Generation of Synthetic Datasets in the Context of Computer Networks using Generative Adversarial Networks

Thiago Caproni Tavares (IFSULDEMINAS)
Ariel Góes de Castro (UNICAMP)
Leandro C. de Almeida (IFPB)
Washington Rodrigo Dias da Silva (UFSCar)
Christian Esteve Rothenberg (UNICAMP)
Fábio Luciano Verdi (UFSCar)

Support

SMARTNESS

INSTITUTO FEDERAL
Sul de Minas Gerais

UNICAMP

INSTITUTO FEDERAL
Paraíba

ufscar

feec

LERIS

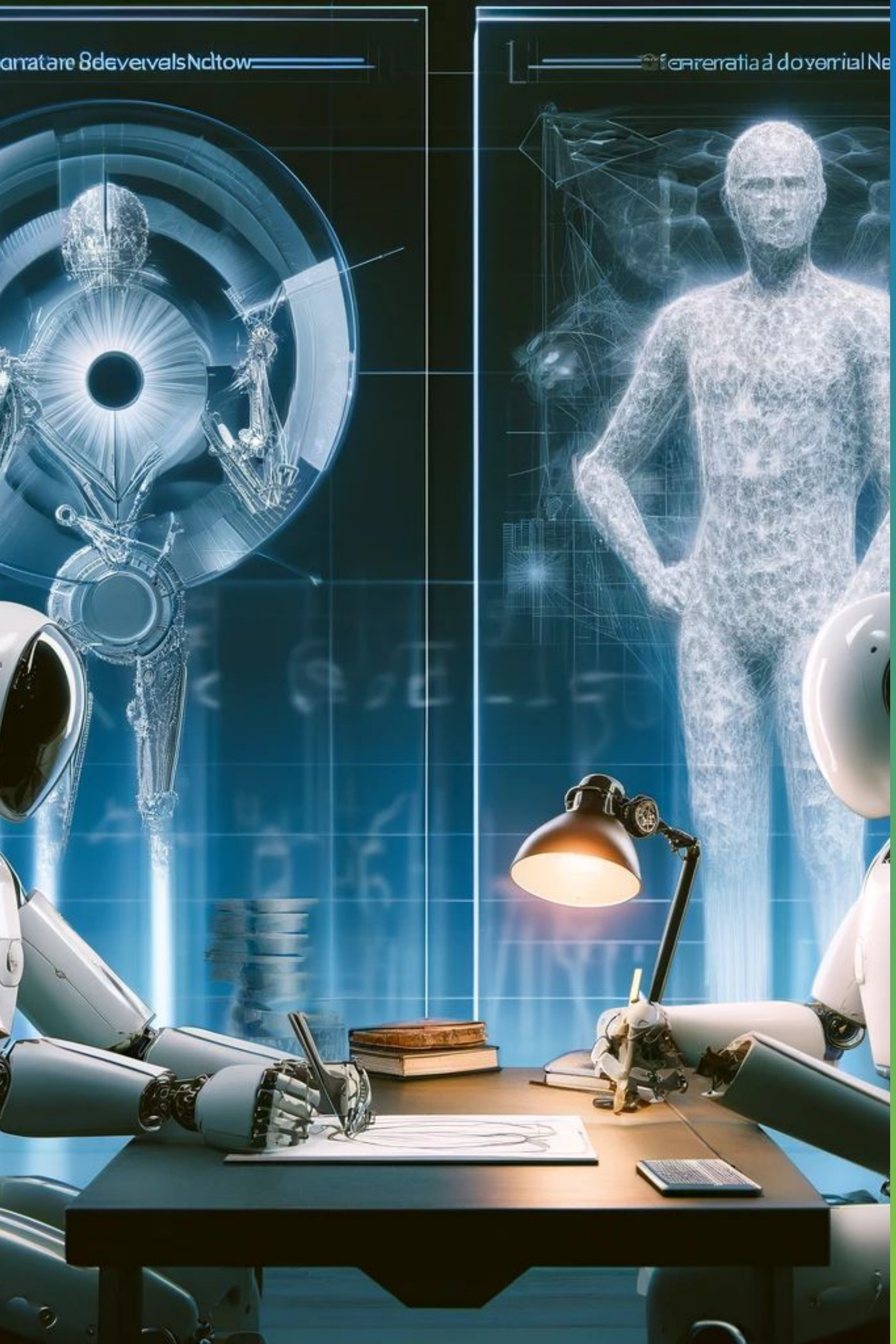
ERICSSON

FAPESP

CNPq

Summary

1. Introduction
2. Fundamentals of Generative Adversarial Networks
3. In-band Network Telemetry and Programmable Data Planes
4. Generation of Telemetry Data
   a. Hands on
5. Generation of Synthetic Network Trace
   a. Hands on
6. Conclusions and future perspectives



Introduction

Motivation

Introduction

Why are datasets important for computer networks?

To understand applications with different demands (e.g., latency, throughput) and propose solutions (e.g, protocols)

What are the real/existing dataset limitations?

Little data available and privacy of companies and users is affected

Why use synthetic data in computer networks?

Existing datasets may be scarce or outdated - i.e., do not reflect existing applications' needs (e.g., TSN, 5G and beyond, video streaming)

Practical Applications of GANs

Introduction

- Generate synthetic data for machine learning model inputs
- Allocation for prediction or classification tasks
- Application in network simulations to maintain data privacy and enhance data quality

Introduction to Generative Artificial Intelligence

Introduction

Generative Artificial Intelligence: An Overview

- Encompasses algorithms and models capable of generating diverse data forms, including images, videos, text, and digital media.
- Has gained significant traction outside academic circles, largely due to advancements like ChatGPT, a Large Language Model (LLM).

Generative Adversarial Networks (GANs)

- Introduced primarily for image synthesis.
- Comprises two neural networks: the generator and the discriminator.
- Operates on game theory principles: the generator creates synthetic data, while the discriminator judges its authenticity.

Applications in Computer Networks

- Extension of GAN applications in computer networks, focusing on data synthesis and privacy.
- Utilization includes dataset augmentation, balancing, and simulation of complex data distributions.

So... can we generate synthetic network data?

Spoiler: yes

Fundamentals of Generative Adversarial Networks

Generative Adversarial Networks (GANs)

Fundamentals of Generative Adversarial Networks

- **Definition:** Machine learning framework consisting of two competing modules: the generator and the discriminator.

- **Function:** The generator creates synthetic data mimicking real data, while the discriminator learns to differentiate between the two.

- **Training Dynamics:** Both modules are trained adversarially until the discriminator cannot distinguish real from synthetic data.

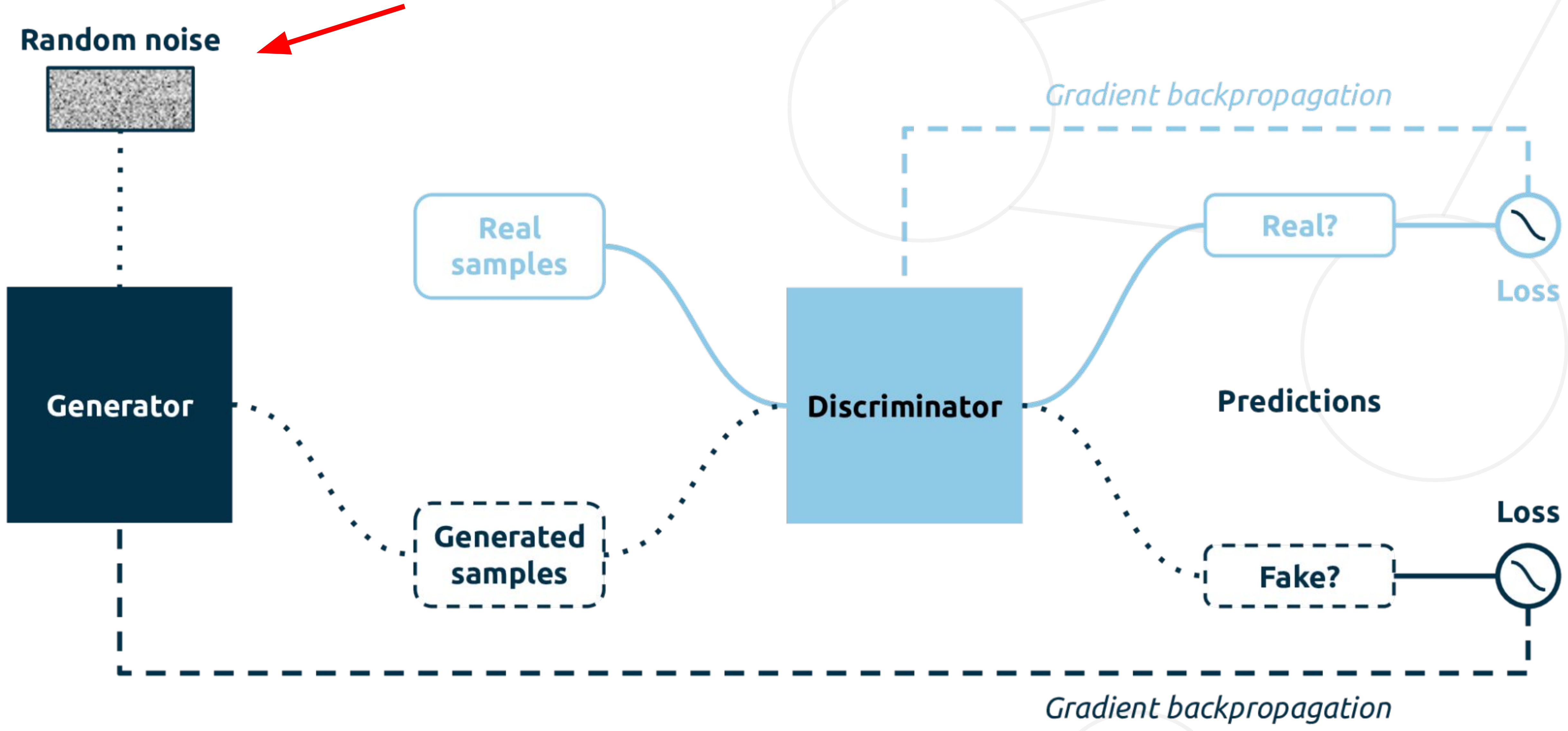
- **Complexity:** The discriminator can be seen as employing supervised learning by using real data to train its judgment capabilities.

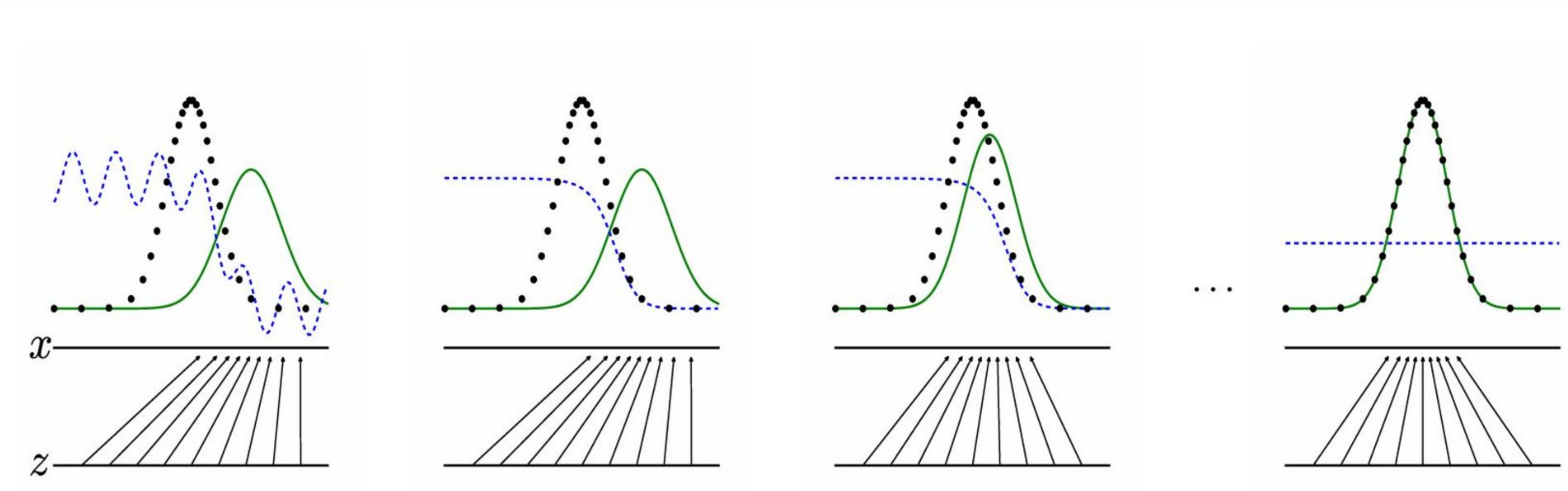
Generative Adversarial Networks (GANs)

Fundamentals of GANs

We summarize the GAN training as follows:

1. First, we feed the generator with (i) noise and a (ii) training sample





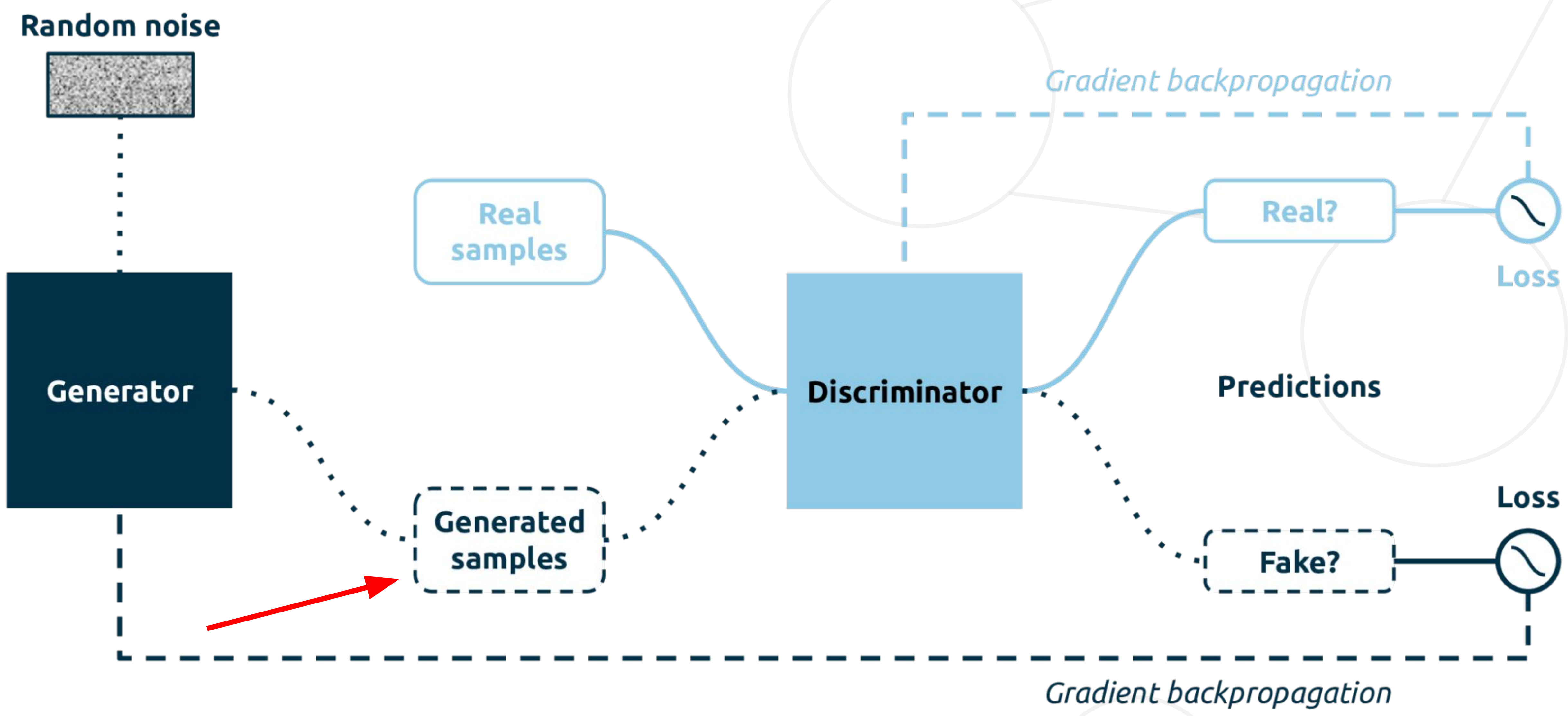
(i) (ii) (iii) (iv)

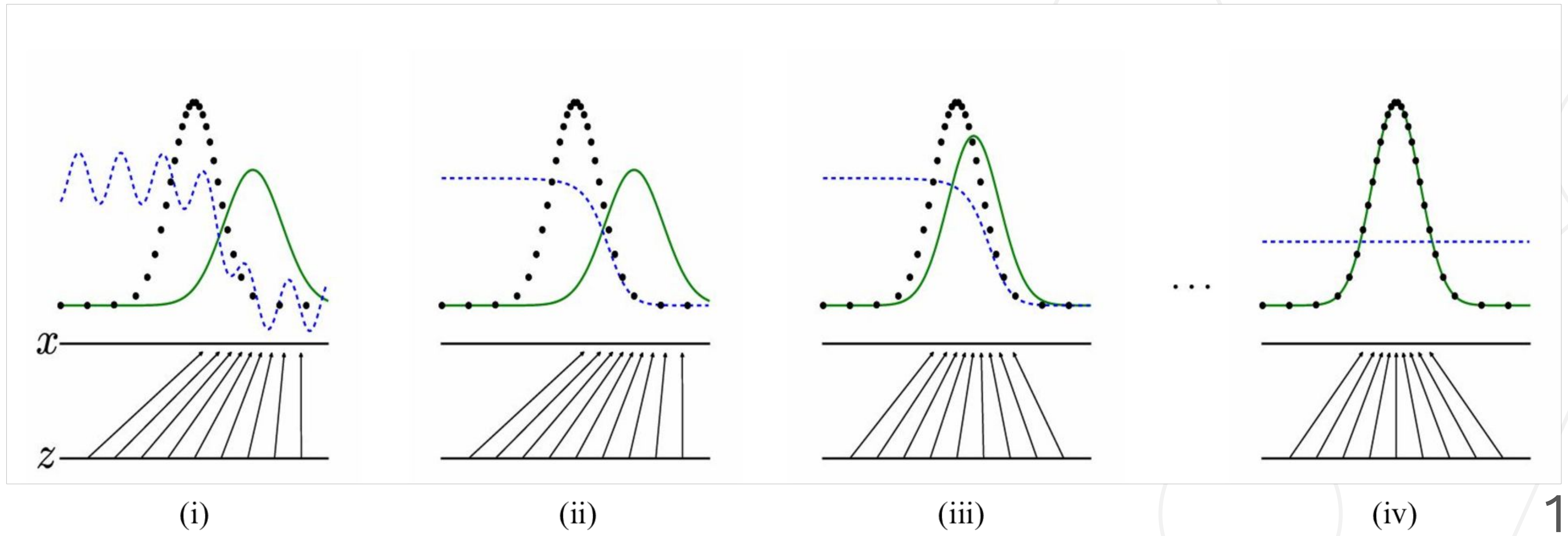
Generative Adversarial Networks (GANs)

Fundamentals of GANs

We summarize the GAN training as follows:

1. First, we feed the generator with (i) noise and a (ii) training sample
2. We merge it and generate a new sample



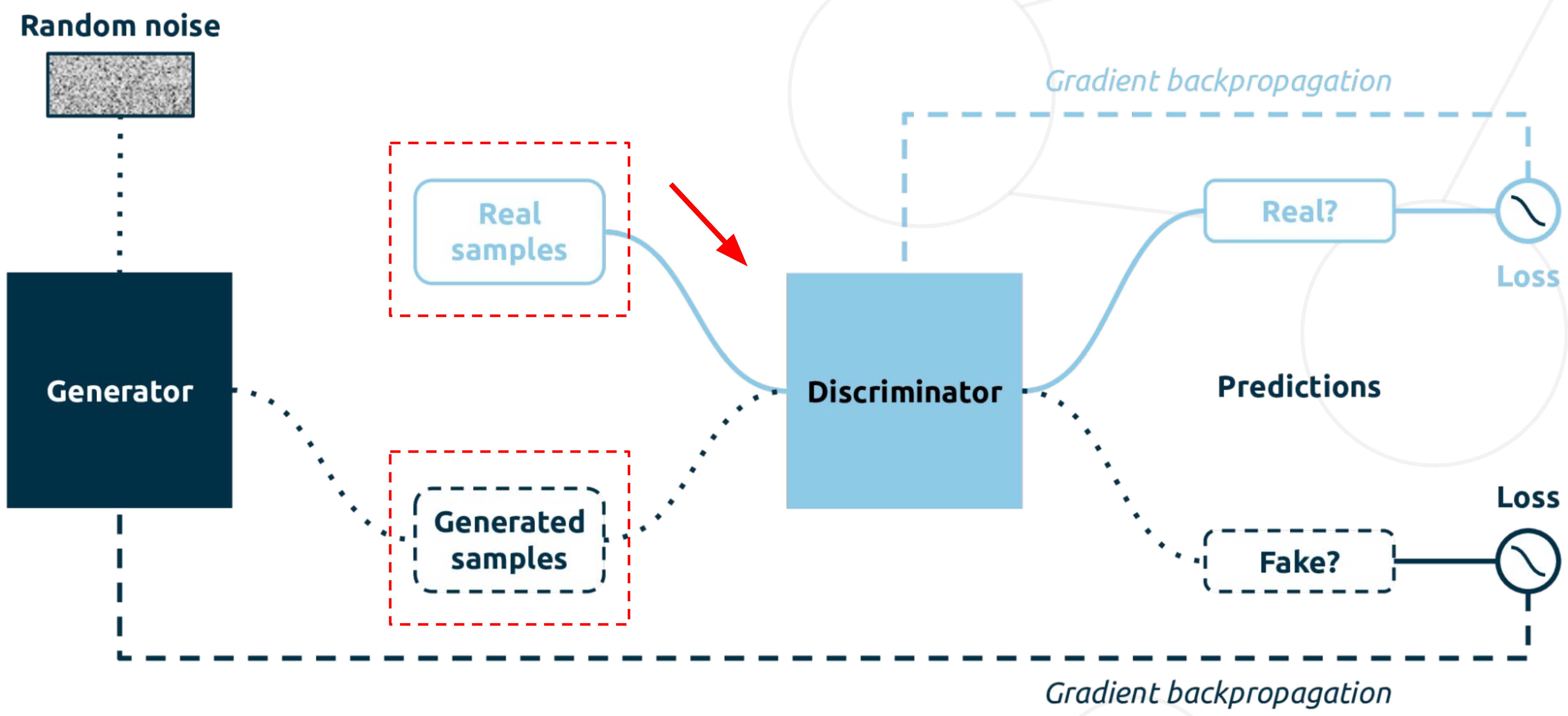


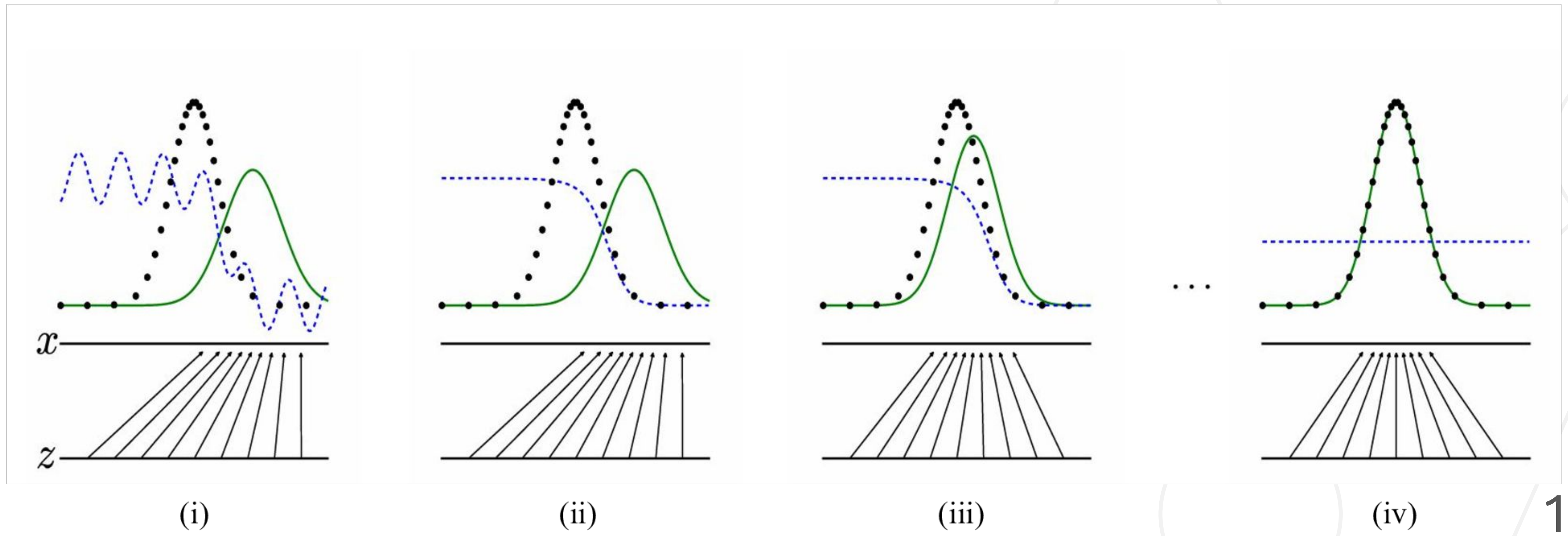
Generative Adversarial Networks (GANs)

Fundamentals of GANs

We summarize the GAN training as follows:

1. First, we feed the generator with (i) noise and a (ii) training sample
2. We merge it and generate a new sample
3. The Discriminator tries to guess which of the entries is the real one



