**Model Architecture and Feature Engineering for Floor Plan Generation**

**Understanding Your Dataset**

From your updated metadata file, I can see:

1. **Detailed spatial information**:
   * Room counts by type
   * Room areas in square feet
   * Total area and dimensions by plot size
   * Consistent patterns within each plot category (5/10/20 Marla)
2. **Data structure**:
   * Multiple versions (V01-V04) of each unique floor plan, likely representing augmented variants
   * Consistent image dimensions for each plot size category
   * Rich feature set including room counts, areas, and spatial relationships

**Recommended Model Architecture**

For your floor plan generation system, I recommend a **Conditional Generative Adversarial Network (CGAN)** with a U-Net backbone for the generator. Here's why and how to structure it:

**1. Generator Architecture**

**U-Net with Conditional Inputs**:

* **Input Layer**:
  + Random noise vector (latent space, e.g., 100-dimensional)
  + Conditional inputs (encoded user requirements)
* **Encoder Path**:
  + Downsampling blocks (Conv2D → BatchNorm → LeakyReLU)
  + Progressive reduction of spatial dimensions while increasing feature channels
  + Feature extraction from conditional inputs at each level
* **Bottleneck**:
  + Dense transformation of conditional parameters
  + Integration with deepest encoder features
* **Decoder Path**:
  + Upsampling blocks (TransposeConv2D → BatchNorm → ReLU)
  + Skip connections from encoder (crucial for preserving spatial information)
  + Conditional feature injection at each level
* **Output Layer**:
  + Multi-channel output (23 channels, one per room type)
  + Softmax activation for room type classification
  + Resolution matching target floor plan dimensions

**2. Discriminator Architecture**

**PatchGAN Discriminator with Conditioning**:

* Evaluate patches of the image rather than the entire image
* Concatenate conditional inputs with image channels
* Output a matrix of realism scores rather than a single value
* Incorporate architectural validity checking

**3. Feature Engineering**

Based on your dataset, here are the critical features to engineer:

**A. Conditional Inputs (User Requirements)**:

1. **Plot Size Category**: One-hot encoding (5 Marla, 10 Marla, 20 Marla)
2. **Room Counts Vector**: Normalized count of each room type
   * Bathrooms (1-4)
   * Bedrooms (1-3)
   * Kitchen (usually 1)
   * Drawing Room
   * Lounge/Sitting Area
   * Etc.
3. **Area Distribution Requirements**:
   * Target total area
   * Percentage allocation for key rooms (bedroom, bathroom, etc.)
   * Derived from your area data
4. **Adjacency Requirements**:
   * Which rooms should be connected
   * Which rooms need external walls
   * Created as a graph representation

**B. Spatial Features**:

1. **Room Arrangement Patterns**:
   * Extract topological relationships between rooms
   * Create a graph representation of room adjacencies
   * Encode common patterns (e.g., bedrooms near bathrooms)
2. **Area-to-Boundary Ratios**:
   * Calculate compactness metrics for each room
   * Help ensure realistic room shapes
3. **Circulation Features**:
   * Identify pathways and connections
   * Encode accessibility patterns

**C. Preprocessing Transformations**:

1. **Image Normalization**:
   * Scale pixel values to [-1, 1]
   * Standardize dimensions within each category
2. **Room Type Encoding**:
   * Create one-hot encoded room type masks
   * Convert RGB color-coding to categorical channels
3. **Area Normalization**:
   * Scale area values relative to the total floor plan area
   * Create percentage-based representations

**Training Approach**

For training this model effectively:

1. **Progressive Growing Strategy**:
   * Start with low-resolution floor plans (64×128)
   * Gradually increase to full resolution (608×1088, 849×1570, 1209×2170)
   * Add detail in stages (room boundaries → room types → fixtures)
2. **Multi-stage Training Pipeline**:
   * Stage 1: Train on room boundary generation
   * Stage 2: Train on room type assignment
   * Stage 3: Fine-tune on full floor plans
3. **Loss Functions**:
   * **Adversarial Loss**: Wasserstein GAN with gradient penalty (WGAN-GP)
   * **Room Type Classification Loss**: Categorical cross-entropy per pixel
   * **Room Count Consistency Loss**: Ensure generated plans match requested room counts
   * **Room Area Loss**: Ensure generated room areas match requirements
   * **Architectural Validity Loss**: Penalize invalid configurations (disconnected rooms, etc.)

**Evaluation Metrics**

To assess the quality of your generated floor plans:

1. **Requirement Satisfaction**:
   * Accuracy of room counts compared to input requirements
   * Room area distribution match with requirements
2. **Architectural Validity**:
   * Room connectivity assessment
   * Wall consistency and thickness
   * Door placement logic
3. **Perceptual Quality**:
   * FID (Fréchet Inception Distance) between generated and real floor plans
   * User preference studies (qualitative)

**Addressing Dataset Challenges**

1. **Limited Data Size**:
   * Implement extensive data augmentation (beyond current V01-V04)
   * Use transfer learning from pretrained networks
   * Synthetic data generation for rare configurations
2. **Aspect Ratio Handling**:
   * Maintain separate generators for each plot size
   * Or use conditional resizing in a single model
3. **Class Imbalance**:
   * Weight loss functions by room type frequency
   * Oversample rare room configurations

**Implementation Considerations**

1. **Framework Selection**:
   * TensorFlow/Keras for easier implementation of custom layers
   * PyTorch for more flexible research experimentation
2. **Computational Requirements**:
   * Train on GPUs with sufficient VRAM (>8GB)
   * Implement mixed precision training to reduce memory footprint
3. **Phased Development**:
   * Start with a single plot size (10 Marla has most samples)
   * Develop a simplified proof-of-concept
   * Gradually add complexity to both model and requirements

Would you like me to elaborate on any specific aspect of this architecture design or feature engineering approach? Or would you prefer to discuss implementation strategies for a particular component?