A Experiment Implementation Details

A.1 Uncertainty Quantification Model

This work employs a deep ensemble method to accurately estimate uncertainty across multiple datasets. Five ensemble models are trained for each dataset, each initialized differently. Except with PartImageNet, all experiments are carried out on an RTX2080Ti GPU using PyTorch, and standard data augmentation techniques such as horizontal flipping and random cropping are applied to all datasets. Below is a summary of the architectures and settings tailored to each dataset:

- C10 Dataset: Utilizing ResNet18 for feature extraction, we add a single fully-connected layer for classification. We use the SGD optimizer with an initial learning rate of 0.1, adjusted to 0.01, 0.001, and 0.0001 at epochs 30, 60, and 90, respectively, and a momentum of 0.9. Training spans a maximum of 100 epochs with a batch size of 128.
- C100 Dataset: ResNet152 serves as the primary feature extractor, coupled with a linear classification layer. The training regime employs SGD, starting with a learning rate of 0.1, which is reduced at epochs 60, 120, and 160 by 90%. The training is conducted with 200 epochs and a batch size of 64.
- SVHN Dataset: A standard convolutional neural network is applied. The network architecture is Conv2D-Relu-Conv2D-Relu-MaxPool2D-Dropout-Dense-Relu-Dropout-Dense-Softmax. Each convolutional layer is comprised of 32 convolutional filters, each with a kernel size of 4×4. Following this, we employ a max-pooling layer that utilizes a 2×2 kernel. The architecture also includes two dense layers, each containing 128 units, and implements a dropout strategy with a probability of 0.5 to prevent overfitting. This model, set with a batch size of 64, is trained for 50 epochs. The learning rate, initially at 0.1, is decreased by 90% at the 15th and 30th epochs. SGD with a momentum of 0.9 is used as the optimizer.
- PartImageNet: A100 GPU has been utilized to perform experiments. Here, DinoV2 serves as the backbone feature extractor of length 1536 and we add three fully connected layers for classification. The number of object classes is 11. We utilize an Adam optimizer with a learning rate of 0.001, with initial decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.99$. Training spans a maximum of 10 epochs with a batch size of 1024. No data augmentation technique has been applied here.

A.2 Attribution Methods

- Ours: There are two hyperparameters, λ and τ, for our method. λ is selected from $[1, 5]$, and τ is selected from $[0.05, 0.2]$. We use $\tau = 0.1$ for all datasets. For the C10, C100, and PartImageNet datasets, $\lambda = 3$ is applied, whereas $\lambda = 2$ is used for the SVHN dataset. It is worth noting that tuning the τ for each dataset can further improve the results. However, the current settings, $\tau = 0.1$, already outperform the baselines. We use the pre-trained ViT-H SAM model available at `https://github.com/facebookresearch/segment-anything`. To solve Eq. (5), we employ gradient descent to iteratively update the learnable parameters $\mathbf{p} = [p_1, p_2, \cdots, p_n]$ and ϕ. To ensure p_i (for $i = 1, 2, \ldots, n$) lies within $[0, 1]$, we parameterize p_i as $\sigma(v_i)$. Both v_i and ϕ are initialized at 0 and are iteratively updated. At each step of gradient descent, we draw one sample from a Logistic(0,1) distribution to calculate the loss function for backpropagation. We use an Adam optimizer with a learning rate of 0.01. We set the maximum number of iterations at 500 and apply an early stop criterion. After every 20 iterations, we assess whether the loss continues to decrease by sampling 20 values of t from Logistic(0,1) to calculate the expected loss. If the loss increases or the magnitude of decrease is less than 0.001, we halt the optimization and return the learned mask.

