In [32]:
```
!pip install tenseal pytorch-msssim scikit-image sewar pytorch_msssim sewar tikzplo
!pip install --upgrade matplotlib tikzplotlib
```

```
Requirement already satisfied: tenseal in /opt/conda/lib/python3.10/site-packages
(0.3.14)
Requirement already satisfied: pytorch-msssim in /opt/conda/lib/python3.10/site-pack
ages (1.0.0)
Requirement already satisfied: scikit-image in /opt/conda/lib/python3.10/site-packag
es (0.22.0)
Requirement already satisfied: sewar in /opt/conda/lib/python3.10/site-packages (0.
4.6)
Requirement already satisfied: tikzplotlib in /opt/conda/lib/python3.10/site-package
s (0.10.1)
Requirement already satisfied: torch in /opt/conda/lib/python3.10/site-packages (fro
m pytorch-msssim) (2.1.2)
Requirement already satisfied: numpy>=1.22 in /opt/conda/lib/python3.10/site-package
s (from scikit-image) (1.26.4)
Requirement already satisfied: scipy>=1.8 in /opt/conda/lib/python3.10/site-packages
(from scikit-image) (1.11.4)
Requirement already satisfied: networkx>=2.8 in /opt/conda/lib/python3.10/site-packa
ges (from scikit-image) (3.2.1)
Requirement already satisfied: pillow>=9.0.1 in /opt/conda/lib/python3.10/site-packa
ges (from scikit-image) (9.5.0)
Requirement already satisfied: imageio>=2.27 in /opt/conda/lib/python3.10/site-packa
ges (from scikit-image) (2.33.1)
Requirement already satisfied: tifffile>=2022.8.12 in /opt/conda/lib/python3.10/site
-packages (from scikit-image) (2023.12.9)
Requirement already satisfied: packaging>=21 in /opt/conda/lib/python3.10/site-packa
ges (from scikit-image) (21.3)
Requirement already satisfied: lazy_loader>=0.3 in /opt/conda/lib/python3.10/site-pa
ckages (from scikit-image) (0.3)
Requirement already satisfied: matplotlib>=1.4.0 in /opt/conda/lib/python3.10/site-p
ackages (from tikzplotlib) (3.7.5)
Requirement already satisfied: webcolors in /opt/conda/lib/python3.10/site-packages
(from tikzplotlib) (1.13)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.10/site-pa
ckages (from matplotlib>=1.4.0->tikzplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-packag
es (from matplotlib>=1.4.0->tikzplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.10/site-p
ackages (from matplotlib>=1.4.0->tikzplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.10/site-p
ackages (from matplotlib>=1.4.0->tikzplotlib) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.10/site-pa
ckages (from matplotlib>=1.4.0->tikzplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.10/sit
e-packages (from matplotlib>=1.4.0->tikzplotlib) (2.9.0.post0)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-packages
(from torch->pytorch-msssim) (3.13.1)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.10/site-p
ackages (from torch->pytorch-msssim) (4.9.0)
Requirement already satisfied: sympy in /opt/conda/lib/python3.10/site-packages (fro
m torch->pytorch-msssim) (1.13.0)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.10/site-packages (fr
om torch->pytorch-msssim) (3.1.2)
Requirement already satisfied: fsspec in /opt/conda/lib/python3.10/site-packages (fr
om torch->pytorch-msssim) (2024.5.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-packages
(from python-dateutil>=2.7->matplotlib>=1.4.0->tikzplotlib) (1.16.0)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.10/site-pac
kages (from jinja2->torch->pytorch-msssim) (2.1.3)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/conda/lib/python3.10/site-
packages (from sympy->torch->pytorch-msssim) (1.3.0)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.10/site-packages
(3.7.5)
Collecting matplotlib
  Downloading matplotlib-3.9.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (11 kB)
Requirement already satisfied: tikzplotlib in /opt/conda/lib/python3.10/site-package
s (0.10.1)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.10/site-pa
ckages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-packag
es (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.10/site-p
ackages (from matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.10/site-p
ackages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.10/site-package
s (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.10/site-pac
kages (from matplotlib) (21.3)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.10/site-packages
(from matplotlib) (9.5.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.10/site-pa
ckages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.10/sit
e-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: webcolors in /opt/conda/lib/python3.10/site-packages
(from tikzplotlib) (1.13)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-packages
(from python-dateutil>=2.7->matplotlib) (1.16.0)
Downloading matplotlib-3.9.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.
whl (8.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.3/8.3 MB 87.5 MB/s eta 0:00:00:00:010
0:01
Installing collected packages: matplotlib
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.7.5
    Uninstalling matplotlib-3.7.5:
      Successfully uninstalled matplotlib-3.7.5
ERROR: pip's dependency resolver does not currently take into account all the packag
es that are installed. This behaviour is the source of the following dependency conf
licts.
beatrix-jupyterlab 2023.128.151533 requires jupyterlab~=3.6.0, but you have jupyterl
ab 4.2.3 which is incompatible.
pointpats 2.5.0 requires shapely>=2, but you have shapely 1.8.5.post1 which is incom
patible.
spopt 0.6.1 requires shapely>=2.0.1, but you have shapely 1.8.5.post1 which is incom
patible.
ydata-profiling 4.6.4 requires matplotlib<3.9,>=3.2, but you have matplotlib 3.9.1 w
hich is incompatible.
ydata-profiling 4.6.4 requires numpy<1.26,>=1.16.0, but you have numpy 1.26.4 which
is incompatible.
Successfully installed matplotlib-3.8.4
```

In [7]:
```
%matplotlib inline

import numpy as np
from pprint import pprint

from PIL import Image
import matplotlib.pyplot as plt
from matplotlib.offsetbox import OffsetImage, AnnotationBbox

import tenseal as ts
import random
random.seed(72)

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import grad
import torchvision
import torchvision.models as models
import torchvision.transforms as transforms
from torchvision import models, datasets, transforms
from torch.optim import LBFGS
from pytorch_msssim import ms_ssim
torch.manual_seed(50)
from sewar.full_ref import vifp  # Correct import for vifp from sewar

from skimage.metrics import structural_similarity as ssim

import tikzplotlib

print(torch.__version__, torchvision.__version__)
```

```
2.1.2 0.16.2
```

In [8]:
```
history_raw = []
msssim_values_raw = []
uqi_values_raw = []

history_20_rand = []
msssim_values_20_rand = []
uqi_values_20_rand = []

history_40_rand = []
msssim_values_40_rand = []
uqi_values_40_rand = []

history_50_rand = []
msssim_values_50_rand = []
uqi_values_50_rand = []

history_10_sens = []
msssim_values_10_sens = []
uqi_values_10_sens = []

history_5_sens = []
```

```
msssim_values_5_sens = []
uqi_values_5_sens = []
```

In [9]:
```
class GlobalContext():
  def __init__(self,
               poly_modulus_degree=2**13,
               coeff_mod_bit_sizes=[31, 26, 26, 26, 26, 26, 26, 31],
               global_scale= 2 ** 26):

    self.context = ts.context(ts.SCHEME_TYPE.CKKS, poly_modulus_degree, coeff_mod_b
    self.context.generate_galois_keys()
    self.context.global_scale = global_scale

global_context = GlobalContext()
```

In [10]:
```
dst = datasets.CIFAR100("/kaggle/working/torch", download=True)
class_labels = dst.classes
aquarium_fish_index = class_labels.index('rabbit')
print("Index of 'aquarium_fish' class in CIFAR-100 dataset:", aquarium_fish_index)

tp = transforms.Compose([
    transforms.Resize(32),
    transforms.CenterCrop(32),
    transforms.ToTensor()
])
tt = transforms.ToPILImage()

device = "cpu"
if torch.cuda.is_available():
    device = "cuda"
print("Running on %s" % device)

def label_to_onehot(target, num_classes=100):
    target = torch.unsqueeze(target, 1)
    onehot_target = torch.zeros(target.size(0), num_classes, device=target.device)
    onehot_target.scatter_(1, target, 1)
    return onehot_target

def cross_entropy_for_onehot(pred, target):
    return torch.mean(torch.sum(- target * F.log_softmax(pred, dim=-1), 1))
```

```
Files already downloaded and verified
Index of 'aquarium_fish' class in CIFAR-100 dataset: 65
Running on cuda
```

In [11]:
```
## for raw model
def weights_init(m):
    if hasattr(m, "weight"):
        m.weight.data.uniform_(-0.5, 0.5)
    if hasattr(m, "bias"):
        m.bias.data.uniform_(-0.5, 0.5)

# # 50 % random
# def weights_init(m):
#     if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
#         if hasattr(m, "weight"):
```

```
#             if random.random() < 0.5:
#                 weight = m.weight.data.view(-1).tolist()
#                 encrypted_weight = ts.ckks_vector(global_context.context, weight)
#                 decrypted_weight = torch.tensor(encrypted_weight.decrypt(), devic
#                 m.weight.data = decrypted_weight.view_as(m.weight.data)
#             else:
#                 m.weight.data.uniform_(-0.5, 0.5)
#         if hasattr(m, "bias"):
#             m.bias.data.uniform_(-0.5, 0.5)


class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        act = nn.Sigmoid
        self.body = nn.Sequential(
            nn.Conv2d(3, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
        )
        self.fc = nn.Sequential(
            nn.Linear(768, 100)
        )

    def forward(self, x):
        out = self.body(x)
        out = out.view(out.size(0), -1)
        # print(out.size())
        out = self.fc(out)
        return out


net = LeNet().to(device)

net.apply(weights_init)
criterion = cross_entropy_for_onehot

# # sensitive
# # Define the weights initialization function
# def weights_init(m):
#     if hasattr(m, "weight"):
#         m.weight.data.uniform_(-0.5, 0.5)
#     if hasattr(m, "bias"):
#         m.bias.data.uniform_(-0.5, 0.5)

# # Define the LeNet model
# class LeNet(nn.Module):
#     def __init__(self):
#         super(LeNet, self).__init__()
#         act = nn.Sigmoid
#         self.body = nn.Sequential(
```

