In [ ]:
```python
# ============================================
# ENVIRONMENT SETUP
# ============================================
import sys
import os

# Detect environment
is_container = os.path.expanduser('~') == '/root'
print(f"Environment: {'Container/Colab' if is_container else 'Local'}")

if is_container:
    # Running in container - clone/update from GitHub
    if os.path.exists('/content/cure'):
        print("Updating to latest from GitHub...")
        os.chdir('/content/cure')
        !git fetch origin
        !git reset --hard origin/main
    else:
        print("Cloning repo...")
        !git clone https://github.com/RobbyPratl/cure.git /content/cure
        os.chdir('/content/cure')

    src_path = '/content/cure/src'
else:
    # Local development
    src_path = os.path.abspath('../src')

if src_path not in sys.path:
    sys.path.insert(0, src_path)
print(f"src path: {src_path}")
```

```
Environment: Container/Colab
📦 Updating to latest from GitHub...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 7 (delta 5), reused 7 (delta 5), pack-reused 0 (from 0)
Unpacking objects: 100% (7/7), 1.42 KiB | 727.00 KiB/s, done.
From https://github.com/RobbyPratl/cure
   4b9de0a..cec2542  main       -> origin/main
HEAD is now at cec2542 Fix relative imports to work with direct module impor
ts
✅ src path: /content/cure/src
```

In [8]:
```python
# Import libraries
import torch
import matplotlib.pyplot as plt
import numpy as np

from degradations import (
    create_gaussian_kernel, degrade_image,
    get_kernel_frequency_response
)
from utils import load_image, show_images, compute_psnr
from wiener import WienerFilter
```

```
print(f"PyTorch: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
```

```
PyTorch: 2.9.0+cpu
CUDA available: False
```

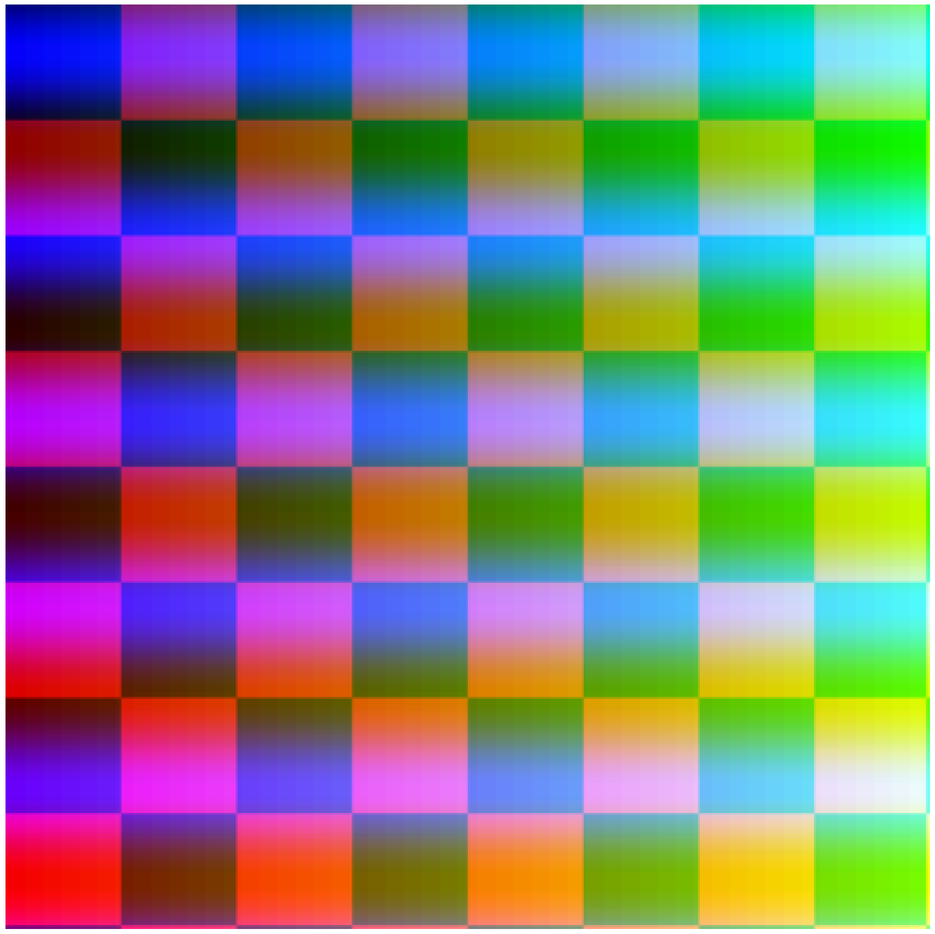
1. Create Test Image

We create a synthetic test image with various frequency content to analyze degradation effects.

