Final Project Report

By Md Humaun Kabir

Introduction

Aerosol Jet Printing (AJP) is a mask-less, direct-write additive-manufacturing technology capable of producing micrometre-scale conductive features on both planar and non-planar substrates. Because it can deposit a wide range of functional inks—including metals, polymers, and biomaterials—AJP has become a key enabler for flexible electronics, wearables, antenna fabrication, biosensors, and multilayer interconnects. The final performance of these devices is tightly linked to the morphology of the printed traces: variations in trace width, edge roughness, continuity, or overspray directly affect electrical resistance, mechanical reliability, and dimensional accuracy.

In practice, morphology is governed by a high-dimensional and non-linear interaction of process parameters such as atomizer voltage (ATM), sheath-to-carrier-gas focusing ratio (FR), carrier-gas flow rate (CGFR), and stage speed (SS).

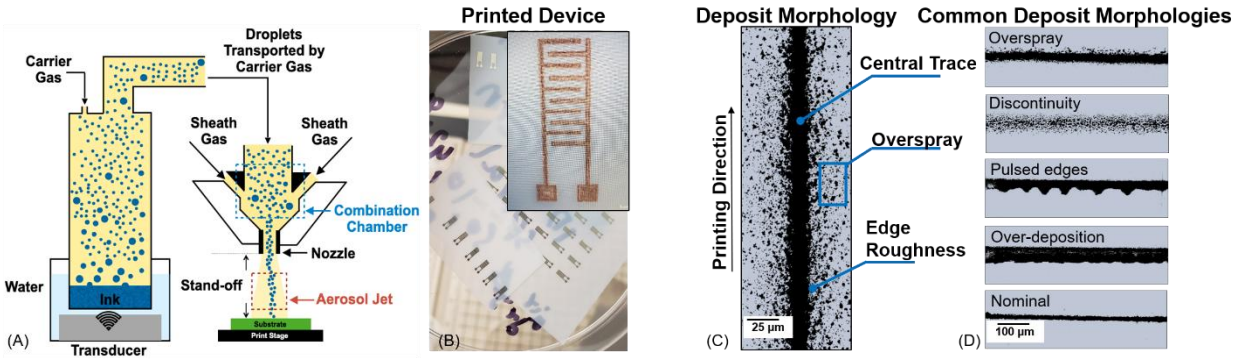

Figure 1: (A) Ultrasonic AJP workflow diagram. (B) Silver traces deposited on a flexible film (1 mm scale bar). (C) Single deposit highlighting key morphology metrics (25 µm scale bar). (D) Examples of typical AJP morphology classes (100 µm scale bar).

Recent breakthroughs in deep-learning—especially GAN (Generative Adversarial Network) a powerful, data-driven way to predict AJP outcomes. By adding process parameters as conditional inputs, a **cGAN** learns to translate a random noise vector and a parameter set into a high-resolution morphology image, effectively standing in for the real print. Unlike straightforward regression, a cGAN embraces the natural randomness of droplet deposition and can produce many plausible variations for the same settings.

Yet AJP itself is highly complex. Key shape features—trace width (L_p), edge roughness (R_x), continuity (ψ), and overspray area (φ)—depend on a tangled, nonlinear mix of factors: atomizer

voltage (ATM), carrier-gas flow rate (CGFR), focusing ratio (FR), stage speed (SS), nozzle–substrate gap, and ink rheology. Traditional tuning, based on design-of-experiments (DoE) trials and operator know-how, is

- Material-intensive – silver nano-inks cost > $400 USD per 100 mL
- Time-consuming – exploring a modest 5-factor, 3-level DoE exceeds 200 prints
- Local in scope – empirical maps rarely generalise beyond the explored window

A digital surrogate capable of predicting morphology a priori would dramatically accelerate process development, reduce ink waste, and enable closed-loop control. Physics-based computational fluid dynamics (CFD) or multiscale droplet-impact simulations can, in principle, predict morphologies, but they require high-fidelity nozzle flow models, contact-line dynamics, and ink rheology inputs—currently infeasible for fast design iteration. Recent breakthroughs in deep generative modelling, particularly Generative Adversarial Networks (GANs), provide a compelling data-driven alternative. By learning the underlying distribution of training images, a GAN can synthesise realistic images that capture subtle stochastic features beyond the reach of deterministic models.