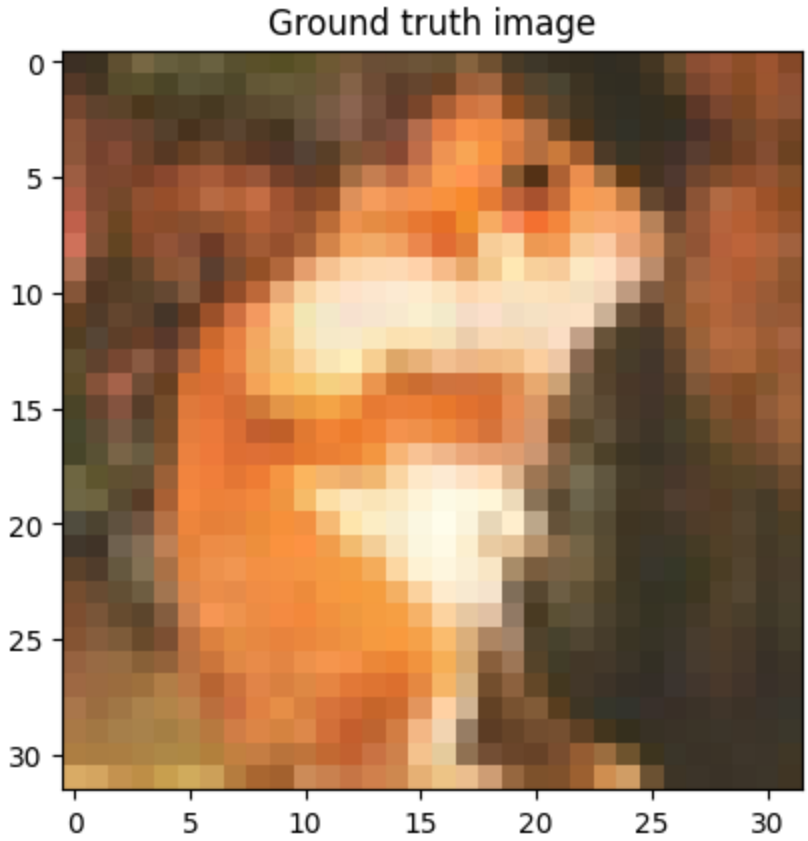
```
#             nn.Conv2d(3, 12, kernel_size=5, padding=5//2, stride=2),
#             act(),
#             nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=2),
#             act(),
#             nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
#             act(),
#             nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
#             act(),
#         )
#         self.fc = nn.Sequential(
#             nn.Linear(768, 100)
#         )

#     def forward(self, x):
#         out = self.body(x)
#         out = out.view(out.size(0), -1)
#         out = self.fc(out)
#         return out

# # Initialize the model and weights
# device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# net = LeNet().to(device)
# net.apply(weights_init)
# criterion = nn.CrossEntropyLoss()  # Assuming cross_entropy_for_onehot is a custo

# # Load CIFAR-100 dataset
# tp_ = transforms.Compose([
#     transforms.Resize(32),
#     transforms.CenterCrop(32),
#     transforms.ToTensor()
# ])
# dst_ = datasets.CIFAR100("/kaggle/working/torch", download=True, transform=tp_)
# data_loader = torch.utils.data.DataLoader(dst_, batch_size=64, shuffle=True)

# # Function to calculate parameter sensitivities
# def calculate_sensitivity(model, dataloader):
#     model.train()
#     criterion = nn.CrossEntropyLoss()
#     gradient_sums = {}

#     for name, param in model.named_parameters():
#         if 'bias' not in name:
#             gradient_sums[name] = 0.0
#             param.requires_grad_(True)

#     for inputs, labels in dataloader:
#         inputs, labels = inputs.to(device), labels.to(device)
#         outputs = model(inputs)
#         loss = criterion(outputs, labels)
#         model.zero_grad()
#         loss.backward()

#         for name, parameter in model.named_parameters():
#             if 'bias' not in name and parameter.requires_grad:
#                 grads = parameter.grad.abs().sum().item()
#                 gradient_sums[name] += grads
```

```
#     return gradient_sums

# # Encrypt the most sensitive model parameters using TenSEAL
# def encrypt_parameters(model, sensitivities, encryption_percentage=0.1):

#     # Sort parameters by sensitivity
#     sorted_sensitivities = sorted(sensitivities.items(), key=lambda item: item[1]
#     num_params_to_encrypt = int(len(sorted_sensitivities) * encryption_percentage

#     encrypted_parameters = {}
#     for i in range(num_params_to_encrypt):
#         name, _ = sorted_sensitivities[i]
#         parameter = dict(model.named_parameters())[name].data.cpu().numpy()
#         encrypted_param = ts.ckks_vector(global_context.context, parameter.flatte
#         encrypted_parameters[name] = encrypted_param

#     return model, encrypted_parameters

# # Calculate sensitivities
# sensitivities = calculate_sensitivity(net, data_loader)

# # Encrypt 10% of the most sensitive parameters
# # encrypted_net, encrypted_parameters = encrypt_parameters(net, sensitivities, en
# encrypted_net, encrypted_parameters = encrypt_parameters(net, sensitivities, encr

# print("Encryption completed.")
```

In [12]:

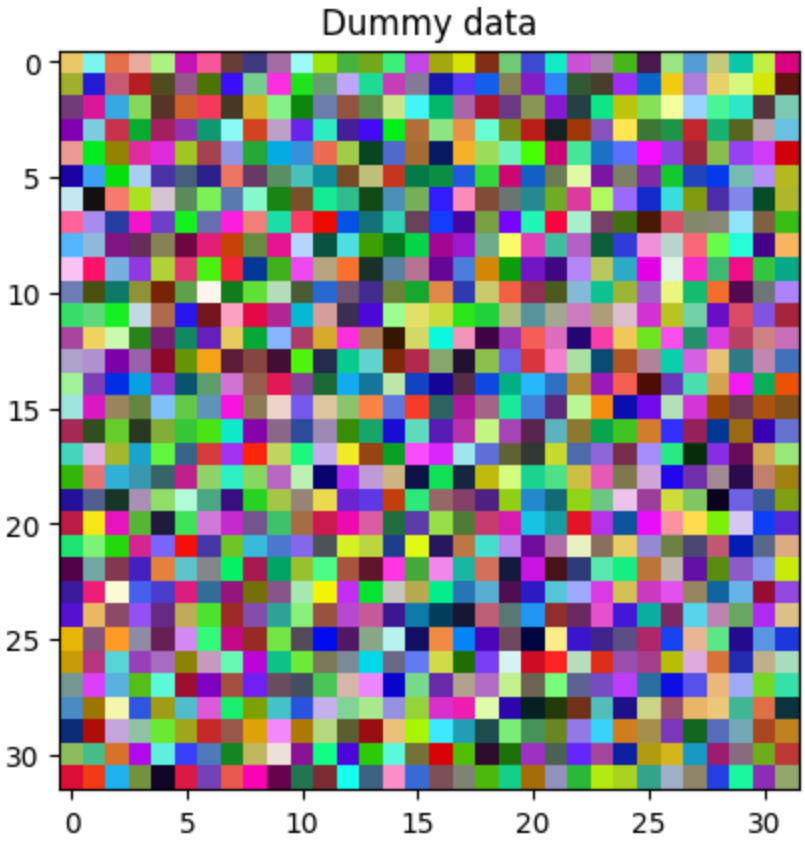
```
######### honest partipant #########
img_index = 60
gt_data = tp(dst[img_index][0]).to(device)
gt_data = gt_data.view(1, *gt_data.size())
gt_label = torch.Tensor([dst[img_index][1]]).long().to(device)
gt_label = gt_label.view(1, )
gt_onehot_label = label_to_onehot(gt_label, num_classes=100)

plt.imshow(tt(gt_data[0].cpu()))
plt.title("Ground truth image")
print("GT label is %d." % gt_label.item(), "\nOnehot label is %d." % torch.argmax(g

# compute original gradient
out = net(gt_data)
y = criterion(out, gt_onehot_label)
dy_dx = torch.autograd.grad(y, net.parameters())


# share the gradients with other clients
original_dy_dx = list((_.detach().clone() for _ in dy_dx))
```

```
GT label is 36.
Onehot label is 36.
```



In [13]:
```
torch.manual_seed(42)
# generate dummy data and label
dummy_data = torch.randn(gt_data.size()).to(device).requires_grad_(True)
dummy_label = torch.randn(gt_onehot_label.size()).to(device).requires_grad_(True)

plt.imshow(tt(dummy_data[0].cpu()))
plt.title("Dummy data")
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())
```

```
Dummy label is 92.
```



In [14]:
```
history_raw = []
msssim_values_raw = []
uqi_values_raw = []

resize_transform = transforms.Resize((161, 161))

optimizer = LBFGS([dummy_data, dummy_label])

# Specific iterations to capture and plot
plot_iterations = [0, 50, 100, 150, 200, 250]

for iters in range(300):
    def closure():
        optimizer.zero_grad()

        pred = net(dummy_data)
        dummy_onehot_label = F.softmax(dummy_label, dim=-1)
        dummy_loss = criterion(pred, dummy_onehot_label)
        dummy_dy_dx = torch.autograd.grad(dummy_loss, net.parameters(), create_grap

        grad_diff = 0
        grad_count = 0
        for gx, gy in zip(dummy_dy_dx, original_dy_dx):
            grad_diff += ((gx - gy) ** 2).sum()
            grad_count += gx.nelement()
        grad_diff.backward()

        return grad_diff
```

```
    optimizer.step(closure)
    if iters % 10 == 0:
        current_loss = closure()
        print(iters, "%.4f" % current_loss.item())

        dummy_data_resized = resize_transform(dummy_data)
        gt_data_resized = resize_transform(gt_data)

        # Calculate MS-SSIM
        msssim_value = ms_ssim(dummy_data_resized, gt_data_resized, data_range=1.0,
        msssim_values_raw.append(msssim_value.item())

        dummy_data_np = dummy_data_resized.detach().permute(0, 2, 3, 1).squeeze().c
        gt_data_np = gt_data_resized.permute(0, 2, 3, 1).squeeze().cpu().numpy()
        uqi_value = vifp(dummy_data_np, gt_data_np)

        uqi_values_raw.append(uqi_value)


    if iters in plot_iterations:
        history_raw.append(dummy_data[0].cpu())
```