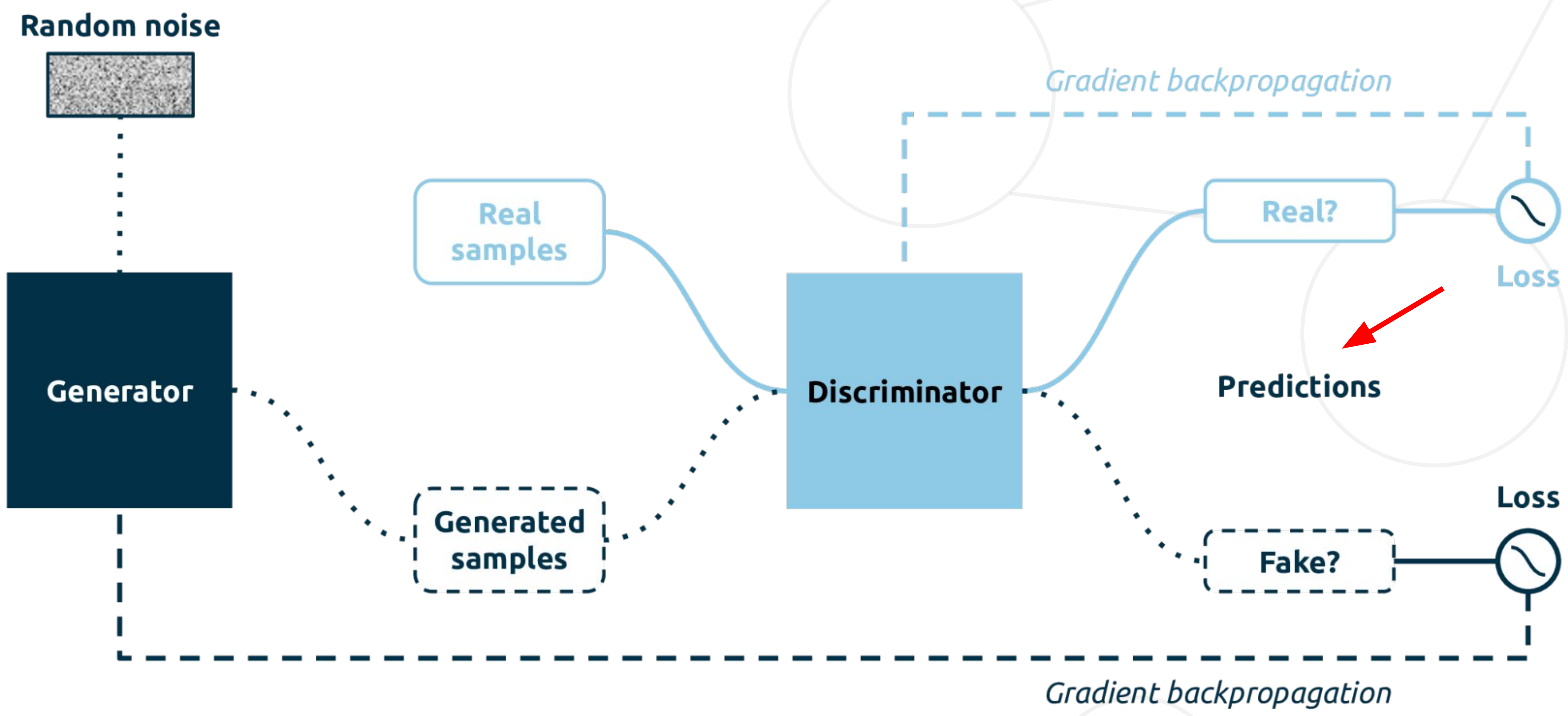


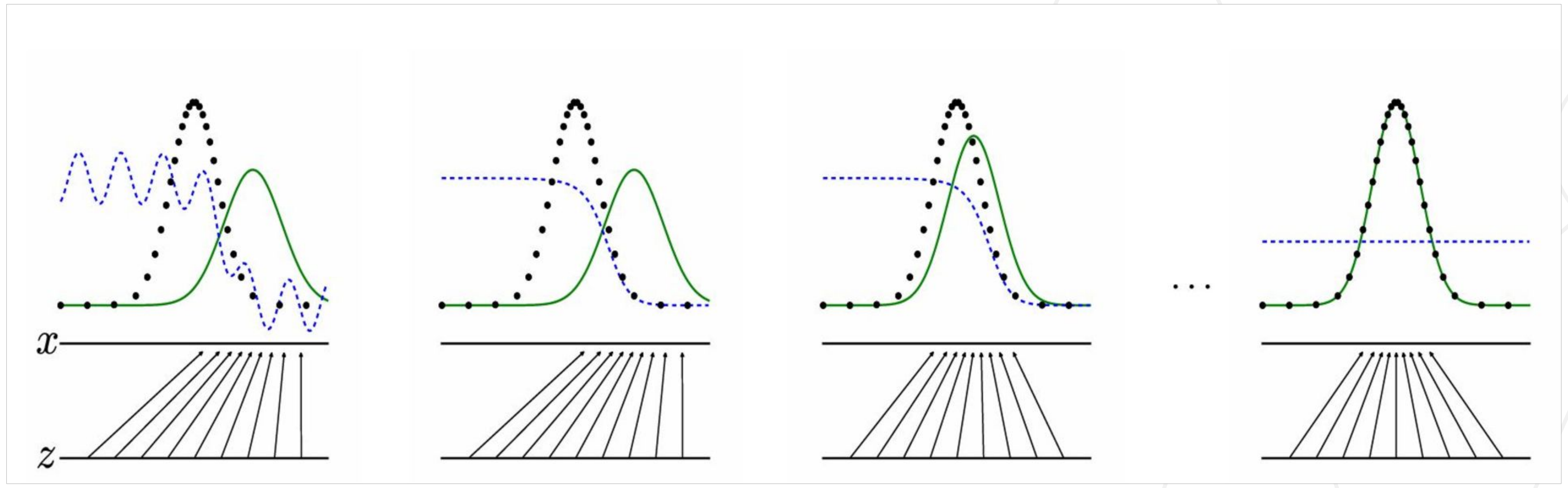
Generative Adversarial Networks (GANs)

Fundamentals of GANs

We summarize the GAN training as follows:

1. First, we feed the generator with (i) noise and a (ii) training sample
2. We merge it and generate a new sample
3. The Discriminator tries to guess which of the entries is the real one
4. The prediction is generally in the probability interval [0,1] (e.g., sigmoid function)





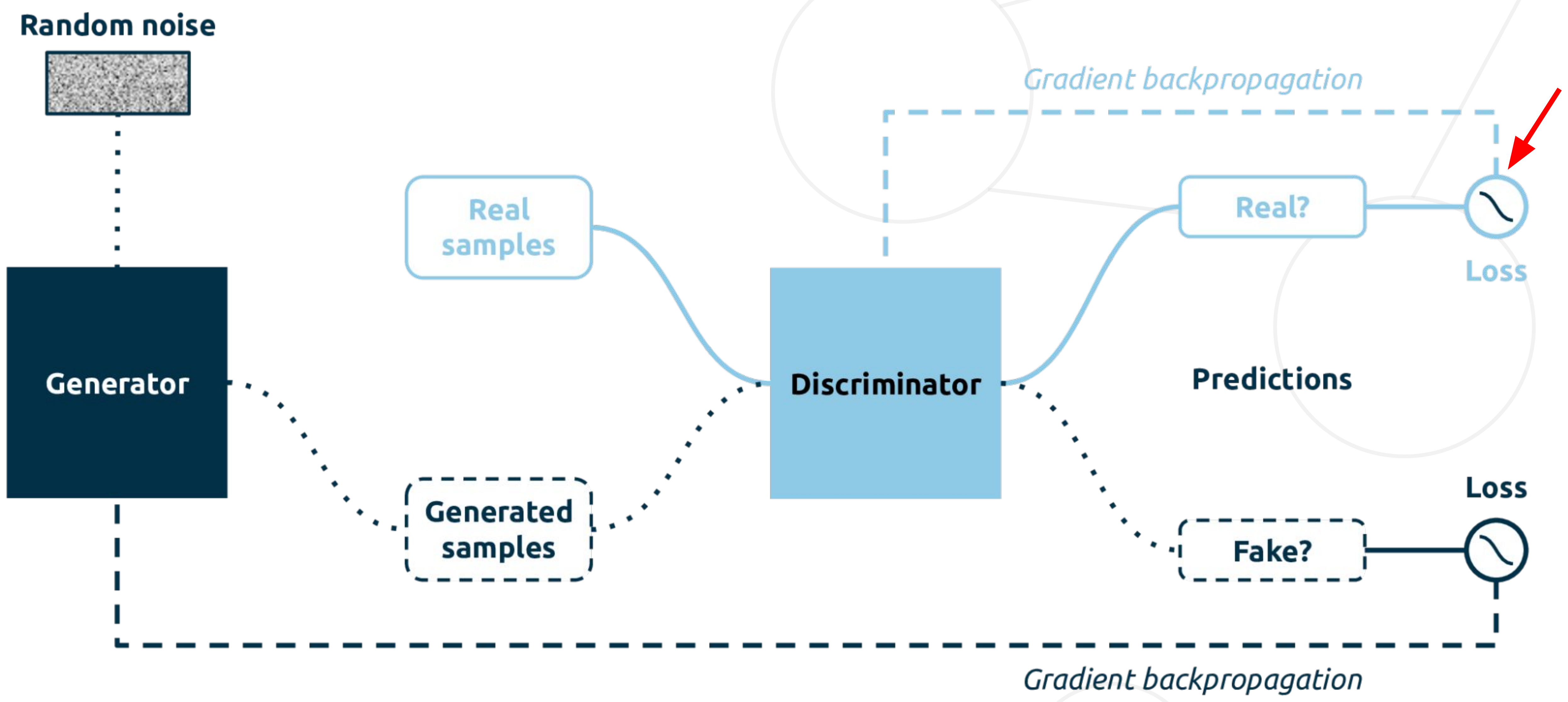
(i) (ii) (iii) (iv)

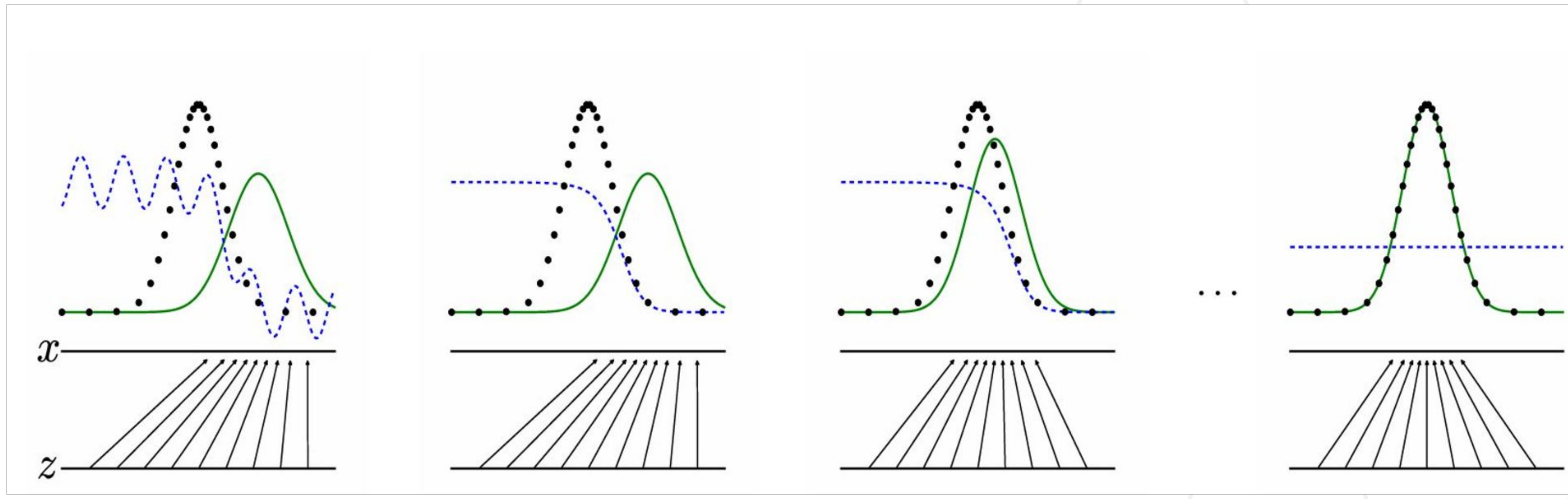
Generative Adversarial Networks (GANs)

Fundamentals of GANs

We summarize the GAN training as follows:

1. First, we feed the generator with (i) noise and a (ii) training sample
2. We merge it and generate a new sample
3. The Discriminator tries to guess which of the entries is the real one
4. The prediction is generally in the probability interval [0,1] (e.g., sigmoid function)
5. The generator is rewarded if it "fools" the discriminator





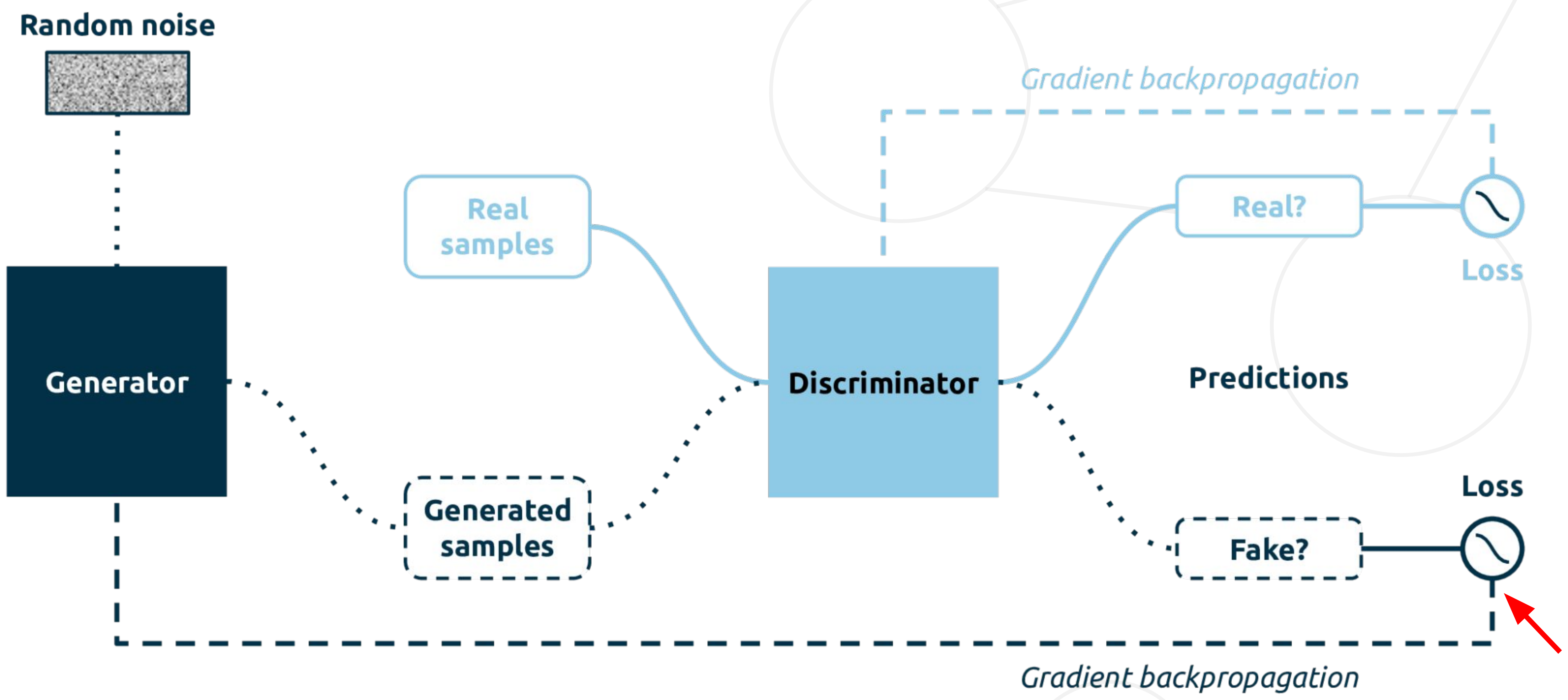
(i) (ii) (iii) (iv)

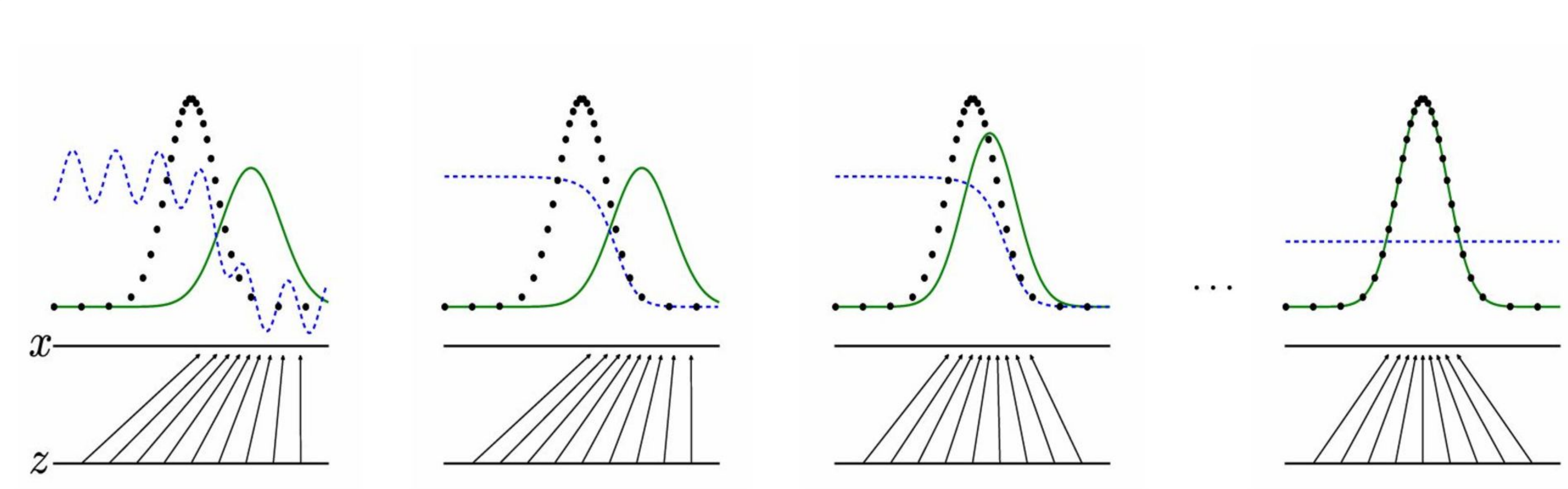
Generative Adversarial Networks (GANs)

Fundamentals of GANs

We summarize the GAN training as follows:

1. First, we feed the generator with (i) noise and a (ii) training sample
2. We merge it and generate a new sample
3. The Discriminator tries to guess which of the entries is the real one
4. The prediction is generally in the probability interval [0,1] (e.g., sigmoid function)
5. The generator is rewarded if it "fools" the discriminator
6. The discriminator function loss is calculated (e.g., using Binary Cross Entropy (BCE))





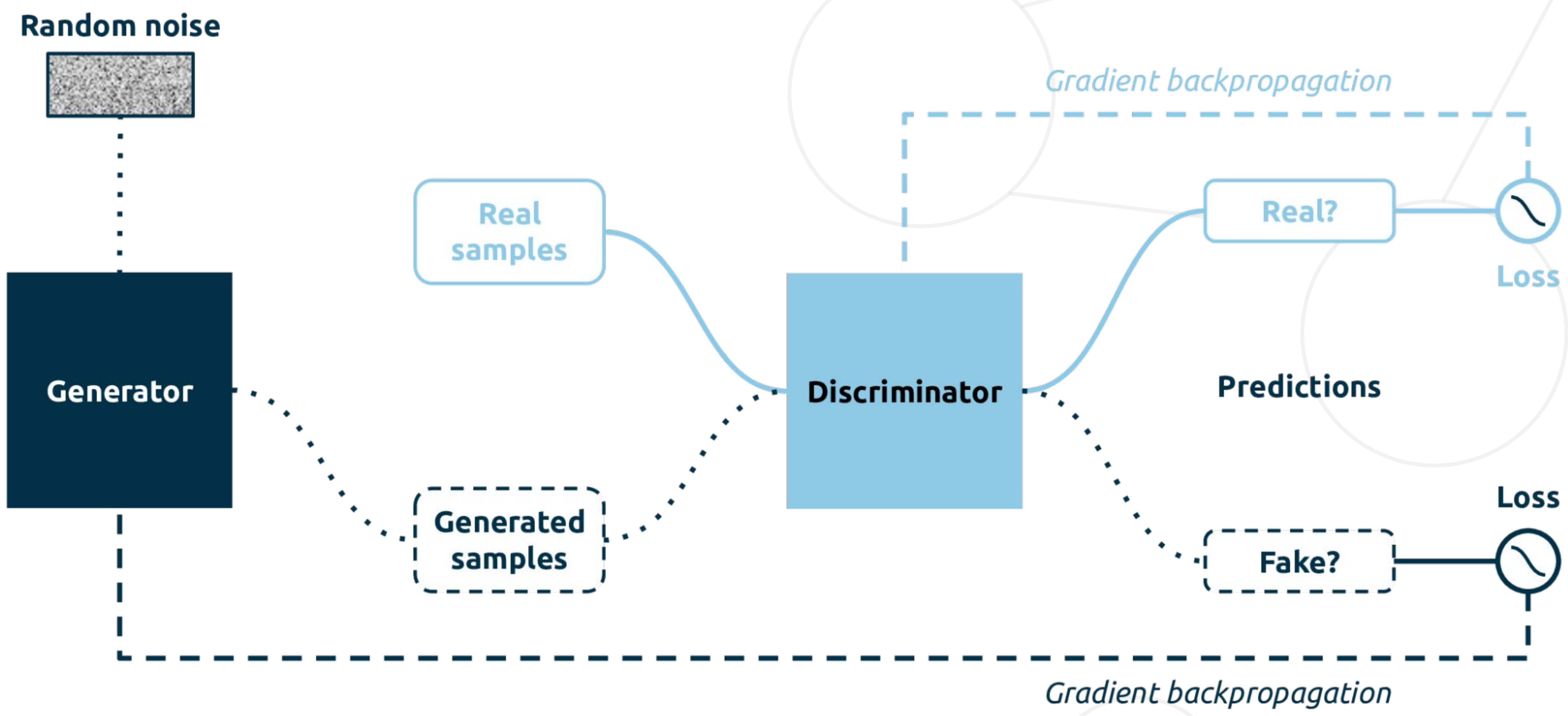
(i) (ii) (iii) (iv)

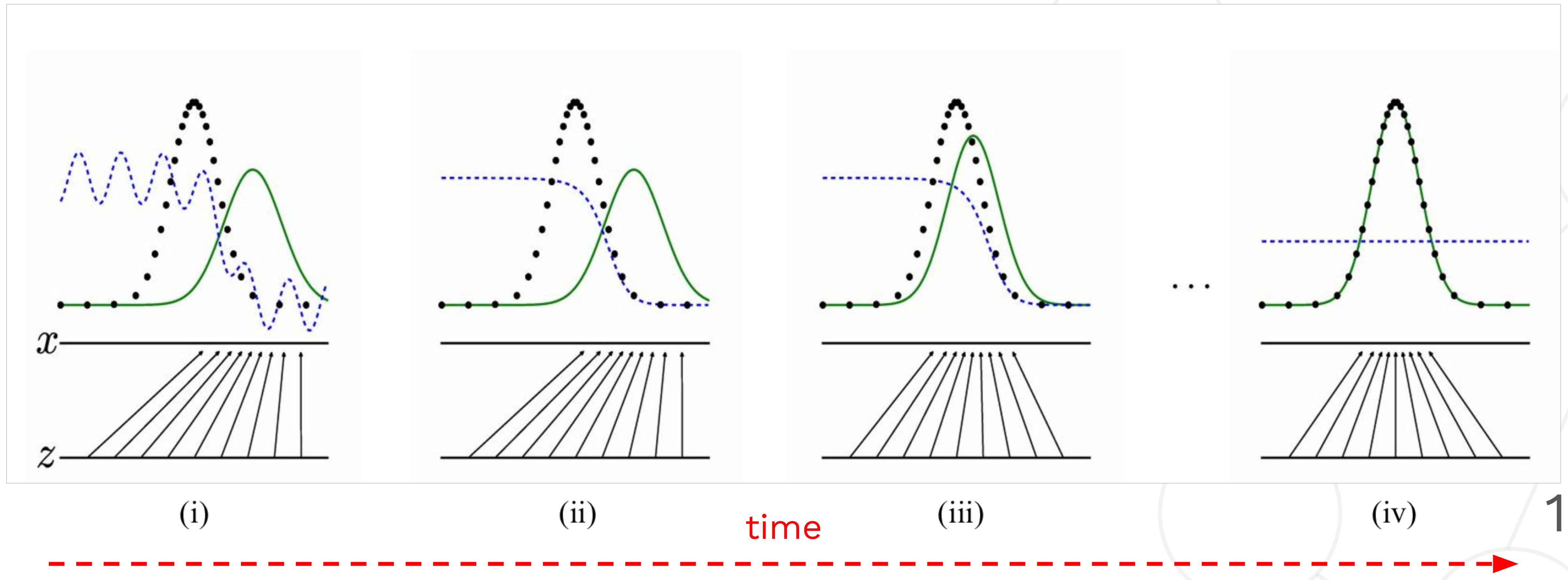
Generative Adversarial Networks (GANs)

Fundamentals of GANs

We summarize the GAN training as follows:

7. During this process, we adjust the learned distribution to the training distribution





Challenges and limitations of GANs

Fundamentals of GANs

- Each scenario may require a different configuration of network hyperparameters:
  - # of neurons, optimization algorithm (e.g., Adam, SGD), Loss Function (e.g., MSE, Binary Cross-Entropy)

Challenges and limitations of GANs

Fundamentals of GANs

- Each scenario may require a different configuration of network hyperparameters:
  - # of neurons, optimization algorithm (e.g., Adam, SGD), Loss Function (e.g., MSE, Binary Cross-Entropy)

- Which of the existing GAN models (e.g., StyleGANs, Diffusion GANs, TimeGANs) is more suitable for our scenario - i.e., packet/telemetry generation?
  - There is not a single “silver bullet” solution

Challenges and limitations of GANs

Fundamentals of GANs

- Each scenario may require a different configuration of network hyperparameters:
  - # of neurons, optimization algorithm (e.g., Adam, SGD), Loss Function (e.g., MSE, Binary Cross-Entropy)

- Which of the existing GAN models (e.g., StyleGANs, Diffusion GANs, TimeGANs) is more suitable for our scenario - i.e., packet/telemetry generation?
  - There is not a single “silver bullet” solution

- How about the timing requirements?
  - Inter- (e.g., three-way-handshake) and intra-packet (e.g., type of service in TCP headers)

GANs with RL in Network Applications

Fundamentals of GANs

Joint Applications of GANs and RL

- ***Potential Benefits:*** Combining GANs' ability to generate realistic data with RL's optimization capabilities offers significant potential for enhancing network configurations and policies.
- ***Sim-to-Real Discrepancy:*** Addressing differences between simulated and real network data is crucial for practical applications.

Case Studies and Research

- **Automated Network Slicing:** Use of GANs for generating synthetic data to train RL models, improving the efficiency of network slicing and reducing simulation-to-real discrepancies.
- **Resource Management in Network:** Integrating deep RL with distributional modeling using GANs to manage resources efficiently.
- **Estimating Channel Coefficients:** Leveraging synthetic data generated by GANs to train RL algorithms for more accurate channel coefficient estimations.

