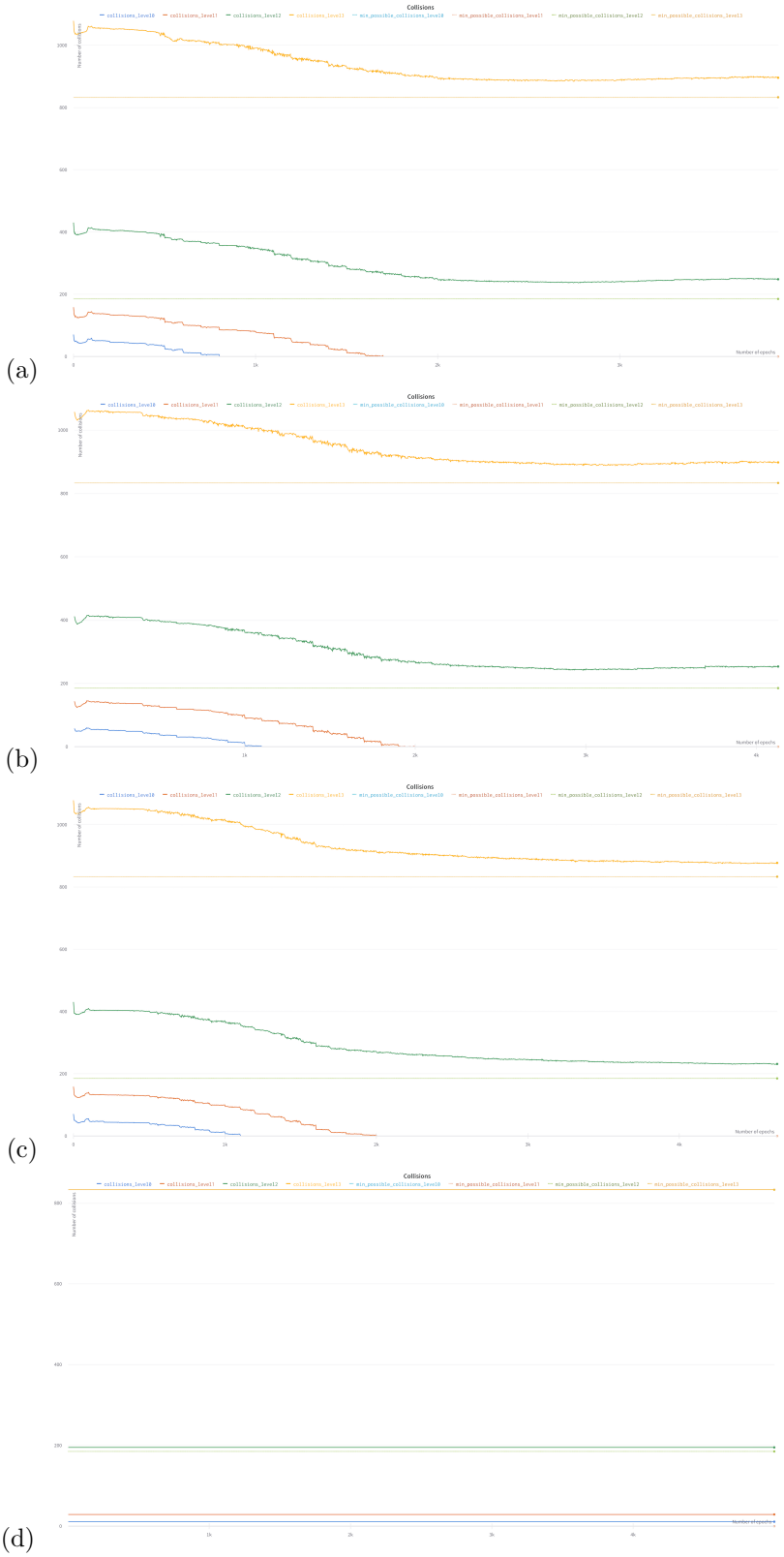
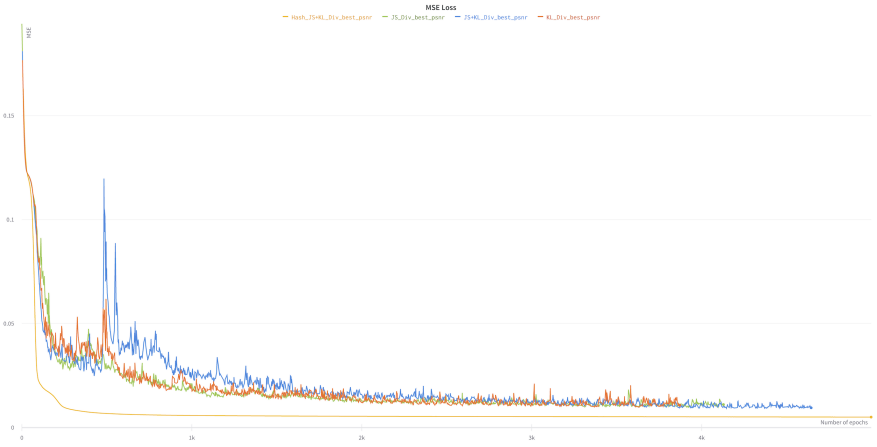
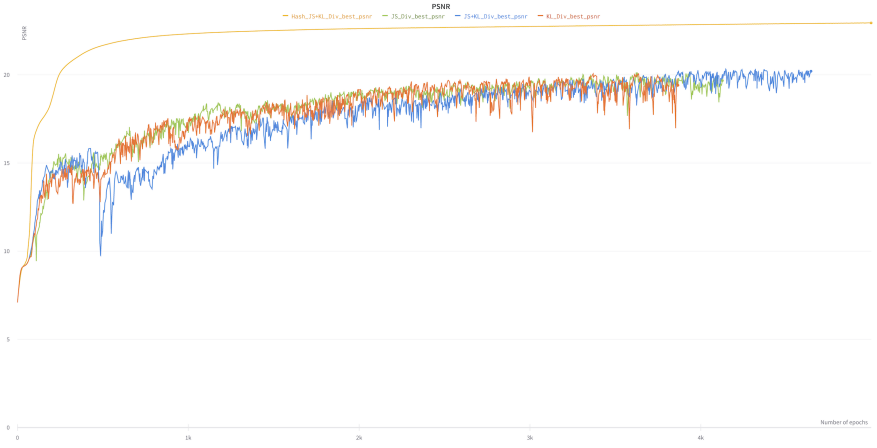