- Grad: We employ Torch.autograd for direct gradient computation between the input and the uncertainty score.
- SmoothGrad: We follow the settings illustrated in Appendix 1.2 of [39]. Essentially, we add random Gaussian noise, $\mathcal{N}(0, 0.1)$, to the original input to generate 100 noisy inputs. Then, we calculate the gradients from the uncertainty to the noisy inputs and average the gradients.
- IG: We adhere to the guidelines provided by the repository at `https://github.com/Featurespace/uncertainty-attribution` for the implementation of uncertainty-adapted versions of IG. For IG computations, we employ 100 path integrations. Additionally, a white image serves as the reference image for the IG process.
- KernelSHAP: This method performs pixel-level perturbations. In each perturbation, 10% of the pixels are replaced by their corresponding blurred counterparts. To blur the pixels, we apply the Gaussian blur function with a kernel size of 5 by 5 and a standard deviation of 2. For each perturbed input, we calculate the uncertainty and then learn a linear regression between the locations of the perturbation and the uncertainty. The weights reveal the uncertainty map. We utilize the implementation from `https://github.com/Featurespace/uncertainty-attribution`. The number of perturbations is set at 2000. This method is not applied to PartImageNet due to the large image size. Linear regression requires the training data to be at least larger than the number of variables to ensure a unique solution. However, performing a number of perturbations larger than the total number of pixels is time-consuming and requires a significant amount of memory.
- KernelSHAP + SAM: The implementation of this method is based on the implementation of KernelSHAP. We perturb directly the combination of segments indicated by SAM. In each perturbation, we perturb two segments and record the uncertainty. The number of perturbations is set at 1000.
- LIME: We utilize the implementation from `https://github.com/Featurespace/uncertainty-attribution` with their default hyperparameters for C10, C100, and SVHN datasets. It is worth noting that this implementation leverages quickshift segmentation for perturbation. The perturbations are performed using binomial masks. For pixel-level LIME, we replace the re-

sults of quickshift segmentation by segmenting uniformly for each pixel. For PartImageNet, we use a kernel size of 40, a maximum distance of 60 and a ratio of 0.2 for quickshift segmentation.

- Occlusion: We leverage a window that slides over the images. This window has a size of 5 by 5 with a stride of 2 for the C10, SVHN, and C100 datasets, and a size of 10 by 10 with a stride of 10 for the PartImageNet dataset. As the window slides, we record the change in uncertainty by blurring the pixels within the window. This change in uncertainty is then directly used to create the uncertainty map. Finally, we resize the uncertainty map to match the original size of the images.

A.3 Experiment Settings

Besides the experiment setting discussed in the main paper, we will further provide undiscussed important details.

- Detection of Problematic Regions: For PartImagetNet, it is worth noting that we have a preprocessing step to resize the original images to size 224 by 224. Since the resize function is not differentiable for gradient-based methods, we compute the gradients and also perform perturbations on the resized image. While we synthetically generate the 100 by 100 problematic areas in the original images, the problematic areas will also be resized as shown in the examples of Fig. 1. Over all the baselines for this experiment, KernelSHAP + SAM, LIME, and Ours leverage the segmentation results, where each segment will have one score. In this experiment, we use total uncertainty map since the problematic regions both behave like the data noise (aleatoric) and input anomaly (epistemic), making the image different from the training data. If the segmentation results are not available, our method still outperforms others at the pixel-level as shown in Appendix C.1.
- Faithfulness Tests: It is important to note that all methods are implemented at the pixel level. Two methods are implemented differently from the baselines used in Table 1: We utilize the pixel-version of LIME and our method, which essentially replaces the segmentation results by randomly segmenting each pixel as a separate segment. We exclude the comparison of KernelSHAP + SAM and only report the results for pixel-level KernelSHAP. For refining the problematic regions using gradient descent, we first precompute the binary mask $\mathbf{M}(\mathbf{x})$ that highlights the pixels with the highest UA score. We set $\eta = 0.2$ and use the Adam optimizer to perform gradient descent. The maximum number of iterations is set to 500, and an early-stop criterion is applied when the uncertainty does not decrease for the updated image. We also use the total uncertainty for the faithfulness test. This is because we believe that an input with high uncertainty can be attributed to multiple sources (both aleatoric and epistemic). Explaining total uncertainty considers all sources and has a more reliable evaluation of the methods. For the blurring test, we strictly follow the settings in [39]. Please refer to [39] for a detailed discussion.