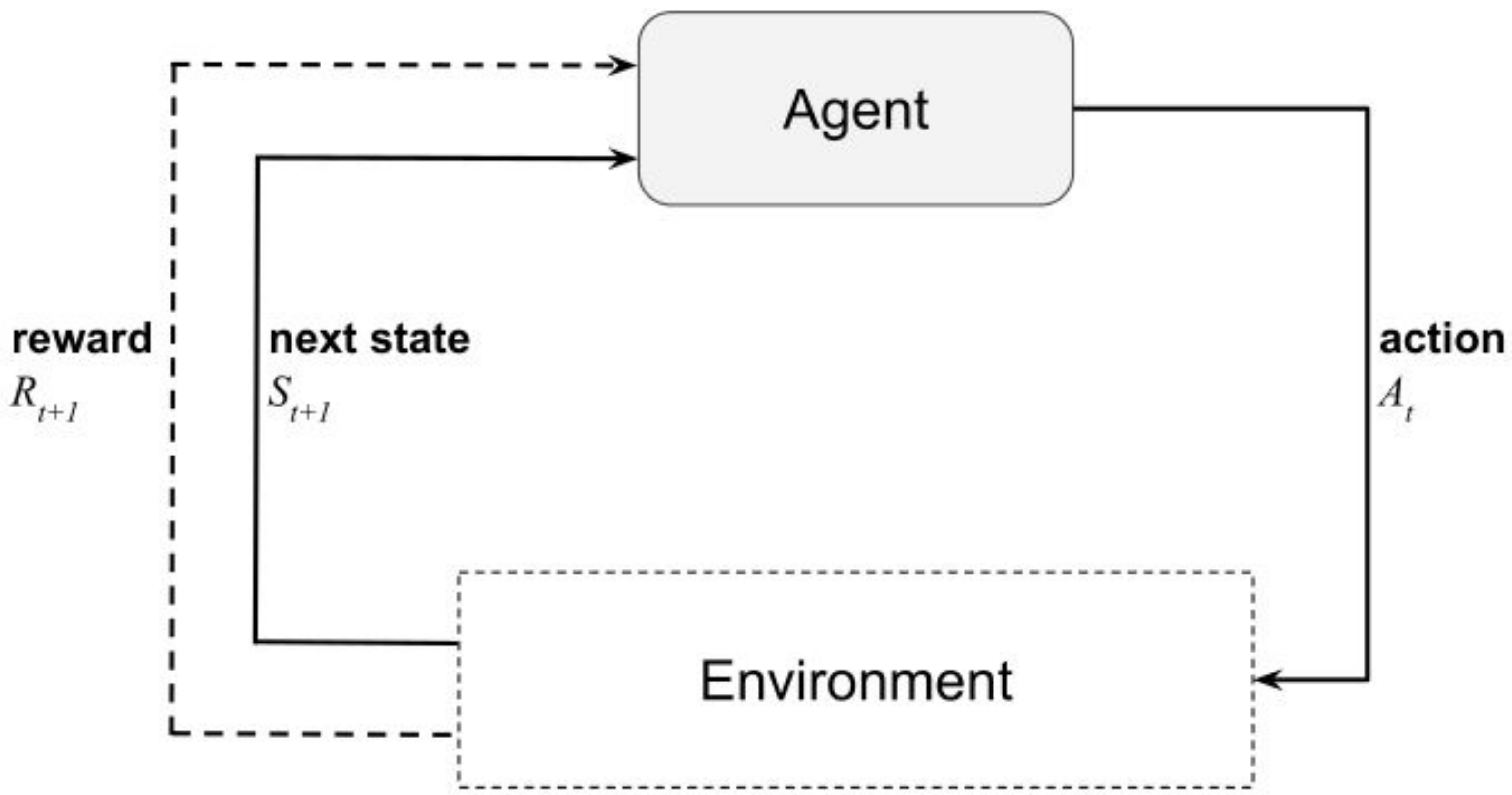
GANs with RL in Network Applications

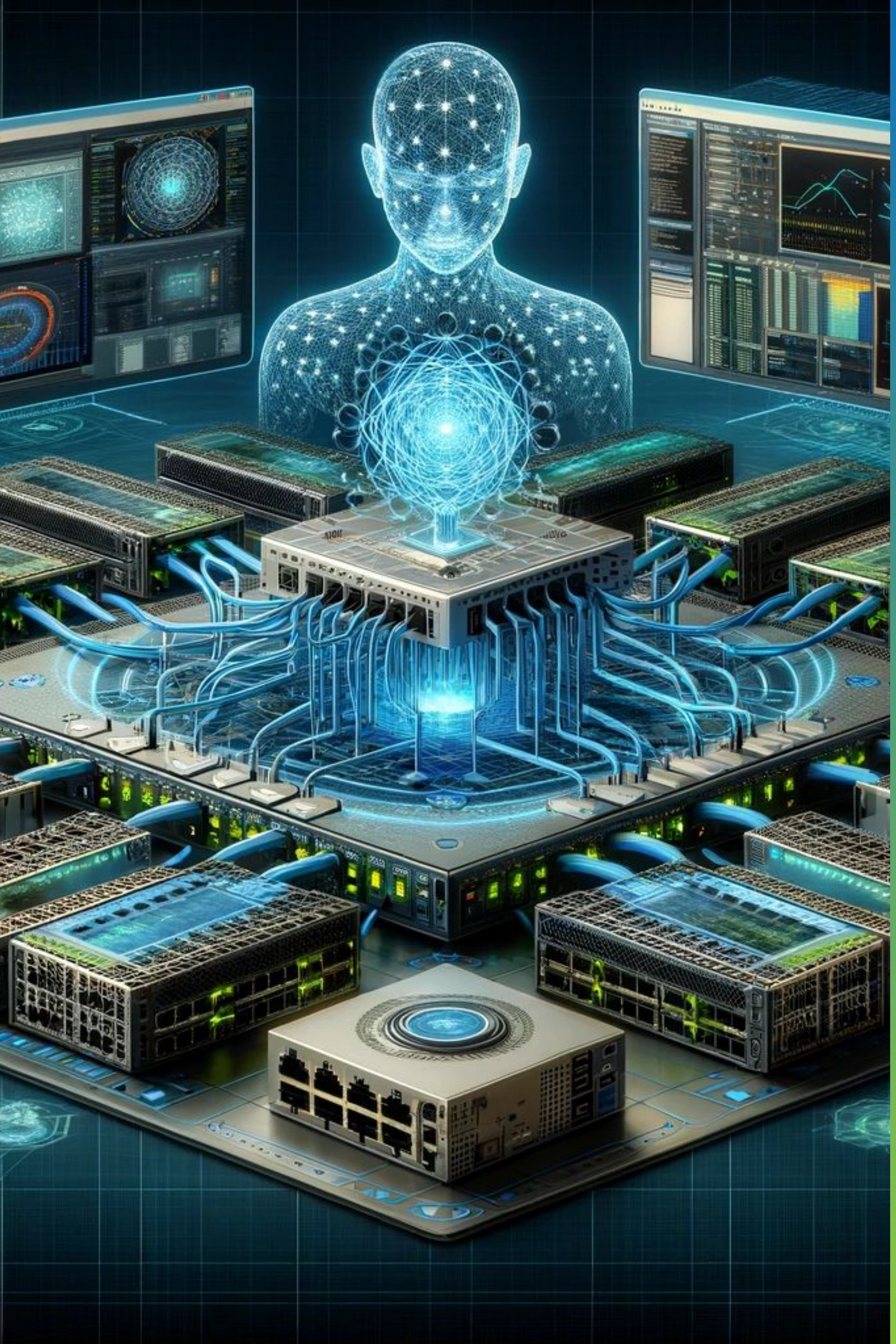
Fundamentals of GANs

Agent-Environment Interaction in MDP (Markov decision process)

1. **Time Step Initiation:** At each specific time t.
2. **Action Taken:** The agent takes an action A.
3. **State Observation:** The agent observes the subsequent state S_{t+1} resulting from its action.
4. **Reward Assessment:** A reward value R_{t+1} is generated for each interaction, assessing the effectiveness of the action.

Objective: The process aims to maximize the reward value throughout the agent's training process, guiding the agent toward optimal decision-making.





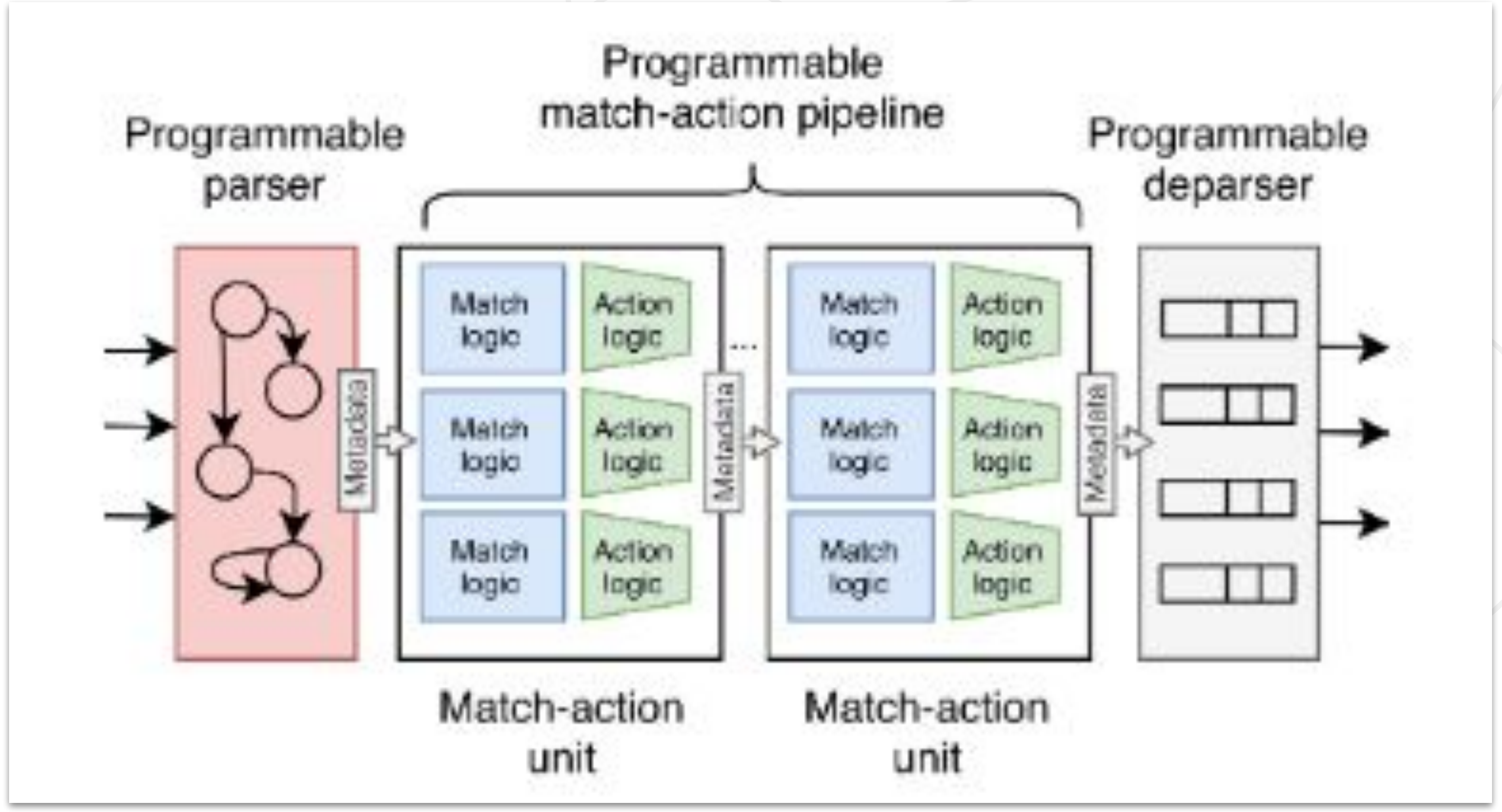
In-band Network Telemetry and Programmable Data Plane

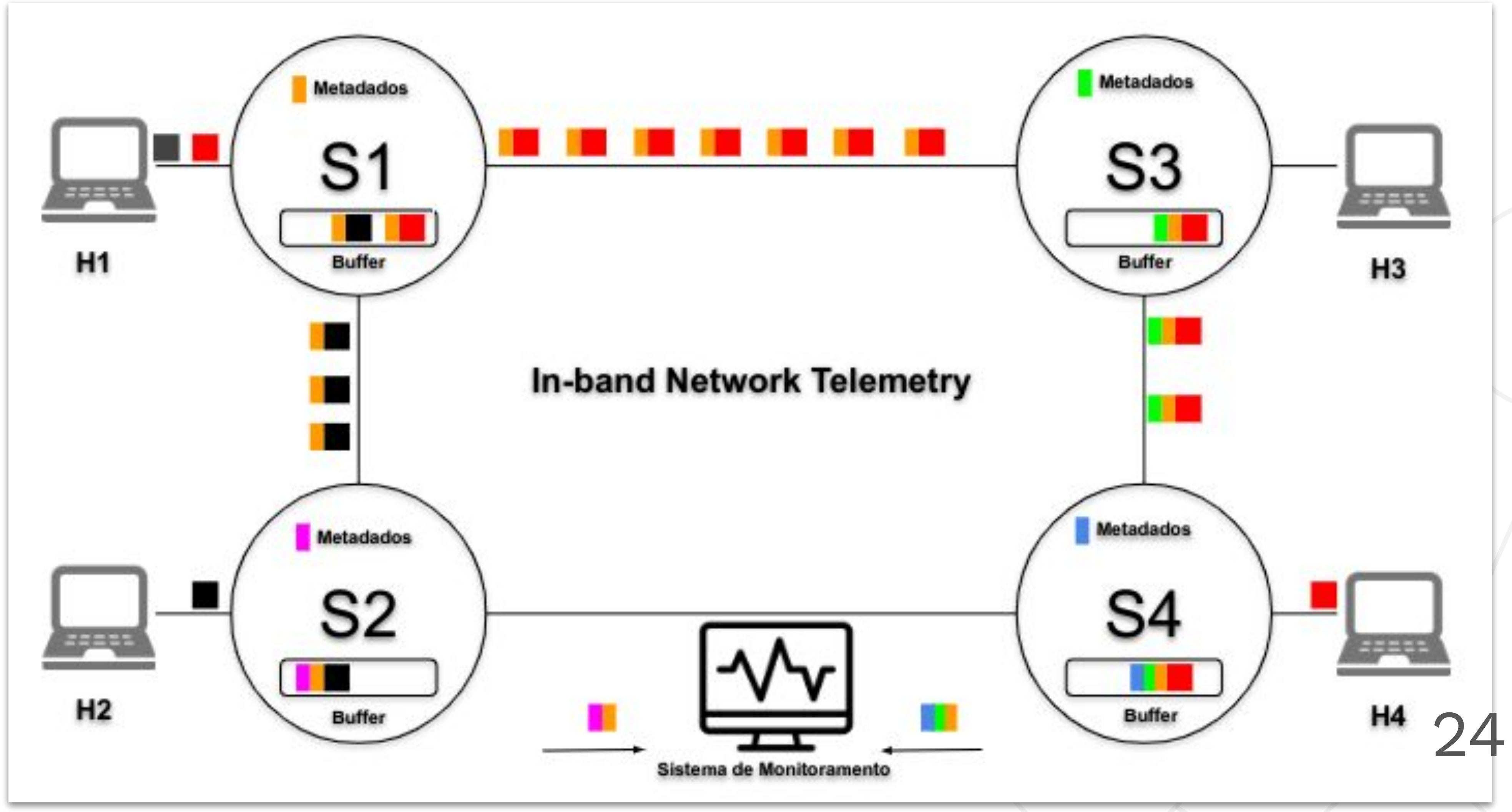
In-band Network Telemetry and Programmable Data Plane

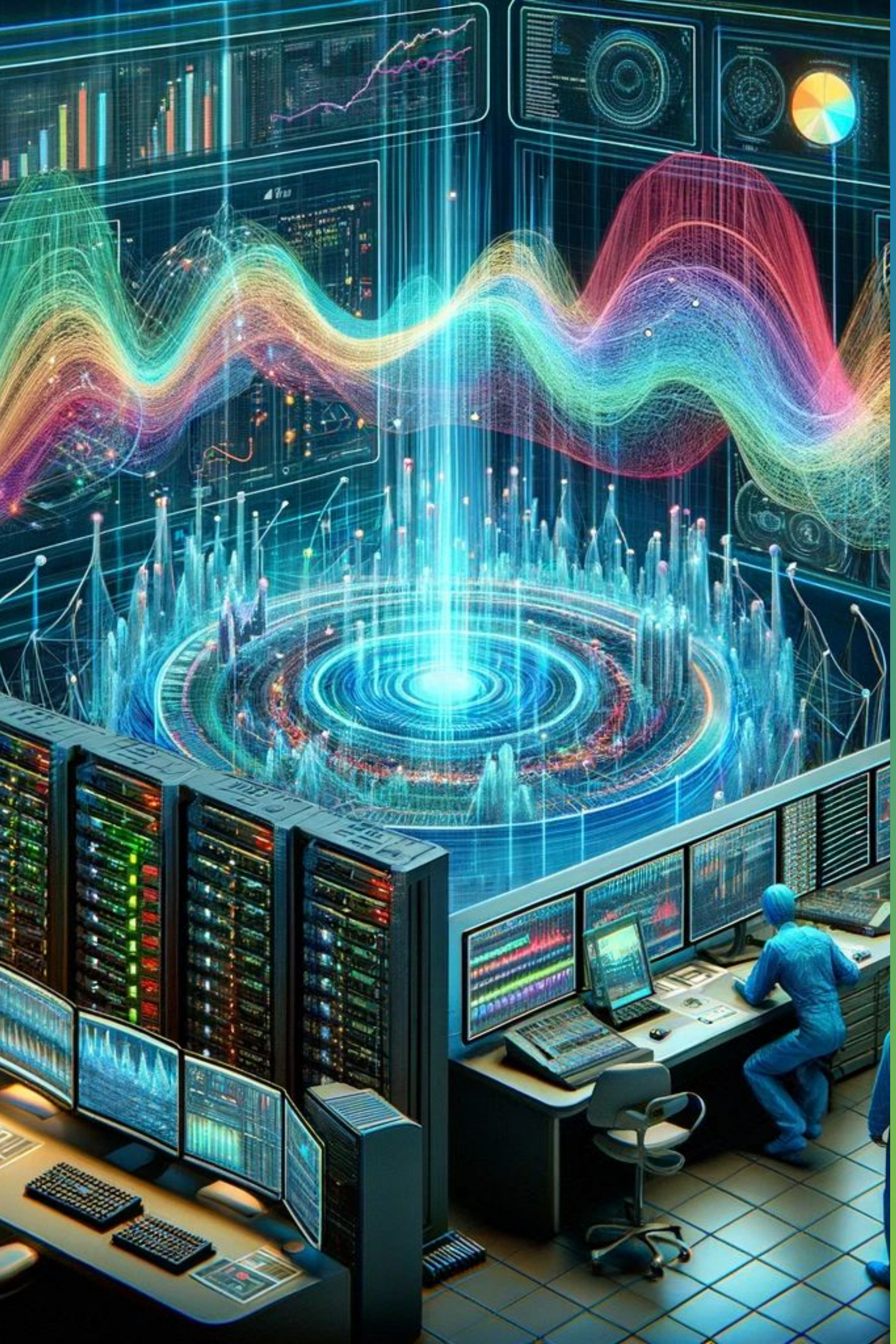
- The data plane programmability facilitates the incorporation of intelligence during packet processing at the hardware's most proximate level, without the necessity for control plane intervention.

- Packets incorporate telemetry instructions within their header fields, facilitating the fine-grained collection and recording of network data.

- At each network hop along these paths, the data plane of the network devices employs telemetry instructions to facilitate the collection and inclusion of metadata within the packets as they traverse each node.



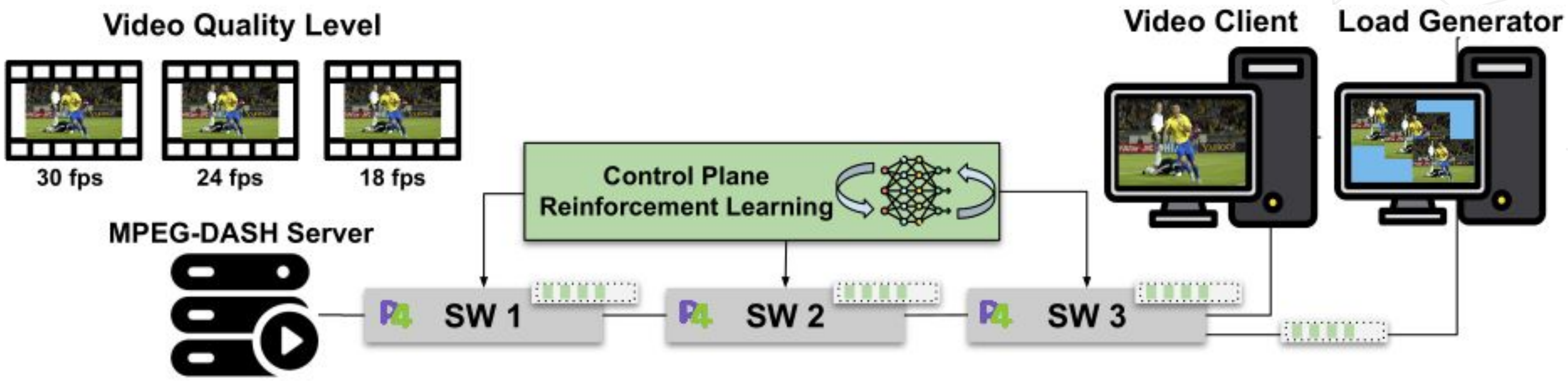




Generation of Telemetry Data

Problem definition

Generation of Telemetry Data

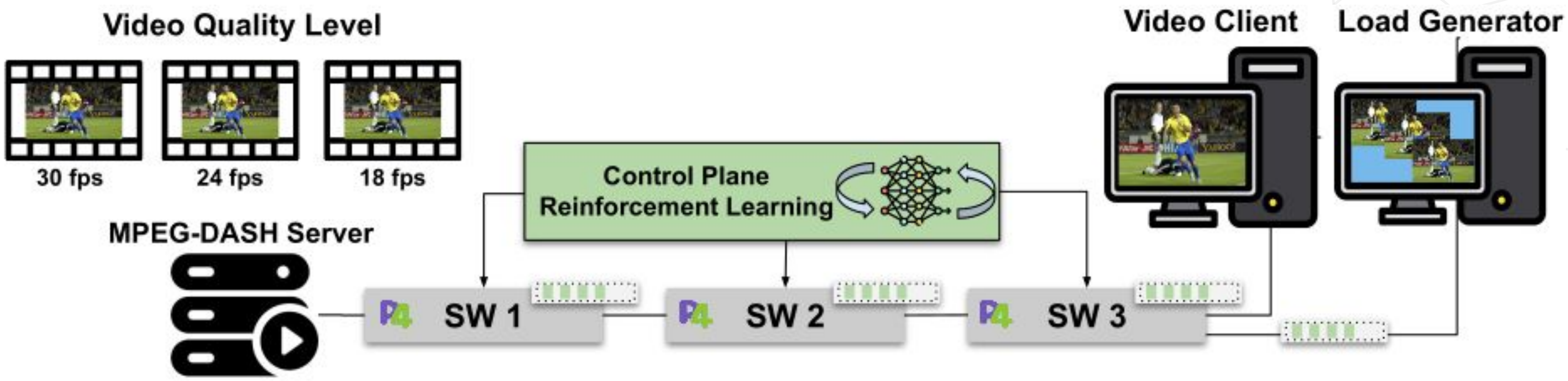


Infrastructure Overview

- **Setup Components:** Virtual machines interconnected via a P4 programmable data plane network.
- **Application Deployment:** CDN supporting MPEG-DASH for live streaming a soccer game.
- **Load Management:** WAVE, is versatile load generator used for orchestrating application instances over time.
- **Network Architecture:** Includes three programmable switches collecting INT telemetry data, complemented by a Video Client for metrics.

Problem definition

Generation of Telemetry Data

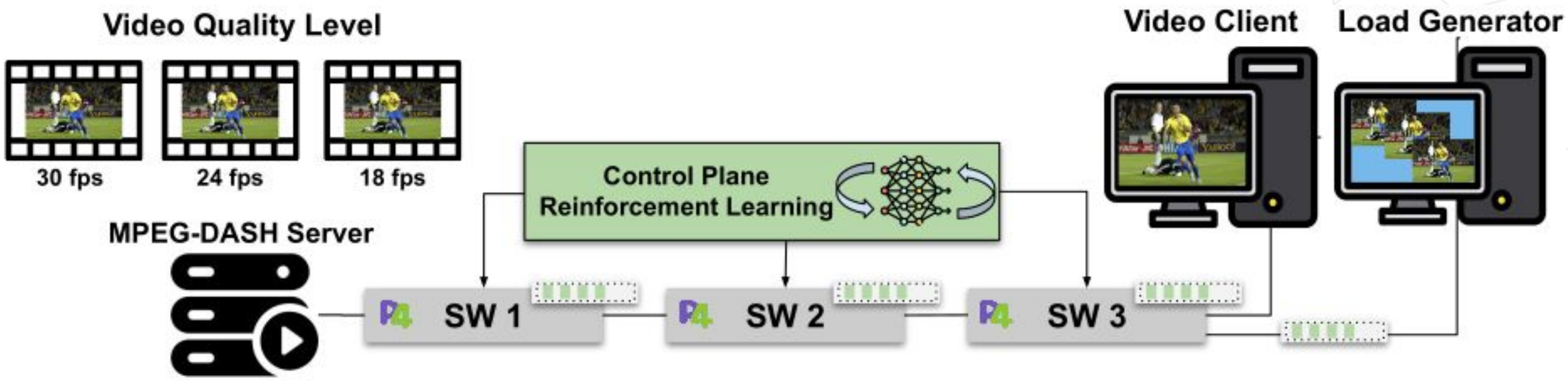


Role of RL Agent

- **Primary Function:** Operates as a data plane optimizer, managing queue sizes in switches to enhance user experience.
- **Goal:** Optimize resource utilization to improve network infrastructure efficiency.