Figure 4: Number of collisions per level throughout the training with respect to the lowest achievable one in scenerarios: (a) KL, (b) JS, (c) JS+KL, (d) Hash.



(a)

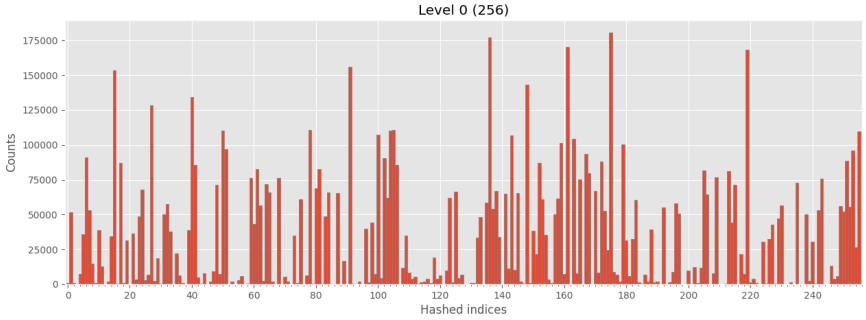
Figure 5: MSE Loss throughout the training.



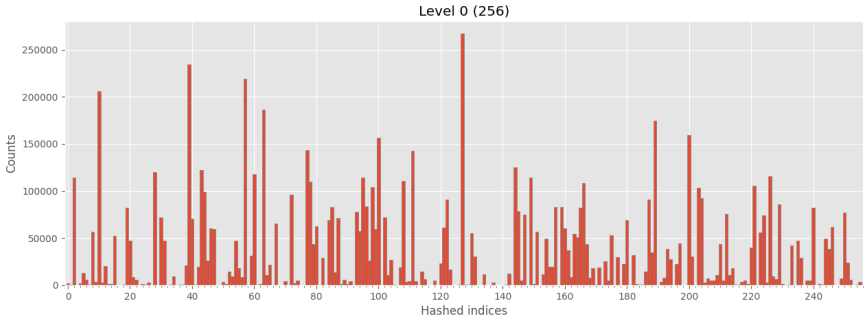
(a)

Figure 6: PSNR value throughout the training.