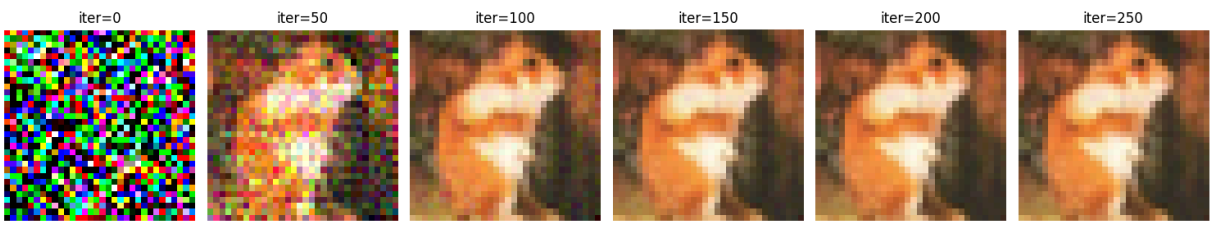
```
0 20.6096
```

```
/opt/conda/lib/python3.10/site-packages/torchvision/transforms/functional.py:1603: U
serWarning: The default value of the antialias parameter of all the resizing transfo
rms (Resize(), RandomResizedCrop(), etc.) will change from None to True in v0.17, in
order to be consistent across the PIL and Tensor backends. To suppress this warning,
directly pass antialias=True (recommended, future default), antialias=None (current
default, which means False for Tensors and True for PIL), or antialias=False (only w
orks on Tensors - PIL will still use antialiasing). This also applies if you are usi
ng the inference transforms from the models weights: update the call to weights.tran
sforms(antialias=True).
  warnings.warn(
```

```
10 0.4442
20 0.0698
30 0.0134
40 0.0032
50 0.0009
60 0.0003
70 0.0001
80 0.0001
90 0.0000
100 0.0000
110 0.0000
120 0.0000
130 0.0000
140 0.0000
150 0.0000
160 0.0000
170 0.0000
180 0.0000
190 0.0000
200 0.0000
210 0.0000
220 0.0000
230 0.0000
240 0.0000
250 0.0000
260 0.0000
270 0.0000
280 0.0000
290 0.0000
```

In [15]:
```python
# Specific iterations to capture and plot
plot_iterations = [0, 50, 100, 150, 200, 250]

plt.figure(figsize=(15, 5))

# Plot images in a 1x6 grid
for i, iter_num in enumerate(plot_iterations):
    plt.subplot(1, 6, i + 1)
    image = history_raw[i].detach().cpu().permute(1, 2, 0).numpy()  # Detach the te
    plt.imshow(image)
    plt.title("iter=%d" % iter_num)
    plt.axis('off')

# Print dummy label
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())

# Adjust layout to make space for titles
plt.tight_layout(rect=[0, 0, 1, 0.9])

# Add a title for the whole plot
plt.suptitle('Reconstructed Images at Different Iterations for Plaintext', y=1.05)

plt.show()
```

```
Dummy label is 36.
```



20 random

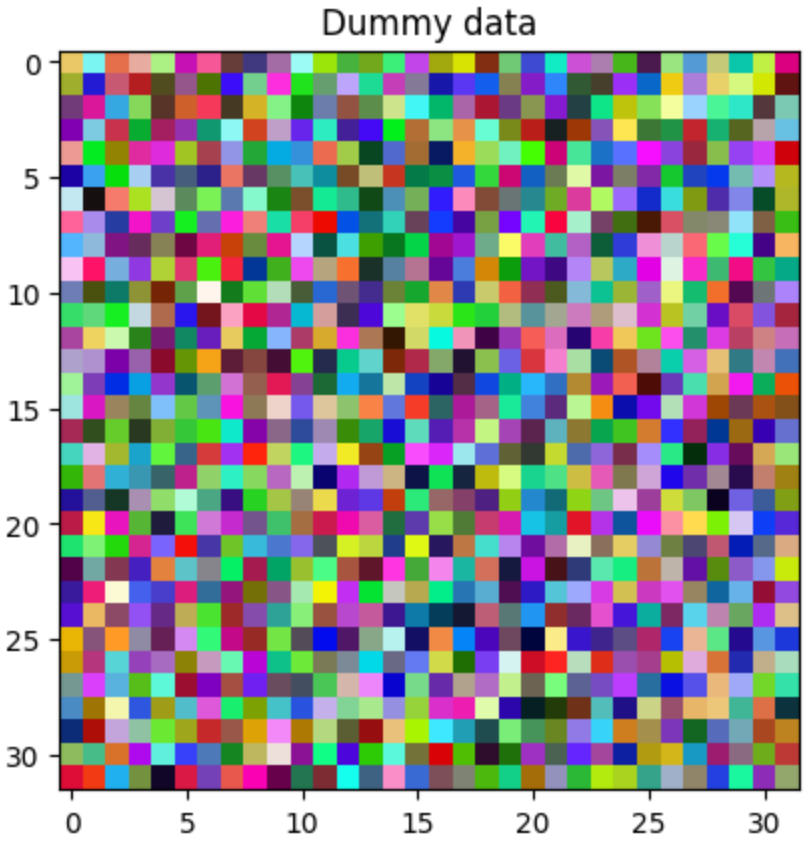
In [16]:
```
history_20_rand = []
msssim_values_20_rand = []
uqi_values_20_rand = []
random.seed(71)

def weights_init(m):
    if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
        if hasattr(m, "weight"):
            if random.random() < 0.2:
                weight = m.weight.data.view(-1).tolist()
                encrypted_weight = ts.ckks_vector(global_context.context, weight)
                decrypted_weight = torch.tensor(encrypted_weight.decrypt(), device=
                m.weight.data = decrypted_weight.view_as(m.weight.data)
            else:
                m.weight.data.uniform_(-0.5, 0.5)
        if hasattr(m, "bias"):
            m.bias.data.uniform_(-0.5, 0.5)


class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        act = nn.Sigmoid
        self.body = nn.Sequential(
            nn.Conv2d(3, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
        )
        self.fc = nn.Sequential(
            nn.Linear(768, 100)
        )

    def forward(self, x):
        out = self.body(x)
        out = out.view(out.size(0), -1)
        # print(out.size())
```

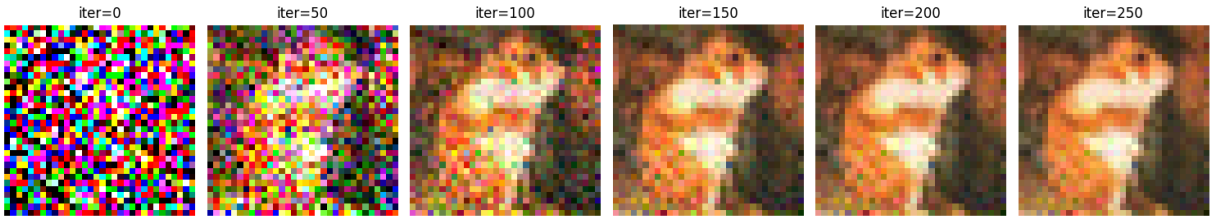
```
        out = self.fc(out)
        return out


net = LeNet().to(device)

net.apply(weights_init)
criterion = cross_entropy_for_onehot



img_index = 60
gt_data = tp(dst[img_index][0]).to(device)
gt_data = gt_data.view(1, *gt_data.size())
gt_label = torch.Tensor([dst[img_index][1]]).long().to(device)
gt_label = gt_label.view(1, )
gt_onehot_label = label_to_onehot(gt_label, num_classes=100)

plt.imshow(tt(gt_data[0].cpu()))
plt.title("Ground truth image")
print("GT label is %d." % gt_label.item(), "\nOnehot label is %d." % torch.argmax(g

# compute original gradient
out = net(gt_data)
y = criterion(out, gt_onehot_label)
dy_dx = torch.autograd.grad(y, net.parameters())


# share the gradients with other clients
original_dy_dx = list((_.detach().clone() for _ in dy_dx))



random.seed(72)
torch.manual_seed(42)
# generate dummy data and label
dummy_data = torch.randn(gt_data.size()).to(device).requires_grad_(True)
dummy_label = torch.randn(gt_onehot_label.size()).to(device).requires_grad_(True)

plt.imshow(tt(dummy_data[0].cpu()))
plt.title("Dummy data")
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())



resize_transform = transforms.Resize((161, 161))

optimizer = LBFGS([dummy_data, dummy_label])

# Specific iterations to capture and plot
plot_iterations = [0, 50, 100, 150, 200, 250]

for iters in range(300):
    def closure():
        optimizer.zero_grad()
```

```
        pred = net(dummy_data)
        dummy_onehot_label = F.softmax(dummy_label, dim=-1)
        dummy_loss = criterion(pred, dummy_onehot_label)
        dummy_dy_dx = torch.autograd.grad(dummy_loss, net.parameters(), create_grap

        grad_diff = 0
        grad_count = 0
        for gx, gy in zip(dummy_dy_dx, original_dy_dx):
            grad_diff += ((gx - gy) ** 2).sum()
            grad_count += gx.nelement()
        grad_diff.backward()

        return grad_diff

    optimizer.step(closure)
    if iters % 10 == 0:
        current_loss = closure()
        print(iters, "%.4f" % current_loss.item())

        dummy_data_resized = resize_transform(dummy_data)
        gt_data_resized = resize_transform(gt_data)

        # Calculate MS-SSIM
        msssim_value = ms_ssim(dummy_data_resized, gt_data_resized, data_range=1.0,
        msssim_values_20_rand.append(msssim_value.item())

                # Calculate UQI
        dummy_data_np = dummy_data_resized.detach().permute(0, 2, 3, 1).squeeze().c
        gt_data_np = gt_data_resized.permute(0, 2, 3, 1).squeeze().cpu().numpy()
        uqi_value = vifp(dummy_data_np, gt_data_np)
        uqi_values_20_rand.append(uqi_value)


    if iters in plot_iterations:
        history_20_rand.append(dummy_data[0].cpu())



plt.figure(figsize=(15, 5))

# Plot images in a 1x6 grid
for i, iter_num in enumerate(plot_iterations):
    plt.subplot(1, 6, i + 1)
    image = history_20_rand[i].detach().cpu().permute(1, 2, 0).numpy()  # Detach th
    plt.imshow(image)
    plt.title("iter=%d" % iter_num)
    plt.axis('off')

# Print dummy label
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())

# Adjust layout to make space for titles
plt.tight_layout(rect=[0, 0, 1, 0.9])

# Add a title for the whole plot
plt.suptitle('Reconstructed Images at Different Iterations for 20 % Random', y=1.05
```

```
plt.show()
```

```
GT label is 36.
Onehot label is 36.
Dummy label is 92.
0 4.8881
10 0.1086
20 0.0242
30 0.0080
40 0.0031
50 0.0012
60 0.0005
70 0.0003
80 0.0001
90 0.0001
100 0.0000
110 0.0000
120 0.0000
130 0.0000
140 0.0000
150 0.0000
160 0.0000
170 0.0000
180 0.0000
190 0.0000
200 0.0000
210 0.0000
220 0.0000
230 0.0000
240 0.0000
250 0.0000
260 0.0000
270 0.0000
280 0.0000
290 0.0000
Dummy label is 36.
```