B Additional Visualizations

In this section, we first show additional figures for detecting problematic regions. In Figure 3, we observe how our method can find out the synthetically created problematic regions compared to other baselines on the CIFAR100 dataset. In addition, we show in Figure 4 how our method can find out the problematic regions of real images although we do not have ground-truth problematic areas. The first four columns pertain to the CIFAR100 dataset, highlighting cases where the real object is indistinguishable, even to humans, due to low resolution. It is often observed that in low-resolution images, what may appear problematic to humans typically involves the usual objects, while the background, being irrelevant to classification, is expected to have a low attribution score. As demonstrated in Figure 4, our method effectively highlights regions associated with the foreground object, rather than inaccurately focusing on the background. Finally, we shed light on the fifth column of Figure 4 where a centaur (human with horse body) is classified as 'Quadruped' and the real problematic region is identified as the human head. Our method correctly identifies the mismatch between the body and the head, indicating both as problematic areas. In summary, our method, not being limited to synthetic perturbations, can generate human-understandable uncertainty maps.

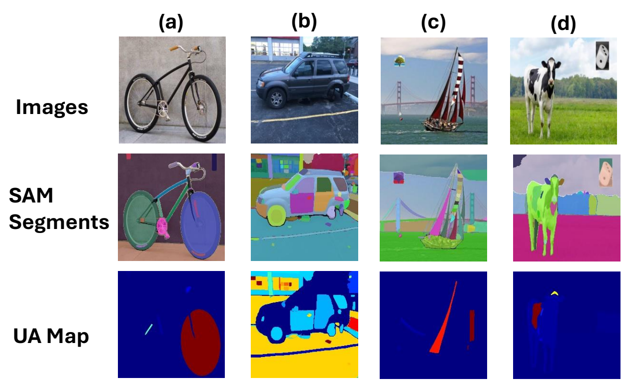

Fig. 2: Examples of failure cases.

Some failure cases are shown in Figure 2. Our method sometimes fails when SAM is unable to locate the problematic regions, as demonstrated in Figure 2 (a) and (b). For human explainees, the problematic region of this bicycle is the missing seat. However, SAM cannot segment the seat location, resulting in an incorrect UA map. Similarly, SAM cannot segment the missing wheels, leading to the failure of our uncertainty maps. Moreover, our methods may also fail when the images have low uncertainty. For example, in Figure 2 (c) and (d), the perturbation on the background only slightly increases uncertainty, making

Table 6: IoU ↑ and ADA ↑ for detection of problematic regions. All the baselines are implemented at the pixel-level without the need for segmentation results.

Method	C10		C100		SVHN	
	IoU	ADA	IoU	ADA	IoU	ADA
Grad	0.109	0.054	0.282	0.250	0.288	0.180
SmoothGrad	0.297	0.226	0.322	0.132	0.312	0.184
IG	0.208	0.152	0.224	0.250	0.310	0.256
UA-Backprop	0.271	0.178	0.360	0.250	0.336	0.258
KernelSHAP	0.324	0.306	0.286	0.250	**0.382**	0.340
Occlusion	0.218	0.160	0.245	0.250	0.350	0.302
Ours (Pixel-based)	**0.380**	**0.365**	**0.560**	**0.750**	0.370	**0.365**

it difficult for the optimization process to locate these uncertainty contributors. Consequently, the optimization may get stuck in local minima in these cases.

C Additional Analysis and Results

C.1 Pixel-level Method Comparison for Detecting Problematic Regions

In Section 4.1, three methods are implemented using segmentation results: KernelSHAP + SAM, LIME, and Ours. We further compare the performance of our method at the pixel level to that of other pixel-level methods that do not require segmentation results. The results, shown in Table 6, indicate that our method outperforms the other baselines. Furthermore, our pixel-level method performs even better than LIME, which is implemented using quickshift segmentation. The results of LIME are available in Table 1.