Problem definition

Generation of Telemetry Data

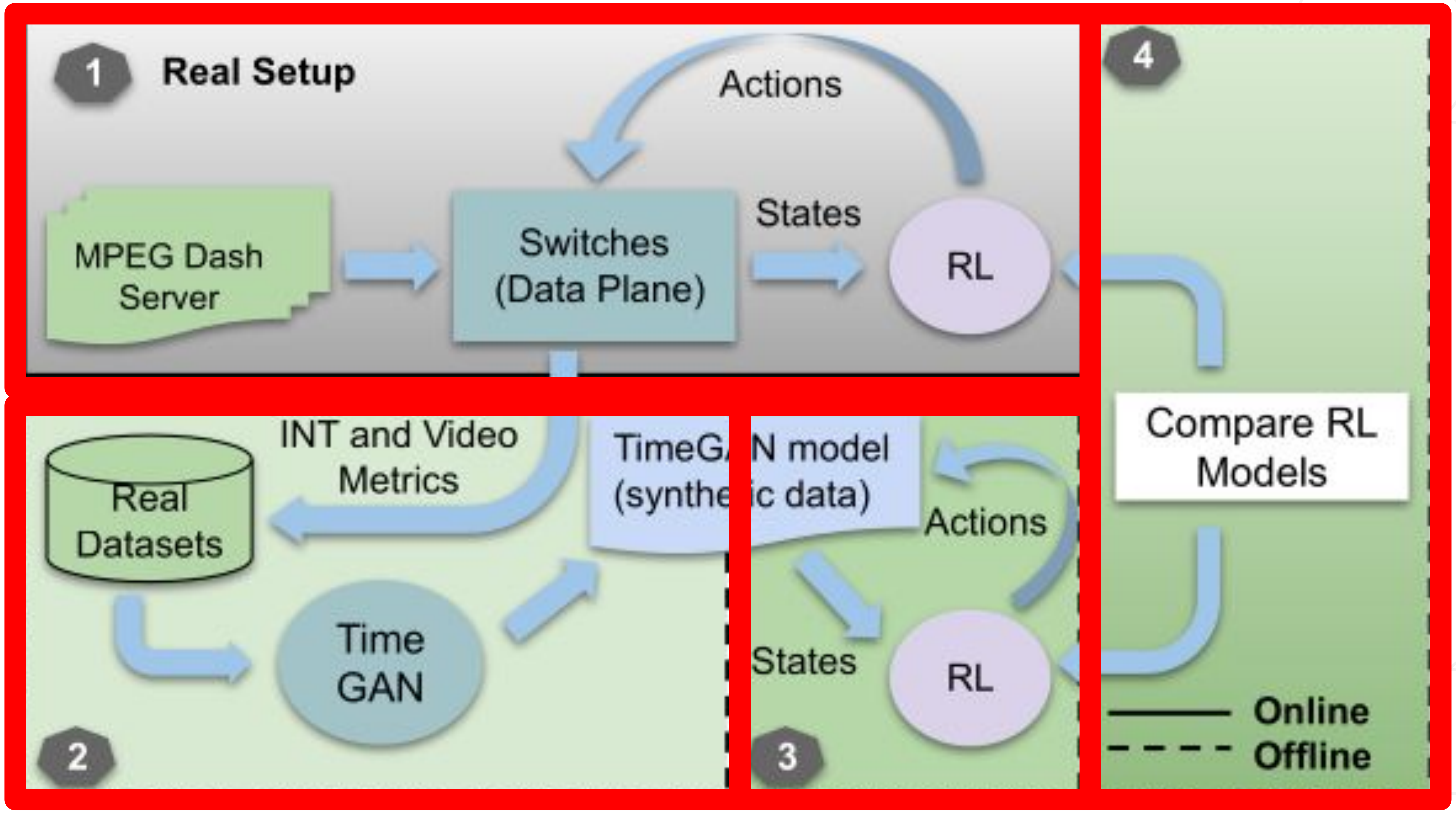


Data Plane Optimization

- **Challenges:** Gaining cooperation from network operators for experiments can be difficult.
- **Alternative Approach:** Use a GAN trained on real data as a simulator to train the RL agent without needing a real setup.

Methodology

Generation of Telemetry Data



Methodology

Generation of Telemetry Data

Training the RL Model

- ***Real Setup:*** RL agent trained using real data collected from the network.
- ***Synthetic Scenario:*** Offline training of the RL model using synthetic data from the TimeGAN to assess generalization capacity.

Importance of GAN in RL Training

- ***Dataset Issues:*** Original datasets may have imbalances, inadequacies, or erroneous values.
- ***GAN Advantages:*** Provides the ability to generate balanced, comprehensive data for diverse experimental scenarios.
- ***Efficiency Comparison:*** Evaluates the time efficiency of RL model training using synthetic versus real data.

Dataset Characterization

Generation of Telemetry Data

Dataset Composition

- ***Video Metrics:*** Frames per second (FPS), bitrate, buffer size.
- ***Network Metrics:*** Queue depth at packet queuing (Enq Qdepth), packet queuing duration in microseconds (Deq Timedelta), and queue depth at packet dequeuing (Deq Qdepth).

Experiments and Data Collection

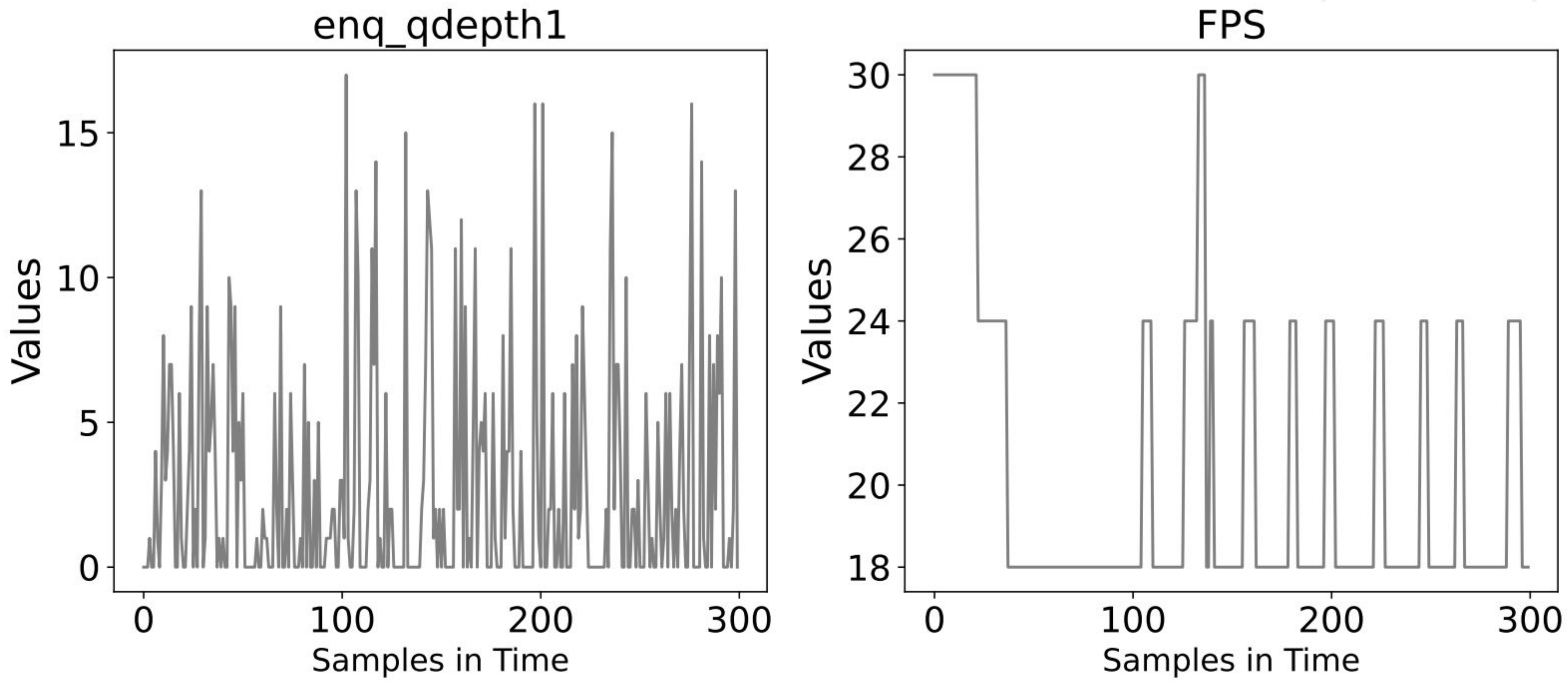
- **Buffer Size Configurations:** Two experiments with switch buffer sizes set at 32 and 64 packets.
- **Data Merging and Filtering:** Datasets merged based on timestamps, filtering out higher `Deq Timedelta' to focus on high-load conditions.

Dataset Characterization

Generation of Telemetry Data

Challenge of Non-Stationary Data

- ***Non-Stationarity:*** features within the Programmable Data Plane application exhibit non-stationary characteristics.
- ***Visualization:*** Non-stationarity visually demonstrated in the figure alongside.
- ***Implication:*** This nature complicates direct comparison between real telemetry data and synthetic data generated by GANs.



TimeGAN Training

Generation of Telemetry Data

Data Preprocessing

- ***Steps:*** Address missing data, remove outliers, and normalize data to prepare for effective training.

Hyperparameter Configuration

- ***Importance:*** Critical for optimizing the training regimen.
- ***Parameters:*** Sequence sizes, sequence length, number of hidden dimensions, batch size.

Hyperparameter Tuning

- ***Method:*** Empirical approach with iterative adjustments based on training outcomes and insights.
- ***Challenges:*** Identifying optimal settings due to the impact on model performance.

Model Selection

Generation of Telemetry Data

SMARTNESS

Lack of Consensus in Evaluating Synthetic Data Generated by GANs

- Highlighted in (Brophy, 2023), there is no agreed method to assess distributions created by GANs, specially for time series data.

Complexity with Non-Stationary Data

- Non-stationary time series show varying statistical properties over time, complicating traditional evaluation methods like KL divergence.
- Real data variability vs. synthetic data constancy can lead to misleading results when using traditional metrics.

Model Selection

Generation of Telemetry Data

SMARTNESS

Developing a New Metric

Designed to assess the similarity between real and synthetic data distributions, focusing on statistical measures.

- **Metric Calculation:**
  - ***Interquartile Discrepancy:*** Measures the difference in dispersion between real (X) and synthetic (y) datasets.
  - ***Median Difference:*** Addresses positional differences between distributions.

$$\mathbf{M} = \sum_{n=1}^{n\_feats} |[Q_3(X_n) - Q_1(X_n)] - [Q_3(y_n) - Q_1(y_n)]| + |[med(X_n) - med(y_n)]|$$

Hands on

Two Possibilities

GitHub

GitHub

Generation of Telemetry Data

- Clone the following project from GitHub:
    - https://github.com/thiagocaproni/tutorial_timegan
- After cloning the project from GitHub, create the environment by running the following command (where 'environment.yml' is located)
    - `conda env create`
- Then, type the following command:
    - `conda activate ydata`

Google Colab

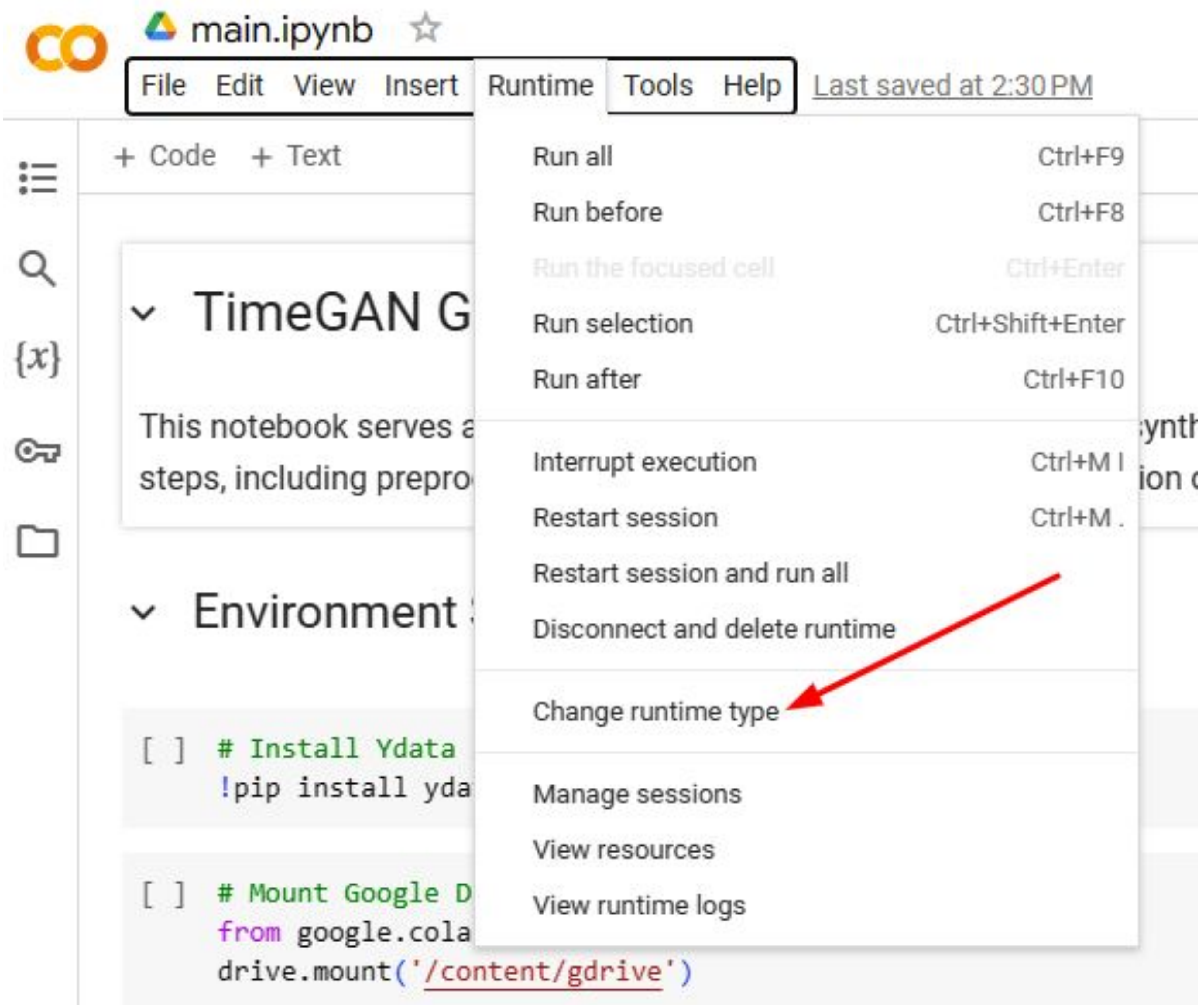
Generation of Telemetry Data

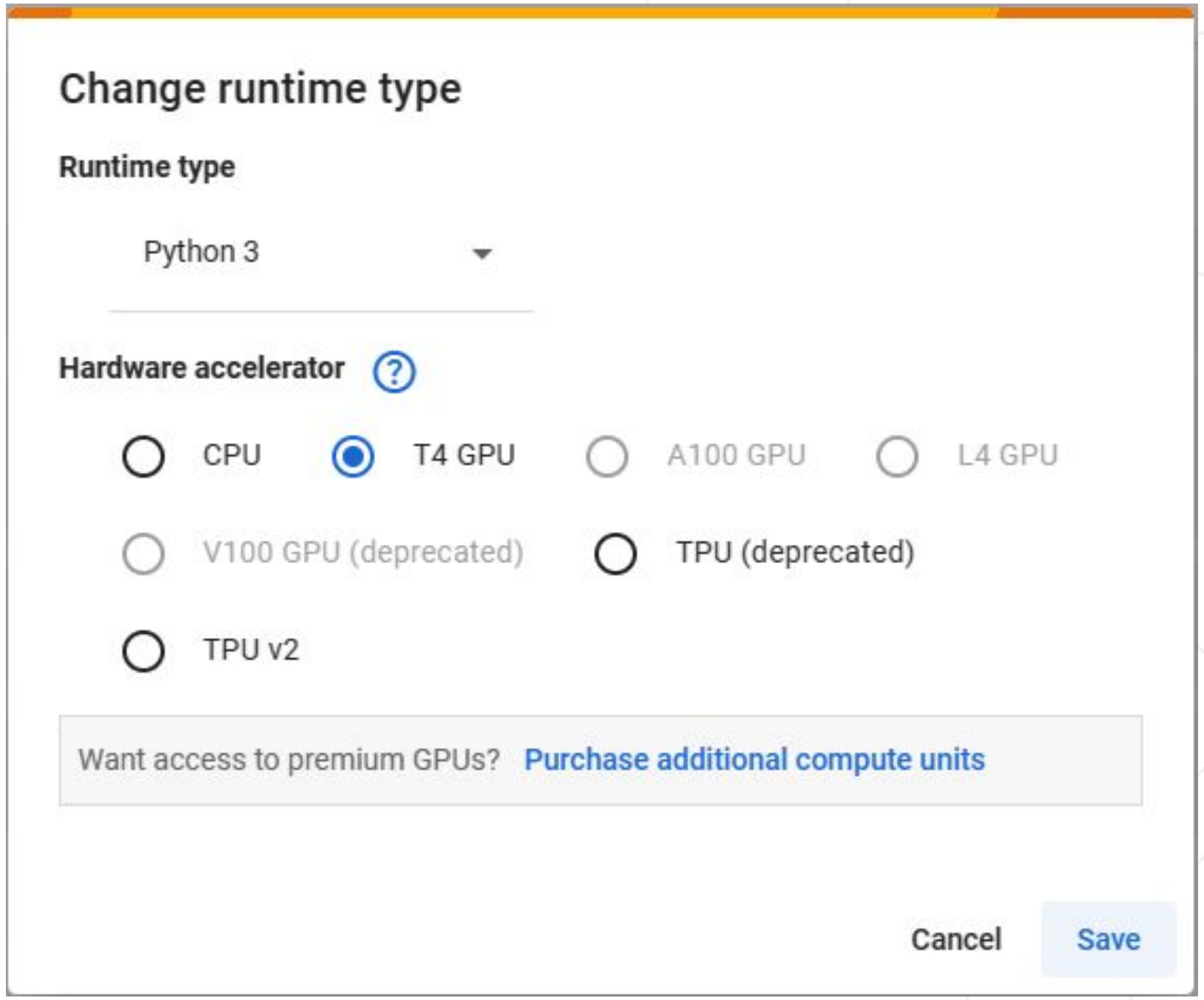
1. Download the zip file from the following link:
    - https://drive.google.com/file/d/1kR0teCHU4Z2jCez75kcM61GArmP0CAfY/view?usp=sharing
2. Unzip the file and upload the "tutorial" folder to the Google Drive root folder
    - All paths used in Python scripts and notebooks are executed considering that the "tutorial" folder is in the Google Drive root directory.
3. Navigate to the following folder
    - tutorial -> code -> timegan
4. Open the notebook:
    - main.ipynb

Google Colab

Generation of Telemetry Data

5. Make sure, you are using a Python 3 session:

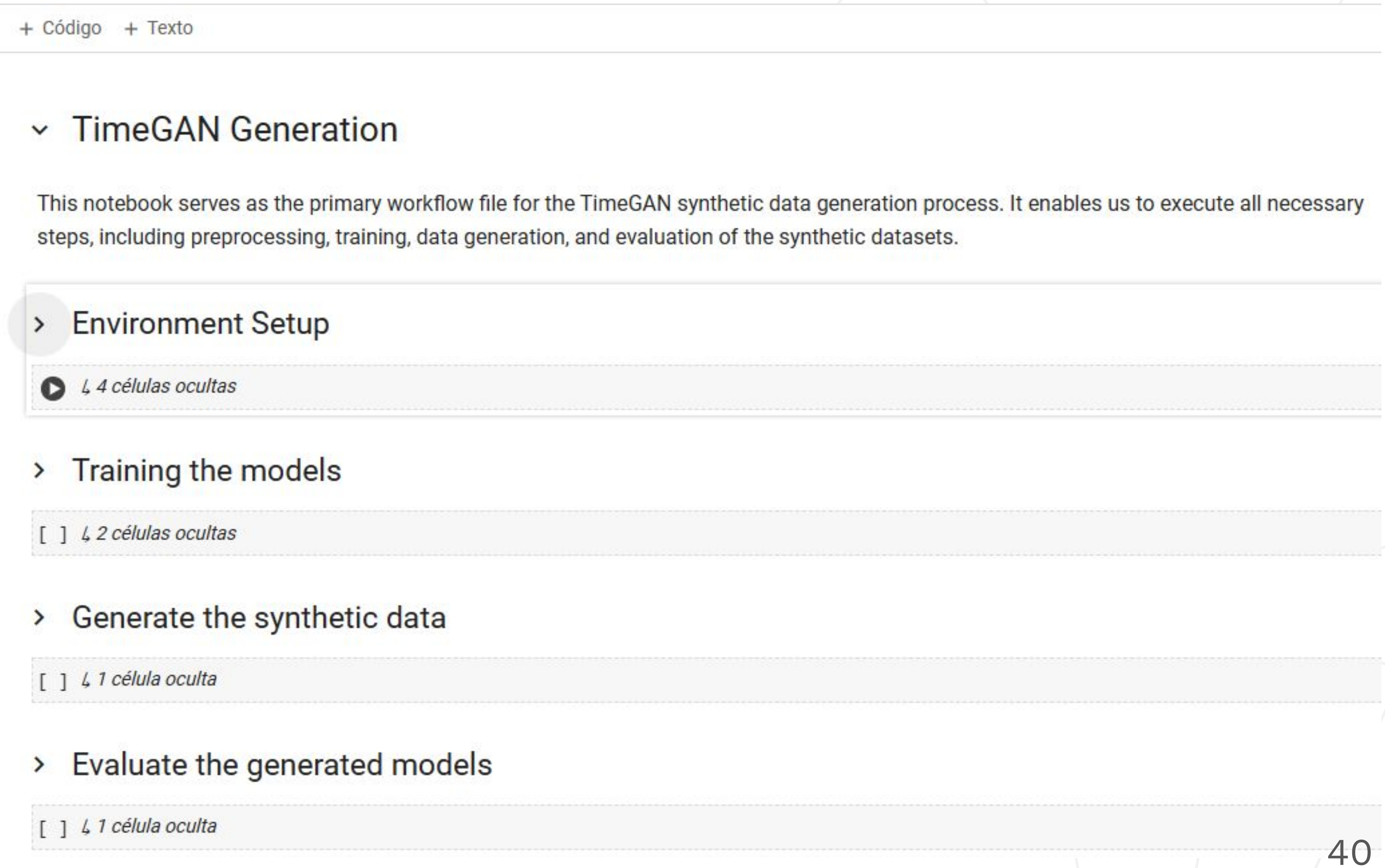




Google Colab

Generation of Telemetry Data

6. The first part is compound with cells to setup the environment:



Open main notebook

Generation of Telemetry Data

7. Run the cells to setup the environment in order to:
    a. Install the ydata-synthetic module (probably you be asked to restart the session)
    b. Mount the Google Drive folder
    c. Access the “timegan” folder
    d. Append the “data_process” folder where is located the script to preprocessing

Environment Setup

```
[ ] # Install Ydata Synthetic
    !pip install ydata-synthetic==1.1.0
```

```
[ ] # Mount Google Drive folder
    from google.colab import drive
    drive.mount('/content/gdrive')
```

```
[ ] # Access the folder timegan
    %cd /content/drive/MyDrive/tutorial/code/timegan
```

```
[ ] import sys
    sys.path.append('../data_process/')
```

Parameters file

Generation of Telemetry Data

1. Open the file: (code/timegan/params.py)
   a. The script has the `amount_of_models` variable
   b. Its value is factored to define the values of the following hyperparameters:
      - `seq_len (i)`: the sequence length would be the size of the temporal window of each sequence used to train the model, that is, how many time steps (lines) each sequence contains.
      - `hidden_dim (j)`: Number of units or neurons in each hidden layer
      - `batch_size (k)`: The batch size determines how many temporal sequences (or how many data examples/lines) are included in a single batch for training.
   c. The `fatNum` function in the script `model_utility.py` returns the values of `i`, `j` and `k` that is used in several other scripts to assign the hyperparameters variations for training, generating and evaluating models.

