Query based Image Synthesizer and multi document summarizer

Prescilla Pragasam
KLA Plus
prescilla.pragasam@kla-tencor.com

Geeta Jakkamsetti
KLA Plus
geeta.jakkamsetti@kla-tencor.com

Abstract

In the world of process control product design, documentation has become an every increasing entity that Users, developers, testers spend enormous amount of time in retrieving information to clarify their queries. In addition, as products evolve engineers depend on these process control tools to perform high quality checks on various algorithms. Due to stringent deadlines, rigid tool time slots are made available for engineers that are insufficient and paced at odd hours.

In this paper, we propose a framework called Generative Imager and Examiner of documents (GenIE) that is aimed at effort optimization in terms of document reliance and process tool dependency. GenIE framework is constructed with 2 models: a NLP based model to solve the documentation reliance problem and a text to image synthesizer that aims to minimize process tool utilization. The proposed solution framework is to build a query based chatbot where user will be able to query and retrieve details regarding any in-house process control SW related topic or generate die images for a user query. A Defect Die Images repository called KLA-DDI dataset comprising of 3000 defect images and KLA-QnA dataset with 100 QnA pairs are created to feed into this framework. The image generation model is trained for 1850 epochs to generate defect images that match the user query. These generated images are validated for correctness through visual inspection and for performance through Human rank and generator/discriminator loss plots. Our model achieved 63 percent test accuracy.

1. Introduction

In the growing semiconductor industry, the need for new features, detection algorithms, different types of wafer handling, high resolutions optics paths grows and process control tools evolve every year. With this, the need to record these changes in documents gains significance. Time involved in searching these documents to resolve a user query is humongous. In addition, rigid tool time slots provide lesser tool time duration for engineers to perform high quality checks for the above stated changes. At times, they are required to work around the clock to honor the allocated time slots. The above stated problems are aimed to be solved with GenIE framework comprising of NLP and Stack-GAN models. While the NLP model aims to resolve the user difficulty to search various documents with a chatbot solution, the Stack GAN model intends to reduce tool dependencies by generating required defect images for a user query. The NLP model solution of this framework converts documents and user queries to respective text embedding. These text embedding are trained by fine-tuned Llama2 7B chatbot pre-trained model along with Guanaco +KLA QnA dataset to provide meaningful interpretation. In addition, the NLP model converts users queries on image generation into text embedding which are used by Stack-GAN model in the framework to generate Defect images images.The details of the NLP model handing user query on existing documents are provided by the co-author in the supplementary section. while this paper focuses on the Stack GAN solution of the framework for image synthesis conditioned by input query text.

1.1. Problem statement

The semiconductor process control tools undergo so many changes to cater to the needs of semiconductor manufacturing. The process tools used in semiconductor manufacturing detect defects in dies. The Algorithms in these process control tools is designed to detect different types of defects for different optics settings. Due the expensive built of these process control tools, only few of them are available in house. In the trend of quick time to market, the need for quick high quality verification becomes inevitable. To reduce the dependency on tools, a GenIE framework with stack-GAN is built to generate die images with desired defects at desired locations. These images will be used as a input to SW tools for quick verification of new and existing algorithms instead of depending on process control tools.This will provide flexibility to engineers and reduce the reliance on process tools.
A typical text to image GAN consisting of two blocks - a generator and a discriminator. This GAN is the basic block

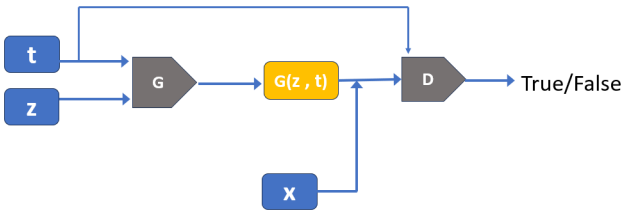


Figure 1. Block diagram of Text conditioned GAN

for image generation and two of these are stacked to generate high quality images. The input fed to the generator is the noise z obtained by random sampling. This noise input vector to GAN varies every time due to random sampling . The output synthesised by the generator will initially be a fake image. This image is fed to the discriminator. The discriminator discriminates the generated image against the ground truth reality until the generator synthesizes fake images that can fool the discriminator. To output the synthesized image as a combination of both image and query text, the GAN will be fed with an input image along with conditional text input t. Hence the synthesized image G(z,t) will now be a function of both input image z and input text t.