40 % Random

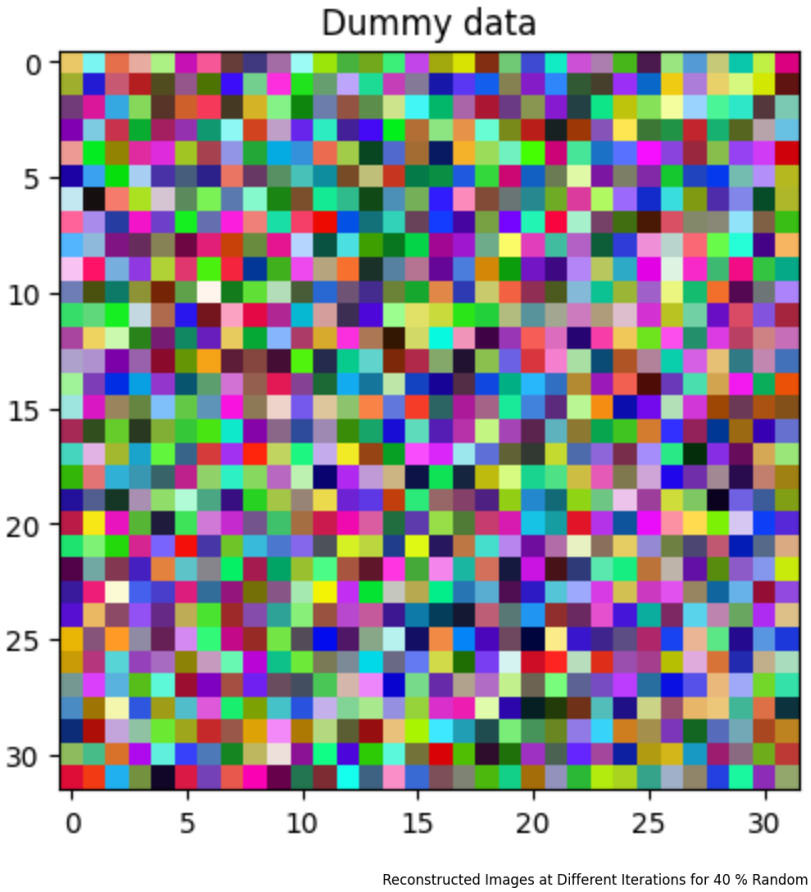
In [17]:
```
history_40_rand = []
msssim_values_40_rand = []
uqi_values_40_rand = []

random.seed(35)

def weights_init(m):
    if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
        if hasattr(m, "weight"):
            if random.random() < 0.4:
                weight = m.weight.data.view(-1).tolist()
                encrypted_weight = ts.ckks_vector(global_context.context, weight)
                decrypted_weight = torch.tensor(encrypted_weight.decrypt(), device=
                m.weight.data = decrypted_weight.view_as(m.weight.data)
            else:
                m.weight.data.uniform_(-0.5, 0.5)
```

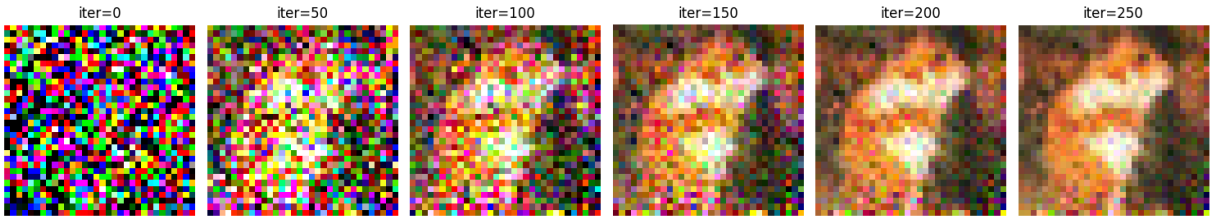
```
        if hasattr(m, "bias"):
            m.bias.data.uniform_(-0.5, 0.5)


class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        act = nn.Sigmoid
        self.body = nn.Sequential(
            nn.Conv2d(3, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
        )
        self.fc = nn.Sequential(
            nn.Linear(768, 100)
        )

    def forward(self, x):
        out = self.body(x)
        out = out.view(out.size(0), -1)
        # print(out.size())
        out = self.fc(out)
        return out


net = LeNet().to(device)

net.apply(weights_init)
criterion = cross_entropy_for_onehot



img_index = 60
gt_data = tp(dst[img_index][0]).to(device)
gt_data = gt_data.view(1, *gt_data.size())
gt_label = torch.Tensor([dst[img_index][1]]).long().to(device)
gt_label = gt_label.view(1, )
gt_onehot_label = label_to_onehot(gt_label, num_classes=100)

plt.imshow(tt(gt_data[0].cpu()))
plt.title("Ground truth image")
print("GT label is %d." % gt_label.item(), "\nOnehot label is %d." % torch.argmax(g

# compute original gradient
out = net(gt_data)
y = criterion(out, gt_onehot_label)
dy_dx = torch.autograd.grad(y, net.parameters())


# share the gradients with other clients
original_dy_dx = list((_.detach().clone() for _ in dy_dx))
```

```
random.seed(72)
torch.manual_seed(42)
# generate dummy data and label
dummy_data = torch.randn(gt_data.size()).to(device).requires_grad_(True)
dummy_label = torch.randn(gt_onehot_label.size()).to(device).requires_grad_(True)

plt.imshow(tt(dummy_data[0].cpu()))
plt.title("Dummy data")
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())



resize_transform = transforms.Resize((161, 161))

optimizer = LBFGS([dummy_data, dummy_label])

# Specific iterations to capture and plot
plot_iterations = [0, 50, 100, 150, 200, 250]

for iters in range(300):
    def closure():
        optimizer.zero_grad()

        pred = net(dummy_data)
        dummy_onehot_label = F.softmax(dummy_label, dim=-1)
        dummy_loss = criterion(pred, dummy_onehot_label)
        dummy_dy_dx = torch.autograd.grad(dummy_loss, net.parameters(), create_grap

        grad_diff = 0
        grad_count = 0
        for gx, gy in zip(dummy_dy_dx, original_dy_dx):
            grad_diff += ((gx - gy) ** 2).sum()
            grad_count += gx.nelement()
        grad_diff.backward()

        return grad_diff

    optimizer.step(closure)
    if iters % 10 == 0:
        current_loss = closure()
        print(iters, "%.4f" % current_loss.item())

        dummy_data_resized = resize_transform(dummy_data)
        gt_data_resized = resize_transform(gt_data)

        # Calculate MS-SSIM
        msssim_value = ms_ssim(dummy_data_resized, gt_data_resized, data_range=1.0,
        msssim_values_40_rand.append(msssim_value.item())

                # Calculate UQI
        dummy_data_np = dummy_data_resized.detach().permute(0, 2, 3, 1).squeeze().c
        gt_data_np = gt_data_resized.permute(0, 2, 3, 1).squeeze().cpu().numpy()
        uqi_value = vifp(dummy_data_np, gt_data_np)
```

```
        uqi_values_40_rand.append(uqi_value)

    if iters in plot_iterations:
        history_40_rand.append(dummy_data[0].cpu())



plt.figure(figsize=(15, 5))

# Plot images in a 1x6 grid
for i, iter_num in enumerate(plot_iterations):
    plt.subplot(1, 6, i + 1)
    image = history_40_rand[i].detach().cpu().permute(1, 2, 0).numpy()  # Detach th
    plt.imshow(image)
    plt.title("iter=%d" % iter_num)
    plt.axis('off')

# Print dummy label
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())

# Adjust layout to make space for titles
plt.tight_layout(rect=[0, 0, 1, 0.9])

# Add a title for the whole plot
plt.suptitle('Reconstructed Images at Different Iterations for 40 % Random', y=1.05

plt.show()
```

```
WARNING: The input does not fit in a single ciphertext, and some operations will be
disabled.
The following operations are disabled in this setup: matmul, matmul_plain, enc_matmu
l_plain, conv2d_im2col.
If you need to use those operations, try increasing the poly_modulus parameter, to f
it your input.
GT label is 36.
Onehot label is 36.
Dummy label is 92.
0 0.3879
10 0.0123
20 0.0037
30 0.0016
40 0.0008
50 0.0005
60 0.0003
70 0.0002
80 0.0001
90 0.0001
100 0.0001
110 0.0000
120 0.0000
130 0.0000
140 0.0000
150 0.0000
160 0.0000
170 0.0000
180 0.0000
190 0.0000
200 0.0000
210 0.0000
220 0.0000
230 0.0000
240 0.0000
250 0.0000
260 0.0000
270 0.0000
280 0.0000
290 0.0000
Dummy label is 36.
```





50 % Random

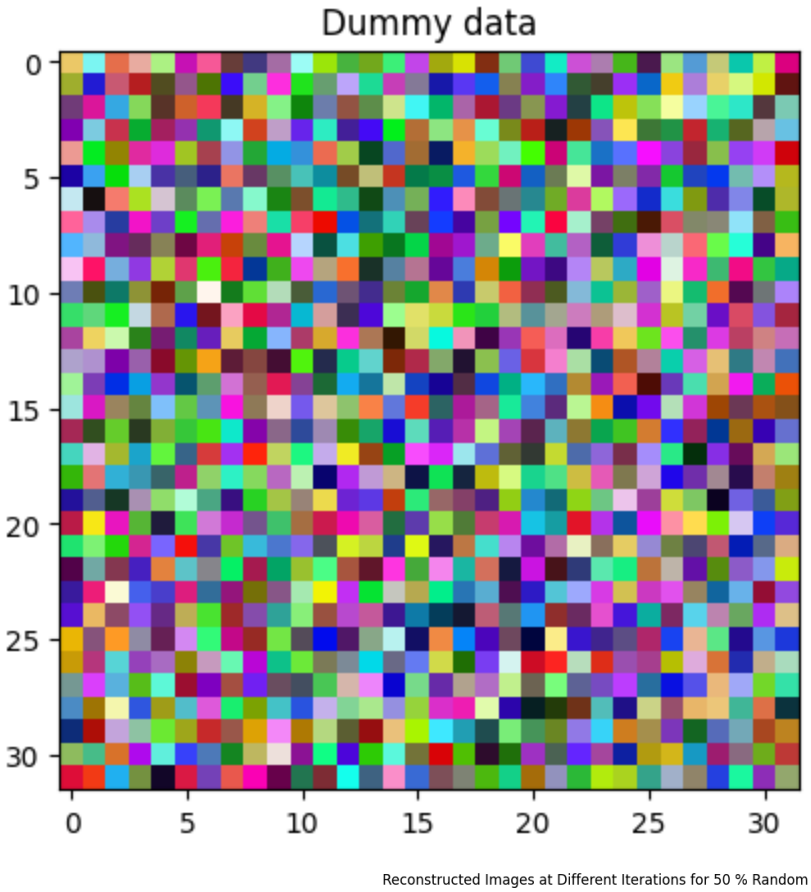
In [18]:
```
history_50_rand = []
msssim_values_50_rand = []
uqi_values_50_rand = []

random.seed(6)

def weights_init(m):
    if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
        if hasattr(m, "weight"):
            if random.random() < 0.5:
                weight = m.weight.data.view(-1).tolist()
                encrypted_weight = ts.ckks_vector(global_context.context, weight)
                decrypted_weight = torch.tensor(encrypted_weight.decrypt(), device=
                m.weight.data = decrypted_weight.view_as(m.weight.data)
            else:
                m.weight.data.uniform_(-0.5, 0.5)
```