Preprocessing
(code/data_process/preprocess_data.py)

Generation of Telemetry Data

```
def loadDataSet(self, path_int, path_dash):
        # Load the INT and DASH datasets from specified paths
        df_int = pd.read_csv(path_int, sep = ',')
        df_dash = pd.read_csv(path_dash, sep = ';')

        # from milliseconds to seconds
        self.transformTimeStamp(df_dash)
        df_int = df_int.loc[df_int.groupby('timestamp')['deq_timedelta1'].idxmax()]

         # Set 'timestamp' as the index for the INT dataset
        df_int.set_index('timestamp', inplace=True)

        # Merge the INT and DASH datasets on their timestamp indices and reset the merged DataFrame's index
        self.dataset = pd.merge(df_int, df_dash, left_index=True, right_index=True).reset_index()
```

Preprocessing
(code/data_process/preprocess_data.py)

Generation of Telemetry Data

```
def preProcessData(self, num_cols, cat_cols, random):
    # Fill missing values in numerical columns with their mean
    for i in num_cols:
        self.dataset[i].fillna(self.dataset[i].mean(), inplace=True)

    # Perform one-hot encoding if there are categorical columns
    if len(cat_cols) > 0:
        self.hotEncode()
        cat_cols = [0,1,2]

    # Create a copy of the dataset with only the processed columns
    self.processed_data = self.dataset[ num_cols + cat_cols ].copy()
    self.cat_cols = cat_cols
    self.num_cols = num_cols

     # Randomly shuffle the dataset if requested
    if random == True:
        idx = np.random.permutation(self.processed_data.index)
        self.processed_data = self.processed_data.reindex(idx)
```

Train (code/timegan/train/timegan32.py)

Generation of Telemetry Data

```
def loadDp(random, outliers):
    dp = DataPre()

    #Loading and mergint INT and DASH datasets
    dp.loadDataSet(path_int='../../../datasets/log_INT_TD-32_100.csv',
                   path_dash='../../../datasets/dash_TD-32_100.csv')

    #preprocessing data
    dp.preProcessData(params.num_cols, cat_cols=params.cat_cols, random=random)

    #removing columns with same values
    dp.removeSameValueAttributes()

    if outliers == False:
        dp.removeOutliers()

    #printing processed data
    dp.processed_data

    return dp
```

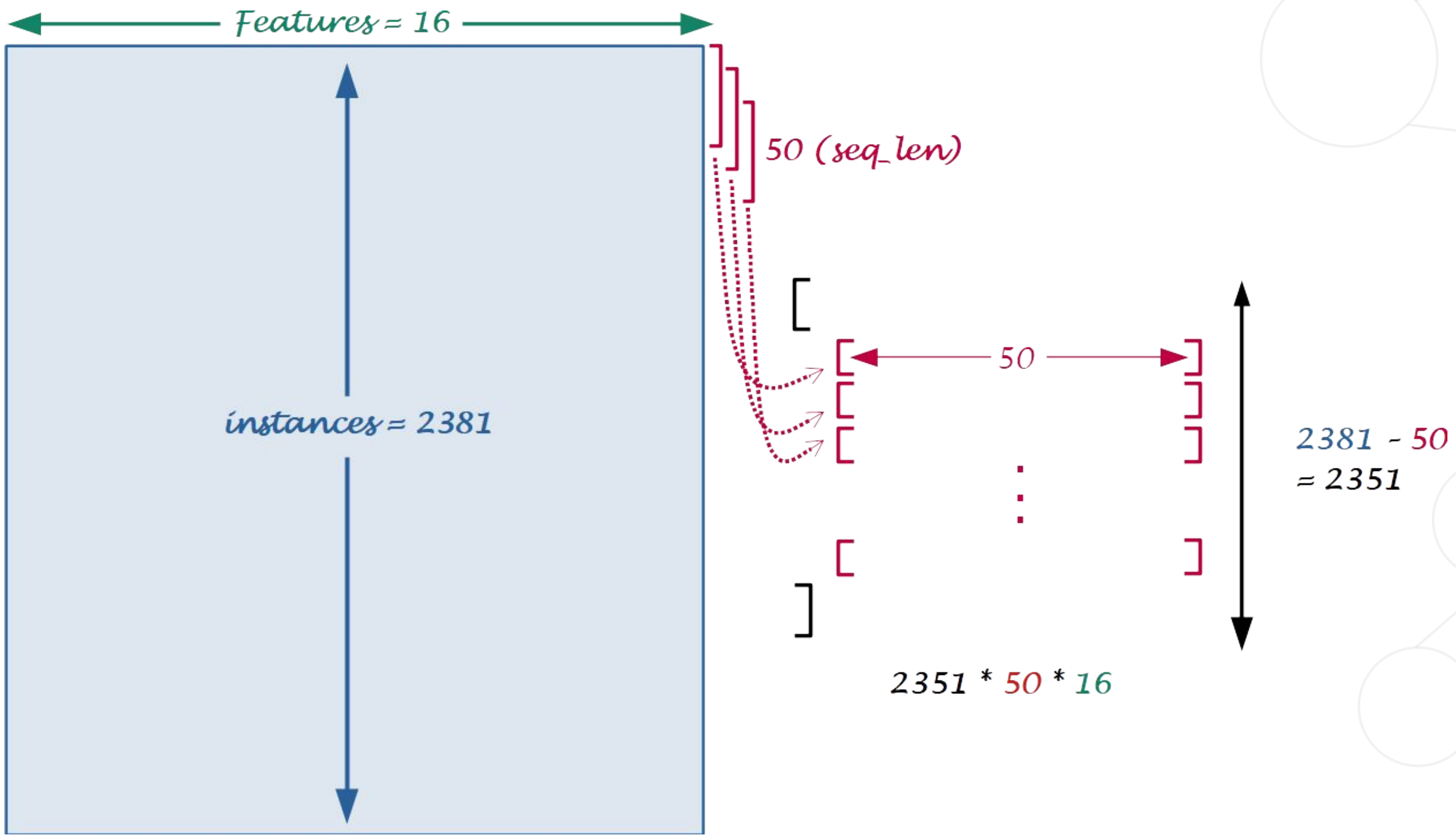
Train (code/timegan/train/timegan32.py)

Generation of Telemetry Data

```
def train(dp, seq_len, n_seq, hidden_dim, noise_dim, dim, batch_size, model, train_steps):
    learning_rate = 5e-4

    gan_args = ModelParameters(batch_size=batch_size,
                               lr=learning_rate,
                               noise_dim=noise_dim,
                               layers_dim=dim)


    #normalizing the data
    processed_data = real_data_loading(dp.processed_data.values, seq_len=seq_len)

    synth = TimeGAN(model_parameters=gan_args, hidden_dim=hidden_dim, seq_len=seq_len, n_seq=n_seq, gamma=1)
    synth.train(processed_data, train_steps=train_steps)
    synth.save(model)
```

Train (code/timegan/train/timegan32.py)

Generation of Telemetry Data



Train (code/timegan/train/timegan32.py)

Generation of Telemetry Data

```
dp = loadDp(random=False, outliers=False)

iMax, jMax, kMax = ModelUtility.fatNum(params.amount_of_models) # Change the file params.py
print("\nNumber of models" + str(params.amount_of_models) + ' iMax: ' + str(iMax) + ' jMax: ' + str(jMax) + ' kMax: ' + str(kMax))

print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

try:
  # Specify an valid GPU device
  with tf.device('/device:GPU:0'):
    for i in range(0,iMax):
      for j in range(0,jMax):
        for k in range(0,kMax):
          train(dp,
            seq_len=(50*(i)+50),
            n_seq=params.merged_columns_len,
            hidden_dim=(20*(j)+20),
            noise_dim=32,
            dim=128,
            batch_size=(28*(k) + 100),
            model=str('../saved_models/so32_seqlen_'+ str((50*(i) + 50)) + '_hidim_' + str(20*(j)+20) + '_batch_' +  str(28*(k) + 100) + '.pkl'),
            train_steps=params.train_steps)
except RuntimeError as e:
  print(e)
```

SMARTNESS

Data Generation

(code/timegan/data_generation/generate_synth_data.py)

Generation of Telemetry Data

```
def loadSynthData(model32, model64, number_of_windows):
    synth_32 = TimeGAN.load(model32)
    synth_data_32 = synth_32.sample(number_of_windows)

    synth_64 = TimeGAN.load(model64)
    synth_data_64 = synth_64.sample(number_of_windows)

    synth_data_32[:,:,13:16][synth_data_32[:,:,13:16] >= 0.5] = 1
    synth_data_32[:,:,13:16][synth_data_32[:,:,13:16] < 0.5] = 0
    synth_data_64[:,:,13:16][synth_data_64[:,:,13:16] >= 0.5] = 1
    synth_data_64[:,:,13:16][synth_data_64[:,:,13:16] < 0.5] = 0


    return synth_data_32, synth_data_64
```

SMARTNESS

Data Generation
(code/timegan/data_generation/generate_synth_data.py)

Generation of Telemetry Data

```
def loadRealData(dsint32, dsint64, dsdash32, dsdash64, num_cols, cat_cols, sample_size, randon, outliers):
    dp32 = DataPre()
    dp32.loadDataSet(path_int=dsint32, path_dash=dsdash32)
    dp32.preProcessData(num_cols, cat_cols=cat_cols, random=randon)
    if outliers == False:
        dp32.removeOutliers()

    real_data_32 = dp32.processed_data
    real_data_32 = real_data_32[0:sample_size].copy()
    real_data_32 = real_data_32.values

    #loading 64 bit buffer dataset
    dp64 = DataPre()
    dp64.loadDataSet(path_int=dsint64, path_dash=dsdash64)
    dp64.preProcessData(num_cols, cat_cols=cat_cols, random=randon)
    if outliers == False:
        dp64.removeOutliers()

    real_data_64 = dp64.processed_data
    real_data_64 = real_data_64[0:sample_size].copy()
    real_data_64 = real_data_64.values

    return real_data_32, real_data_64
```

Data Generation
(code/timegan/data_generation/generate_synth_data.py)

Generation of Telemetry Data

```
def getStatistics(data):
    median = np.median(data)
    percentile_25 = np.percentile(data, 25)
    percentile_75 = np.percentile(data, 75)

    return [percentile_25, median, percentile_75]

def genStatisctics(real_32, synth_32, real_64, synth_64, sample_size, num_cols):
    dict = {}

    for j, col in enumerate(num_cols):
        dict[col] = [getStatistics(real_32[:,j][:sample_size]),
                     getStatistics(synth_32[:,j][:sample_size]),
                     getStatistics(real_64[:,j][:sample_size]),
                     getStatistics(synth_64[:,j][:sample_size])]

    return dict
```

Data Generation
(code/timegan/data_generation/generate_synth_data.py)

Generation of Telemetry Data

```
def createDataSet(seq_len, data):
    lines =  int(params.synth_sample_size/seq_len)
    dataset = np.zeros(lines * seq_len * params.merged_columns_len).reshape(lines*seq_len, params.merged_columns_len)

    for i in range(0,lines):
        for j in range(0, seq_len):
            dataset[(i*seq_len) + j] = data[i][j][:]

    return dataset
```

Data Generation
(code/timegan/data_generation/generate_synth_data.py)

Generation of Telemetry Data

```
def getMetrics(statistic_data):
    metric32 = (abs( (statistic_data[0][2] - statistic_data[0][0]) - (statistic_data[1][2] - statistic_data[1][0])) +
                abs(  statistic_data[0][1] - statistic_data[1][1] ) )

    metric64 = (abs( (statistic_data[2][2] - statistic_data[2][0]) - (statistic_data[3][2] - statistic_data[3][0])) +
                abs(  statistic_data[3][1] - statistic_data[2][1] ) )

    return metric32, metric64

def get_allfeatures_metrics(metrics, model_index, statistic_data):
    for j, col in enumerate(params.num_cols):
        metrics[0][model_index][j], metrics[1][model_index][j] = getMetrics(statistic_data.get(col))
```

Model Selection

code/timegan/evaluation/analyze_data_models.ipynb

Generation of Telemetry Data

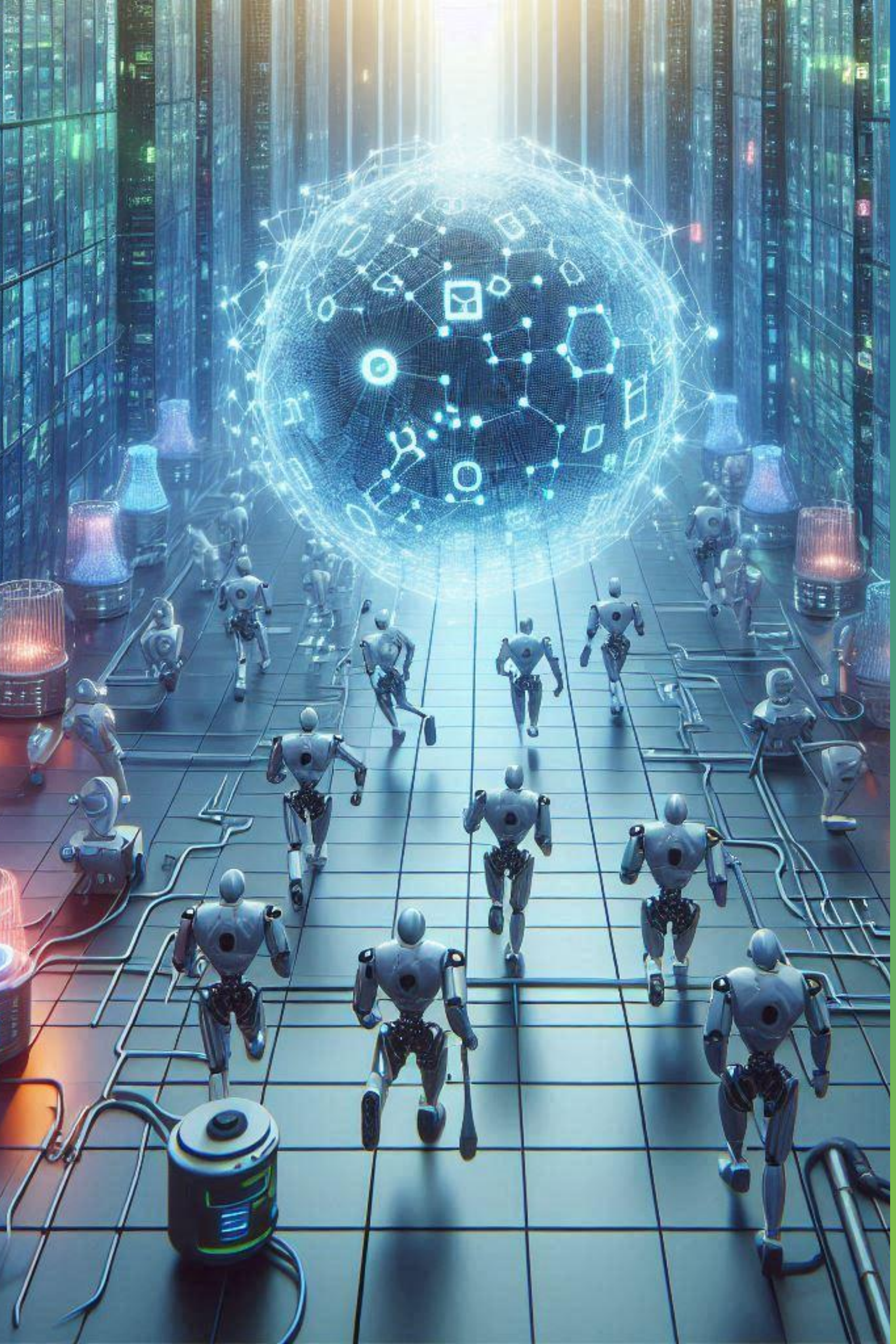
```
def getFeaturesBestMetricsOfModels(models, metrics):
    sum32, sum64 = sumFeatureMetricsOfModels(models, metrics)

    index = np.argmin(sum32)
    model = getModelNameByIndex(index)
    #print('bestmodel_int32: ' + model + ' index: ' + str(index))
    best_32 = models.get(model)[0]

    index = np.argmin(sum64)
    model = getModelNameByIndex(index)
    #print('bestmodel_int64: ' + model + ' index: ' + str(index))
    best_64 = models.get(model)[0]

    index = np.argmax(sum32)
    model = getModelNameByIndex(index)
    #print('worst_int32: ' + model + ' index: ' + str(index))
    worst_32 = models.get(model)[0]

    index = np.argmax(sum64)
    model = getModelNameByIndex(index)
    #print('worst_int32: ' + model + ' index: ' + str(index))
    worst_64 = models.get(model)[0]

    return best_32, worst_32, best_64, worst_64
```

Model Selection

code/timegan/evaluation/analyze_data_models.ipynb

Generation of Telemetry Data

```
def sumFeatureMetricsOfModels(models, data_metrics):
    sum32 = np.zeros(len(models))
    sum64 = np.zeros(len(models))

    for i in range(len(models)):
            sum32[i] = sum(data_metrics[0,i,:])
            sum64[i] = sum(data_metrics[1,i,:])

    return sum32, sum64
```



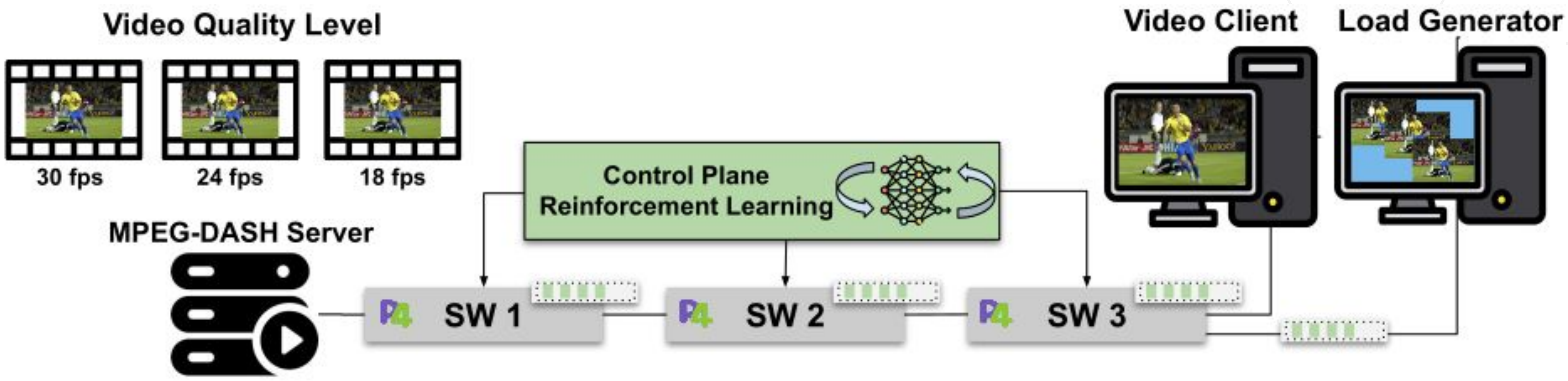
SMARTNESS

Applying synthetic data to an RL agent

Applying synthetic data to an RL agent

Generation of Telemetry Data

Real setup



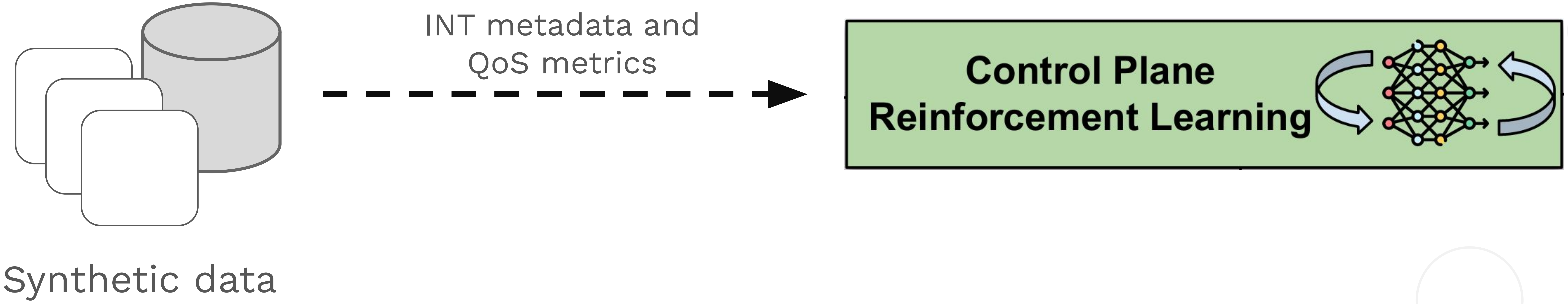
Challenges

- The infrastructure requirements may not be feasible for a real setup;
- The agent training time depends on the video streaming duration.

Applying synthetic data to an RL agent

Generation of Telemetry Data

Possible solution



Applying synthetic data to an RL agent

Generation of Telemetry Data

How can we implement it?

Read INT metadata regarding the 32-bit and 64-bit queue sizes from their respective CSV files

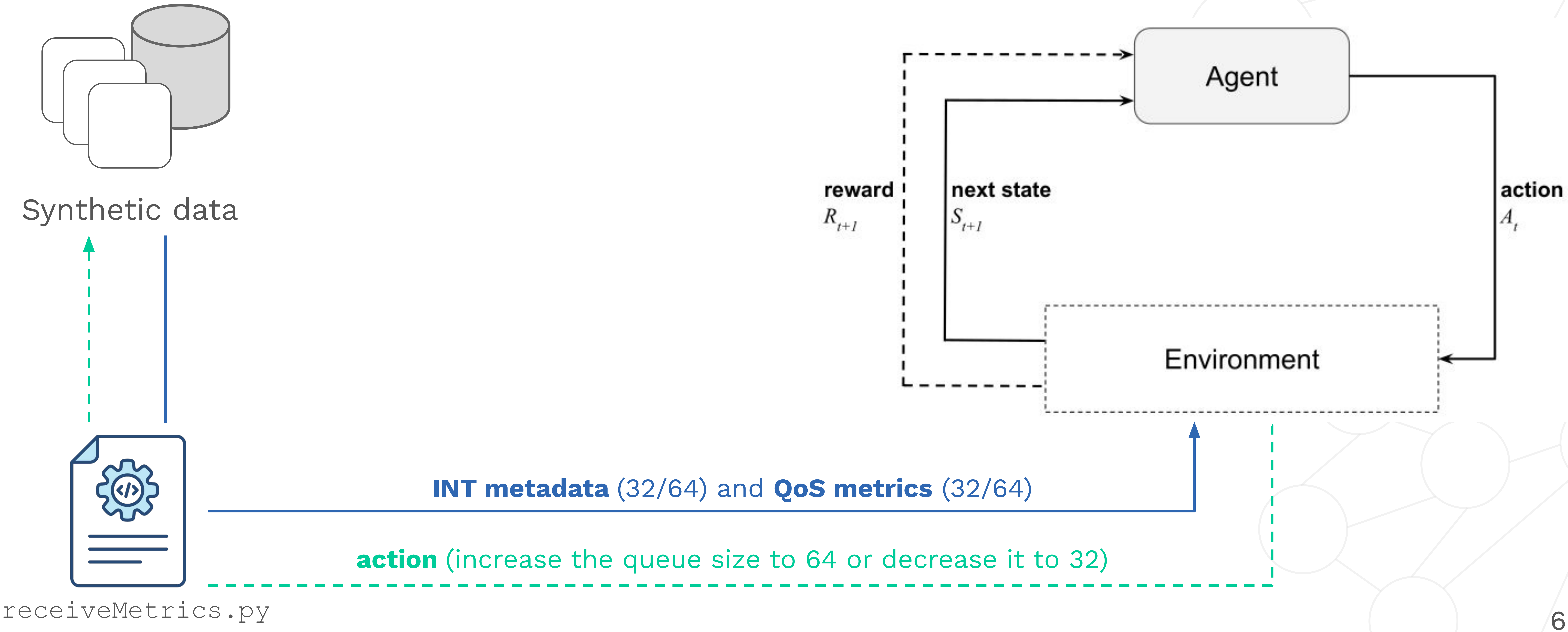
Join the data obtained in the previous step into a global dictionary

Send data to the RL Environment and store the transition in an experience replay buffer

Learn from experience

Applying synthetic data to an RL agent

Simulating the real network behavior



Applying synthetic data to an RL agent

receiveMetrics.py

Simulating the real network behavior

```
def readFile32():
    global sample32

    # Define the columns of interest
    cols = ['enq_qdepth1', 'deq_timedelta1', 'deq_qdepth1',
            ' enq_qdepth2', ' deq_timedelta2', ' deq_qdepth2',
            'enq_qdepth3', 'deq_timedelta3', 'deq_qdepth3',
            'FPS', 'Buffer', 'CalcBitrate', 'ReportedBitrate']

    # Read the CSV file in chunks of 4 seconds
    for sample32 in pd.read_csv('synthetic_data/best_modelsum_32.csv', chunksize=4):
        # Select only the specified columns
        sample32 = sample32[cols]
        # Process the data using the 'jointoRL' function with TYPE_32
        jointoRL(sample32, TYPE_32)


# Start reading files in separate threads
thread64 = threading.Thread(target=readFile64)
thread64.start()

thread32 = threading.Thread(target=readFile32)
thread32.start()
```

Applying synthetic data to an RL agent

receiveMetrics.py

Simulating the real network behavior

```python
def jointoRL(sample, t):
    global sampleJoin

    # Store the data sample in the global dictionary using the specified type 't'
    sampleJoin[t] = sample

    # If two data samples have been collected (INT and DASH metrics related to the 32 and 64 queue sizes),
    # call the 'sendtoRl' function
    if len(sampleJoin) == 2:
        sendtoRl(sampleJoin)
```

Applying synthetic data to an RL agent

receiveMetrics.py

Simulating the real network behavior

```
def sendtoRl(sample):
    global ddqn
    global env
    global experiment_id

    # Verify whether the agent has already taken actions
    if len(env.actions_history) == 0:
        # Determine the type of data sample and retrieve the DataFrame from the global dictionary accordingly
        if list(sample).index(0) == TYPE_32:
            df_INT = sample[TYPE_32].iloc[:,:9]
            df_dash = sample[TYPE_32].iloc[:,9:12]
        else:
            df_INT = sample[TYPE_64].iloc[:,:9]
            df_dash = sample[TYPE_64].iloc[:,9:12]
    # If the agent has already taken actions, verify which action was taken
    else:
        if env.actions_history[-1] == 0:
            df_INT = sample[TYPE_64].iloc[:,:9]
            df_dash = sample[TYPE_64].iloc[:,9:12]
        else:
```

Applying synthetic data to an RL agent

receiveMetrics.py

Simulating the real network behavior

```
        df_INT = sample[TYPE_32].iloc[:,:9]
        df_dash = sample[TYPE_32].iloc[:,9:12]

# Convert data to numpy arrays
current_state = df_INT.to_numpy()
dash_state = df_dash.to_numpy()

# Get state dimensionality
state_dim = df_INT.shape[1]

# Choose an action using epsilon-greedy policy
action = ddqn.epsilon_greedy_policy(current_state[FOURTH_SECOND].reshape(-1, state_dim))

# Take the chosen action and observe the next state, reward, and done flag
current_state, next_state, reward, done, _ = env.take_action(action, current_state, dash_state)

# If next state is available, memorize the transition and perform experience replay
if next_state is not None:
    print("next state received, memorizing transition")
```

SMARTNESS

Applying synthetic data to an RL agent

receiveMetrics.py

Simulating the real network behavior

```
        ddqn.memorize_transition(current_state[FOURTH_SECOND],
                                 env.actions_history[-2], # Action performed before reward calculation
                                 reward,
                                 next_state[FOURTH_SECOND],
                                 0.0 if done else 1.0)

        if ddqn.train:
            ddqn.experience_replay()

    print("\n========================================\n")
    print("last action: {0} | reward: {1} | fps: {2} | "
          "Buffer size: {3}".format(
           env.actions_history[-1], env.reward_history[-1],
           env.fps_history[-1], env.buffer_size[-1]))
    print("\n========================================\n")
```

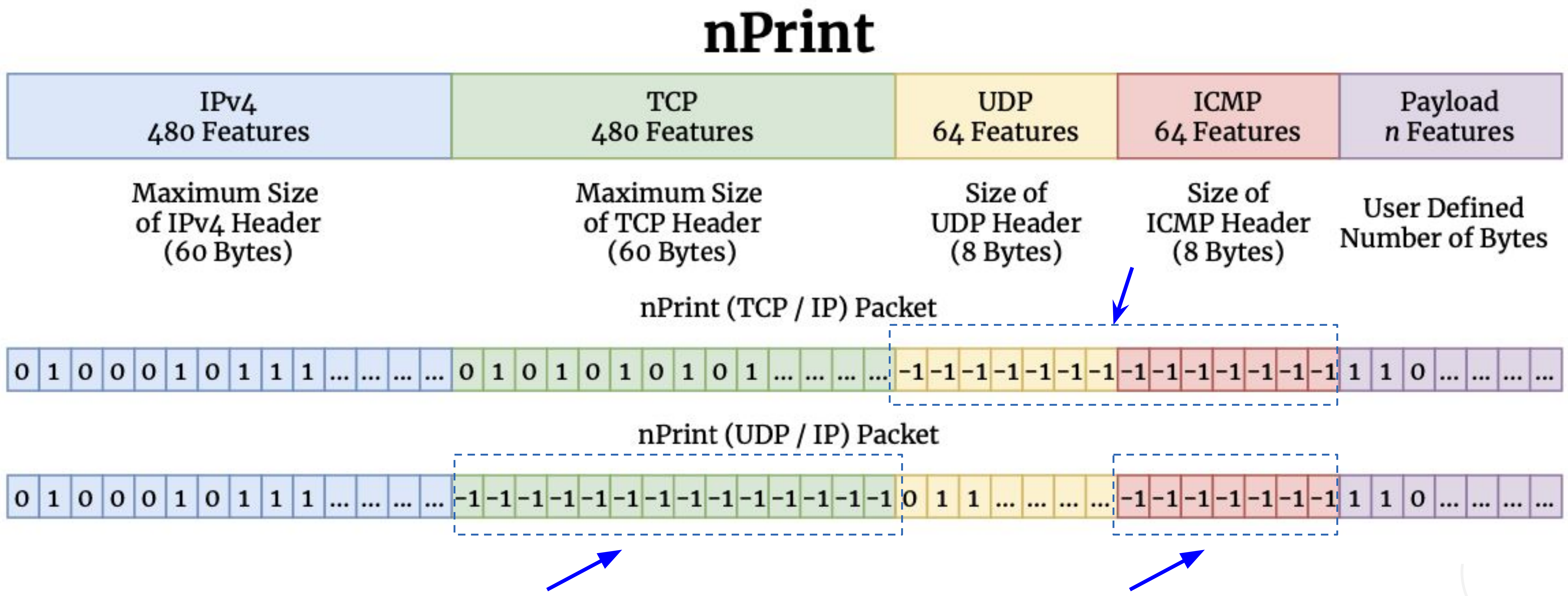
Generation of Synthetic Network Trace

NetDiffusion: Network Data Augmentation Through Protocol-Constrained Traffic Generation