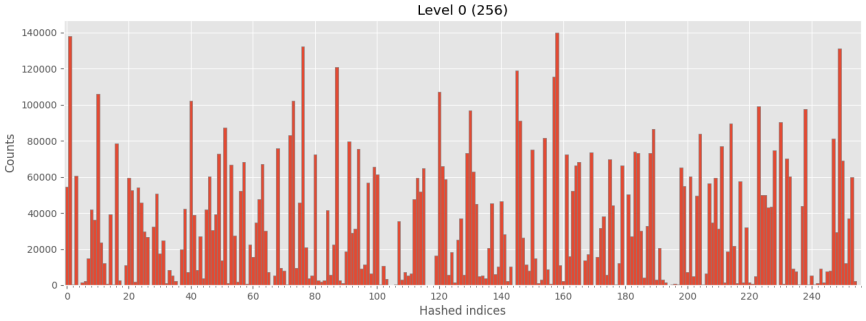
(a)



(b)



(c)



(d)

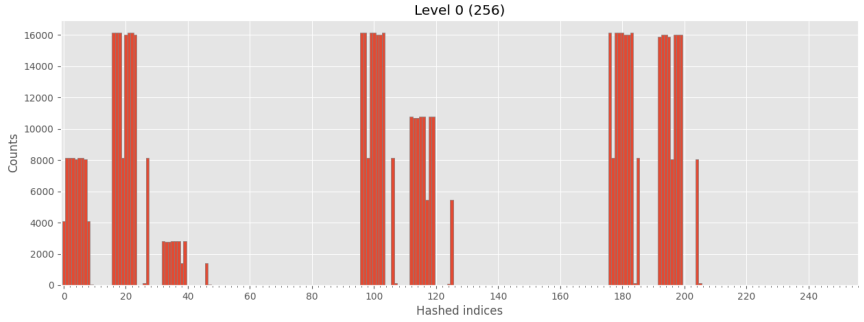
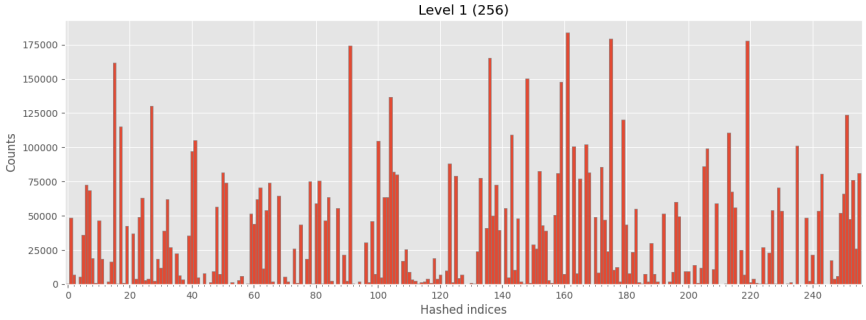
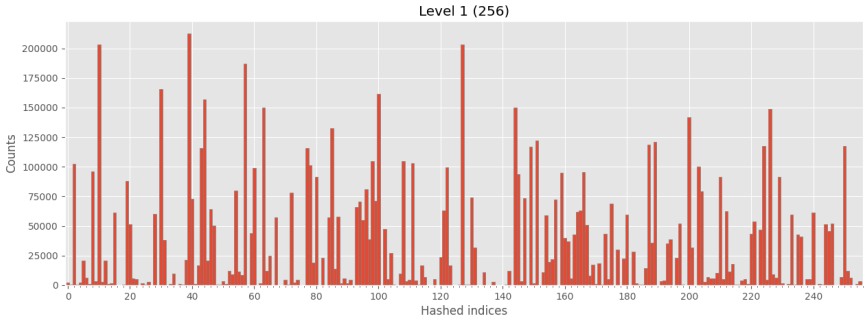


Figure 7: Hashed Indices distribution at level 0 with scenarios: (a) KL, (b) JS, (c) JS+KL, (d) Hash.

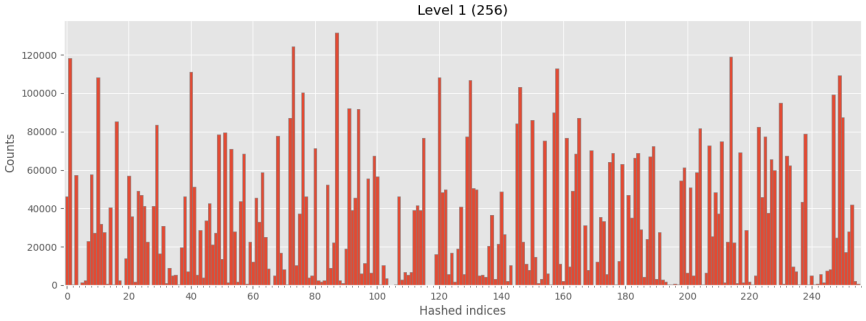
(a)