In [9]:
```
# Create test image with varying frequency content
def make_test_image(size=256):
    x = torch.linspace(0, 1, size)
    xx, yy = torch.meshgrid(x, x, indexing='ij')
    # Checkerboard pattern (high frequency)
    r = ((xx*8).long() + (yy*8).long()) % 2 * 0.5 + xx * 0.5
    # Gradient (low frequency)
    g = yy
    # Sinusoidal (medium frequency)
    b = torch.sin(xx * 10 * np.pi) * 0.5 + 0.5
    return torch.stack([r, g, b])

clean = make_test_image(256)
print(f"Image shape: {clean.shape}")

plt.figure(figsize=(6, 6))
plt.imshow(clean.permute(1, 2, 0))
plt.title('Test Image (mixed frequencies)')
plt.axis('off')
plt.show()
```

```
Image shape: torch.Size([3, 256, 256])
```

Test Image (mixed frequencies)

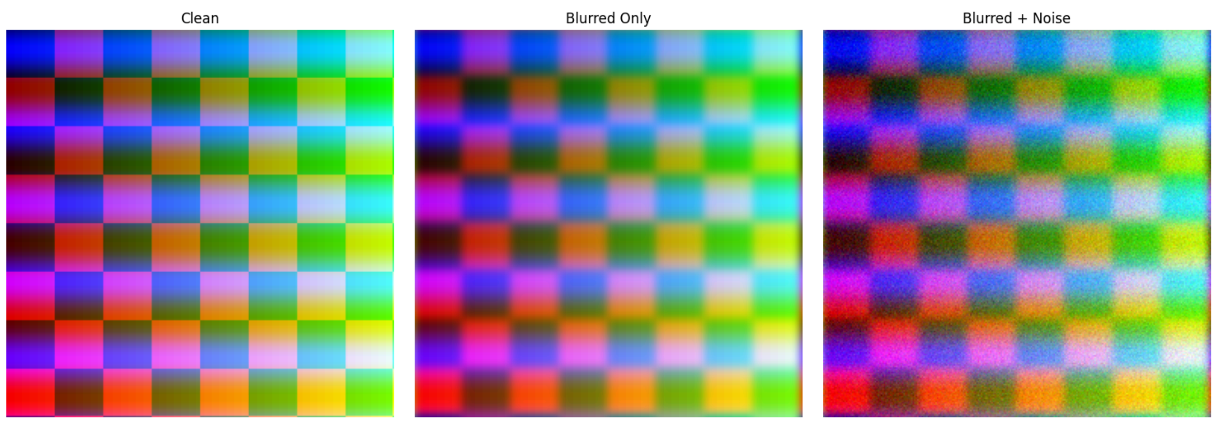


2. Apply Degradation

Apply Gaussian blur and additive noise to simulate a degraded observation.

In [11]:
```
# Degrade image with blur and noise
result = degrade_image(clean, blur_sigma=3.0, noise_sigma=0.05, seed=42)

print(f"Blur kernel shape: {result['kernel'].shape}")
print(f"Noise sigma: {result['params']['noise_sigma']}")

show_images(
    [clean, result['blurred'], result['degraded']],
    ['Clean', 'Blurred Only', 'Blurred + Noise']
)
```

```
Blur kernel shape: torch.Size([19, 19])
Noise sigma: 0.05
```