Jiang, Xi and Liu, Shinan and Gember-Jacobson, Aaron and Bhagoji, Arjun Nitin and Schmitt, Paul and Bronzino, Francesco and Feamster, Nick

nPrint

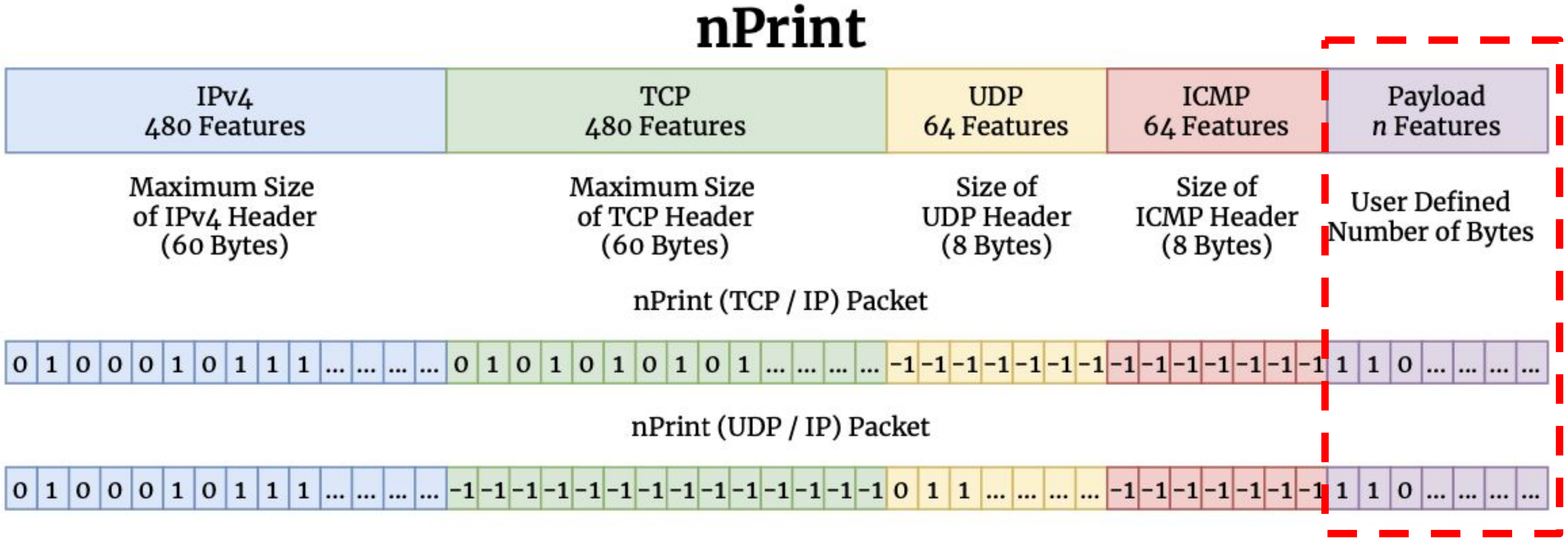
Generation of Synthetic Network Trace



nPrint

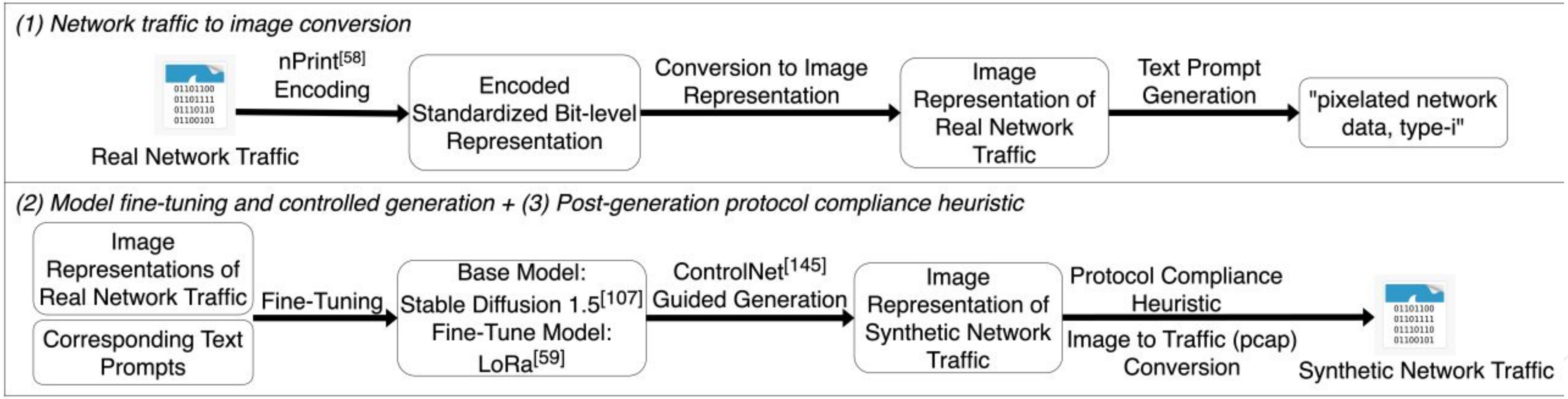
Generation of Synthetic Network Trace

We can be infer the payload length from header fields such as the IP "Total Length"



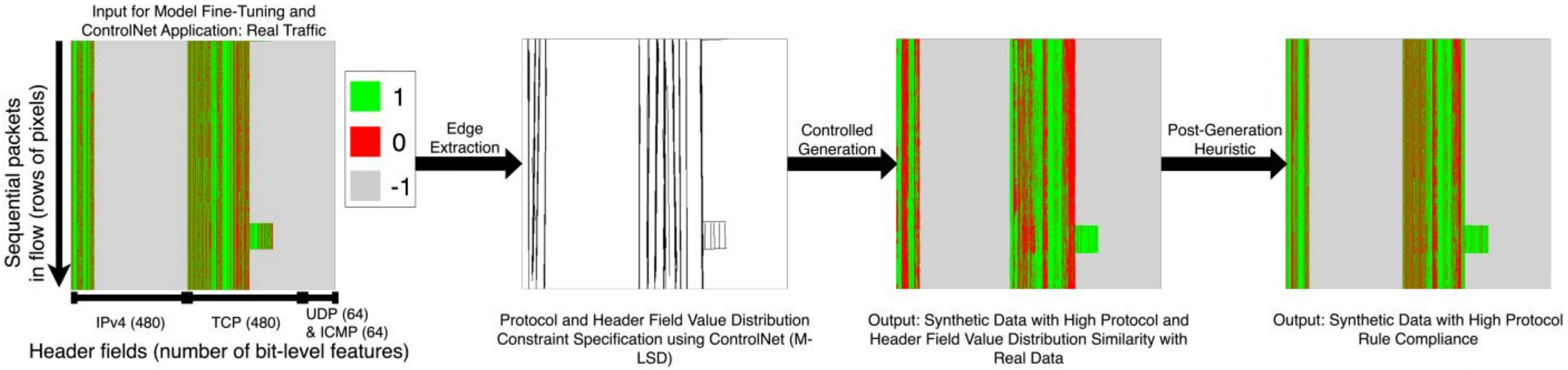
NetDiffusion: Workflow

Generation of Synthetic Network Trace



NetDiffusion: Workflow (Example)

Generation of Synthetic Network Trace



Synthetic Amazon network traffic outputs: (1) Using ControlNet, it detect regions present in the original traffic and ensure protocol and header field value distribution conformance by generating within specified regions. (2) Applying a post-generation heuristic to refine field details for protocol conformance.

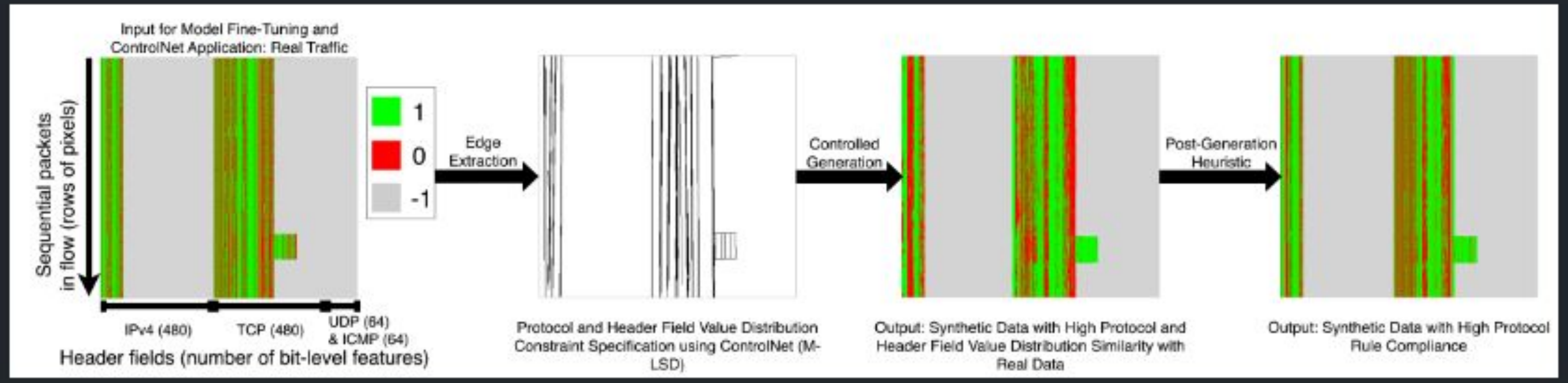
Hands on

Generation of Synthetic Network Trace

https://github.com/arielgoes/NetDiffusion_Generator.git

README License

NetDiffusion: High-Fidelity Synthetic Network Traffic Generation



NetDiffusion tutorial

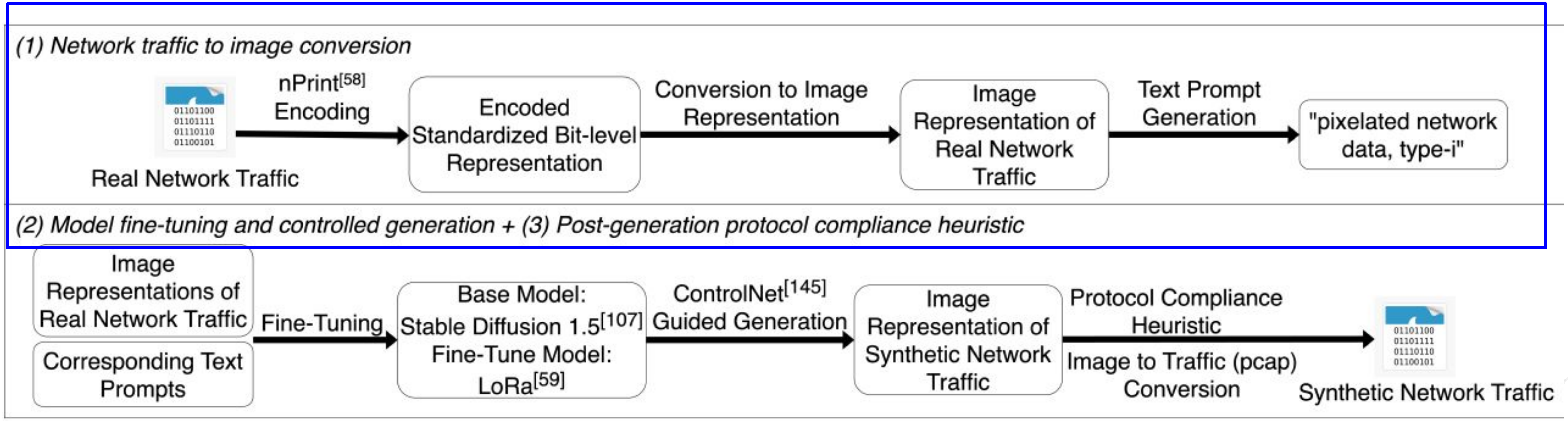
This tutorial is based on the original guide steps for generating PCAPs with NetDiffusion.

SMARTNESS

NetDiffusion: Workflow

Generation of Synthetic Network Trace

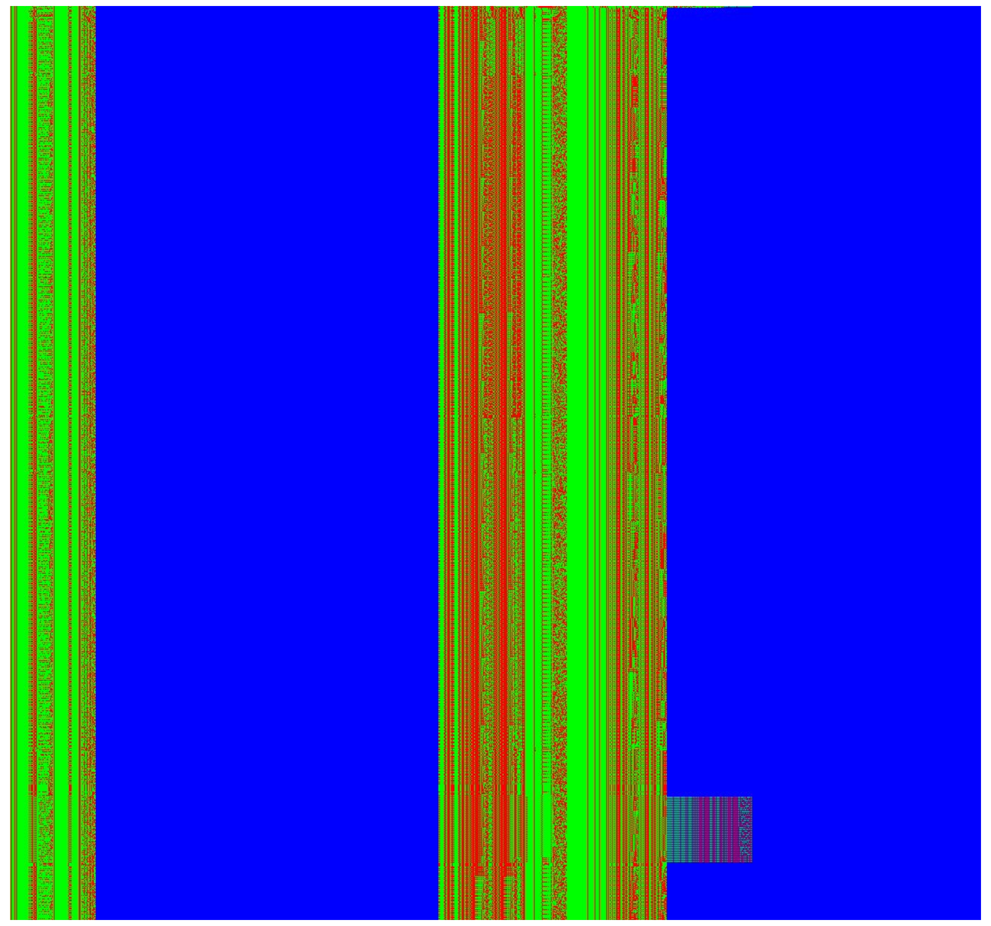
Steps 1-4



NetDiffusion: Main Steps

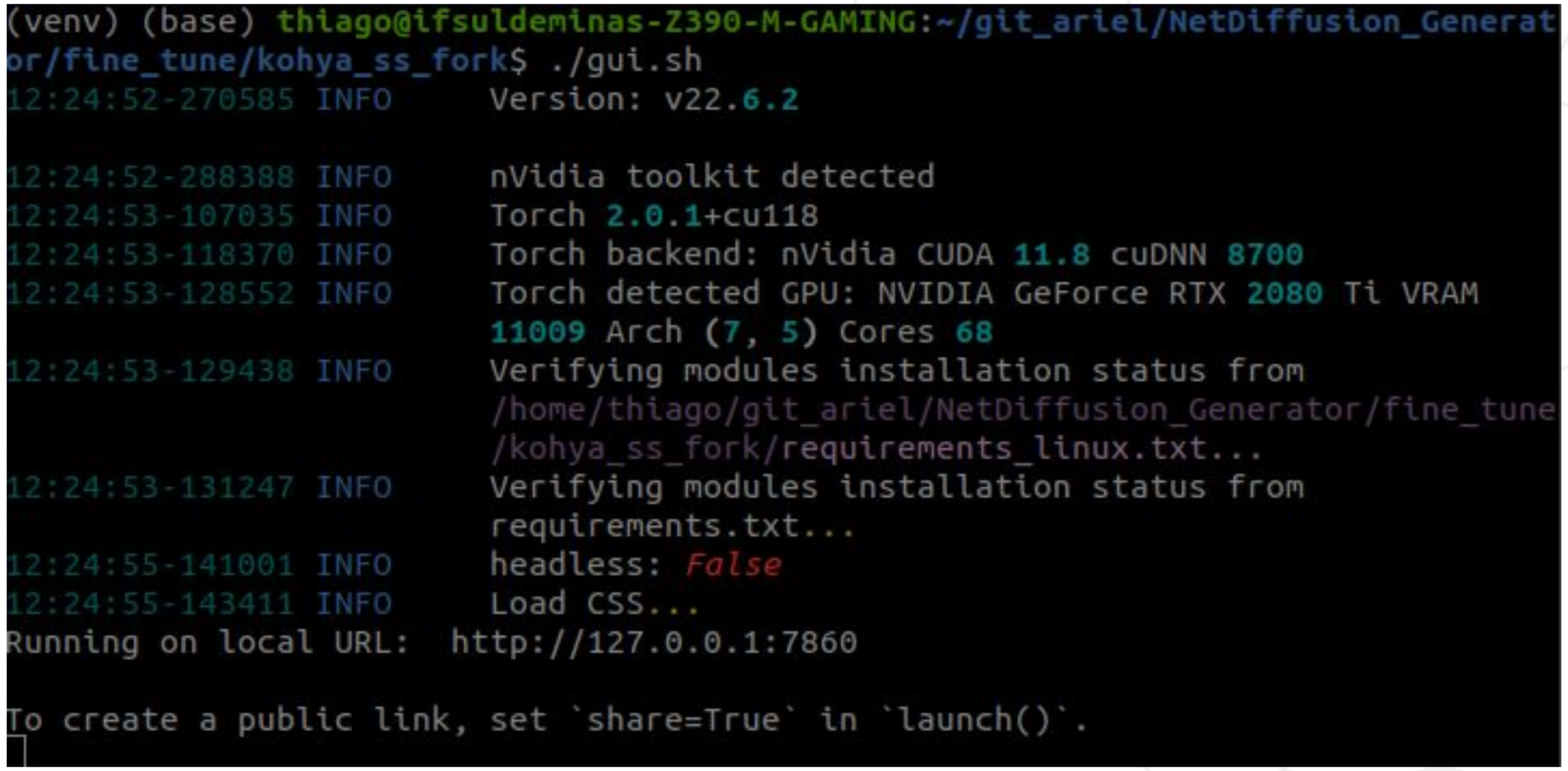
Generation of Synthetic Network Trace

1. Input converted nPrint to image (1088x1024):



blue = -1 green = 1 red = 0
caption: “*pixelated network data, type-0*”