1.2. Inputs and outputs

The inputs and outputs for the image generation model are the input noise image z and text embedding. Defect images collected from process control tools are classified, captioned, pre-processed, resized and organised into train and test images. These real images belong to six classes namely - small particle, large particle, clusters, chippings, scratch, peelings. These images which are processed by stack-GAN comprises of two GANs: Stage1 GAN and Stage II GAN. The input and output flow of each block of Stack-GAN are explained in this section.

Conditioning Augmentation network
NLP model receives input {user query} and outputs {text embedding vector}. Conditioned Augmentation Network (CAN) processes {text embedding} to {text conditioning variable}

Stage I GAN
Stage I generator processes two inputs: {noise image, text conditioning variable}, and synthesises a low resolution {stage I defect image}.stage I discriminator gets three inputs: {stage I generated image, real image, text embedding}, evaluates and returns probabilities {0,1} for real and fake images based on its evaluation.

Stage II GAN
Stage II generator gets two inputs {Stage I defect image, text conditioning variable}and generates a high resolution {stage II defect image}. Stage 2 discriminator processes three inputs: {Stage 2 generated image, real image, text embedding}, validates low resolution image and outputs probabilities of real and fake images{0,1}.

2. Related Work

Ever since GAN was published by Goodfellow, et al in 2014,[4] many image generation variations like DC-GAN, Stack-GAN, Attn-GAN were proposed and experimented. For the above stated problem, the advantages, disadvantages, approaches of the different GANs are discussed. Additionally, why conditioned stack-GAN solution is chosen as a solution is validated in this section.

cGAN
In the same year 2014, Mirza, et, al[7] proposed conditioned version of GAN where both the generator and discriminator were conditioned through class label y. The model described how cGAN learns models with multi modal inputs. The generated images were conditioned by class labels. The discriminator evaluated the generated image based on real image and the class labels. This paved way for more and more multi modal image generation model especially the text to image Models. The primitive cGAN handled class labels and tags. They were not robust enough to handle large sentences. A similar kind of cGAN with convolutional layers was applied in facial recognition for human- robot interaction by Deng, et al [2]. This model used facial expression class labels with respective images as inputs to generate images of specific facial expression. This model in fact used the same generator twice and three discriminators to evaluate and provide feedback on generated images.

DC-GAN
Alec Radford et al [8]proposed a modified GAN called DC-GAN that replaced multi-layer perceptrons of vanilla GAN with deep conditional layers. The pooling layers in GAN were replaced with strided convolutional layers (discriminator) and fractional strided convolutional layers (generator). Batch norm layers were used in both generators and discriminators. Fully connected layers were removed. Generator ReLU layers were replaced with Tanh activation layers but discriminator still retained ReLU layers for activation. This network with deep convolutional GAN provided improved accuracy and reduced error rate compared to previous models. This model however had modal collapse problems and images generated were of lower resolution.
A modified DC-GAN with conditioned text inputs were experimented by Reed et al [10] where text encoders

encoded text descriptions to text embedding. These text embedding were concatenated with images and fed to both generators and discriminators of GAN. The outputs of both generator and discriminator of this DC-GAN were controlled and conditioned by text embedding as demonstrated in Figure 1. While Reed's model provided the advantage of getting accurate synthesized real images through text conditioning, those images were not of intended high resolution. There was room for improvement in terms of high resolution image generation and handing different types of texts. In another research work performed by Reed et al [9], performance of GANs from raw text are experimented with different text encoders. Text- based CNNs, CNN- RNN, LSTM text coders were evaluated. Of these CNN-RNN based model had higher accuracy in their experiments. Several algorithms for test embedding are being experimented every day.

Stack-GAN

The images produced by text conditioned DC- GAN were of better accuracy than the basic DC-GAN , but still the images produced were of lower resolution. To mitigate this problem, Zhang, et al [12] proposed stack-GAN where DC-GANs were stacked together. The Stack-GAN has two stages of 2 GANs - Stage I GAN and Stage II GAN. Stage 1 generator produced low resolution images with lesser accuracy while stage 2 generator processed these low resolution from stage 1 and produced high resolution image of size 256X256. Discriminators of both Stage 1 and Stage 2 GANs were fed with text embedding to ensure the generated images were matching the text information. In the follow up research of the stack-GAN, Zhang, et al [13] came up with version 2 of stack-GAN called stack-GAN++ for conditional and non conditional generative tasks. Stack-GAN ++ comprised of multiple generators and discriminators arranged in a tree like fashion. Images of different scales for the same scene were generated from each of these GAN branches.Stack-GAN++ results showed no collapsed mode. The stack-GAN++ work comprised of various features like Conditioning Augmentation and Color Consistency regulation, which had led to further improvement in image generation. Stack-GANs were used for variety of applications. Jain, et al [5]has applied stack GAN to process user query to text embedding that are fed to GAN to generate fashion clothing images. A variation of stack- GAN called Perception-GAN was introduced by Garg, et al [3] that aims in improving the quality of the initial image generated by stack-GAN. This method employs introduction of captioner loss in lower dimensional vector of real and generated images to ensure most of the perceptual aspects of the image are captured in stage I image generation itself.