(b)



(c)



(d)

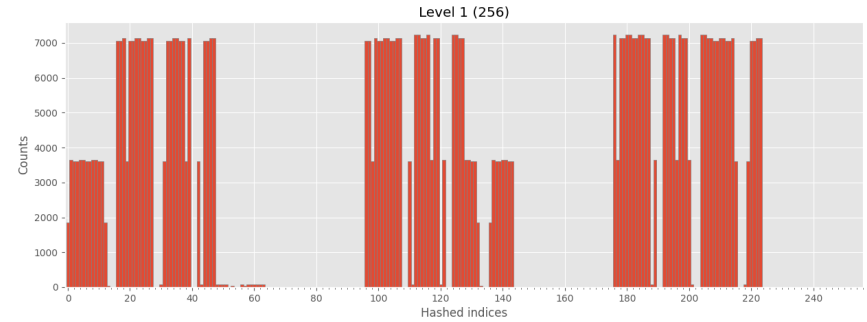
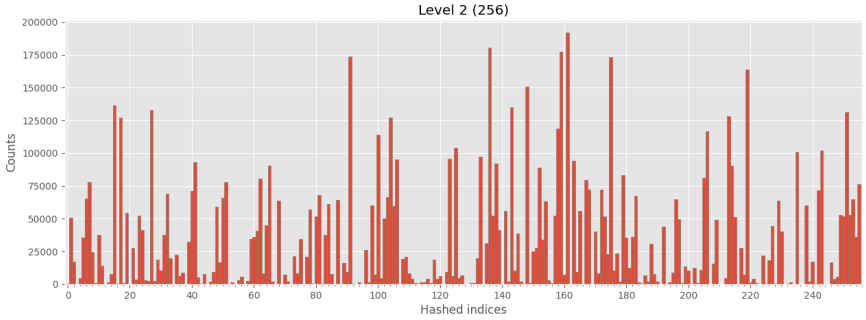
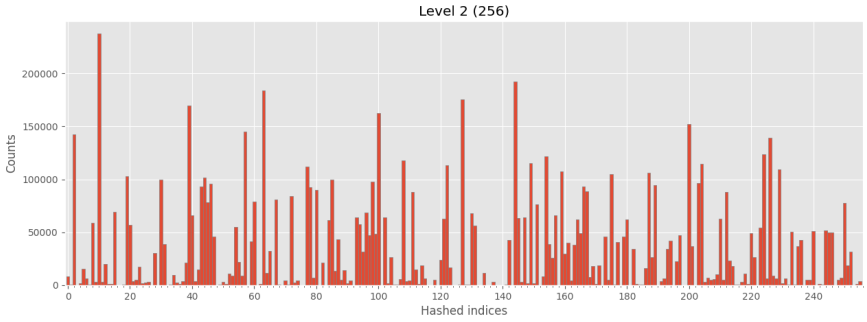


Figure 8: Hashed Indices distribution at level 1 with scenarios: (a) KL, (b) JS, (c) JS+KL, (d) Hash.

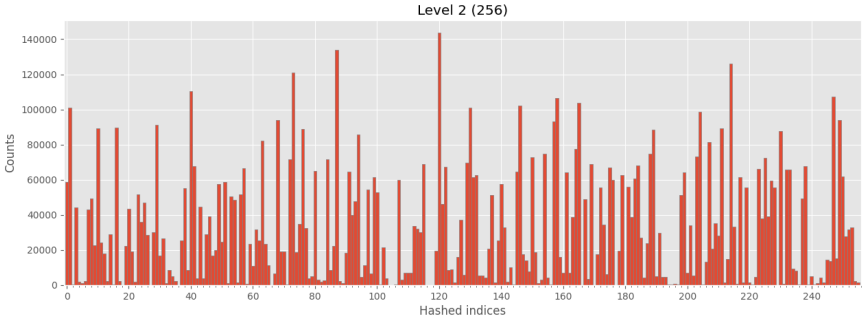
(a)



(b)



(c)



(d)