C.2 Contributions of Each Component

In this section, we further demonstrate how each component contributes to improving the performance of the faithfulness test. Since the faithfulness test is applied at the pixel level, we do not include SAM segmentation in our method. Table 7 shows the performance improvement attributable to two components: Learnable Perturbation (LP) and Gumbel-sigmoid reparameterization. We can see that each individual component has its own contribution to the performance. Gumbel-sigmoid reparameterization seems to have a slightly more contribution to the performance.

C.3 Hyperparameter Sensitivity

In Table 8, we further explore the sensitivity of hyperparameters for the C100 and SVHN datasets in detecting problematic regions in semi-synthetic data. The conclusion remains the same: our algorithm is not sensitive to hyperparameters

Table 7: Contributions of each component for faithfulness test on C100 and SVHN datasets.

Method	Gradient Refinement				Blurring Test			
	C100		SVHN		C100		SVHN	
	0.5%	1%	0.5%	1%	2%	5%	2%	5%
Eq. (3)	0.742	0.791	0.982	0.985	0.355	0.434	0.608	0.713
Eq. (3) + LP	0.756	0.799	0.983	0.988	0.364	0.458	0.745	0.812
Eq. (3) + LP + Gumbel	**0.766**	**0.808**	**0.988**	**0.992**	**0.386**	**0.487**	**0.869**	**0.939**

within certain ranges. It is also worth noting that the performance may potentially depend on the random sampling used in the Gumbel-sigmoid function.

Table 8: Hyperparameter sensitivity for detecting problematic regions on C100 and SVHN datasets. '*' indicates the setting used in the main paper.

Method	C100	
	IoU	ADA
$\lambda = 3, \tau = 0.1$*	0.750	**0.750**
$\lambda = 3, \tau = 0.07$	0.733	0.734
$\lambda = 3, \tau = 0.15$	**0.772**	0.744
$\lambda = 2.5, \tau = 0.1$	0.726	0.708
$\lambda = 2, \tau = 0.1$	0.736	0.704

Method	SVHN	
	IoU	ADA
$\lambda = 2, \tau = 0.1$*	0.829	0.830
$\lambda = 2, \tau = 0.08$	0.816	0.828
$\lambda = 2, \tau = 0.12$	0.793	0.800
$\lambda = 1.8, \tau = 0.1$	0.815	0.812
$\lambda = 2.2, \tau = 0.1$	**0.831**	**0.840**

C.4 Convergence Analysis

It is worth noting that we observe hundreds of iterations are often sufficient for the algorithm to converge. Hence, we have set the maximum number of iterations to 500 and also implemented an early stop criterion. Additionally, it is worth mentioning that adjusting the learning rate can often facilitate faster convergence. For our method, we believe that SAM-guided learning could potentially expedite convergence since fewer parameters need to be learned. To prove this, we aim to empirically study the number of iterations required for the method to converge. We exclude Gumbel-sigmoid reparameterization from the algorithm and solely parameterize the mask using a general sigmoid function. This approach is taken because Gumbel-sigmoid reparameterization introduces randomness to the algorithm, making convergence also dependent on random sampling. Without Gumbel-sigmoid reparameterization, we examine the convergence speed of applying a SAM-guided mask compared to learning a pixel-wise mask directly. The same stopping criteria are applied, such that the algorithm will stop if the loss increases or if the magnitude of the decrease is less than 0.001. The number of iterations required for the algorithm to converge is shown

Table 9: Average number of iterations needed for convergence of our algorithm to attribution one image.

Method	C10	C100	SVHN
	# of iterations	# of iterations	# of iterations
Ours(Pixel-based)	194.3	180.4	141.0
Ours(SAM-based)	76.0	77.4	76.8

in Table 9, where the application of SAM-guided mask parameterization can speed up the process by more than 50%.

With Gumbel-sigmoid reparameterization, our full algorithm illustrated in Eq. (5) may take a little longer time for convergence. Empirically, we stop optimization if the decrease in the expected objective function in Eq. (5) is below 0.001, which is detailed in Appendix A.2. Our algorithm requires about 72, 94, 84, and 181 gradient steps on average for C10, C100, SVHN, and PartImageNet, respectively. High-resolution PartImageNet requires more complex models and may need more steps, but they are still fewer than the thousands needed by many perturbation-based methods.