Attn-GAN

Attn-GAN proposed by Xu, et al [11] emphasizes the importance of applying attention mechanism to each word/sentence in an input text sequence. The solution used 2 kinds of models: A LSTM model as text encoder that encoded incoming sequence into word/sentence features and 3 stage GANs that processed encoded sentence embedding and noise inputs to generate images. The word features retrieved by attention model were fed to the subsequent generator stages (from stage 2 onwards) to generate fine grain images in each stages.

This paper explores the application of stack-GAN, augmented by transformer based text encoding that generates high resolution images for an input query. The aim is to generate defect die images for user query similar to the state of the art text to image generation stage-GAN models. The only variation is our experiments is that its carried out with text embedding performed by transformers in an attempt to advance the state of the art performance.

3. Methods

The solution approach uses GenIE framework which consists of a Llama2 and Stack-GAN module. The Llama2 model details are included in co-author's paper. This paper focuses on the text conditioned image generation model of the framework. The Stack-GAN module has two GANs stacked together. These GANs are made of fully connected convolutional and transpose convolutional layers. The value function used to optimize is given by Equation 1, where pdata(x) denotes the true data distribution and pz(z) denotes the noise distribution.

$$\min_G \max_D V(D,G) = E_{x\sim pdata(x)}[logD(x)] + E_{z\sim pz(z)}[\log(1 - D(G(z)))] \quad (1)$$

The above equation for GAN is modified to handle the text input as a conditional input.

3.1. Stack-GAN Architecture

Our solution framework GenIE consists of 2 models: a NLP model that uses sentence transformers to convert user queries to text embedding and Stack GAN model.

NLP model

The user query is processed by NLP model that uses sentence transformer and converts them to key word captions that specify defect type and defect location ex: {Large particle, center}. These captions are converted to text embedding. The text embedding are converted to text conditioning variable by Conditioning Augmentation Network (CAN). The CAN consists of fully connected layer to generate μ_0 and $\sigma_0$for the Gaussian distribution

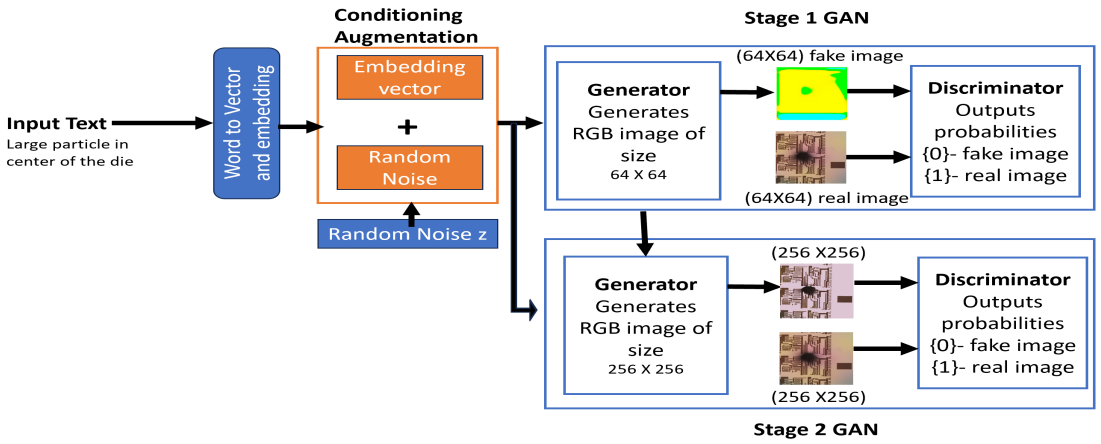


Figure 2. Block diagram of Stack GAN.