A Conditional GAN (cGAN) extends this paradigm by incorporating control variables. In an AJP context, concatenating latent noise z with encoded process parameters c = {ATM, FR, CGFR, SS} allows the generator G(z|c) to output morphology images specific to a chosen parameter set. The discriminator D(x|c) assesses realism and parametric consistency, guiding G to honour physical causality as learned from data.

We present AJP-cGAN, the first (to our knowledge) conditional GAN trained on an extensive experimental dataset to predict AJP deposit morphologies. Our work makes the following contributions:

1. Comprehensive Image Dataset – 1950 traces were printed under 650 parameter combinations, imaged at 300×, binarised, cropped, and augmented to yield 23,400 labelled images (256 × 256 px).
2. Lightweight, Parameter-Aware Architecture – We designed a generator–discriminator pair with embedded parameter vectors that fits within a single consumer-GPU memory footprint while maintaining high resolution.
3. Multi-level Evaluation Framework – We introduce a dual-metric strategy combining AJP-specific morphological descriptors (Lw, ψ, Rq, φ) with canonical image similarity metrics (IoU, Precision, F1) for rigorous validation.
4. Insight into Process–Structure Links – By performing virtual parameter sweeps, we reveal how each process variable shapes morphology and identify regimes where the model diverges, informing targeted experimentation.

Dataset Description

2.1 Experimental Setup

- Ink: UTDAg40X silver nano-ink (40 wt.% concentration)

- Substrate: Glass microscope slides
- Nozzle: 30 G (233 ± 3 μm inner diameter)
- Printer: IDS NanoJet system
- Print Parameter Matrix: Four primary variables (defined below) were combinatorially varied to span an industrially relevant design space.
- Total Experiments: 650 unique parameter sets × 3 replicates = 1950 traces

2.2 Printing Parameters

The dataset used for training the residual cGAN was constructed from systematically varied process parameters in Aerosol Jet Printing (AJP), with each parameter discretized into a finite set of representative levels. As shown in Table 1, the four key parameters include Carrier Gas Flow Rate (CGFR), Focusing Ratio (FR), Atomizer Voltage (ATM), and Stage Speed (SS). These variables were selected for their direct influence on aerosol transport dynamics, deposition resolution, and trace morphology. Due to the challenges associated with modeling continuous variables in GAN frameworks—such as instability and poor generalization in sparsely sampled regions—each parameter was treated as a categorical variable. Discrete levels were chosen based on experimentally validated process windows to ensure both physical feasibility and sufficient representation across the parameter space. This discrete encoding enabled robust conditional embedding within the GAN architecture and allowed the model to learn distinct morphology patterns associated with specific parameter combinations.

Table 1: Key AJP process parameters used as conditional inputs for the cGAN and their primary effects on deposit morphology

Parameter	Range / Levels	Functional Role	Parameter
Carrier Gas Flow Rate (CGFR)	6, 9, 12 sccm	Controls the number of droplets transported in the aerosol stream.	Carrier Gas Flow Rate (CGFR)
Focusing Ratio (FR)	1 – 15	Governs aerosol stream collimation, directly affecting jet diameter and trace width.	Focusing Ratio (FR)
Atomizer Voltage (ATM)	23.5, 30, 36.5 V	Influences droplet size and production rate, thereby modifying overspray and surface coverage.	Atomizer Voltage (ATM)
Stage Speed (SS)	0.2, 0.4, 0.6, 0.8, 1 mm s^{-1}	Controls the effective material deposition rate, impacting trace continuity and thickness.	Stage Speed (SS)

2.3 Image Acquisition and Preprocessing Pipeline

All morphology images were acquired using an optical microscope at 300× magnification, resulting in native RGB images with a resolution of 2048 × 1536 pixels. To prepare the dataset for deep learning training, a structured preprocessing pipeline was implemented, as illustrated in Figure 3.