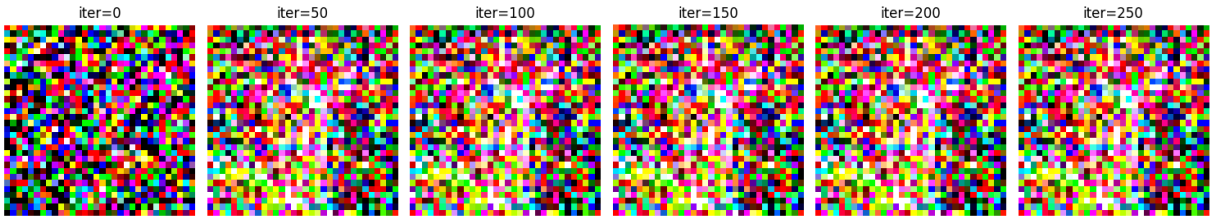
```
        if hasattr(m, "bias"):
            m.bias.data.uniform_(-0.5, 0.5)


class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        act = nn.Sigmoid
        self.body = nn.Sequential(
            nn.Conv2d(3, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
        )
        self.fc = nn.Sequential(
            nn.Linear(768, 100)
        )

    def forward(self, x):
        out = self.body(x)
        out = out.view(out.size(0), -1)
        # print(out.size())
        out = self.fc(out)
        return out


net = LeNet().to(device)

net.apply(weights_init)
criterion = cross_entropy_for_onehot



img_index = 60
gt_data = tp(dst[img_index][0]).to(device)
gt_data = gt_data.view(1, *gt_data.size())
gt_label = torch.Tensor([dst[img_index][1]]).long().to(device)
gt_label = gt_label.view(1, )
gt_onehot_label = label_to_onehot(gt_label, num_classes=100)

plt.imshow(tt(gt_data[0].cpu()))
plt.title("Ground truth image")
print("GT label is %d." % gt_label.item(), "\nOnehot label is %d." % torch.argmax(g

# compute original gradient
out = net(gt_data)
y = criterion(out, gt_onehot_label)
dy_dx = torch.autograd.grad(y, net.parameters())


# share the gradients with other clients
original_dy_dx = list((_.detach().clone() for _ in dy_dx))
```

```
torch.manual_seed(42)
# generate dummy data and label
dummy_data = torch.randn(gt_data.size()).to(device).requires_grad_(True)
dummy_label = torch.randn(gt_onehot_label.size()).to(device).requires_grad_(True)

plt.imshow(tt(dummy_data[0].cpu()))
plt.title("Dummy data")
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())



resize_transform = transforms.Resize((161, 161))

optimizer = LBFGS([dummy_data, dummy_label])

# Specific iterations to capture and plot
plot_iterations = [0, 50, 100, 150, 200, 250]

for iters in range(300):
    def closure():
        optimizer.zero_grad()

        pred = net(dummy_data)
        dummy_onehot_label = F.softmax(dummy_label, dim=-1)
        dummy_loss = criterion(pred, dummy_onehot_label)
        dummy_dy_dx = torch.autograd.grad(dummy_loss, net.parameters(), create_grap

        grad_diff = 0
        grad_count = 0
        for gx, gy in zip(dummy_dy_dx, original_dy_dx):
            grad_diff += ((gx - gy) ** 2).sum()
            grad_count += gx.nelement()
        grad_diff.backward()

        return grad_diff

    optimizer.step(closure)
    if iters % 10 == 0:
        current_loss = closure()
        print(iters, "%.4f" % current_loss.item())

        dummy_data_resized = resize_transform(dummy_data)
        gt_data_resized = resize_transform(gt_data)

        # Calculate MS-SSIM
        msssim_value = ms_ssim(dummy_data_resized, gt_data_resized, data_range=1.0,
        msssim_values_50_rand.append(msssim_value.item())
                # Calculate UQI
        dummy_data_np = dummy_data_resized.detach().permute(0, 2, 3, 1).squeeze().c
        gt_data_np = gt_data_resized.permute(0, 2, 3, 1).squeeze().cpu().numpy()
        uqi_value = vifp(dummy_data_np, gt_data_np)
        uqi_values_50_rand.append(uqi_value)
```

```
    if iters in plot_iterations:
        history_50_rand.append(dummy_data[0].cpu())



plt.figure(figsize=(15, 5))

# Plot images in a 1x6 grid
for i, iter_num in enumerate(plot_iterations):
    plt.subplot(1, 6, i + 1)
    image = history_50_rand[i].detach().cpu().permute(1, 2, 0).numpy()  # Detach th
    plt.imshow(image)
    plt.title("iter=%d" % iter_num)
    plt.axis('off')

# Print dummy label
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())

# Adjust layout to make space for titles
plt.tight_layout(rect=[0, 0, 1, 0.9])

# Add a title for the whole plot
plt.suptitle('Reconstructed Images at Different Iterations for 50 % Random', y=1.05

plt.show()
```

```
WARNING: The input does not fit in a single ciphertext, and some operations will be
disabled.
The following operations are disabled in this setup: matmul, matmul_plain, enc_matmu
l_plain, conv2d_im2col.
If you need to use those operations, try increasing the poly_modulus parameter, to f
it your input.
GT label is 36.
Onehot label is 36.
Dummy label is 92.
0 0.0004
10 0.0000
20 0.0000
30 0.0000
40 0.0000
50 0.0000
60 0.0000
70 0.0000
80 0.0000
90 0.0000
100 0.0000
110 0.0000
120 0.0000
130 0.0000
140 0.0000
150 0.0000
160 0.0000
170 0.0000
180 0.0000
190 0.0000
200 0.0000
210 0.0000
220 0.0000
230 0.0000
240 0.0000
250 0.0000
260 0.0000
270 0.0000
280 0.0000
290 0.0000
Dummy label is 36.
```





5 % sens

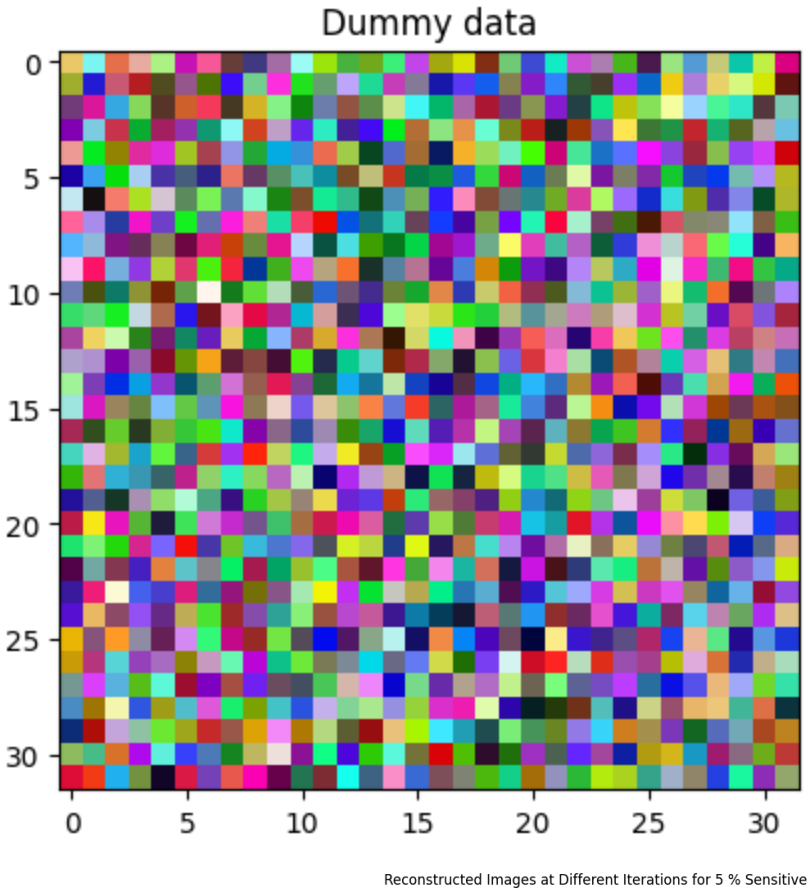
In [19]:
```
history_5_sens = []
msssim_values_5_sens = []
uqi_values_5_sens = []
# # sensitive
# Define the weights initialization function
def weights_init(m):
    if hasattr(m, "weight"):
        m.weight.data.uniform_(-0.5, 0.5)
    if hasattr(m, "bias"):
        m.bias.data.uniform_(-0.5, 0.5)

# Define the LeNet model
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        act = nn.Sigmoid
```