$N(\mu\_0(t), \sigma_0(t))$. ˆTCV c_0 are then sampled from the Gaussian distribution. Leaky ReLU layers normalises text embedding and converts them to Text Conditioning Variable (TCV). These text conditioning variables are fed to both Stage I/stage II generators and discriminators for conditioned image generation and validation.

Stage-I GAN

Stage I-GAN consists of a discriminator and a generator. The generator receives input noise and Text Conditioning Variable (TCV). The input noise z is generated through random sampling for every image generation. The input noise z and TCV c_0 are concatenated and passed through several convolutional layers each followed by a batch normalization layer. Tanh activation is used in generator. Low resolution image of size (64x64x3) is generated by the stage I generator G_0.Generator uses Binary Cross Entrophy (BCE) loss for generator loss computation.. Stage I discriminator D_0 receives stage I generated low resolution image, TCV of size (384) and real image for evaluation as depicted in Figure 2. The input image and TCV are convoluted, flattened and activated by sigmoid to generate probabilities for real and fake image. Batch normalization is used after every convolutional layer as a regularizer. The discriminator is denoted by D_0.
For the real image I_0, input noise z, text embedding t, TCV c_0, the generator loss and discriminator loss are given by Equation (2) and Equation (3)

$$L_{D_0} = E_{(I_0,t)\sim pdata}[logD_0(I_0,t)] + E_{z\sim pz, t\sim pdata}[\log(1 - D_0(G_0(z,c_0),t))] \quad (2)$$

$$L_{G_0} = E_{z\sim pz, t\sim pdata}[\log(1 - D_0(G_0(z,c_0),t))] \quad (3)$$

Stage II GAN

The stage II generator G receives the low resolution generated image s_0 from stage I generator along with the TCV t and synthesizes high resolution image (256x256x3). It concatenates input image and TCV, downsamples concatenated output, passes through residual block, upsamples the image to an image of size (256x256x3). The stage II discriminator evaluates the high resolution stage II generated image against the TCV and real images to be real or fake.. It learns the generated combinations - {real image, fake text}, {real image, real text} and {fake image, fake text}. The output of discriminator is true when it finds the correct image and correct text combination. This correct image synthesized by generator is sent as output. When the discriminator does not find correct image and correct text, it's output is false and generator regenerates image until the discriminator is unable to distinguish generated image/text against real image/ text. The generator and discriminator loss of stage-1 GAN are given by Equation (4) and (5)

$$L_D = E_{(I,t)} \sim pdata[logD(I,t)] + E_{s_0} \sim p_{G_0}, t \sim p_{data}[\log(1 - D(G(s_0,c),t))] + \lambda DKL(N(\mu_0(t),\sigma_0(t))||N(0,I)) \quad (4)$$

$$L_G = E_{s_0} \sim p_{G_0}, t \sim p_{data}[\log(1 - D(G(s_0,c),t))] + \lambda DKL(N(\mu(t),\sigma(t))||N(0,I)) \quad (5)$$

3.2. Algorithm

The GAN plays the min max game as specified in Equation (1) where the generator generates fake images till discriminator gets fooled and discriminator validates the generated image to be real or fake. The details of the algo

implemented in stack-GAN is summarised in Algorithm 1. The sample images I_0, batch size of n_{batch} and TCV c_0 are fed to the stage I generator G_0. The generated image from Stage I GAN is denoted by S_0. The gradients with respect to discriminator loss $\alpha\partial L_{D_0}/\partial D_0$ and generator loss $\alpha\partial L_{G_0}/\partial G_0$ are computed and learning step is updated by adam optimizer. s_r in the algorithm section is the score for $\{$real image, right text$\}$, s_w is the score for $\{$real image, wrong text$\}$ and s_f is the score for $\{$wrong image, right text$\}$

The stage II GAN uses the same algo and CAN module but stage I generated image s_0 is used as input in place of noise image z. Stage II discriminator uses learning rate of 0.0002. Batch normalization is implemented after every convolutional layers in generator and discriminator as a in built regulariser. Adam optimizers are leveraged for learning step update and to prevent exploding and vanishing gradients. Binary cross entropy loss is computed for discriminator and generator in both GANs. Both Stage I and stage II Adversarial losses are computed. 1850 epoch for batch size of 64 were run on in house GPU.