1. Otsu Thresholding was applied to convert high-resolution RGB micrographs into binary masks, segmenting the printed trace from the background. This unsupervised, histogram-based thresholding technique automatically identifies the optimal cutoff value to separate foreground (deposited material) from background, ensuring consistent and reproducible morphology extraction. Binary representation simplifies the learning task, focuses the model on structural features (e.g., line width, continuity, overspray), and reduces computational complexity during training.
2. Symmetric Cropping was performed to isolate the central region containing the trace morphology. Each image was cropped to a size of 1024 × 512 pixels, removing uninformative background and ensuring consistent framing across the dataset.
3. Data Augmentation was introduced to improve generalization and model robustness. Each cropped image was transformed using 180° rotation and horizontal/vertical flips, resulting in 12 augmented variants per original image. This augmentation preserved the geometric and morphological integrity of the deposit while increasing sample diversity across the training set.
4. Final Resize was applied to convert each image to a grayscale 512 × 512 pixel format, compatible with GPU memory constraints and optimized for convolutional network input. The grayscale representation retains all relevant morphological features while significantly reducing the input dimensionality.

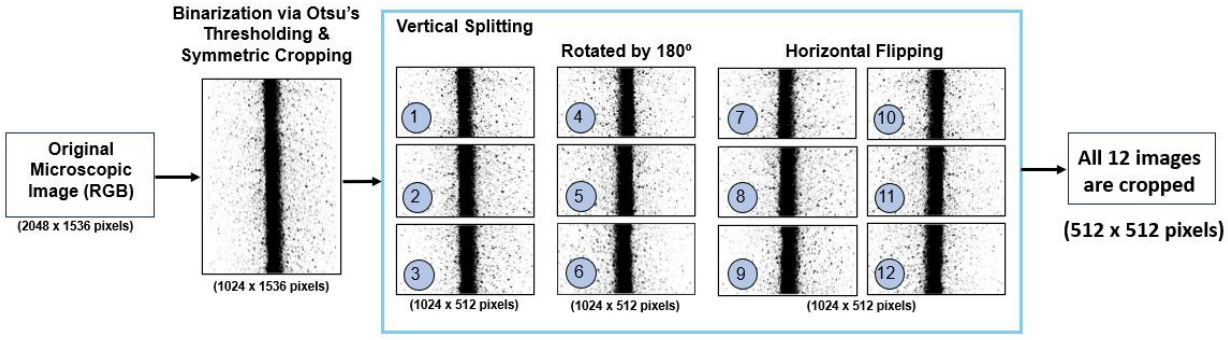

Figure 3: Overview of the image preprocessing and augmentation pipeline applied to microscope-captured AJP morphologies.

2.4 Final Dataset Statistics

The final dataset consisted of 23,400 grayscale binary images, derived from 1,950 unique experimental samples and their 12-fold augmented versions. Each image was labeled with a 4-dimensional vector representing the corresponding process parameters: Atomizer Voltage (ATM), Focusing Ratio (FR), Carrier Gas Flow Rate (CGFR), and Stage Speed (SS). These labels were

discretized into categorical levels as described in Section 2.3 and encoded for conditional generation.

Network Description

The predictive engine behind AJP-cGAN is a Conditional Generative Adversarial Network consisting of a Generator G and a Discriminator D. Training is an adversarial game: G tries to synthesise morphology images that fool D, while D learns to distinguish real from synthetic given the same process parameters.

Latent Noise and Conditioning Vector

- Latent vector z (dimension = 100) – sampled from $\mathcal{N}(0, 1)$, it injects stochasticity so G can produce diverse but plausible morphologies even for identical process settings.
- Conditioning vector c = [ATM, CR, FR, PS] – each categorical parameter is first mapped to an integer class index and then to a learnable embedding. The concatenated embeddings form a low-dimensional description of the physical state.

