Stony Brook University

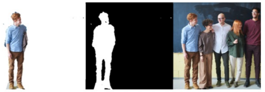
# **Peekaboo**: Text to Image Diffusion Models are Zero-Shot Segmentors

Ryan Burgert, Kanchana Ranasinghe, Xiang Li, Michael Ryoo

← ryanndagreat.github.io/peekaboo

JUNE 18-22, 2023
CVPR
VANCOUVER, CANADA

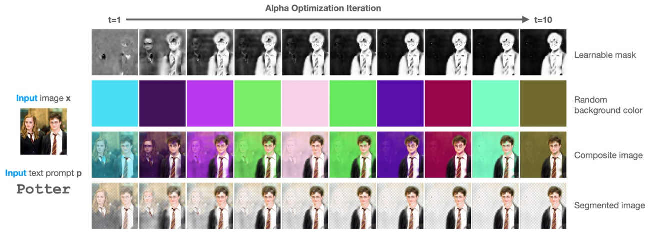
## Goal: open-vocabulary zero-shot segmentation, without any new training. Given an image and a prompt, Peekaboo aims to segment that region.



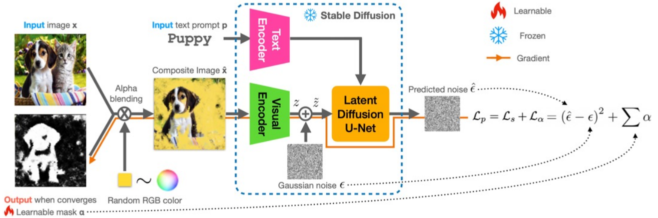
man with blonde hair in blue shirt and brown pants

bald man wearing glasses with white shirt and black pants

## Peekaboo segments images by iteratively optimizing an alpha mask

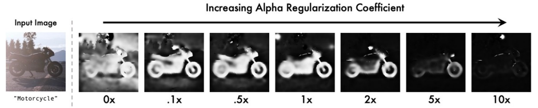


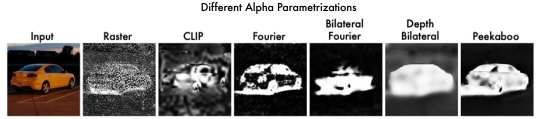

## Here's the full inference pipeline! There is no training pipeline. Peekaboo uses off-the-shelf stable diffusion with no further training.



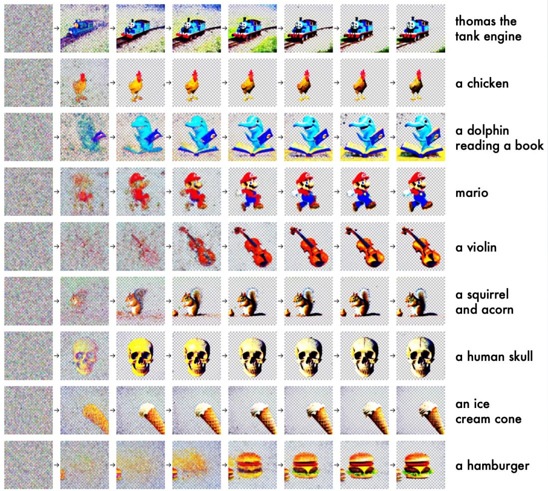

$$\mathcal{L}_p = \mathcal{L}_s + \mathcal{L}_\alpha = (\hat{\epsilon} - \epsilon)^2 + \sum \alpha$$

## Alpha Regularization and Parametrizations




## Peekaboo can also be used to generate images with transparency (An RGB image and mask are jointly optimized with Peekaboo loss)

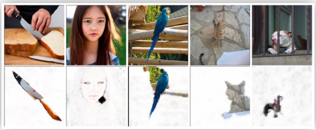



## Simple Peekaboo Pseudocode

Peekaboo Pseudocode

```
def segment_image_via_peekaboo(image, prompt):
    alpha = torch.rand(image.height, image.width)
    optim = torch.optim.SGD(alpha)
    for _ in range(num_iterations):
        loss=peekaboo_loss(image, prompt,
      bilateral_blur(alpha,image))
        loss.backward() ; optim.step()
    return bilateral_blur(alpha, image)

def peekaboo_loss(image, prompt, alpha):
    #Core of our paper. Blends image with random
    #color and returns loss guiding alpha mask.
    background = torch.rand(3) # random RGB color
    composite_img = torch.lerp(background, image, alpha)
    loss = alpha_regularization_loss = alpha.sum()
    loss += score_distillation_loss(image, prompt)
    return loss

def score_distillation_loss(image , prompt):
    # Same loss proposed in DreamFusion
    timestep = random_int(0, _diffusion_step)
    noise = diffusion.get_noise(timestep)
    noisy_image = diffusion.add_noise(image, noise)

    with torch.no_grad():
        pred_noise = diffusion.unet(noisy_image, prompt)

    return (noise - pred_noise).abs().sum()

def text_to_rgba_image(prompt):
  # Generates an RGB image and alpha mask from prompt
    alpha = torch.rand(image_height, image_width)
    image = torch.rand(3, image_height, image_width)
    optim = torch.optim.SGD([alpha, image])
    for _ in range(num_iterations):
        loss = peekaboo_loss(image, prompt, alpha)
        loss.backward() ; optim.step()
    return image, alpha
```

## Failure Cases: Peekaboo can generate hallucinations



## Quantitative Segmentation Results

| Type | Method | Prec@0.2 | Prec@0.4 | Prec@0.6 | Prec@0.8 | mIoU |
|---|---|---|---|---|---|---|
| Baselines | Random | .141 | .022 | .003 | .000 | .102 |
| | GroupVIT [11] | .212 | .075 | .020 | .002 | .112 |
| | LSeg [7] | .512 | .212 | .051 | .008 | .235 |
| **Peekaboo** Variants (ours) | Depth Bilateral | .359 | .135 | .037 | .003 | .204 |
| | **RGB Bilateral** | .318 | .099 | .018 | .002 | .163 |

RefCOCO Dataset

| Type | Method | Prec@0.2 | Prec@0.4 | Prec@0.6 | Prec@0.8 | mIoU |
|---|---|---|---|---|---|---|
| Baselines | Random | .670 | .198 | .032 | .012 | .281 |
| | Clippy [12] | .757 | .459 | .263 | .049 | .539 |
| | GroupViT [11] | .862 | .778 | .602 | .205 | .578 |
| | LSeg [7] | .952 | .897 | .758 | .311 | .678 |
| **Peekaboo** Variants (ours) | Raster | .756 | .323 | .103 | .012 | .340 |
| | CLIP | .918 | .488 | .093 | .023 | .430 |
| | Fourier | .862 | .598 | .231 | .084 | .454 |
| | Bilateral Fourier | .845 | .608 | .281 | .123 | .470 |
| | Depth Bilateral | .929 | .707 | .455 | .187 | .551 |
| | **RGB Bilateral** | .892 | .709 | .331 | .130 | .520 |

Cropped Pascal-VOC Dataset

## Qualitative Segmentation Results