Algorithm 1 Text conditioned Stack-GAN Stage-I GAN

$hyper\ parameters : epochs = 1850, z_{dim} = 500, batch - size = 64, lr_{g0} = 0.0002, lr_{d0} = 0.0002, \beta1 = 0.5, \beta2 = 0.99$

Input: sample images I_0, batch-size n_{batch}, text embedding t, noise image z

Output: generated images S_0, probabilities$\{0,1\}$

1. Text encoder generates text description from query
2. CAN generates TCV c_0
3. n_{batch}=[n_{sample}/batch-size]

for e in (epochs): **do**
  for i in n_{batch} **do**
    $Generated - image S_0 \leftarrow G_0(z,t)$
    $Compute\ s_r \leftarrow D_0(I_0,t)$ ▷ real image,right text
    $Compute\ s_r \leftarrow D_0(I_0,t')$▷ real image,wrong text
    $Compute\ s_f \leftarrow D_0(S_0,t)$ ▷ fake image, right text
    $Compute\ L_{D_0} = 0.5 * (s_r + s_w + (s_f)/2)$
    $Update - discriminator : D_0 - \alpha\partial L_{D_0}/\partial D_0$
    $Compute - generatorloss : L_{G_0}$
    $updategenerator : G_0 - \alpha\partial L_{G_0}/\partial G_0$
    $Compute\ adversarial\ loss$
  end for
end for

The existing code base for Stack- GAN by [1]AarohiSingla is modified and used for KLA-DDI dataset. The input sizes of images are modified from (100x100) to (500X500) and text embedding based TCV are modified from 1024 to 384 to process KLA-DDI dataset. 3000 defect images are used for experiments. The code base is modified to include code for computing loss plots, generating text embedding and pickling filenames and classcodes. The batch size, learning rate of 0.0002 is used in both discriminator and generator for training KLA-DDI dataset. The 1850 epochs were used for training.

Defect Type	Train/Test dataset
Small particle	400/100
Large particle	400/100
Scratch	400/100
Peeling	400/100
Clusters	400/100
Chippings	400/100

Table 1. **Compilation of Data set**

4. Dataset

KLA Defect Die Image (KLA-DDI) data set is created and used for this experiments. Defect images of six different defect classes like small particle/large particle/ clusters/ chippings/ scratch/ peelings at different locations like middle/ top left/ top right/ bottom left/All over/center/top/bottom/left/Right/bottom right are collected from in-house process control tools. These images are pre-processed, labelled and made ready for training.

4.1. Image Pre-processing

3000 images are collected from process control tools by running many inspection scans on patterned wafers. Every inspection scan setting is fine-tuned with appropriate defect detection algorithms to grab the required die defect images for each classes. The collected defect images are reviewed individually to categorise them to each of the six defect classes as in table 1 and to caption each of them with two parameters - defect class and defect position on the die Example: $\{$large particle, center$\}$. These images are resized to size (500x500) and captioned with appropriate text to convey the class and defect position. These resized images with respective class names and captions are stores to KLA-DDI dataset for future use.Pixel normalization is performed on images by subtracting and dividing them by 127.5 to have the inputs normalized to range [-1,1] before passing on to the Conv layers in GAN.

4.2. Text Pre-processing

The query from the end user are converted to captions and in-turn into text embedding by the NLP model. The text embedding are converted to text conditioning variable by Conditioning augmentation network. In CAN, the text are also normalised by computing mean and standard deviation and applying them. The defect images are fed along with TCV and real images to train our model. The filenames

Sample images						
Class	Small particle	Large particle	Scratch	Clusters	Peelings	Chippings
Caption	{small particle, bottom}	{large particle, center}	{Scratch, center}	{clusters, all over}	{Peelings, all over}	{Chippings, top right}

Figure 3. Sample images from KLA-DDI dataset

Metric	Data set	GAN-INT-CLS	GAWWN	Stack GAN
Human Rank	CUB	2.81 ± .03	1.99 ± .04	1.37 ± .02
	Oxford	1.87 ± .03	/	1.13 ± .03
	COCO	1.89 ± .04	/	1.11 ± .03
	KLA-DDI			1.33±.02

Figure 4. Human Rank

and classnames are converted to pickle files to serialize them and preserve them in order, throughout the model so that the discriminator retrieves correct class name and image files for generator image evaluation.

4.3. Sample images

Sample images for each class and respective captions are listed as in Figure 3. An example for each of the defect class: Small particle, large particle, scratch, clusters, peelings and chippings are mapped to respective captions and displayed for understanding. code base.

