Core Concept

Haveli-GAN is a **conditional GAN** based on an encoder-decoder architecture. The "condition" is not just the damaged image but also a **damage mask** and a **style vector**, making it a multi-modal system. This allows the model to learn not only *what* to reconstruct but also *how* to reconstruct it in the correct artistic style of Rajasthani frescoes.

1. Model Architecture

The model consists of three main components: a Generator, a Style Encoder, and a Discriminator.

A. The Generator

The Generator's job is to take the damaged fresco and restore it. We'll use a sophisticated U-Net-like architecture with attention and style-modulation.

- **Architecture: Style-Semantic U-Net**
  - **Encoder:** A series of down-sampling convolutional blocks. The initial layers should use **Gated Convolutions**. Unlike standard convolutions, gated convolutions learn to handle irregular holes and damage masks, which is perfect for the diverse damage types you listed (flaking, plaster loss, etc.).
  - **Bottleneck:** At the deepest part of the U-Net, a **Self-Attention mechanism** is crucial. This allows the model to learn long-range dependencies across the image. For example, to restore a damaged face, it can draw information from an intact shoulder or other parts of the body to ensure semantic consistency.
  - **Decoder:** A series of up-sampling (Transposed Convolution) blocks with skip connections from the encoder. The key innovation here is the use of **Adaptive Instance Normalization (AdaIN)** layers. Instead of standard normalization, AdaIN injects the "style" of the fresco into the network at each resolution level.

Concept

- **Inputs to Generator:**
    1. **Damaged Image:** The input fresco needing restoration.
    2. **Damage Mask:** A binary mask where white pixels indicate missing/damaged areas and black pixels are intact. This explicitly tells the generator where to focus its efforts.
    3. **Style Vector (w):** A latent vector that encapsulates the specific Rajasthani art style. This is produced by the Style Encoder.

B. The Style Encoder

This small network's only job is to look at a reference painting and distil its artistic style into a single vector.

- **Architecture:** A simple Convolutional Neural Network (CNN). You can use the initial layers of a pre-trained network like VGG-16 or ResNet-50.
- **Input:** A **Style Reference Image**. This can be another, intact fresco from the same haveli or artist, or even a high-quality digital swatch of the target style.
- **Output:** A style vector w. This vector is then fed into the AdaIN layers of the Generator.

C. The Discriminator

The Discriminator learns to differentiate between real, original frescoes and the fakes created by the Generator.

- **Architecture: Conditional PatchGAN** A standard GAN discriminator outputs a single probability (real/fake). A **PatchGAN** is more effective for images; it outputs a grid of probabilities, where each cell evaluates the "realness" of a different patch of the image. This forces the generator to produce realistic textures and details across the entire fresco.
- **Conditional Input:** The Discriminator doesn't just look at the output image. It looks at the **output image concatenated with the original damaged image**. This makes it a "conditional" discriminator. It asks a more sophisticated question: "Is this a plausible restoration *given this specific damage*?" This prevents the generator from ignoring the original context and hallucinating unrelated content.

2. The Loss Function: Teaching the Model

To train the network, you need a combination of loss functions to ensure the output is simultaneously realistic, faithful to the original, semantically correct, and stylistically accurate.

The total loss for the generator (L_G) will be a weighted sum:

$$L_G = \lambda_{adv} L_{adv} + \lambda_{rec} L_{rec} + \lambda_{perc} L_{perc} + \lambda_{style} L_{style}$$

- **Adversarial Loss (L_adv):** The classic GAN loss. It pushes the Generator to create images that can fool the Discriminator.
- **Reconstruction Loss (L_rec):** A pixel-by-pixel comparison between the generated image and the ground-truth original image. An **L1 Loss** (Mean Absolute Error) is preferred over L2 (MSE) as it produces sharper, less blurry results.
- **Perceptual Loss (L_perc):** This is for semantic understanding. The generated image and the ground truth image are passed through a pre-trained VGG-16 network. The loss is the difference between their feature map activations at deeper layers. This ensures that a restored eye *looks like an eye* in a high-level feature sense, not just a collection of correct pixels.
- **Style Loss (L_style):** This is crucial for being "style-aware." It's similar to perceptual loss, but it compares the Gram matrices of the VGG feature maps. The Gram matrix captures texture and style information. This loss forces the generated restoration to match the brushwork, color palette, and texture of the original art style.

3. Training Workflow

1. **Dataset Preparation:** For each of your 5000+ images, you need to create a training triplet:
    - **Ground Truth:** The original, undamaged fresco image.
    - **Damaged Input:** The ground truth image with artificial damage applied (cracks, fading, missing sections). You can script this to simulate the types of damage you listed.
    - **Damage Mask:** The binary mask corresponding to the artificial damage.
2. **Training Loop:**
    - Fetch a batch of training triplets. For the **Style Reference Image**, you can simply use the Ground Truth image itself, forcing the model to learn to reconstruct in the *same* style.
    - **Train the Discriminator:** Show it a mix of real frescoes (ground truth) and fake ones (output from the generator). Update its weights.
    - **Train the Generator:** Feed a damaged image, its mask, and its style reference into the system. Calculate the combined loss (L_G) and update the Generator's and Style Encoder's weights.
    - Repeat for many epochs until the generated results are of high quality.

This **Haveli-GAN** architecture is robust and specifically designed to tackle the challenges you've outlined, providing a strong foundation for your project on restoring Rajasthani heritage frescoes.

How to Use the Model

1. **Set Up Your Dataset:**
   - Place your training images into the data/ subfolders as described in the project structure.
   - **Crucially**, ensure that for every image_001.jpg in train_damaged, there is a corresponding image_001.jpg in train_ground_truth and train_masks. If your file names are different, you MUST modify the __getitem__ method in dataset.py.
2. **Start Training:**
   - Open your terminal or command prompt and navigate to the haveli-gan/ directory.
   - Run the training script:

     Bash : python train.py
   - The training will begin, showing a progress bar from tqdm. This process will be very computationally intensive and will take a long time, likely days, on a good GPU (like an NVIDIA RTX 3080/4080 or better).
   - Every 5 epochs, a sample restored image will be saved to the outputs/ folder, and model checkpoints will be saved to checkpoints/.
3. **Perform Inference:**
   - Once training is complete (or you have a checkpoint you're happy with), open inference.py.
   - Update the GEN_CHECKPOINT and STYLE_ENC_CHECKPOINT variables to point to your saved model files.
   - Update the DAMAGED_IMG_PATH, MASK_PATH, and STYLE_REF_PATH to point to the new images you want to process.
   - Run the script:

     Bash: python inference.py
   - The restored image will be saved to the location specified by OUTPUT_PATH.

Project Structure

```
Haveli-GAN/
├── data/
|   ├── ...
├── checkpoints/
├── outputs/
├── model.py          # Generator, Discriminator, Style Encoder
├── loss.py           # VGG Perceptual Loss (which imports the official vgg19)
├── dataset.py
├── train.py
└── inference.py
```