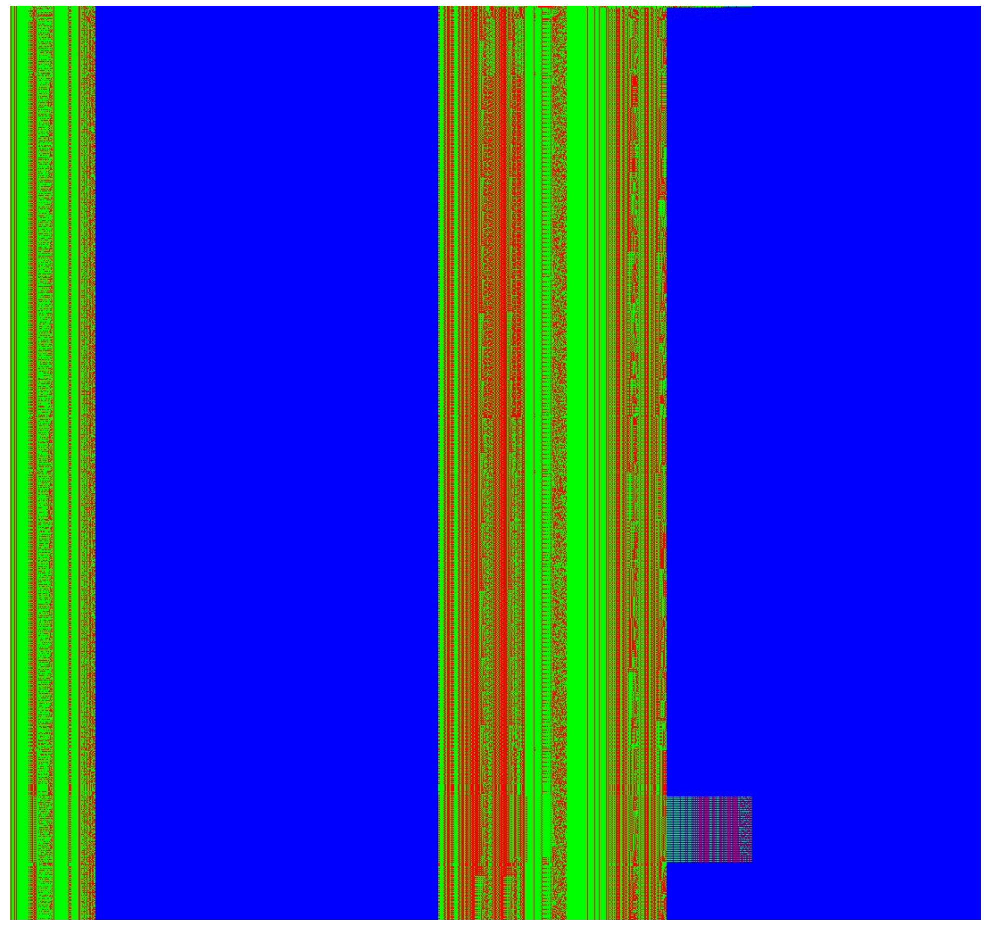
2. Accessing the GUI



NetDiffusion: Main Steps

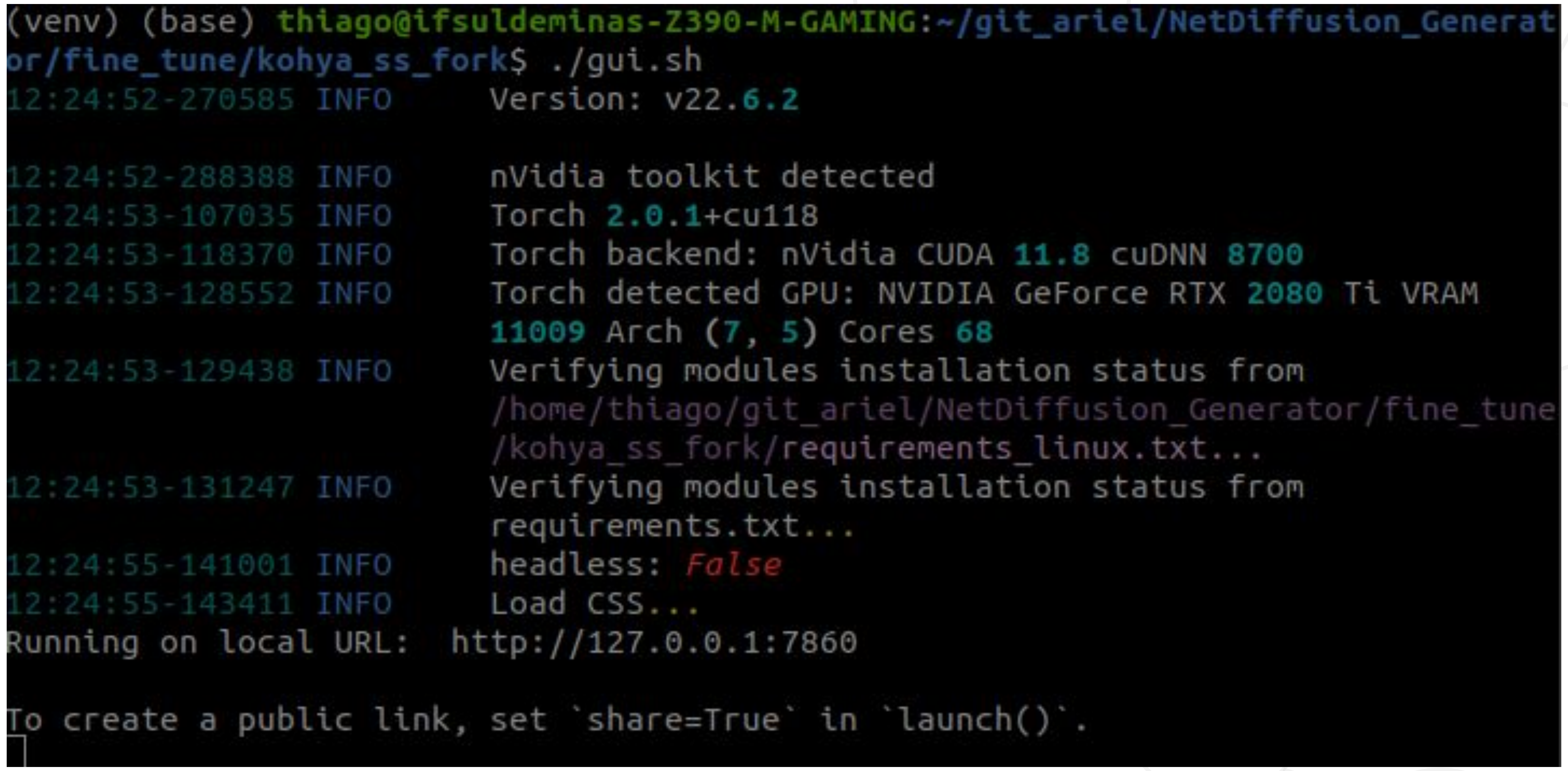
Generation of Synthetic Network Trace

1. Input converted nprint to image (1088x1024):



blue = -1 green = 1 red = 0
caption: *“pixelated network data, type-0”*

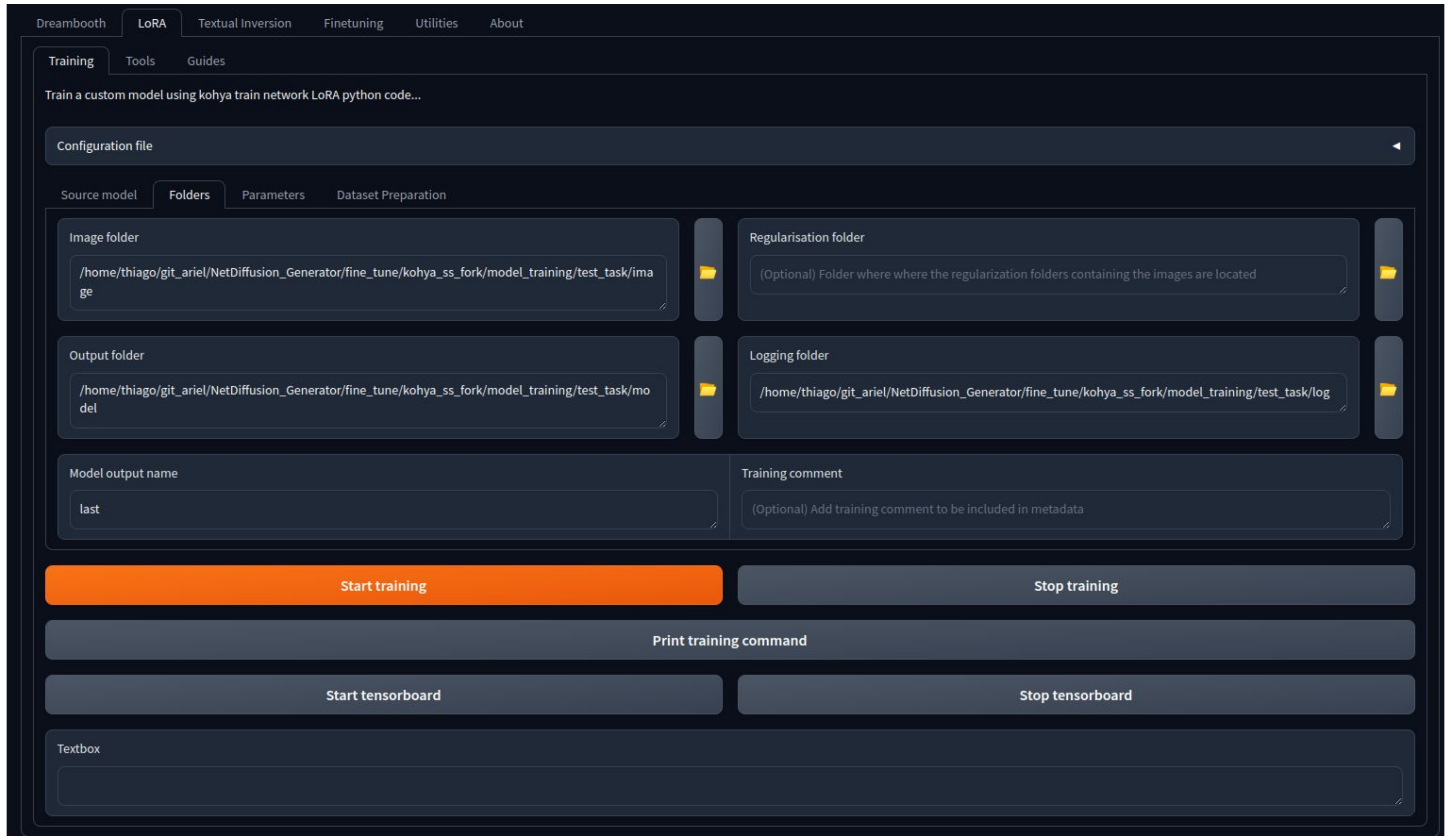
2. Accessing the GUI



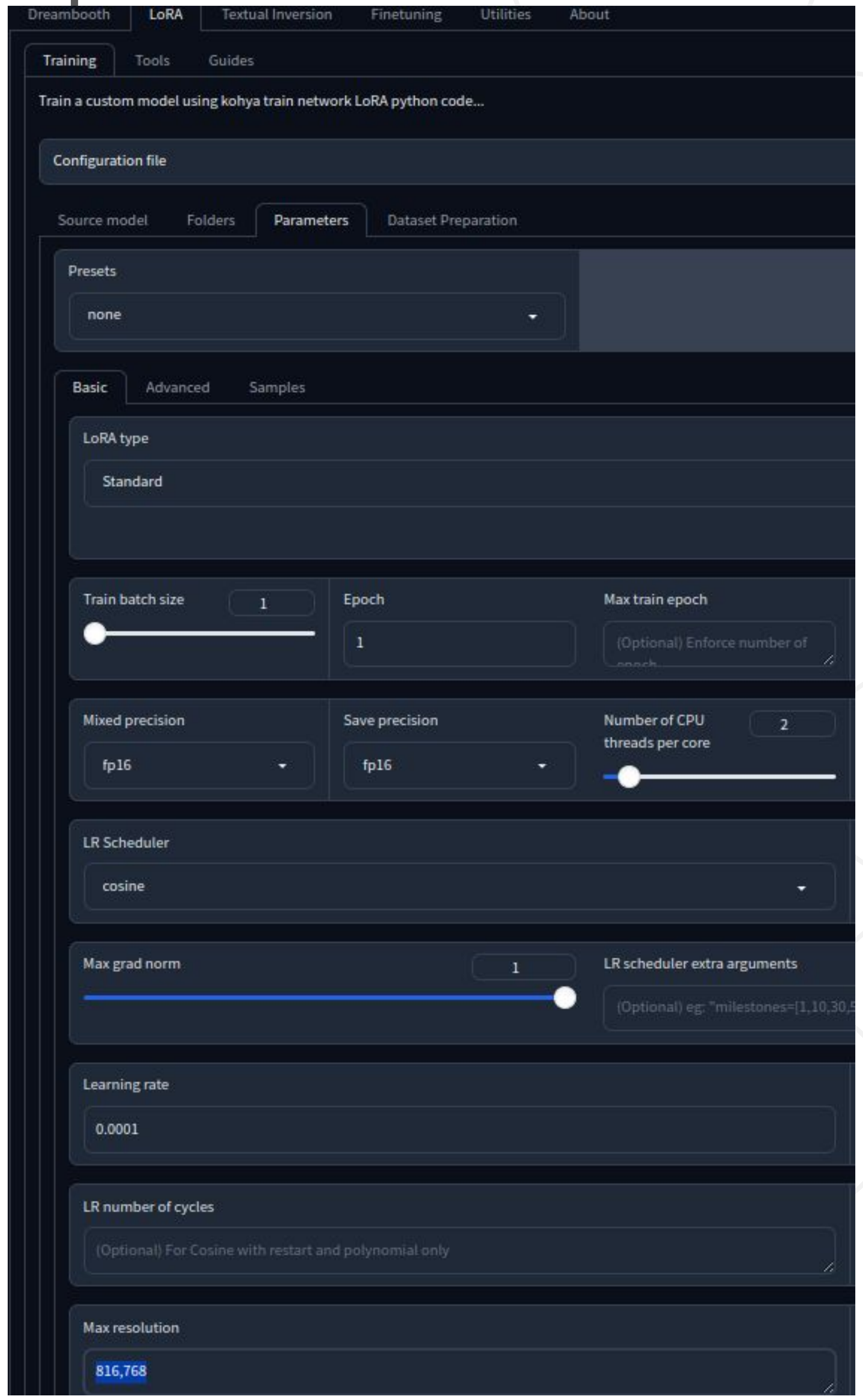
NetDiffusion: Main Steps

Generation of Synthetic Network Trace

3. Fine-tuning image/model/log paths



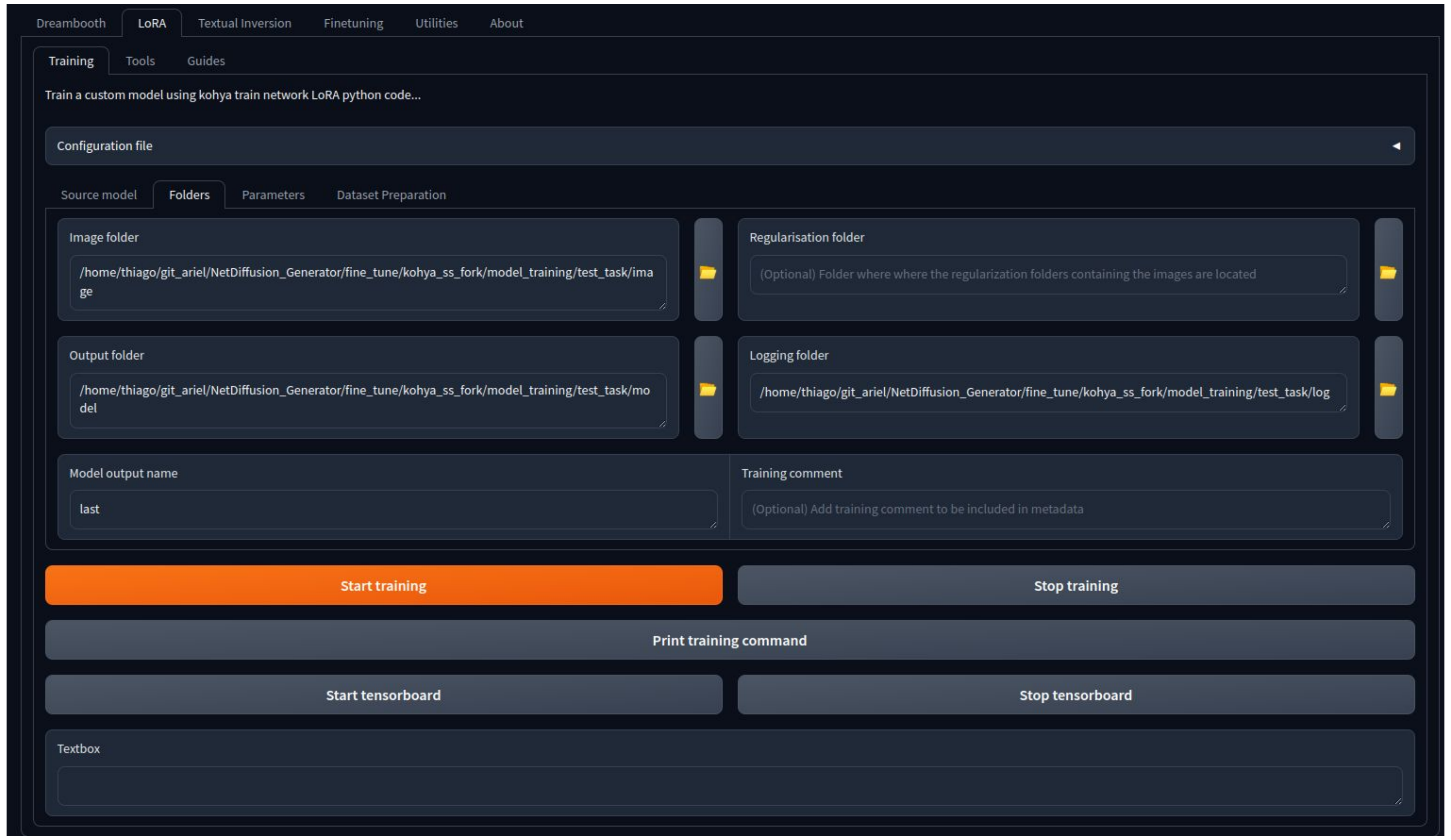
4. Set max resolution to 816x768 and enabling caption



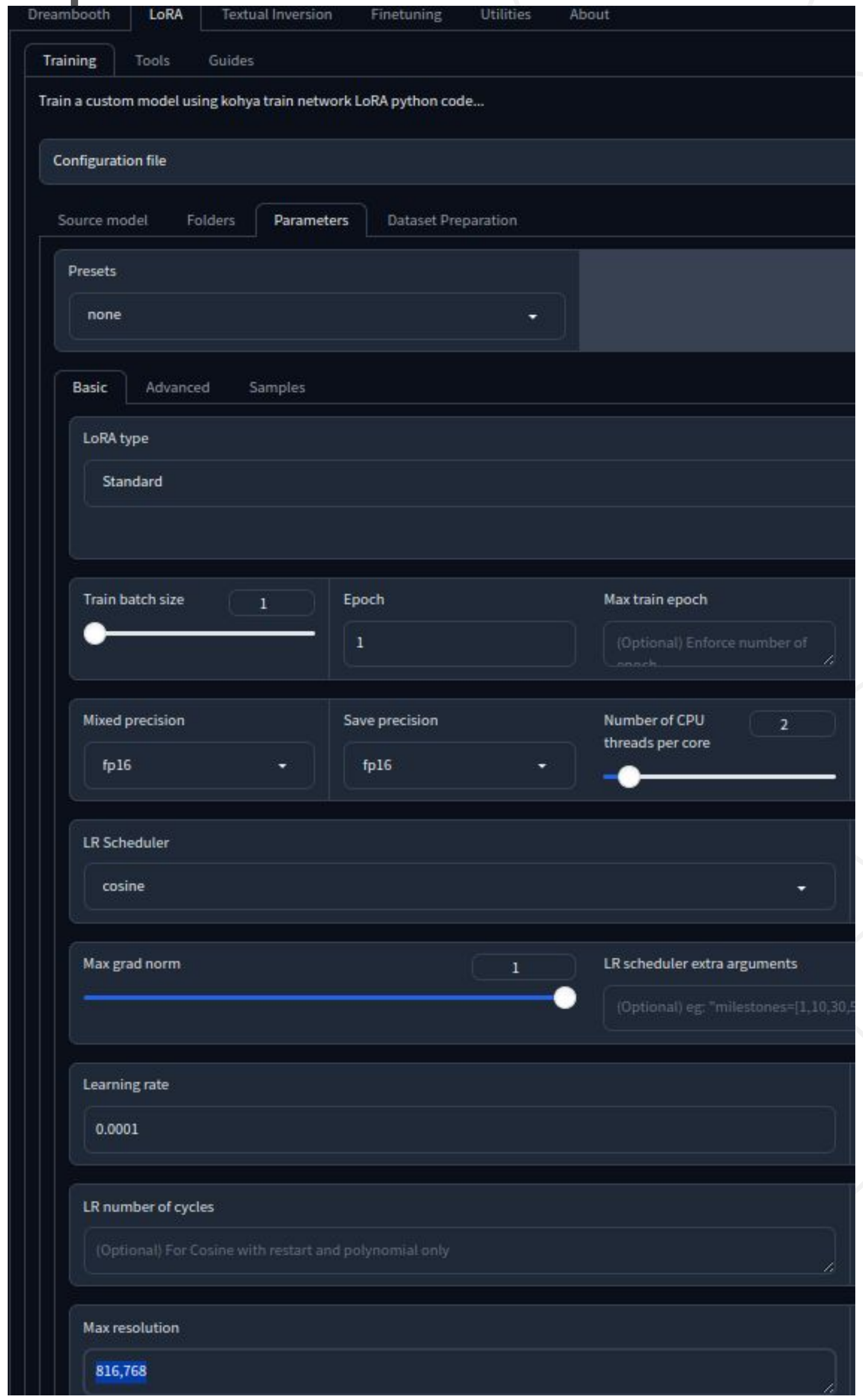
NetDiffusion: Main Steps

Generation of Synthetic Network Trace

3. Fine-tuning image/model/log paths

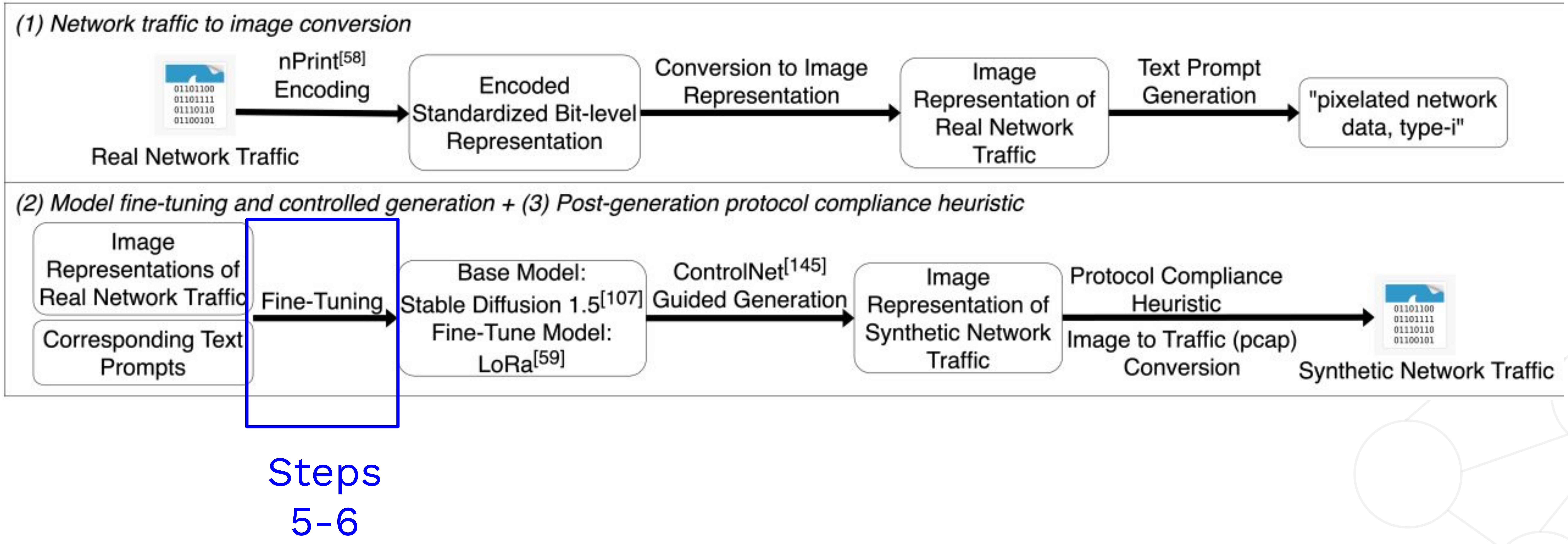


4. Set max resolution to 816x768 and enabling caption



NetDiffusion: Workflow

Generation of Synthetic Network Trace



NetDiffusion: Main Steps

Generation of Synthetic Network Trace

5. Training the fine-tuned image

```
[Dataset 0]
caching latents.
checking cache validity...
100%|██████████████████████████████████████████████| 4/4 [00:00<00:00, 75573.05it/s]
caching latents...
100%|██████████████████████████████████████████████| 4/4 [00:00<00:00,  6.81it/s]
create LoRA network. base dim (rank): 8, alpha: 1.0
neuron dropout: p=None, rank dropout: p=None, module dropout: p=None
create LoRA for Text Encoder:
create LoRA for Text Encoder: 72 modules.
create LoRA for U-Net: 192 modules.
enable LoRA for text encoder
enable LoRA for U-Net
prepare optimizer, data loader etc.
use 8-bit AdamW optimizer | {}
running training / 学習開始
  num train images * repeats / 学習画像の数×繰り返し回数: 80
  num reg images / 正則化画像の数: 0
  num batches per epoch / 1epochのバッチ数: 80
  num epochs / epoch数: 1
  batch size per device / バッチサイズ: 1
  gradient accumulation steps / 勾配を合計するステップ数 = 1
  total optimization steps / 学習ステップ数: 80
steps:   0%|                                          | 0/80 [00:00<?, ?it/s]
epoch 1/1
steps:  15%|██████                    | 12/80 [00:04<00:27,  2.50it/s, avr_loss=0.0283]
```

NetDiffusion: Main Steps

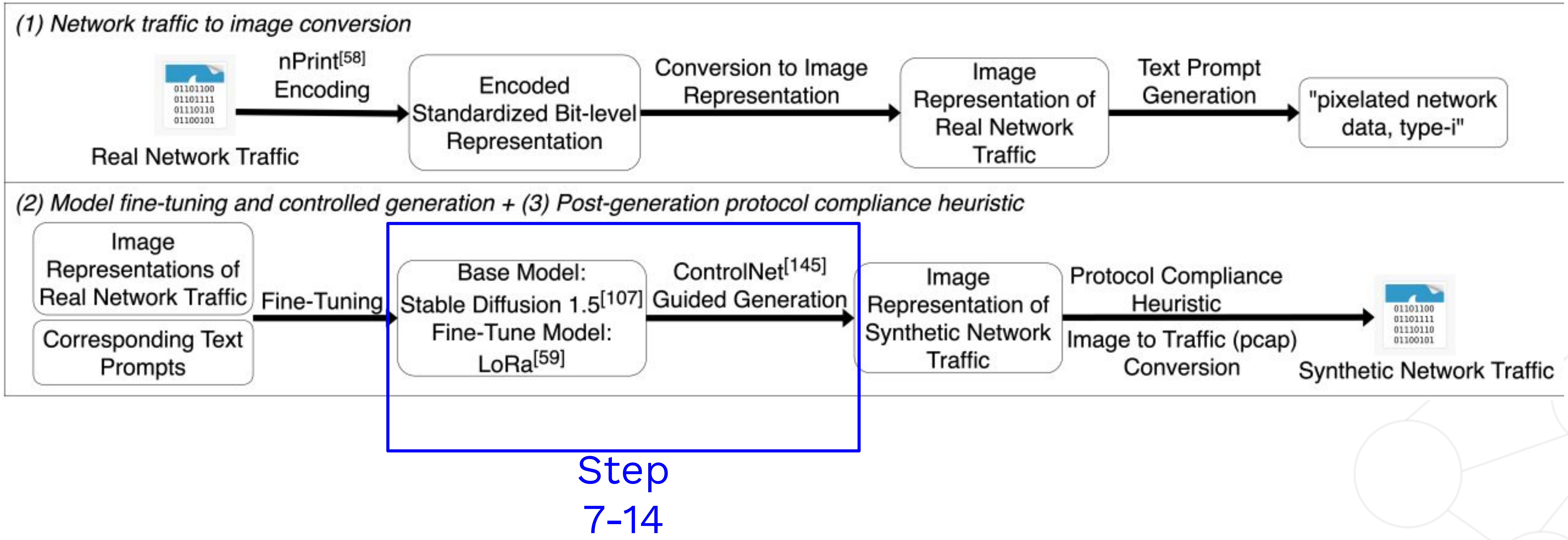
Generation of Synthetic Network Trace

6. Showing the image caption

```
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/fine_tune/kohya_ss_fork/model_training/test_task/image/20_network$ ls
discord_01.png  discord_01.txt  netflix_01_1.png  netflix_01_1.txt  netflix_01_2.png  netflix_01_2.txt  netflix_01.png  netflix_01.txt
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/fine_tune/kohya_ss_fork/model_training/test_task/image/20_network$ cat netflix_01.txt
pixelated network data, type-0
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/fine_tune/kohya_ss_fork/model_training/test_task/image/20_network$
```

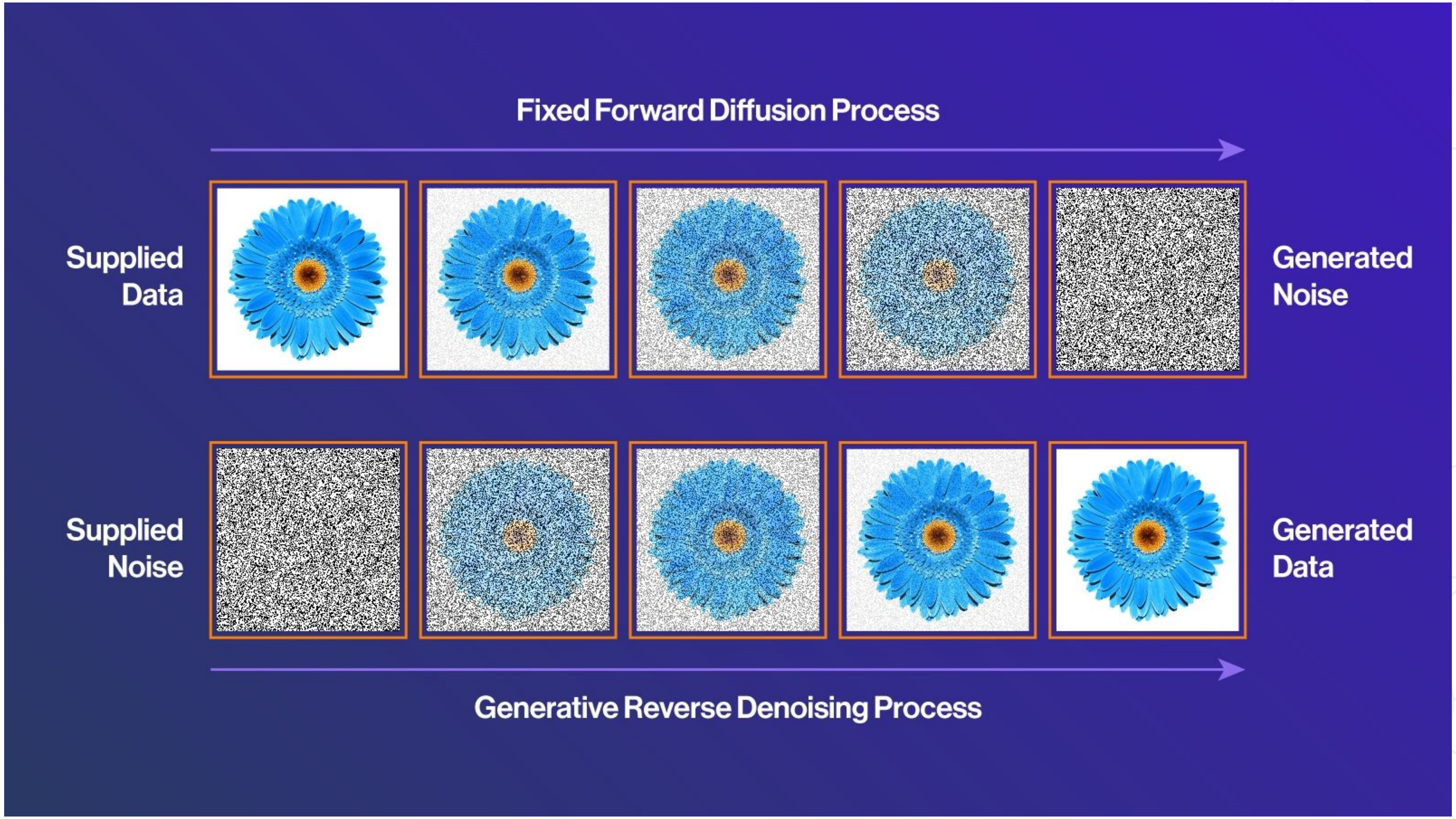
NetDiffusion: Workflow

Generation of Synthetic Network Trace



What is a diffusion model (in Generative AIs)?