5. Experiments

In order to validate our designed model, KLA in-house GPU machine is used. SW code is written using python tensorflow libraries. The defect images in KLA-DDI datatset has key features such as die patterns, defect shadows, scratch lines and group of small defects. For our experiments, KLA-DDI dataset created for this project is used. With our experiments, various qualitative and quantitative metrics are derived and discussed. 3000 defect images, 400 images in each class are categorised as training images and the model is trained. 100 images in each class is reserved for testing. During test, user queries were sent and based on the embedding received, the images are generated. The images generated during test mostly retained spatial features and defect information as expected. Our objective in running the following experiments were to validate if the model captures key defect features like particle, scratch, etc.. during training.The hyper-parameters like learning rate(generator:0.0002 and discriminator:0.0002), batch size(64), no of epochs(1850) are modified and experimented for this goal. The learning rate of 0.001 was used initially. The training process was fast and learning failed to capture few key details during training. So, the learning rate of 0.0002 is used in both generator and discriminator of both stage I and Stage II GANs which yielded images with better spacial information. Adam optimizer is used in order to keep exploding gradient and vanishing gradient problems in check. Also, initially batch size of 32 and epoch of 500 was used for training. with these values, learning is not capturing all intended features. so the number of epoch is increased by 250 and batch size is retained at 64 for every subsequent runs. The loss curves(in section 5.2) were observed for every runs and finally concluded that epoch of 1250 is sufficient for this model and data set.

5.1. Performance metrics

The qualitative metrics define the quality of generated images. Visual inspection is used as qualitative metrics in our experiment results. The quantitative metrics are numerical/graphical measure of the performance of the model. Human rank, loss curves results are discussed as part of quantitative metrics.

Qualitative metrics

Visual interpretation

In evaluating any Computer vision models, visual inspection and interpretation becomes an essential, cost efficient and intuitive method of evaluating a model's performance.[6] By looking at generated defect images and performing a visual check against real defect images and class names, the clarity of the defect image, defect position and defect type, defect relevance to class names are evaluated for the input query. Blurred image, incorrect defect position, incorrect defect types are easy to identify through visual inspection. This visual inspection results will serve as input to compute Human rank metric. Visually analysing the images for true/false positives and true/false negatives are easy.

Quantitative metrics

Human rank

To access if the generated images are truly conditioned by the input, Human rank adopted by Zhang, et al [12] is leveraged. Human rank is an evaluation metrics for GANs where humans validate the images by comparing them with real images. After validating the generated images, Humans rank the images in terms of realism and image quality. The rankings for our experiments is between 1 to 5. Human evaluation is important in computer vision tasks as it gives us insights on how good the generated image matches with human perception.

Loss curves

Loss curves are graphical representation of a model's performance over time.It provides insights of how the

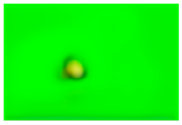
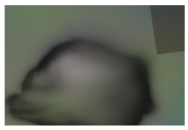
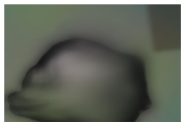
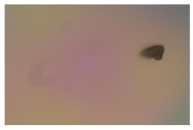
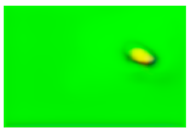
Class	captions	Real image	Initial Epoch Stage I GAN image	Stage II GAN Image
Small particle	{Small particle, center}			
Large particle	{Large particle, center}			
Peeling	{Peeling, Right}			

Figure 5. Visual representation of generated defect images of 3 classes.