The final input to the generator is [z ‖ embed(c)], while the discriminator ingests the image together with spatially broadcast label maps derived from the same embeddings. This design forces both networks to internalise the causal link between parameters and geometry.

Generator:

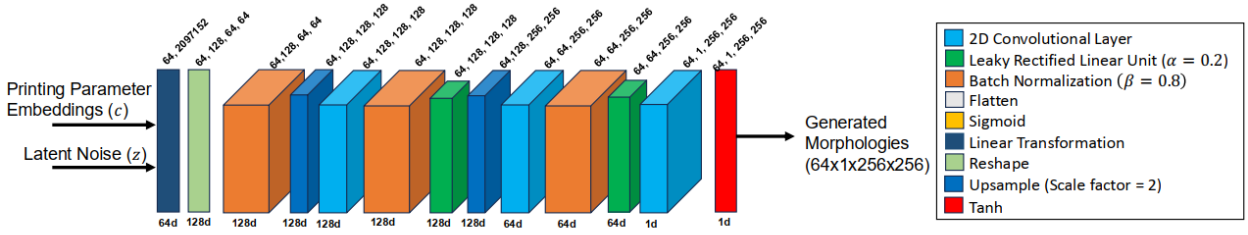

Figure 3: Generator Architecture

```
class Generator(nn.Module):
    def __init__(self, latent_dim, num_classes_ATM, num_classes_CR,
                 num_classes_FR, num_classes_PS, img_size=256):
        super(Generator, self).__init__()
```

Creates a subclass of nn.Module; we pass latent size, label vocab sizes, and image size so the constructor stays flexible.

```
self.label_emb_ATM = nn.Embedding(num_classes_ATM, num_classes_ATM)
self.label_emb_CR  = nn.Embedding(num_classes_CR,  num_classes_CR)
self.label_emb_FR  = nn.Embedding(num_classes_FR,  num_classes_FR)
self.label_emb_PS  = nn.Embedding(num_classes_PS,  num_classes_PS)
```

Each discrete label is converted into a dense vector; we purposely use one-hot–like length (e.g., 3 → 3 dims) to keep interpretation simple.

```
self.init_size = img_size // 4
self.l1 = nn.Sequential(
    nn.Linear(latent_dim + num_classes_ATM + num_classes_CR +
              num_classes_FR + num_classes_PS,
              128 * self.init_size ** 2))
```

Concatenates noise + all embeddings, then projects to a 128-channel 64×64 tensor—our starting feature map.

```
self.conv_blocks = nn.Sequential(
    nn.BatchNorm2d(128),
    nn.Upsample(scale_factor=2),           # 64→128
    nn.Conv2d(128, 128, 3, 1, 1),
    nn.BatchNorm2d(128, 0.8),
    nn.LeakyReLU(0.2, inplace=True),

    nn.Upsample(scale_factor=2),           # 128→256
    nn.Conv2d(128, 64, 3, 1, 1),
    nn.BatchNorm2d(64, 0.8),
    nn.LeakyReLU(0.2, inplace=True),

    nn.Conv2d(64, 1, 3, 1, 1),
    nn.Tanh())
```

Uses nearest-neighbour upsample + conv to minimise checkerboard artefacts. Final Tanh() maps pixel range to [-1, 1].

```
def forward(self, noise, labels_ATM, labels_CR, labels_FR, labels_PS):
    # 4  embed → concatenate with noise
    x = torch.cat((noise,
                   self.label_emb_ATM(labels_ATM),
                   self.label_emb_CR(labels_CR),
                   self.label_emb_FR(labels_FR),
                   self.label_emb_PS(labels_PS)), dim=-1)
    out = self.l1(x)                                  # FC → vector
    out = out.view(out.size(0), 128, self.init_size, self.init_size)
    return self.conv_blocks(out)                      # upsample to 256²
```

Produces a 1 × 256 × 256 binary morphology mask conditioned on all four parameters.