D Limitations, Societal Impact, and Future Work

Limitations The first limitation of our method is that it requires two hyperparameters. Although we demonstrate the general range for these hyperparameters and show that they are not sensitive within certain ranges, tuning based on specific datasets is still necessary. The optimal hyperparameters can vary across different datasets and uncertainty quantification methods. For instance, hyperparameters suitable for the ensemble method as the UQ approach may not directly apply to those using another UQ method, given the potential for different scales of uncertainty estimation. Another limitation is our method's lower inference efficiency than gradient-based methods, though it has comparably complexity to perturbation-based methods. It is also worth noting that there are some general limitations of post-hoc explanation methods such as disagreements with the same input and model and susceptibility to adversarial attacks.

Future Work In our future work, we plan to develop a hyperparameter learning algorithm to automatically adjust the hyperparameters for new tasks, eliminating the need for manual tuning. To enhance efficiency, we could simultaneously optimize all testing samples, allowing parallel learning of the uncertainty maps. Furthermore, by modeling the uncertainty map as a function of the input, parameterized by a neural network, we can expedite the generation of uncertainty maps for new images once the mapping neural network is trained. These techniques hold the potential to significantly accelerate the process. Finally, applying our method to generate uncertainty attribution maps for real-world safety-critical applications, such as medical diagnoses, presents an intriguing avenue for future research.

Societal Impact: Our model improves both model explainability and trustworthiness. UA maps can also help humans enhance the model and address high-uncertainty predictions with mitigation strategies.

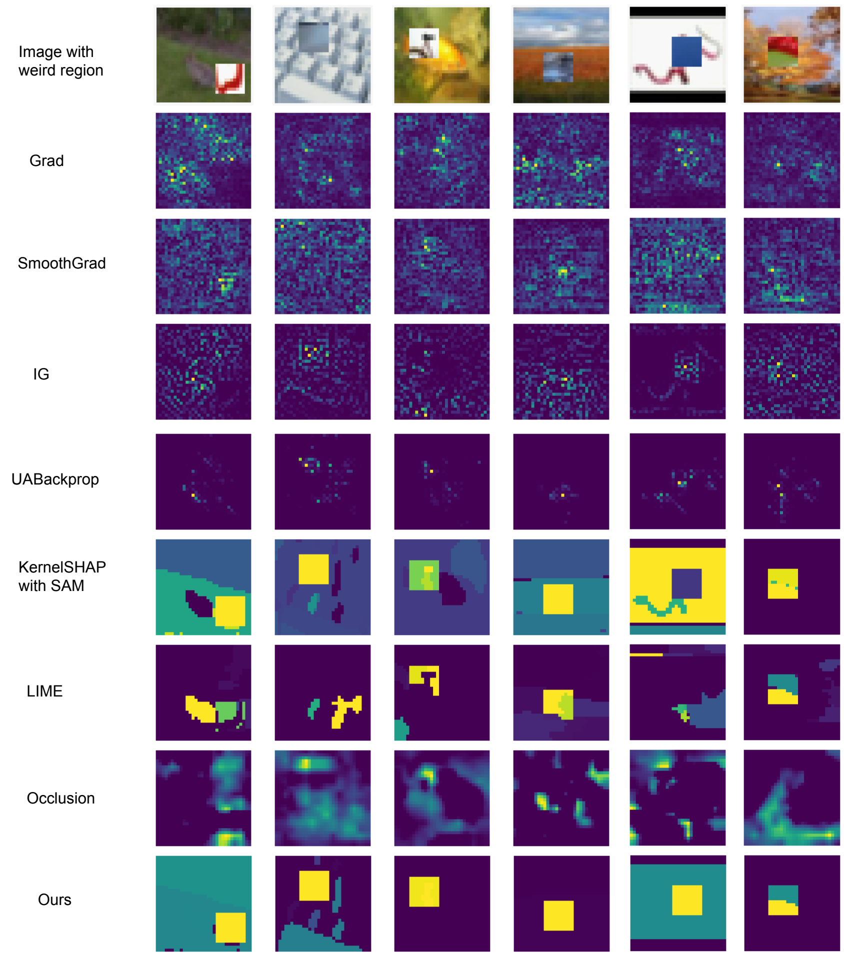

Fig. 3: Examples of uncertainty maps for identifying problematic regions on CIFAR100. The problematic regions are created synthetically. Brighter regions indicate a higher contribution to the uncertainty.