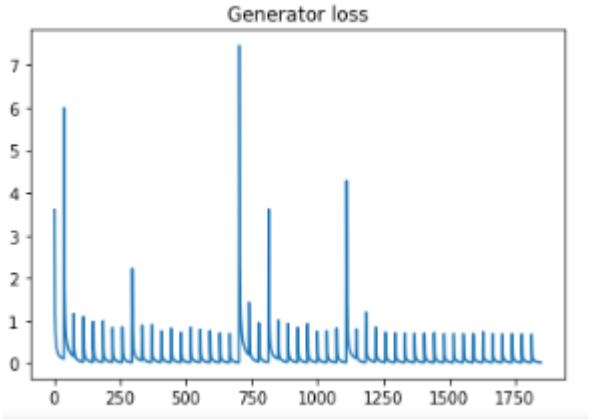


Figure 6. Generator loss of Stage-II GAN

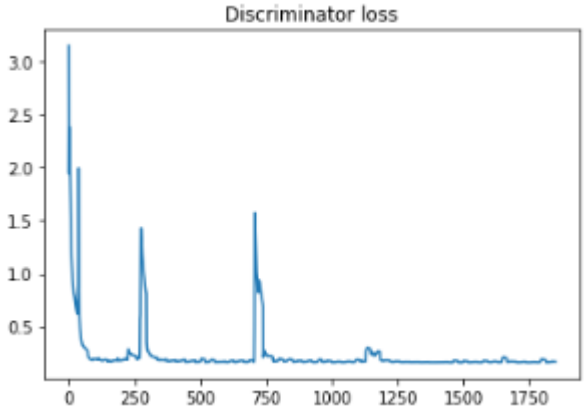


Figure 7. Discriminator loss of Stage-II GAN

model's generated images are close to reality. In general, we expect the losses to be high during initial training phase and as time rolls on, losses starts decreasing. By observing the graphs, the user identifies where to stop training. Discriminator and generator loss plots of both Stage I and Stage II GANs are discussed.

For the six defect classes, qualitative and quantitative metric results are discussed. Both human and system generated metrics are considered for evaluation.

5.2. Results and discussion

The visual interpretation shows the images that were obtained during initial epochs in GAN I and later epoch in GAN II. As we can see in the images, during initial epoch, the spatial features are being learnt by GAN I and later as models learning advances, we see clearer images pertaining to user query are generated. The table shows the initial and final images generated for three classes. Random captions from test set (100 images and corresponding captions) were chosen and passed to the model for image synthesis. We see from Stage 1 initial epoch results that defect information is attended by the model and is learnt by the model. For the given defect classes, we see the spatial features are learnt by the model in stage I and stage 2 synthesis a finer high resolution image similar to stage I image.

To compute Human rank, 50 random captions for each class are selected and 10 Defect images are generated for those captions. Those images are provided to 12 users and are evaluated by them against the captions. The ranks provided by users for 5 images in each class are collected and averaged across each class to arrive Human rank. The

computed Human rank is almost same as the state of the art models as depicted in Figure4. For further experimentation, users are given set of real defect images and generated defect images and are asked to pick the images corresponding to the defect class codes. For this exercise, 10 images from each class are generated and provided to users for image selection. It was observed that nearly 6 out of 10 times, users chose the images generated by our model. Considering visual perception, this model has achieved 63 percent accuracy.

The stack-GAN is trained for 1850 epochs to analyse the training of KLA-DDI dataset by the model. The generator loss computed during this training presented in Figure 5. The loss oscillates initially at high frequency for initial epochs as discriminator validates the fake images produced by the generator. The variation in loss values are very frequent during initial epoch and as time progresses and when epoch 1250 is reached, too much variation in losses are not seen. Our model reduces generator loss, stabilizes and converges as expected.

The discriminator losses starts with a very high loss and time progresses, the discriminator loss decreases and fluctuates very mildly. After epoch 250, spike in losses occur in the interval of 500 epochs. When epoch 1250 is reached, spikes get suppressed and oscillates at around 32 to 38 percent. At about 1250 epoch, we see both the discriminator and generator losses are converging to minimal values and not much variance in losses are seen beyond this point. It indicates training to be stopped at 1250 epoch.

6. Conclusion and future work

In this paper, we used a solution framework of GenIE that employs NLP and Stack-GAN to experiment on KLA DDI image dataset. We generated images that are conditioned by user query to solve our tool time challenges. Various qualitative and quantitative results are summarised to display the model's performance. We succesfully generated images for six defect classes. We observe the model is able to successfully learn the defect features. Our future work will be focused on generating more number of realistic images for a user query and to improve accuracy. We aim to focus on improving the model to handle more number of defect classes and to produce higher resolution images as this will solve the engineer's problem of tool dependency.

7. Appendices

The layers used in CAN, Stage 1 GAN and Stage 2 GAN are summarised in supplementary files for reference. It lists the number of layers, parameters in each layer and output shape of tensor from each layer. An example of the