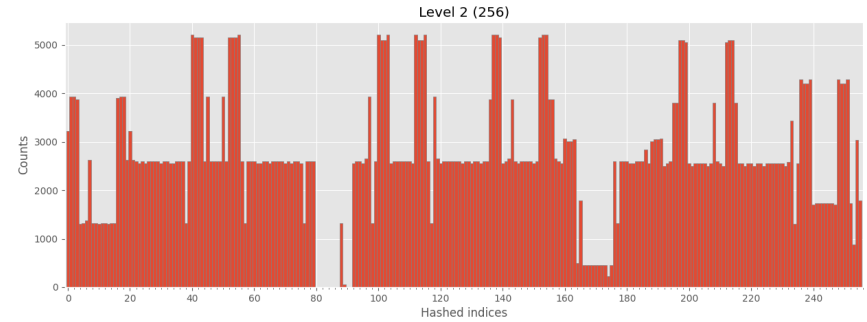
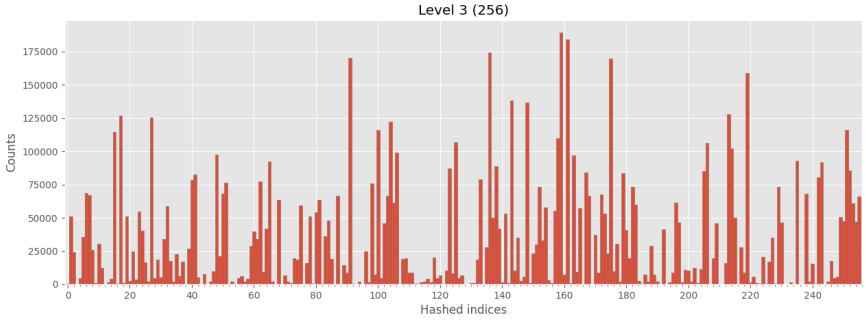
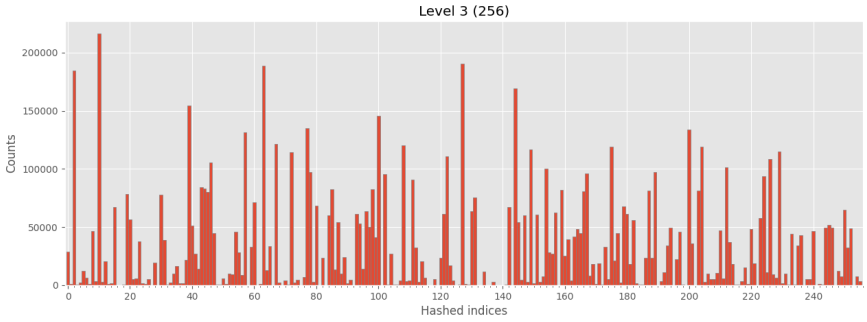


Figure 9: Hashed Indices distribution at level 2 with scenarios: (a) KL, (b) JS, (c) JS+KL, (d) Hash.

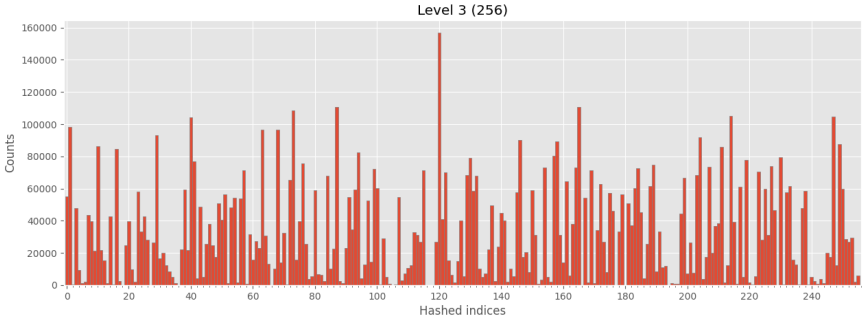
(a)



(b)



(c)



(d)

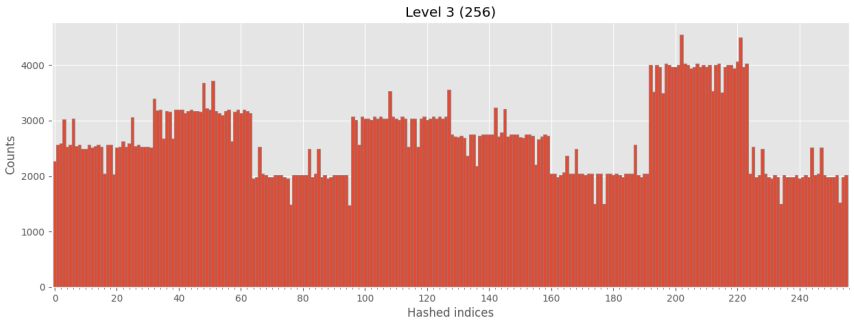


Figure 10: Hashed Indices distribution at level 3 with scenarios: (a) KL, (b) JS, (c) JS+KL, (d) Hash.