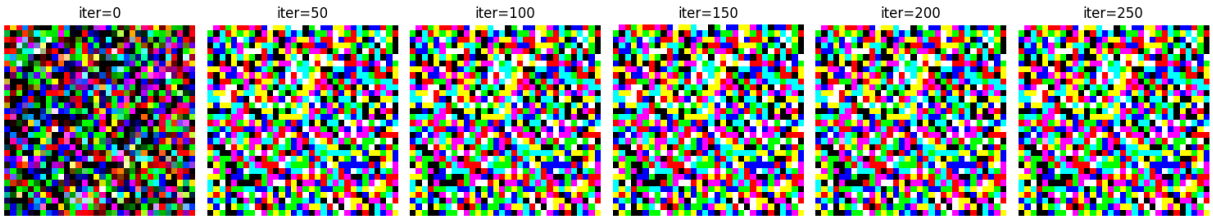
```
        self.body = nn.Sequential(
            nn.Conv2d(3, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
        )
        self.fc = nn.Sequential(
            nn.Linear(768, 100)
        )

    def forward(self, x):
        out = self.body(x)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out

# Initialize the model and weights
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = LeNet().to(device)
net.apply(weights_init)
criterion = nn.CrossEntropyLoss()  # Assuming cross_entropy_for_onehot is a custom

# Load CIFAR-100 dataset
tp_ = transforms.Compose([
    transforms.Resize(72),
    transforms.CenterCrop(32),
    transforms.ToTensor()
])
dst_ = datasets.CIFAR100("/kaggle/working/torch", download=True, transform=tp_)
data_loader = torch.utils.data.DataLoader(dst_, batch_size=64, shuffle=True)

# Function to calculate parameter sensitivities
def calculate_sensitivity(model, dataloader):
    model.train()
    criterion = nn.CrossEntropyLoss()
    gradient_sums = {}

    for name, param in model.named_parameters():
        if 'bias' not in name:
            gradient_sums[name] = 0.0
            param.requires_grad_(True)

    for inputs, labels in dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        model.zero_grad()
        loss.backward()

        for name, parameter in model.named_parameters():
            if 'bias' not in name and parameter.requires_grad:
                grads = parameter.grad.abs().sum().item()
```

```
                gradient_sums[name] += grads

    return gradient_sums

# Encrypt the most sensitive model parameters using TenSEAL
def encrypt_parameters(model, sensitivities, encryption_percentage=0.1):

    # Sort parameters by sensitivity
    sorted_sensitivities = sorted(sensitivities.items(), key=lambda item: item[1],
    num_params_to_encrypt = int(len(sorted_sensitivities) * encryption_percentage)

    encrypted_parameters = {}
    for i in range(num_params_to_encrypt):
        name, _ = sorted_sensitivities[i]
        parameter = dict(model.named_parameters())[name].data.cpu().numpy()
        encrypted_param = ts.ckks_vector(global_context.context, parameter.flatten(
        encrypted_parameters[name] = encrypted_param

    return model, encrypted_parameters

# Calculate sensitivities
sensitivities = calculate_sensitivity(net, data_loader)

# Encrypt 10% of the most sensitive parameters
# encrypted_net, encrypted_parameters = encrypt_parameters(net, sensitivities, encr
encrypted_net, encrypted_parameters = encrypt_parameters(net, sensitivities, encryp

print("Encryption completed.")

random.seed(72)
torch.manual_seed(42)
# generate dummy data and label
dummy_data = torch.randn(gt_data.size()).to(device).requires_grad_(True)
dummy_label = torch.randn(gt_onehot_label.size()).to(device).requires_grad_(True)

plt.imshow(tt(dummy_data[0].cpu()))
plt.title("Dummy data")
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())


resize_transform = transforms.Resize((161, 161))

optimizer = LBFGS([dummy_data, dummy_label])

# Specific iterations to capture and plot
plot_iterations = [0, 50, 100, 150, 200, 250]

for iters in range(300):
    def closure():
        optimizer.zero_grad()

        pred = net(dummy_data)
        dummy_onehot_label = F.softmax(dummy_label, dim=-1)
        dummy_loss = criterion(pred, dummy_onehot_label)
        dummy_dy_dx = torch.autograd.grad(dummy_loss, net.parameters(), create_grap
```

```
        grad_diff = 0
        grad_count = 0
        for gx, gy in zip(dummy_dy_dx, original_dy_dx):
            grad_diff += ((gx - gy) ** 2).sum()
            grad_count += gx.nelement()
        grad_diff.backward()

        return grad_diff

    optimizer.step(closure)
    if iters % 10 == 0:
        current_loss = closure()
        print(iters, "%.4f" % current_loss.item())

        dummy_data_resized = resize_transform(dummy_data)
        gt_data_resized = resize_transform(gt_data)

        # Calculate MS-SSIM
        msssim_value = ms_ssim(dummy_data_resized, gt_data_resized, data_range=1.0,
        msssim_values_5_sens.append(msssim_value.item())

                # Calculate UQI
        dummy_data_np = dummy_data_resized.detach().permute(0, 2, 3, 1).squeeze().c
        gt_data_np = gt_data_resized.permute(0, 2, 3, 1).squeeze().cpu().numpy()
        uqi_value = vifp(dummy_data_np, gt_data_np)
        uqi_values_5_sens.append(uqi_value)
    if iters in plot_iterations:
        history_5_sens.append(dummy_data[0].cpu())

plt.figure(figsize=(15, 5))

# Plot images in a 1x6 grid
for i, iter_num in enumerate(plot_iterations):
    plt.subplot(1, 6, i + 1)
    image = history_5_sens[i].detach().cpu().permute(1, 2, 0).numpy()  # Detach the
    plt.imshow(image)
    plt.title("iter=%d" % iter_num)
    plt.axis('off')

# Print dummy label
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())

# Adjust layout to make space for titles
plt.tight_layout(rect=[0, 0, 1, 0.9])

# Add a title for the whole plot
plt.suptitle('Reconstructed Images at Different Iterations for 5 % Sensitive', y=1.

plt.show()
```

```
Files already downloaded and verified
Encryption completed.
Dummy label is 92.
0 89.5979
10 62.7126
20 61.5839
30 943.1993
40 943.1993
50 943.1994
60 943.1992
70 943.1995
80 943.1994
90 943.1994
100 943.1992
110 943.1994
120 943.1995
130 943.1992
140 943.1995
150 943.1993
160 943.1994
170 943.1992
180 943.1994
190 943.1993
200 943.1994
210 943.1993
220 943.1992
230 943.1995
240 943.1992
250 943.1995
260 943.1994
270 943.1995
280 943.1995
290 943.1994
Dummy label is 17.
```





10 % Sens

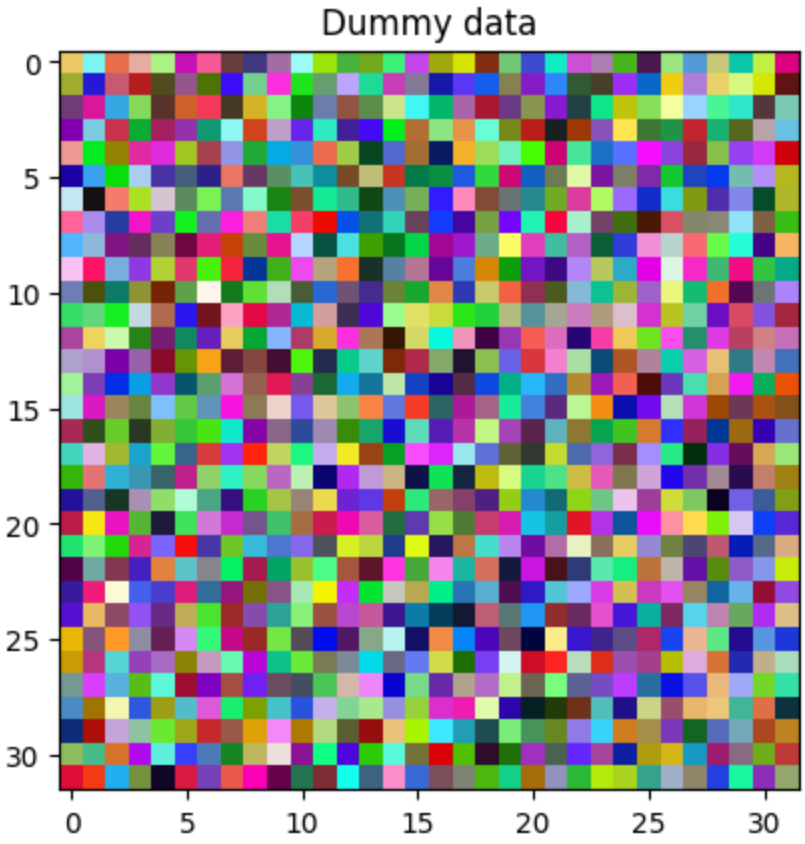
In [20]:
```
history_10_sens = []
msssim_values_10_sens = []
uqi_values_10_senss = []
# # sensitive
# Define the weights initialization function
def weights_init(m):
    if hasattr(m, "weight"):
        m.weight.data.uniform_(-0.5, 0.5)
    if hasattr(m, "bias"):
        m.bias.data.uniform_(-0.5, 0.5)

# Define the LeNet model
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        act = nn.Sigmoid
```

```
        self.body = nn.Sequential(
            nn.Conv2d(3, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=2),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
            nn.Conv2d(12, 12, kernel_size=5, padding=5//2, stride=1),
            act(),
        )
        self.fc = nn.Sequential(
            nn.Linear(768, 100)
        )

    def forward(self, x):
        out = self.body(x)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out

# Initialize the model and weights
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = LeNet().to(device)
net.apply(weights_init)
criterion = nn.CrossEntropyLoss()  # Assuming cross_entropy_for_onehot is a custom

# Load CIFAR-100 dataset
tp_ = transforms.Compose([
    transforms.Resize(32),
    transforms.CenterCrop(32),
    transforms.ToTensor()
])
dst_ = datasets.CIFAR100("/kaggle/working/torch", download=True, transform=tp_)
data_loader = torch.utils.data.DataLoader(dst_, batch_size=64, shuffle=True)

# Function to calculate parameter sensitivities
def calculate_sensitivity(model, dataloader):
    model.train()
    criterion = nn.CrossEntropyLoss()
    gradient_sums = {}

    for name, param in model.named_parameters():
        if 'bias' not in name:
            gradient_sums[name] = 0.0
            param.requires_grad_(True)

    for inputs, labels in dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        model.zero_grad()
        loss.backward()

        for name, parameter in model.named_parameters():
            if 'bias' not in name and parameter.requires_grad:
                grads = parameter.grad.abs().sum().item()
```