1. Inputs
   • Latent noise z → shape (B, 100).
   • Four label indices → ATM (0-2), CR (0-2), FR (0-14), PS (0-4).
2. Label Embedding
   nn.Embedding turns each index into a dense vector:
   ATM→3 D, CR→3 D, FR→15 D, PS→5 D.
   Resulting shapes: (B, 3), (B, 3), (B, 15), (B, 5).
   Total label length = 26.
3. Concatenate noise + embeddings → (B, 126) (100 + 26).
4. Fully-Connected Projection
   Linear(126 → 128 × 64 × 64 = 524 288) flattens to (B, 524 288) then reshapes to a 4-D tensor (B, 128, 64, 64).
   This 64 × 64 feature map is the coarse "canvas."
5. BatchNorm2d(128) stabilises channel statistics before upsampling.
6. Upsample-Conv Block #1
   • Upsample(scale_factor=2) doubles the grid → 128 × 128.
   • Conv2d(128→128, 3×3, stride 1, pad 1) blends neighbours.
   • BatchNorm2d(128, momentum 0.8) + LeakyReLU(0.2) add stability & non-linearity.
7. Upsample-Conv Block #2
   • Upsample again → 256 × 256.
   • Conv2d(128→64, 3×3, stride 1, pad 1) reduces channels for memory efficiency.
   • BatchNorm2d(64, 0.8) + LeakyReLU(0.2).
8. Output Layer
   Conv2d(64→1, 3×3, stride 1, pad 1) converts 64 features to 1 grayscale channel.
9. Tanh Activation scales pixels to [-1, 1], matching discriminator expectations.
10. Final Output: tensor shape (B, 1, 256, 256)—a synthetic AJP morphology mask conditioned on ATM, CR, FR, and PS.

Although, generator and discriminator was designed separately, their interdependency has led us to cross-check and design it simultaneously.

- A summary of our approaches is below:
  Generator → Discriminator The images G produces become D's "fake" inputs.
- **Shared hyper-parameters** Batch size, learning rates, and loss functions must match.
- **Consistent normalisation** Both blocks expect pixel values scaled to [-1, 1].
- **Opposite sampling paths** G repeatedly **upsamples** noise to a 256 × 256 image, while D must **downsample** that same resolution back to a compact feature vector to decide **REAL** vs **FAKE**.

How we wired this up

1. **Discriminator step** During each training batch D receives
   • real_images, and
   • fake_images = generator(noise, labels) — so D's forward pass depends on G's output.
2. **Generator step** Right after updating D, we freeze D's weights and evaluate
   validity = discriminator(fake_images, labels);
   G's loss is computed from that validity score, so G's optimisation depends on D's forward pass.

Sampling symmetry

Generator (upsampling)	**Spatial size**
Seed reshape	64 × 64
Upsample × 1	128 × 128
Upsample × 2	256 × 256
Discriminator (downsampling)	**Spatial size**
Input image	256 × 256
Conv stride 2 × 1	128 × 128
Conv stride 2 × 2	64 × 64
Conv stride 2 × 3	32 × 32
Conv stride 2 × 4	16 × 16
Conv stride 2 × 5	8 × 8 → 4 × 4 (final)

This mirrored up-/down-sampling ensures that every detail the generator adds can be examined at the right scale by the discriminator, keeping the training adversarial yet balanced.

Training Hyperparameters

```
optimizer_G = Adam(generator.parameters(), lr=2e-4, betas=(0.5, 0.999))
optimizer_D = Adam(discriminator.parameters(), lr=2e-4, betas=(0.5, 0.999))
adversarial loss = nn.BCELoss()
```

Standard DCGAN settings: lr = 0.0002 and $\beta_1 = 0.5$ foster stable convergence.

Why This Works for AJP

- Conditional embeddings let the network learn unique texture/edge characteristics for each process setting.
- Down-sampling discriminator with progressive channels efficiently captures multi-scale edge roughness.
- Nearest-neighbour upsampling in the generator minimises ringing artefacts critical for thin lines.