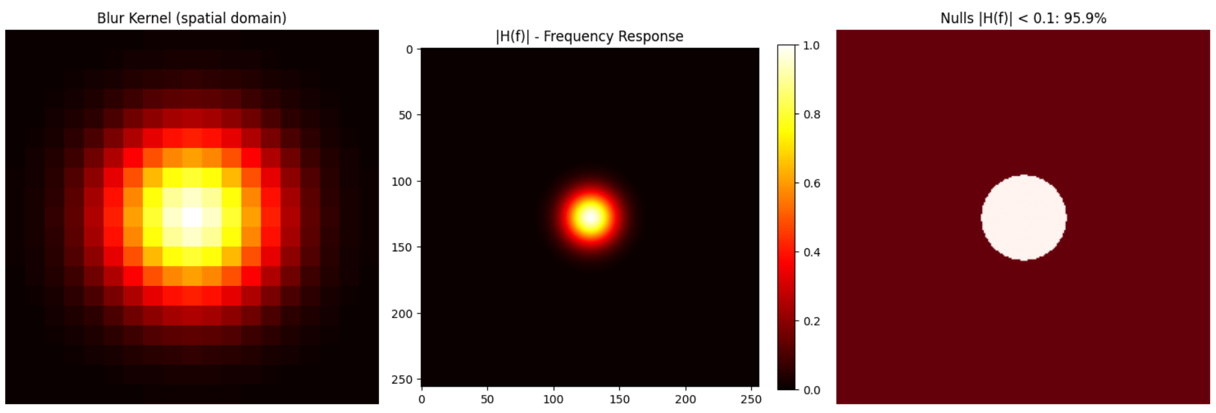


3. Frequency Response Analysis

Key Insight: The blur kernel's frequency response |H(f)| shows which frequencies are attenuated. Where |H(f)| ≈ 0, information is destroyed - these are the "null" frequencies where we hypothesize diffusion models will be miscalibrated.

In [ ]:

```python
# Analyze frequency response of the blur kernel
kernel = result['kernel']
H_mag = get_kernel_frequency_response(kernel, (256, 256))
H_centered = torch.fft.fftshift(H_mag)

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Spatial kernel
axes[0].imshow(kernel.numpy(), cmap='hot')
axes[0].set_title('Blur Kernel (spatial domain)')
axes[0].axis('off')

# Frequency response magnitude
im = axes[1].imshow(H_centered.numpy(), cmap='hot', vmin=0, vmax=1)
axes[1].set_title('|H(f)| - Frequency Response')
plt.colorbar(im, ax=axes[1], fraction=0.046)

# Null frequencies (where info is lost)
nulls = (H_centered < 0.1).float()
axes[2].imshow(nulls.numpy(), cmap='Reds')
axes[2].set_title(f'Nulls |H(f)| < 0.1: {nulls.mean()*100:.1f}%')
axes[2].axis('off')

plt.tight_layout()
plt.show()

print(f"KEY: {nulls.mean()*100:.1f}% of frequencies have |H(f)| < 0.1")
print("Hypothesis: Diffusion will be miscalibrated at these null frequencies
```



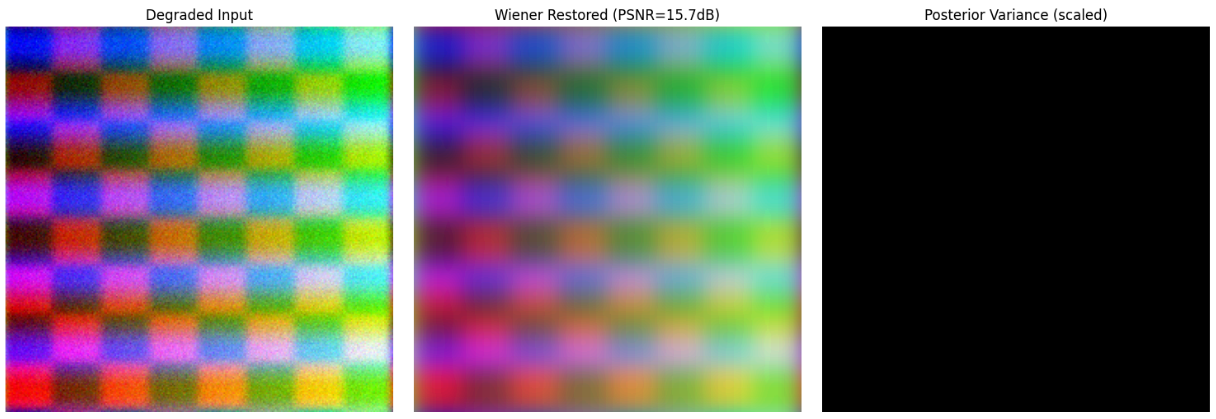
```
🔑 KEY: 95.9% of frequencies have |H(f)| < 0.1
Hypothesis: Diffusion will be miscalibrated at these null frequencies!
```