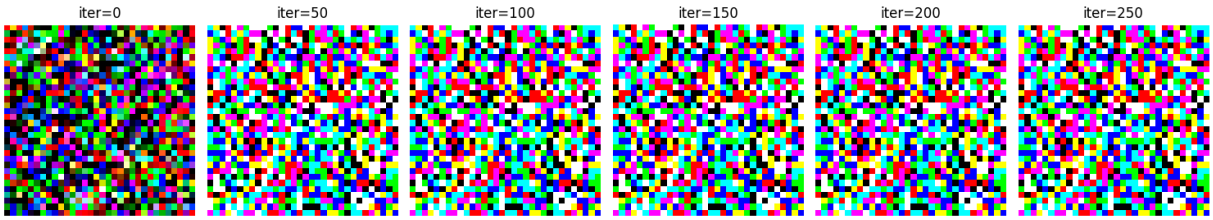
```
                gradient_sums[name] += grads

    return gradient_sums

# Encrypt the most sensitive model parameters using TenSEAL
def encrypt_parameters(model, sensitivities, encryption_percentage=0.1):

    # Sort parameters by sensitivity
    sorted_sensitivities = sorted(sensitivities.items(), key=lambda item: item[1],
    num_params_to_encrypt = int(len(sorted_sensitivities) * encryption_percentage)

    encrypted_parameters = {}
    for i in range(num_params_to_encrypt):
        name, _ = sorted_sensitivities[i]
        parameter = dict(model.named_parameters())[name].data.cpu().numpy()
        encrypted_param = ts.ckks_vector(global_context.context, parameter.flatten(
        encrypted_parameters[name] = encrypted_param

    return model, encrypted_parameters

# Calculate sensitivities
sensitivities = calculate_sensitivity(net, data_loader)

# Encrypt 10% of the most sensitive parameters
encrypted_net, encrypted_parameters = encrypt_parameters(net, sensitivities, encryp

print("Encryption completed.")

random.seed(72)
torch.manual_seed(42)
# generate dummy data and label
dummy_data = torch.randn(gt_data.size()).to(device).requires_grad_(True)
dummy_label = torch.randn(gt_onehot_label.size()).to(device).requires_grad_(True)

plt.imshow(tt(dummy_data[0].cpu()))
plt.title("Dummy data")
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())


resize_transform = transforms.Resize((161, 161))

optimizer = LBFGS([dummy_data, dummy_label])

# Specific iterations to capture and plot
plot_iterations = [0, 50, 100, 150, 200, 250]

for iters in range(300):
    def closure():
        optimizer.zero_grad()

        pred = net(dummy_data)
        dummy_onehot_label = F.softmax(dummy_label, dim=-1)
        dummy_loss = criterion(pred, dummy_onehot_label)
        dummy_dy_dx = torch.autograd.grad(dummy_loss, net.parameters(), create_grap

        grad_diff = 0
```

```
        grad_count = 0
        for gx, gy in zip(dummy_dy_dx, original_dy_dx):
            grad_diff += ((gx - gy) ** 2).sum()
            grad_count += gx.nelement()
        grad_diff.backward()

        return grad_diff

    optimizer.step(closure)
    if iters % 10 == 0:
        current_loss = closure()
        print(iters, "%.4f" % current_loss.item())

        dummy_data_resized = resize_transform(dummy_data)
        gt_data_resized = resize_transform(gt_data)

        # Calculate MS-SSIM
        msssim_value = ms_ssim(dummy_data_resized, gt_data_resized, data_range=1.0,
        msssim_values_10_sens.append(msssim_value.item())

                # Calculate UQI
        dummy_data_np = dummy_data_resized.detach().permute(0, 2, 3, 1).squeeze().c
        gt_data_np = gt_data_resized.permute(0, 2, 3, 1).squeeze().cpu().numpy()
        uqi_value = vifp(dummy_data_np, gt_data_np)
        uqi_values_10_senss.append(uqi_value)
    if iters in plot_iterations:
        history_10_sens.append(dummy_data[0].cpu())

plt.figure(figsize=(15, 5))

# Plot images in a 1x6 grid
for i, iter_num in enumerate(plot_iterations):
    plt.subplot(1, 6, i + 1)
    image = history_10_sens[i].detach().cpu().permute(1, 2, 0).numpy()  # Detach th
    plt.imshow(image)
    plt.title("iter=%d" % iter_num)
    plt.axis('off')

# Print dummy label
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())

# Adjust layout to make space for titles
plt.tight_layout(rect=[0, 0, 1, 0.9])

# Add a title for the whole plot
plt.suptitle('Reconstructed Images at Different Iterations for 10 % Sensitive', y=1

plt.show()
```

```
Files already downloaded and verified
Encryption completed.
Dummy label is 92.
0 89.6210
10 63.0515
20 62.9397
30 609.4541
40 609.4542
50 609.4542
60 609.4542
70 609.4541
80 609.4541
90 609.4542
100 609.4542
110 609.4542
120 609.4542
130 609.4541
140 609.4542
150 609.4541
160 609.4542
170 609.4541
180 609.4542
190 609.4541
200 609.4542
210 609.4541
220 609.4542
230 609.4542
240 609.4542
250 609.4541
260 609.4542
270 609.4541
280 609.4542
290 609.4542
Dummy label is 83.
```



Reconstructed Images at Different Iterations for 10 % Sensitive

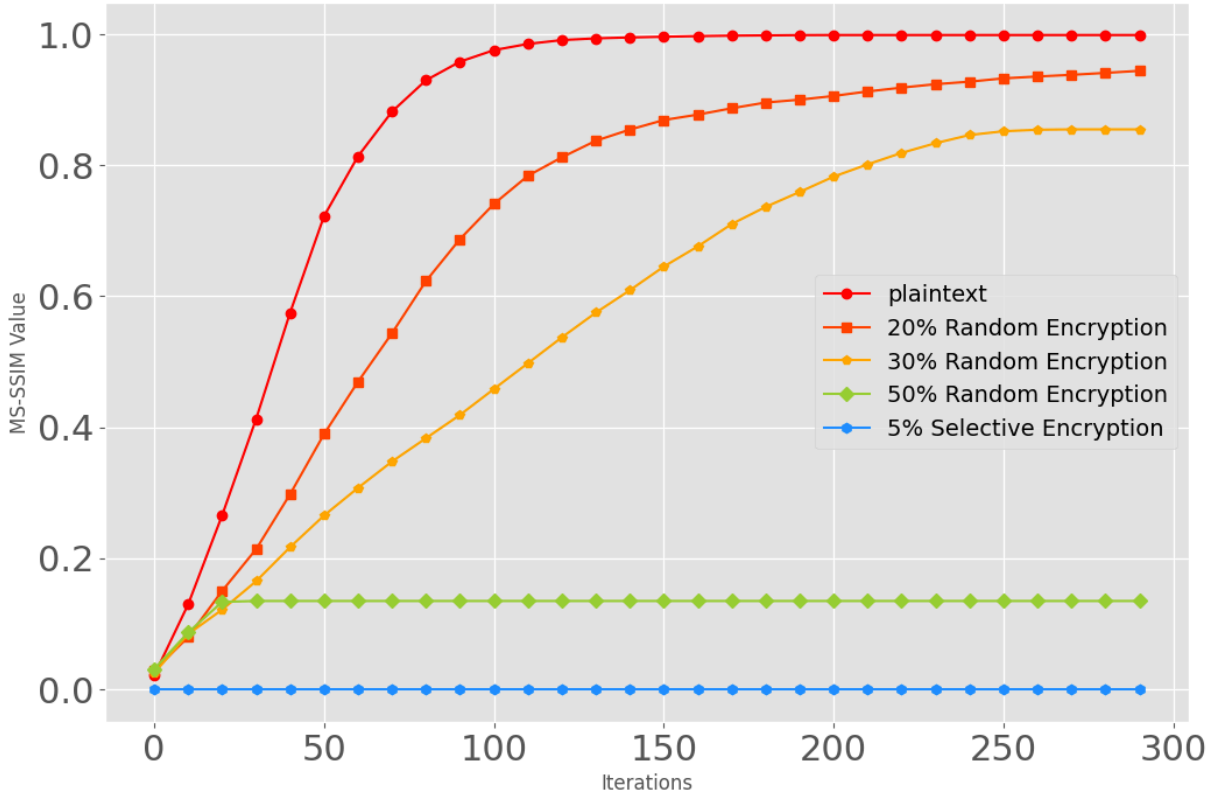


In [36]:
```
np.save('msssim_values_raw.npy', msssim_values_raw)
np.save('msssim_values_20_rand.npy', msssim_values_20_rand)
np.save('msssim_values_40_rand.npy', msssim_values_40_rand)
np.save('msssim_values_50_rand.npy', msssim_values_50_rand)
np.save('msssim_values_5_sens.npy', msssim_values_5_sens)

np.save('uqi_values_raw.npy', msssim_values_raw)
np.save('uqi_values_20_rand.npy', msssim_values_20_rand)
np.save('uqi_values_40_rand.npy', msssim_values_40_rand)
np.save('uqi_values_50_rand.npy', msssim_values_50_rand)
np.save('uqi_values_5_sens.npy', msssim_values_5_sens)

np.save('history_raw.npy', msssim_values_raw)
np.save('history_20_rand.npy', msssim_values_20_rand)
np.save('history_40_rand.npy', msssim_values_40_rand)
np.save('history_50_rand.npy', msssim_values_50_rand)
np.save('history_5_sens.npy', msssim_values_5_sens)
np.save('history_10_sens.npy', msssim_values_5_sens)
```

In [35]:
```
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
import tikzplotlib

# Set font properties globally
rcParams['font.family'] = 'serif'
rcParams['font.serif'] = ['Times New Roman']
rcParams['axes.titlesize'] = 24
rcParams['xtick.labelsize'] = 22
rcParams['ytick.labelsize'] = 22

# Assuming msssim_values_raw, msssim_values_20_rand, msssim_values_40_rand, msssim_
# Also, assuming iterations is defined

plt.figure(figsize=(12, 8))
plt.plot(iterations, msssim_values_raw, label='plaintext', color='red', marker='o')
plt.plot(iterations, msssim_values_20_rand, label='20% Random Encryption', color='o
plt.plot(iterations, msssim_values_40_rand, label='30% Random Encryption', color='o
plt.plot(iterations, msssim_values_50_rand, label='50% Random Encryption', color='y
plt.plot(iterations, msssim_values_5_sens, label='5% Selective Encryption', color='

plt.xlabel('Iterations')
plt.ylabel('MS-SSIM Value')
plt.legend(fontsize=14, ncol=1)
plt.grid(True)


tikzplotlib.save("/kaggle/working/msssim.tex")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[35], line 29
     25 plt.legend(fontsize=14, ncol=1)
     26 plt.grid(True)
---> 29 tikzplotlib.save("/kaggle/working/msssim.tex")

File /opt/conda/lib/python3.10/site-packages/tikzplotlib/_save.py:262, in save(filepath, encoding, *args, **kwargs)
    252 def save(filepath: str | Path, *args, encoding: str | None = None, **kwargs):
    253     """Same as `get_tikz_code()`, but actually saves the code to a file.
    254
    255     :param filepath: The file to which the TikZ output will be written.
   (...)
    260     :returns: None
    261     """
--> 262     code = get_tikz_code(*args, filepath=filepath, **kwargs)
    263     with open(filepath, "w", encoding=encoding) as f:
    264         f.write(code)

File /opt/conda/lib/python3.10/site-packages/tikzplotlib/_save.py:213, in get_tikz_code(figure, filepath, axis_width, axis_height, textsize, tex_relative_path_to_data, externalize_tables, override_externals, externals_search_path, strict, wrap, add_axis_environment, extra_axis_parameters, extra_groupstyle_parameters, extra_tikzpicture_parameters, extra_lines_start, dpi, show_info, include_disclaimer, standalone, float_format, table_row_sep, flavor)
    210     _print_pgfplot_libs_message(data)
    212 # gather the file content
--> 213 data, content = _recurse(data, figure)
    215 # Check if there is still an open groupplot environment. This occurs if not
    216 # all of the group plot slots are used.
    217 if "is_in_groupplot_env" in data and data["is_in_groupplot_env"]:

File /opt/conda/lib/python3.10/site-packages/tikzplotlib/_save.py:352, in _recurse(data, obj)
    349 data["current axes"] = ax
    351 # Run through the child objects, gather the content.
--> 352 data, children_content = _recurse(data, child)
    354 # populate content and add axis environment if desired
    355 if data["add axis environment"]:

File /opt/conda/lib/python3.10/site-packages/tikzplotlib/_save.py:380, in _recurse(data, obj)
    378     content.extend(cont, child.get_zorder())
    379 elif isinstance(child, mpl.legend.Legend):
--> 380     data = _legend.draw_legend(data, child)
    381     if data["legend colors"]:
    382         content.extend(data["legend colors"], 0)

File /opt/conda/lib/python3.10/site-packages/tikzplotlib/_legend.py:81, in draw_legend(data, obj)
     78 if alignment:
     79     data["current axes"].axis_options.append(f"legend cell align={{{alignment}}}")
---> 81 if obj._ncol != 1:
```

```
     82     data["current axes"].axis_options.append(f"legend columns={obj._ncol}")
     84 # Write styles to data

AttributeError: 'Legend' object has no attribute '_ncol'
```