Evaluation

File Handling and Data Processing Infrastructure

Zip Files Script: Configures Google Drive mounting, defines paths (zip_path, output_csv, temp_extract_dir), creates temp_extract_dir with os.makedirs, and implements a zipfile.ZipFile loop with tqdm to extract/process ~23k images, saving metrics to CSV (df.to_csv) and cleaning up with os.remove/os.rmdir.

Batches Script: Sets up Drive mounting, defines input_folder/output_folder, creates output_folder with os.makedirs, and writes a loop with os.listdir and tqdm to process folder images, saving metrics to CSV (e.g., A23_C9_F15_P2_metrics.csv), integrating process_image.

Integration: Links process_image function, handling inputs (e.g., image_path, original_name) and errors for robust batch processing.

The other part of our combined work is the design of code for

- Defining Adam optimization
- Defining Loss Function to plot for generator and discriminator
- Appropriate label generation

Results

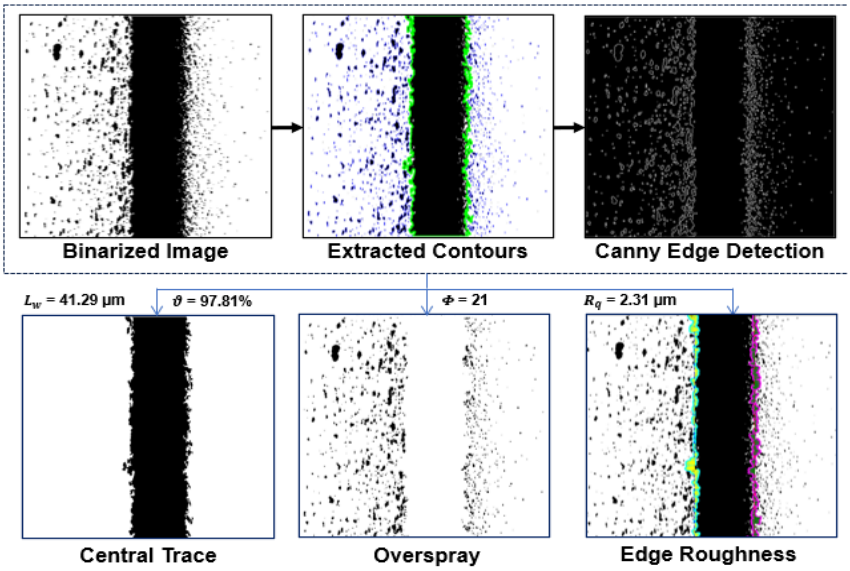

Figure 4: Workflow for quantifying AJP-specific morphology attributes using OpenCV. Example trace printed with ATM = 30 V, CGFR = 6 sccm, FR = 13, and SS = 0.4 mm/s. Measured values: L_w = 41.29 µm, ϑ = 97.81%, R_q = 2.31 µm, and Φ of 21%.