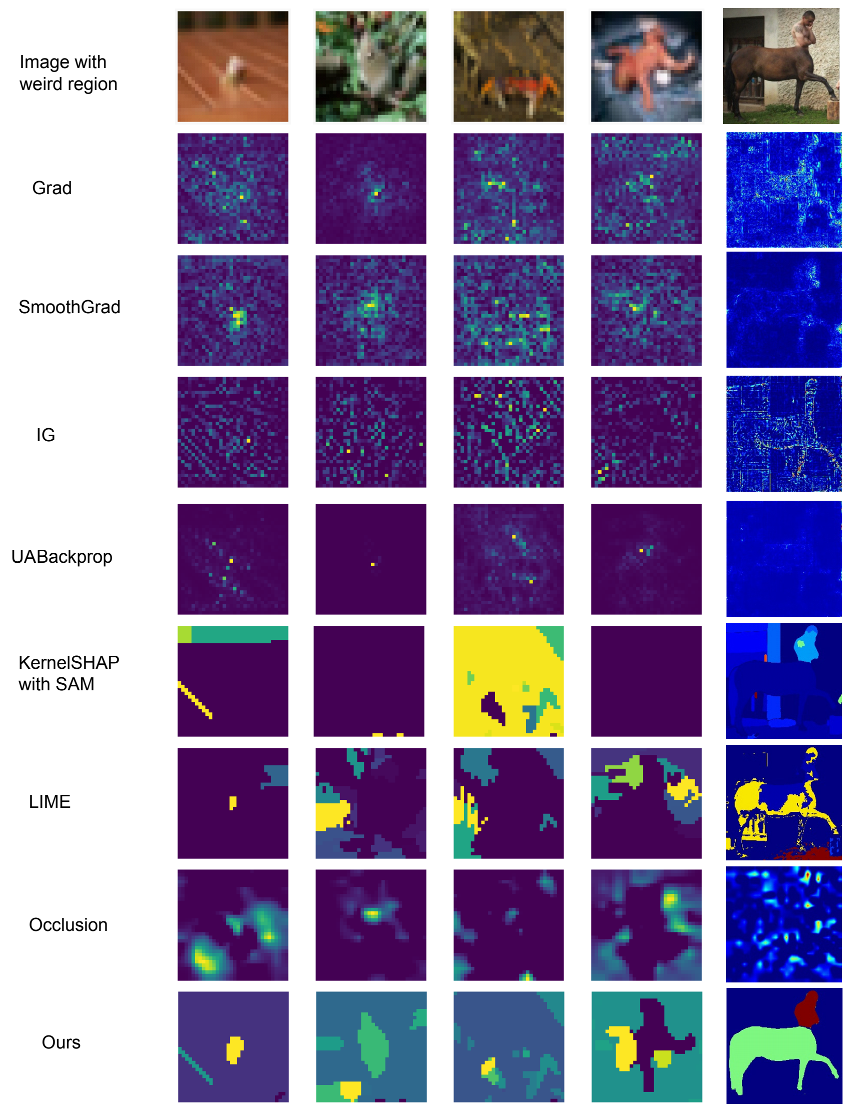

Fig. 4: Examples of uncertainty maps for identifying problematic regions on CIFAR100 and PartImageNet. The first four columns correspond to CIFAR100 and the last row correspond to PartImageNet. The problematic images in these figures are not synthetic. Brighter regions indicate a higher contribution to the uncertainty.