4. Wiener Filter Restoration

The Wiener filter provides the optimal linear estimator under Gaussian assumptions. Critically, it also gives us **closed-form posterior variance** - our calibration reference.

In [13]:
```python
# Wiener filter restoration with uncertainty quantification
wf = WienerFilter(result['kernel'], noise_sigma=0.05, image_shape=(256, 256)

# Restore and get variance
restored, var_spatial, var_freq = wf.restore(result['degraded'])

# Compute quality metric
psnr = compute_psnr(restored, clean)
print(f"PSNR: {psnr:.2f} dB")

# Visualize results
show_images(
    [result['degraded'], restored, var_spatial.unsqueeze(0).repeat(3, 1, 1)
    ['Degraded Input', f'Wiener Restored (PSNR={psnr:.1f}dB)', 'Posterior Va
)
```

```
PSNR: 15.74 dB
```



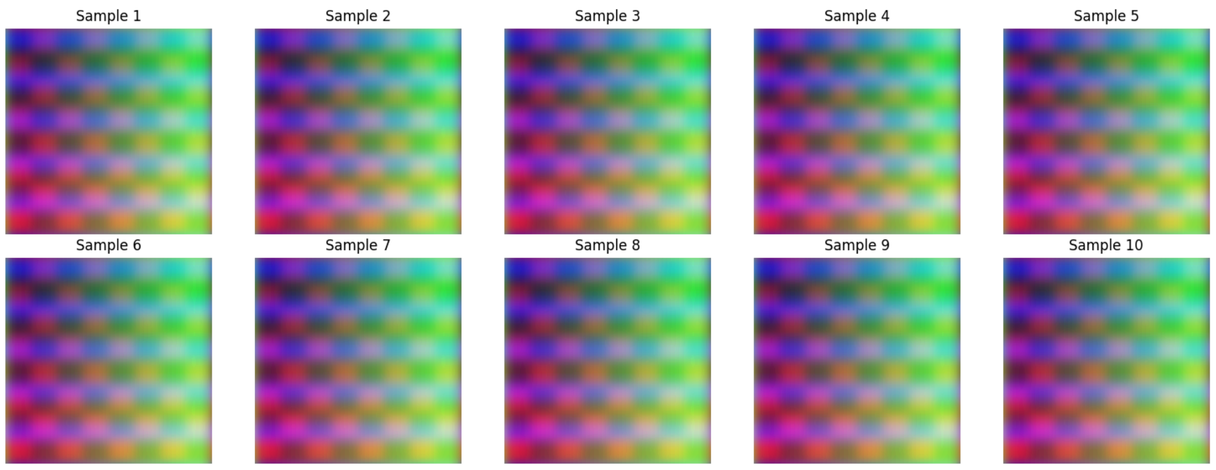
5. Posterior Sampling

Sample from the Gaussian posterior to visualize uncertainty. The diversity in samples shows where the model is uncertain.

In [14]:
```python
# Sample from Wiener posterior
samples = wf.sample_posterior(result['degraded'], n_samples=20)
print(f"Samples shape: {samples.shape}")

# Display sample diversity
fig, axes = plt.subplots(2, 5, figsize=(15, 6))
for i in range(10):
    ax = axes[i // 5, i % 5]
    ax.imshow(samples[i].permute(1, 2, 0).clip(0, 1).numpy())
    ax.axis('off')
    ax.set_title(f'Sample {i+1}')
plt.suptitle('Wiener Posterior Samples - Diversity Shows Uncertainty', fonts
plt.tight_layout()
plt.show()

# Compute sample statistics
sample_mean = samples.mean(dim=0)
sample_std = samples.std(dim=0)
print(f"Sample mean PSNR: {compute_psnr(sample_mean, clean):.2f} dB")
print(f"Mean pixel std: {sample_std.mean():.4f}")
```

```
Samples shape: torch.Size([20, 3, 256, 256])
```