Layer (type)	Output Shape	Param #	Connected to
input_48 (InputLayer)	[(None, 384)]	0	
dense_24 (Dense)	(None, 256)	98560	input_48[0][0]
leaky_re_lu_55 (LeakyReLU)	(None, 256)	0	dense_24[0][0]
lambda_12 (Lambda)	(None, 128)	0	leaky_re_lu_55[0][0]
input_49 (InputLayer)	[(None, 500)]	0	
concatenate_9 (Concatenate)	(None, 628)	0	lambda_12[0][0] input_49[0][0]
dense_25 (Dense)	(None, 16384)	10289152	concatenate_9[0][0]
re_lu_71 (ReLU)	(None, 16384)	0	dense_25[0][0]
reshape_4 (Reshape)	(None, 4, 4, 1024)	0	re_lu_71[0][0]
up_sampling2d_28 (UpSampling2D)	(None, 8, 8, 1024)	0	reshape_4[0][0]
conv2d_117 (Conv2D)	(None, 8, 8, 512)	4718592	up_sampling2d_28[0][0]
batch_normalization_102 (BatchN	(None, 8, 8, 512)	2048	conv2d_117[0][0]
re_lu_72 (ReLU)	(None, 8, 8, 512)	0	batch_normalization_102[0][0]
up_sampling2d_29 (UpSampling2D)	(None, 16, 16, 512)	0	re_lu_72[0][0]
conv2d_118 (Conv2D)	(None, 16, 16, 256)	1179648	up_sampling2d_29[0][0]
batch_normalization_103 (BatchN	(None, 16, 16, 256)	1024	conv2d_118[0][0]
re_lu_73 (ReLU)	(None, 16, 16, 256)	0	batch_normalization_103[0][0]
up_sampling2d_30 (UpSampling2D)	(None, 32, 32, 256)	0	re_lu_73[0][0]
conv2d_119 (Conv2D)	(None, 32, 32, 128)	294912	up_sampling2d_30[0][0]
batch_normalization_104 (BatchN	(None, 32, 32, 128)	512	conv2d_119[0][0]
re_lu_74 (ReLU)	(None, 32, 32, 128)	0	batch_normalization_104[0][0]
up_sampling2d_31 (UpSampling2D)	(None, 64, 64, 128)	0	re_lu_74[0][0]
conv2d_120 (Conv2D)	(None, 64, 64, 64)	73728	up_sampling2d_31[0][0]
batch_normalization_105 (BatchN	(None, 64, 64, 64)	256	conv2d_120[0][0]
re_lu_75 (ReLU)	(None, 64, 64, 64)	0	batch_normalization_105[0][0]
conv2d_121 (Conv2D)	(None, 64, 64, 3)	1728	re_lu_75[0][0]
activation_12 (Activation)	(None, 64, 64, 3)	0	conv2d_121[0][0]

Total params: 16,660,160
Trainable params: 16,658,240
Non-trainable params: 1,920

Figure 8. Layers of Stage I generator

Stage1 generator's layer information is added for quick reference here.

8. Contributions and acknowledgements

We sincerely thank Stanford professors, Teaching Assistants, lecturers who has enabled us build this project by imparting knowledge through lectures, ed discussions and discussions. We thank our project guide who has provided feedback and encouragement through project milestone evaluation. We sincerely thank KLA for providing us in house GPU machine to train and test our algorithms. We thank our colleagues who helped perform visual inspection and rank the images generated by this model. Geeta Jakkamsetti belonging to CS224N course class is the co-author of this paper. She has been instrumental in converting user queries into text embedding using transformer model and query based document summarization. We thank AarohiSingla for making the stack-GAN code base public. This code base was tailored for the stack-GAN module of our GenIE framework.

References

[1] AarohiSingla. Stackgan.

[2] J. Deng, G. Pang, Z. Zhang, Z. Pang, H. Yang, and G. Yang. cgan based facial expression recognition for human-robot interaction. *IEEE Access*, 7:9848–9859, 2019.

[3] K. Garg, A. K. Singh, D. Herremans, and B. Lall. Perceptiongan: real-world image construction from provided text through perceptual understanding. In *2020 Joint 9th International Conference on Informatics, Electronics & Vision (ICIEV) and 2020 4th International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, pages 1–7. IEEE, 2020.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[5] A. Jain, D. Modi, R. Jikadra, and S. Chachra. Text to image generation of fashion clothing. In *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 355–358. IEEE, 2019.

[6] D. Meena, H. Katragadda, K. Narva, A. Rajesh, and J. Sheela. Text-conditioned image synthesis using tac-gan: A unique approach to text-to-image synthesis. In *2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS)*, pages 454–462. IEEE, 2023.

[7] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[8] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[9] S. Reed, Z. Akata, H. Lee, and B. Schiele. Learning deep representations of fine-grained visual descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 49–58, 2016.

[10] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In *International conference on machine learning*, pages 1060–1069. PMLR, 2016.

[11] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1316–1324, 2018.

[12] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5907–5915, 2017.

[13] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1947–1962, 2018.