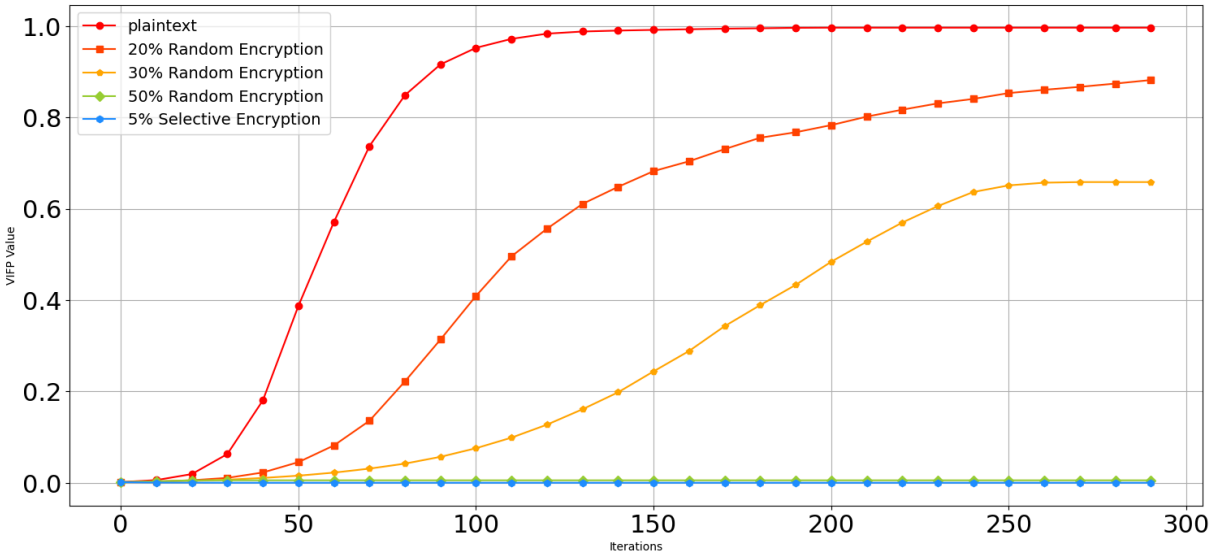
UQI

In [22]:

```
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.offsetbox import OffsetImage, AnnotationBbox

# Set font properties globally
rcParams['font.family'] = 'serif'
rcParams['font.serif'] = ['Times New Roman']
rcParams['axes.titlesize'] = 24
rcParams['xtick.labelsize'] = 22
rcParams['ytick.labelsize'] = 22

gt_image = gt_data[0].cpu().permute(1, 2, 0).numpy()

iterations = list(range(0, 300, 10))

# really is vifp

plt.figure(figsize=(18, 8))
plt.plot(iterations, uqi_values_raw, label='plaintext', color='red', marker='o')
plt.plot(iterations, uqi_values_20_rand, label='20% Random Encryption', color='oran
plt.plot(iterations, uqi_values_40_rand, label='30% Random Encryption', color='oran
plt.plot(iterations, uqi_values_50_rand, label='50% Random Encryption', color='yell
plt.plot(iterations, uqi_values_5_sens, label='5% Selective Encryption', color='dod
```

```
print(uqi_values_raw)

plt.xlabel('Iterations')
plt.ylabel('VIFP Value')
plt.legend(fontsize=14)  # Adjust the fontsize for the legend
plt.grid(True)
plt.show()

tikzplotlib.save("/kaggle/working/vifp.tex")
```

```
[0.00152912409707847, 0.0058088448211621, 0.019019833305154003, 0.0629110578975688,
0.18048530825476847, 0.38731001207890614, 0.5712060710404138, 0.7372918821938917, 0.
8490268948712184, 0.9162920818624013, 0.9526221749044123, 0.9722489531742514, 0.9836
527991214897, 0.9883466478036835, 0.9904185787072189, 0.9919356613099369, 0.99328115
37508731, 0.9946208340631908, 0.9954427950880328, 0.9964644314321546, 0.996806819610
9252, 0.9968068196109252, 0.9968068196109252, 0.9968068196109252, 0.996806819610925
2, 0.9968068196109252, 0.9968068196109252, 0.9968068196109252, 0.9968068196109252,
0.9968068196109252]
```



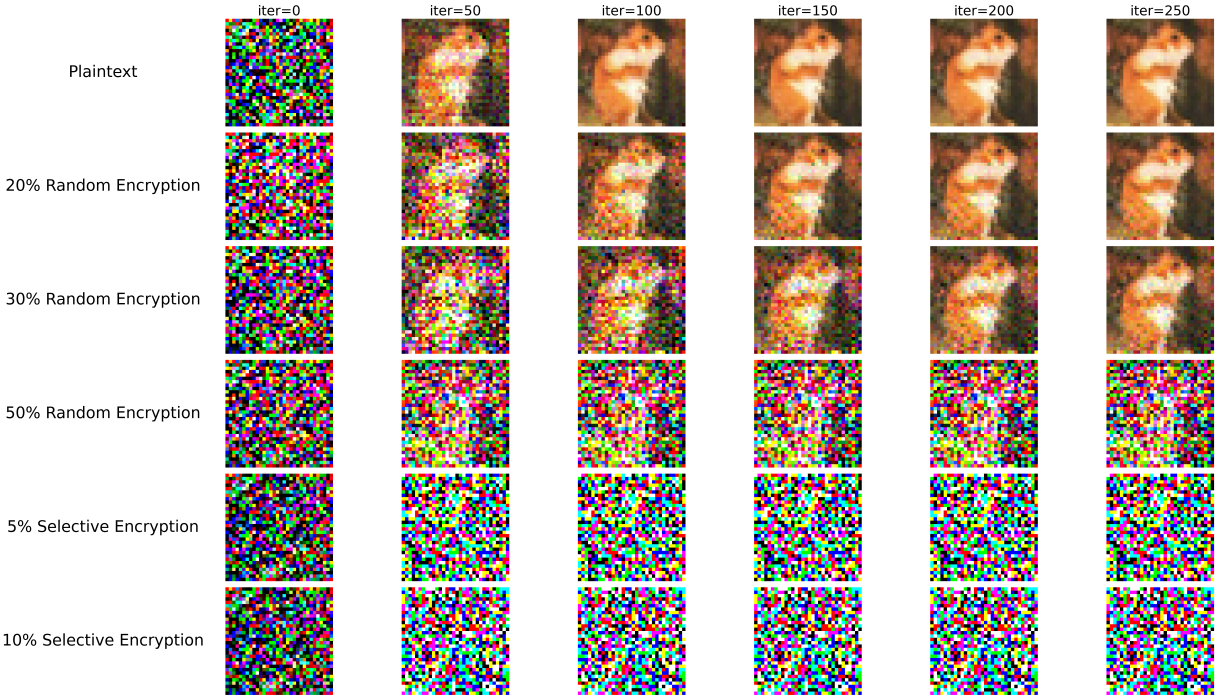

```
<Figure size 640x480 with 0 Axes>
```

All in one history plot

In [23]:
```
import matplotlib.pyplot as plt
from matplotlib import rcParams

# Set font properties globally
rcParams['font.family'] = 'serif'
rcParams['font.serif'] = ['Times New Roman']
rcParams['axes.titlesize'] = 24
rcParams['axes.labelsize'] = 24
rcParams['xtick.labelsize'] = 22
rcParams['ytick.labelsize'] = 22
rcParams['legend.fontsize'] = 24
rcParams['figure.titlesize'] = 32

# Specific iterations to capture and plot
plot_iterations = [0, 50, 100, 150, 200, 250]

# Titles for each history
```

```
titles = [
    'Plaintext',
    '20% Random Encryption',
    '30% Random Encryption',  # really is 40%
    '50% Random Encryption',
    '5% Selective Encryption',
    '10% Selective Encryption'
]

# List of histories
histories = [
    history_raw,
    history_20_rand,
    history_40_rand,
    history_50_rand,
    history_5_sens,
    history_10_sens
]

# Create the figure
fig, axs = plt.subplots(6, 7, figsize=(31, 18), dpi=300)

# Plot images in a 6x7 grid (6 histories, 7 columns including subtitle column)
for row, (history, title) in enumerate(zip(histories, titles)):
    for col in range(7):
        ax = axs[row, col]
        if col == 0:
            ax.text(0.5, 0.5, title, ha='center', va='center', fontsize=28)
            ax.axis('off')
        else:
            image = history[col - 1].detach().cpu().permute(1, 2, 0).numpy()  # Det
            ax.imshow(image)
            ax.axis('off')
            if row == 0:
                ax.set_title("iter=%d" % plot_iterations[col - 1], fontsize=24)

# Adjust layout to make space for titles
plt.tight_layout(rect=[0, 0, 1, 0.96])

# Add a title for the whole plot
# plt.suptitle('Reconstructed Images at Different Iterations for Various Methods',

# Print dummy label
print("Dummy label is %d." % torch.argmax(dummy_label, dim=-1).item())

plt.show()
tikzplotlib.save("/kaggle/working/all_images_attack.tex")
```

```
Dummy label is 83.
```



<Figure size 640x480 with 0 Axes>