Wiener Posterior Samples - Diversity Shows Uncertainty



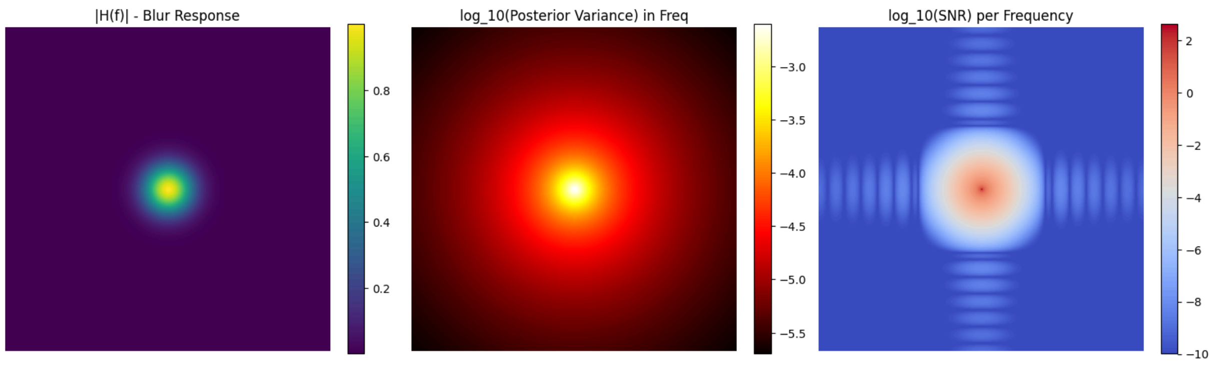
```
Sample mean PSNR: 15.74 dB
Mean pixel std: 0.0000
```

6. Variance vs Frequency Response

Core CURE hypothesis: Posterior variance should be HIGH where |H(f)| is LOW (information destroyed by blur).

In [ ]:
```python
# Visualize relationship between |H(f)| and variance
var_freq_centered = torch.fft.fftshift(var_freq)
snr = wf.get_snr_per_frequency()
snr_centered = torch.fft.fftshift(snr)

fig, axes = plt.subplots(1, 3, figsize=(15, 5))
```

```
# Frequency response
im0 = axes[0].imshow(H_centered.numpy(), cmap='viridis')
axes[0].set_title('|H(f)| - Blur Response')
plt.colorbar(im0, ax=axes[0], fraction=0.046)

# Posterior variance in frequency domain
im1 = axes[1].imshow(torch.log10(var_freq_centered + 1e-10).numpy(), cmap='h
axes[1].set_title('log_10(Posterior Variance) in Freq')
plt.colorbar(im1, ax=axes[1], fraction=0.046)

# SNR per frequency
im2 = axes[2].imshow(torch.log10(snr_centered + 1e-10).numpy(), cmap='coolwa
axes[2].set_title('log_10(SNR) per Frequency')
plt.colorbar(im2, ax=axes[2], fraction=0.046)

for ax in axes:
    ax.axis('off')
plt.tight_layout()
plt.show()

print("Notice: High variance (bright) corresponds to low |H(f)| (dark in fir
print("This is expected - uncertainty is highest where blur destroys informa
```



```
Notice: High variance (bright) corresponds to low |H(f)| (dark in first plo
t)
This is expected - uncertainty is highest where blur destroys information!
```

Summary & Next Steps

What we've established:

1. Blur destroys high-frequency information (|H(f)| -> 0)
2. Wiener filter provides provably correct posterior variance
3. Variance is HIGH where |H(f)| is LOW (as expected)

Next steps for CURE:

1. Compare Wiener calibration vs DPS (diffusion) calibration
2. Compute frequency-resolved ECE to quantify miscalibration
3. Train recalibrator to correct DPS uncertainty using Wiener as reference