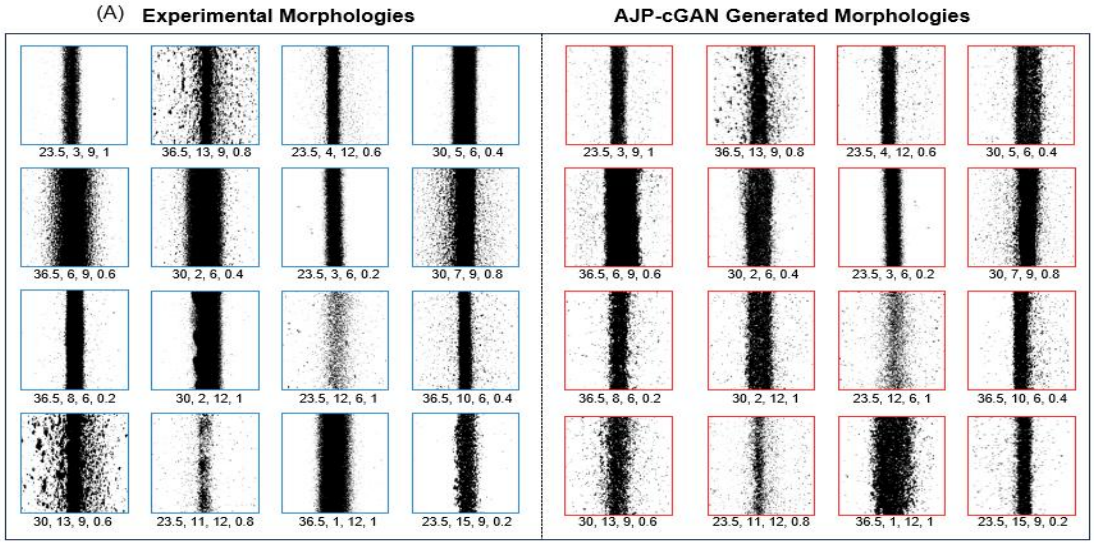

Figure 5: Comparison of experimental and AJP-cGAN-generated morphologies

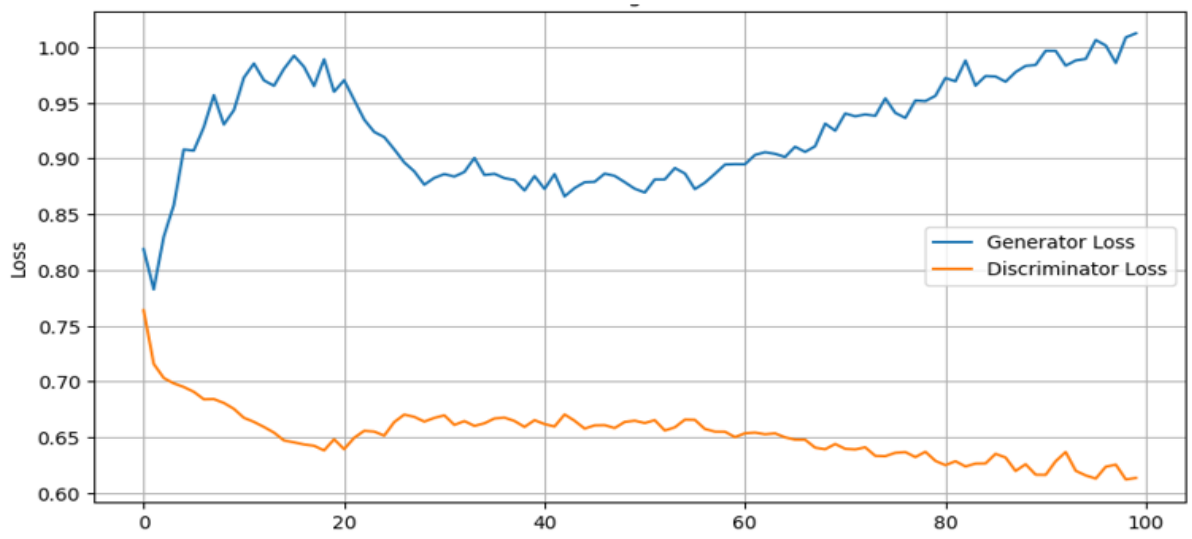

Figure 6: Training Loss for Generator and Discriminator

We are targeting for a particular iteration sample where loss from Generator is minimum and loss from discriminator is minimum. From the observation, we tend to get it around 42 epochs.

Summary and Conclusion

We demonstrated that a cGAN-based model can predict AJP morphology conditioned on process parameters. The approach showed:

- High-quality image generation
- Preservation of statistical morphology features
- Improved workflow efficiency via virtual prototyping

Future Work

- Integrate physics-informed losses and constraints
- Explore conditional diffusion models for higher fidelity
- Model dynamic processes (time evolution of morphology)
- Predict functional performance (e.g., conductivity)