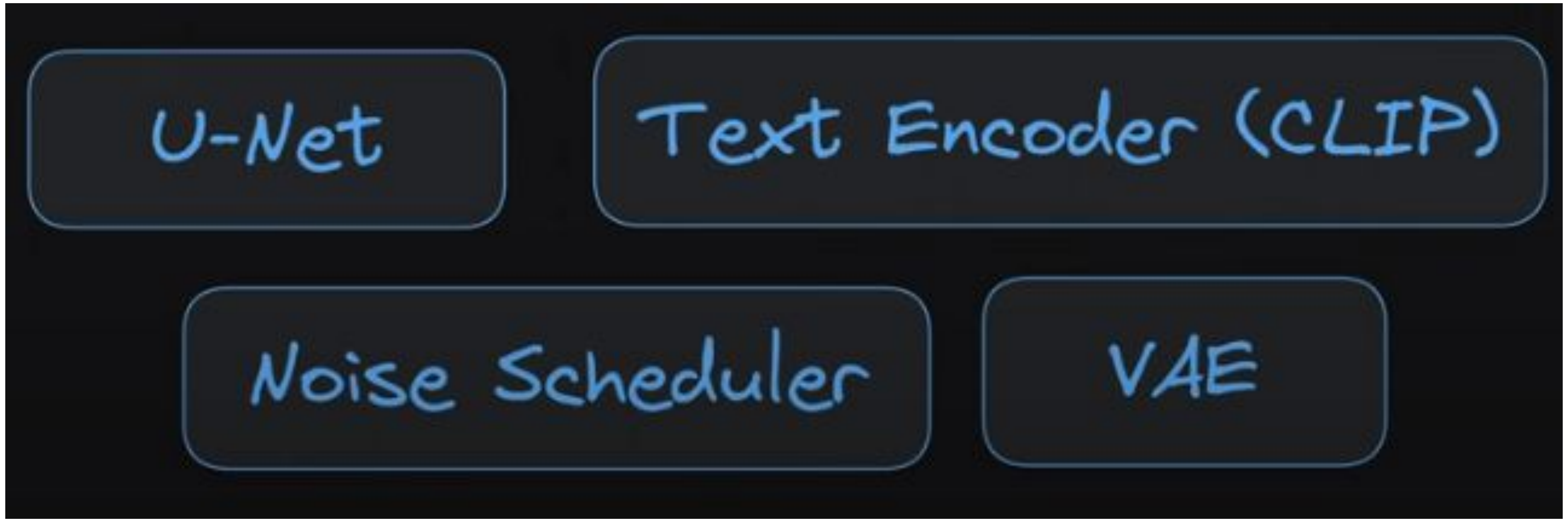
Generation of Synthetic Network Trace



What is Stable Diffusion?

Generation of Synthetic Network Trace

Main components:

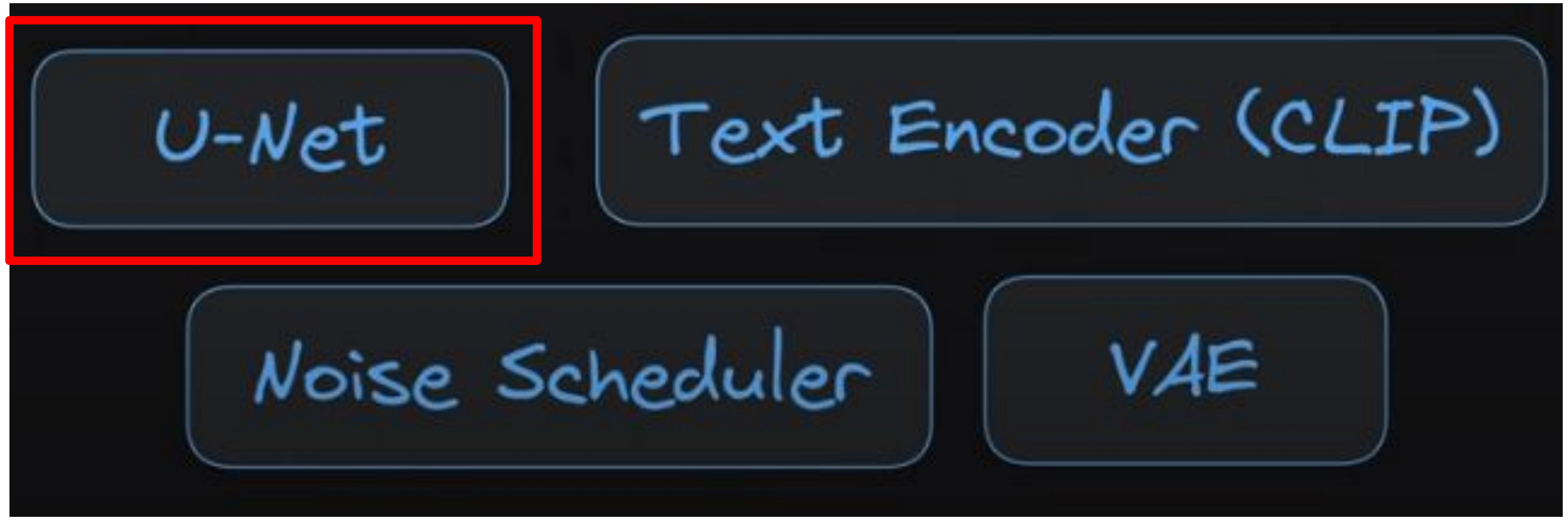


source: https://www.youtube.com/watch?v=_JZPKbEp6gk

What is Stable Diffusion?

Generation of Synthetic Network Trace

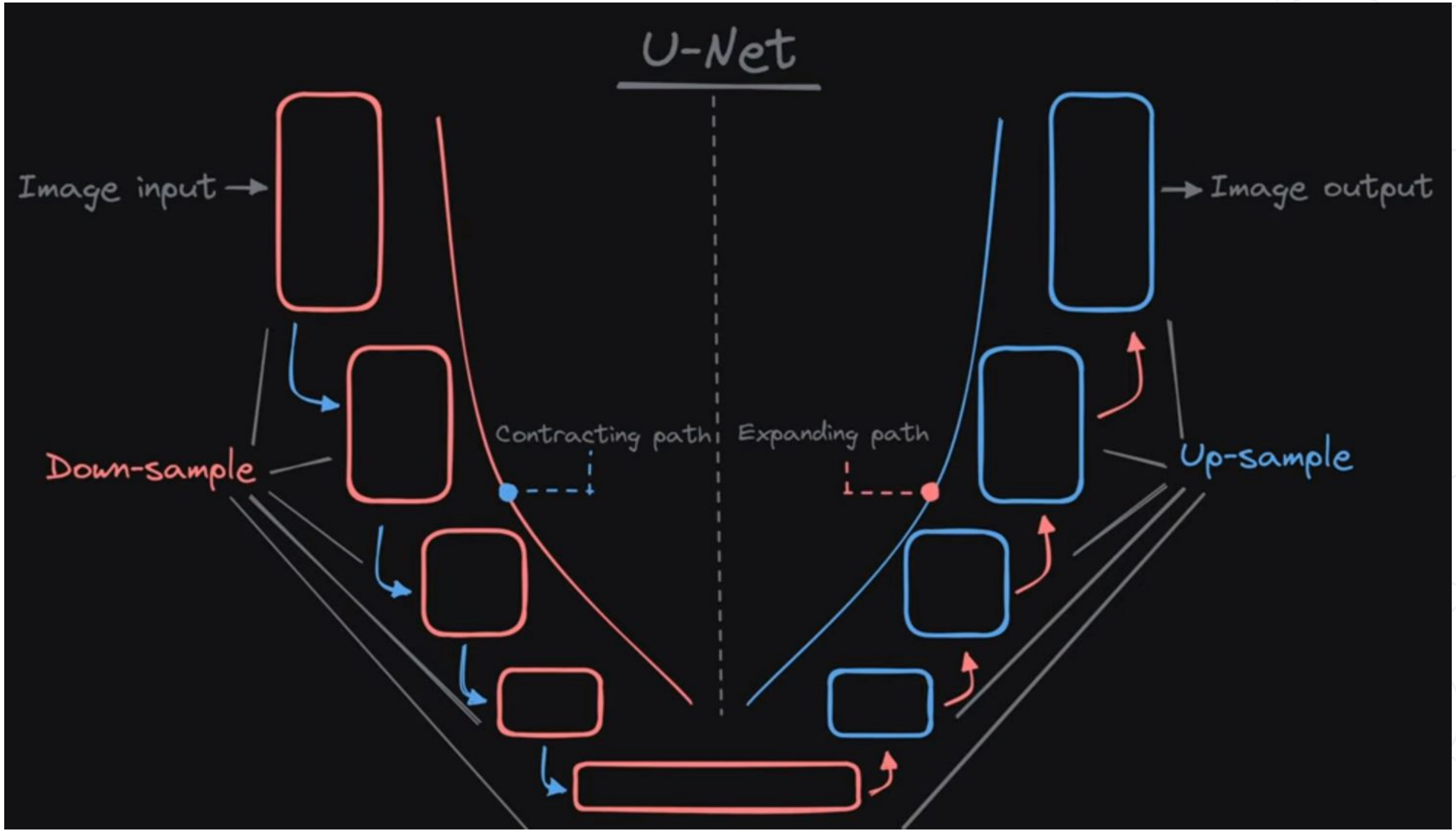
Main components:



source: https://www.youtube.com/watch?v=_JZPKbEp6gk

Stable Diffusion: U-Net

Generation of Synthetic Network Trace

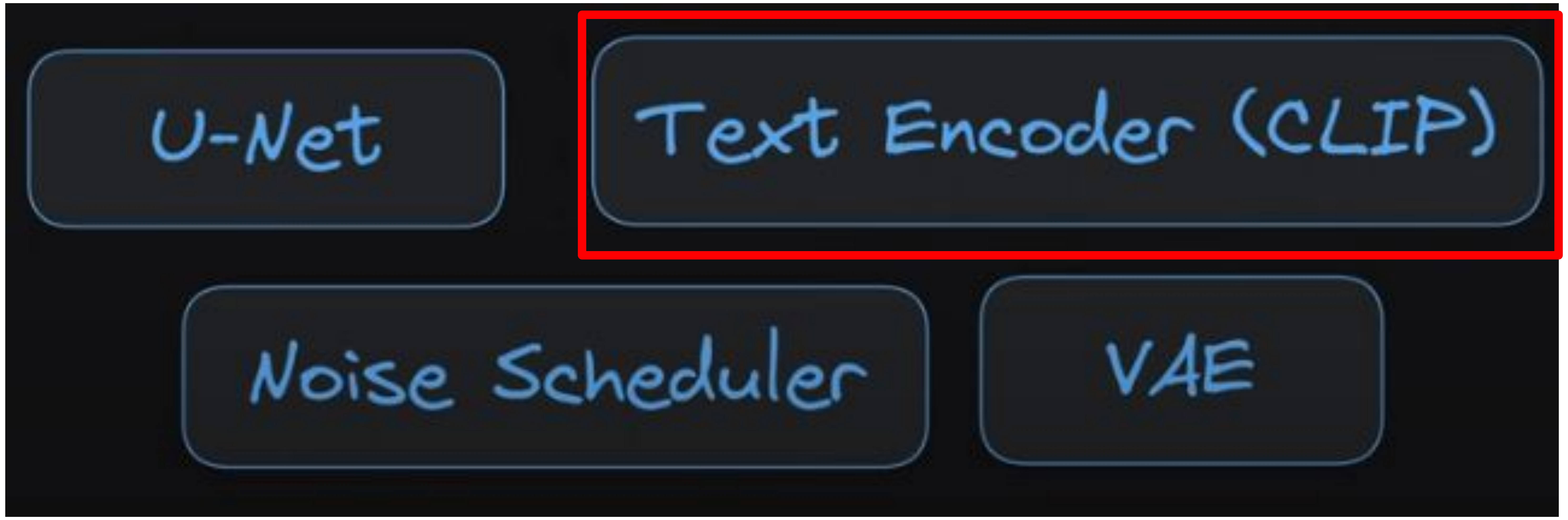


source: https://www.youtube.com/watch?v=_JZPKbEp6gk

Stable Diffusion: Text Encoder (CLIP)

Generation of Synthetic Network Trace

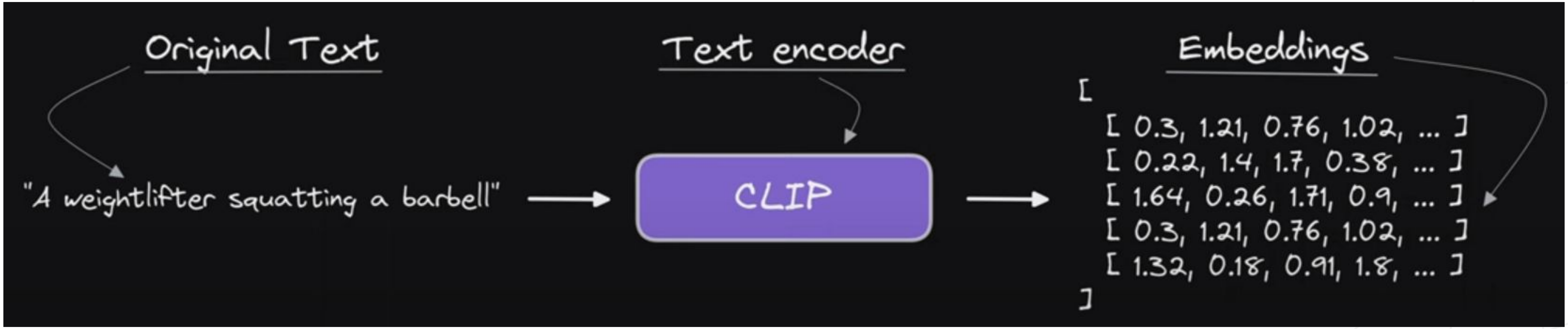
Main components:



source: https://www.youtube.com/watch?v=_JZPKbEp6gk

Stable Diffusion: Text Encoder (CLIP)

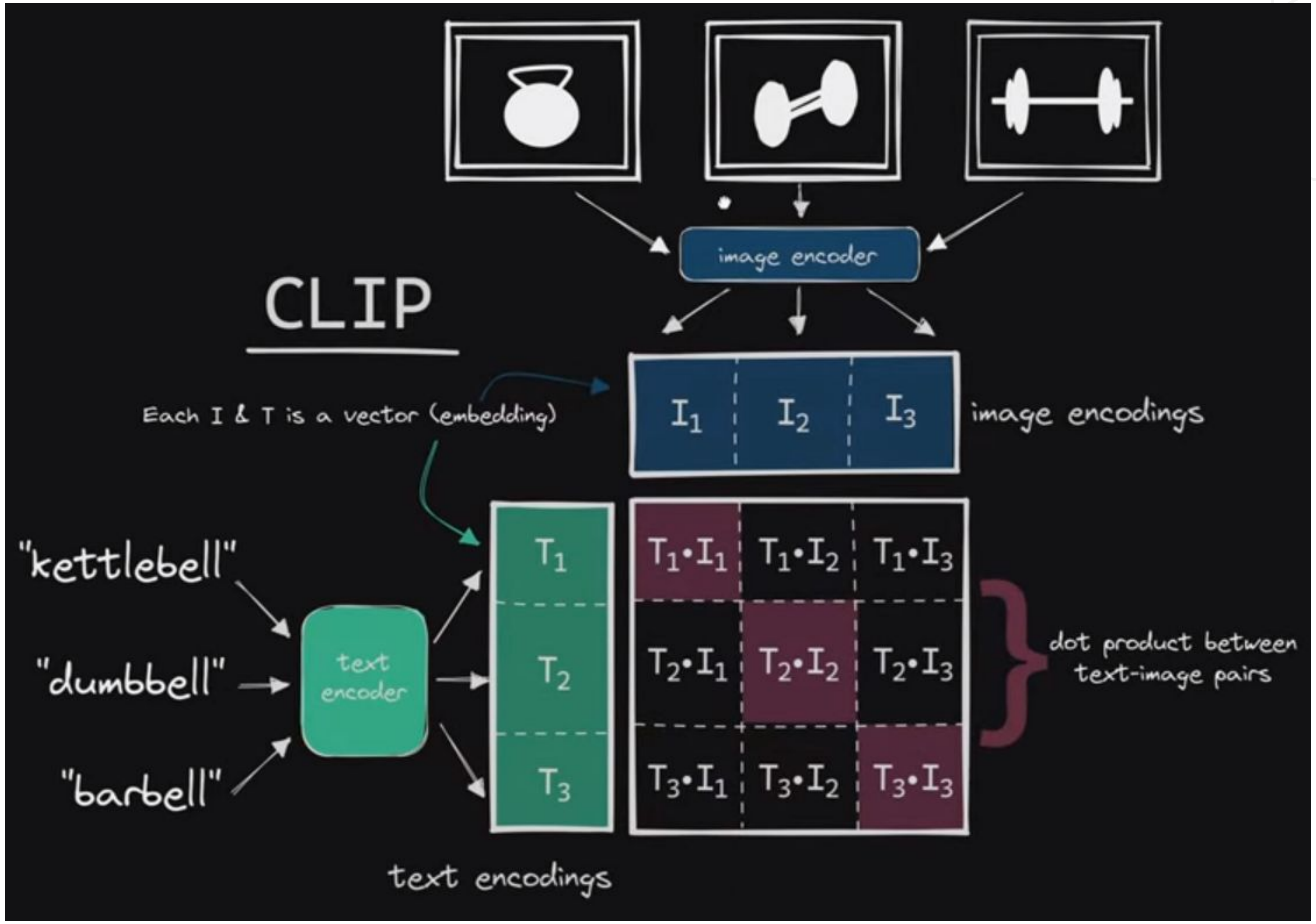
Generation of Synthetic Network Trace



source: https://www.youtube.com/watch?v=_JZPKbEp6gk

Stable Diffusion: Text Encoder (CLIP)

Generation of Synthetic Network Trace

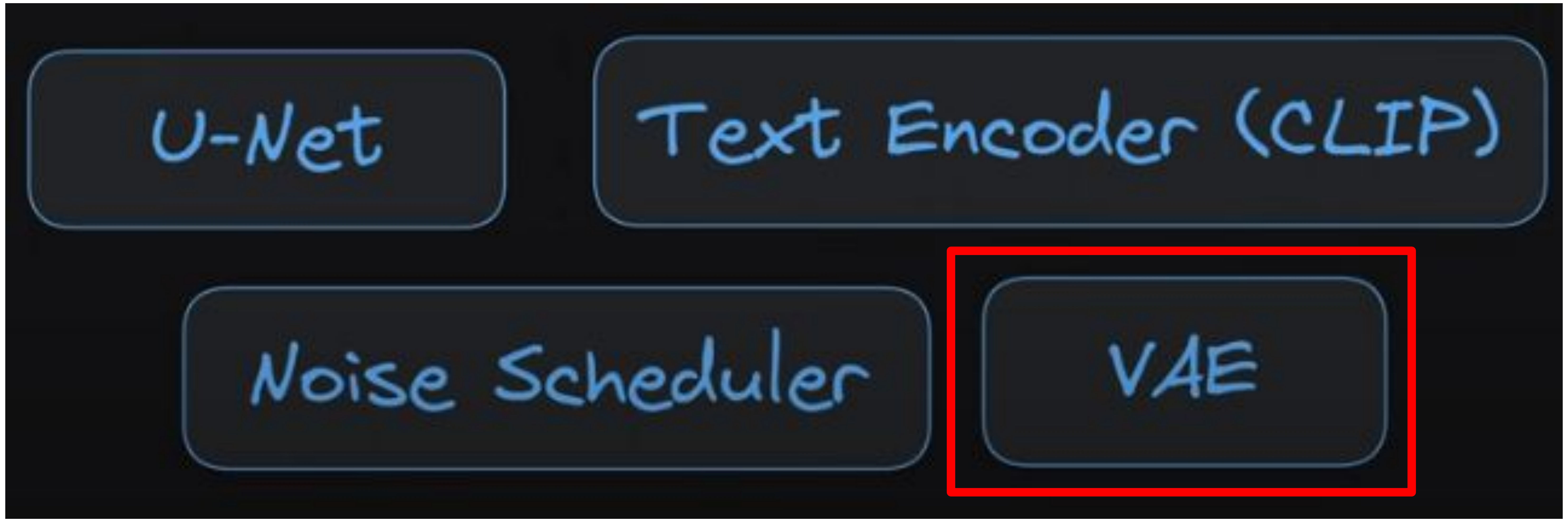


source: https://www.youtube.com/watch?v=_JZPKbEp6gk

Stable Diffusion: Variational Autoencoder (VAE)

Generation of Synthetic Network Trace

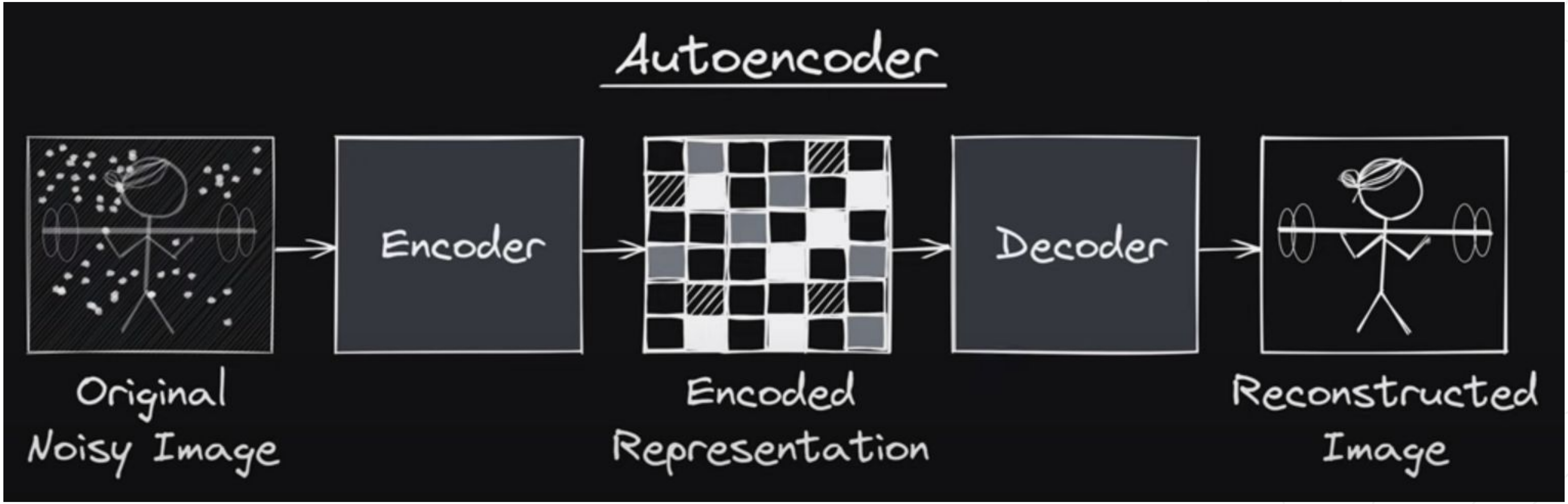
Main components:



source: https://www.youtube.com/watch?v=_JZPKbEp6gk

Stable Diffusion: Variational Autoencoder (VAE)

Generation of Synthetic Network Trace

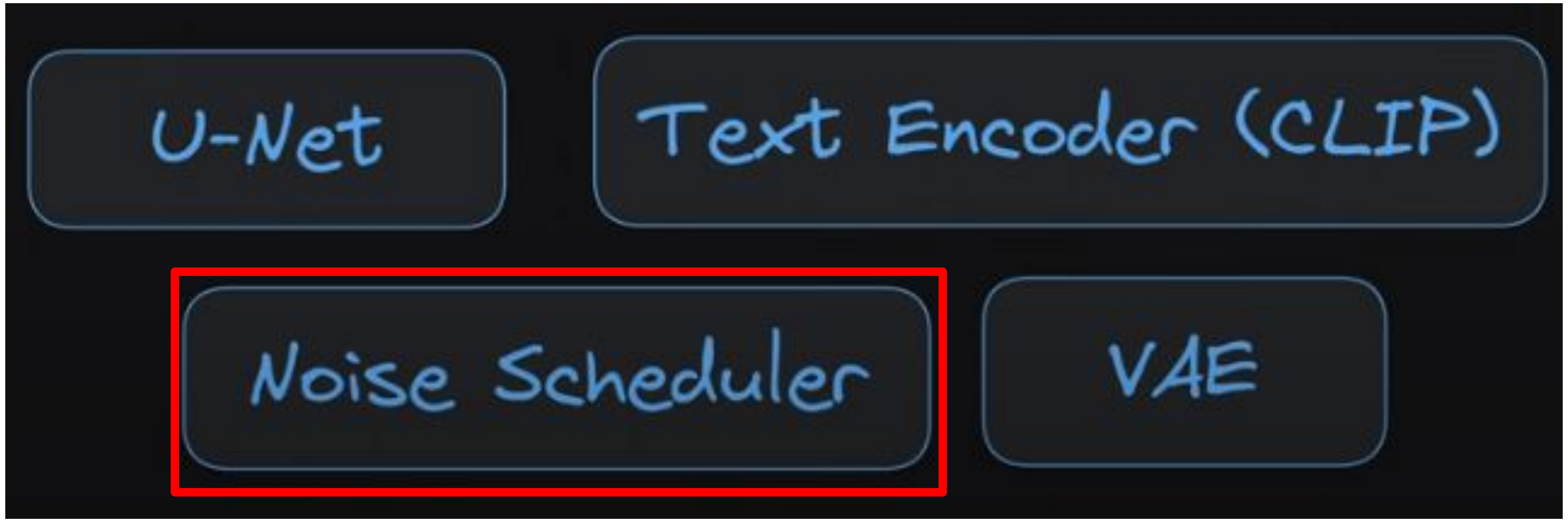


source: https://www.youtube.com/watch?v=_JZPKbEp6gk

Stable Diffusion: Noise Scheduler

Generation of Synthetic Network Trace

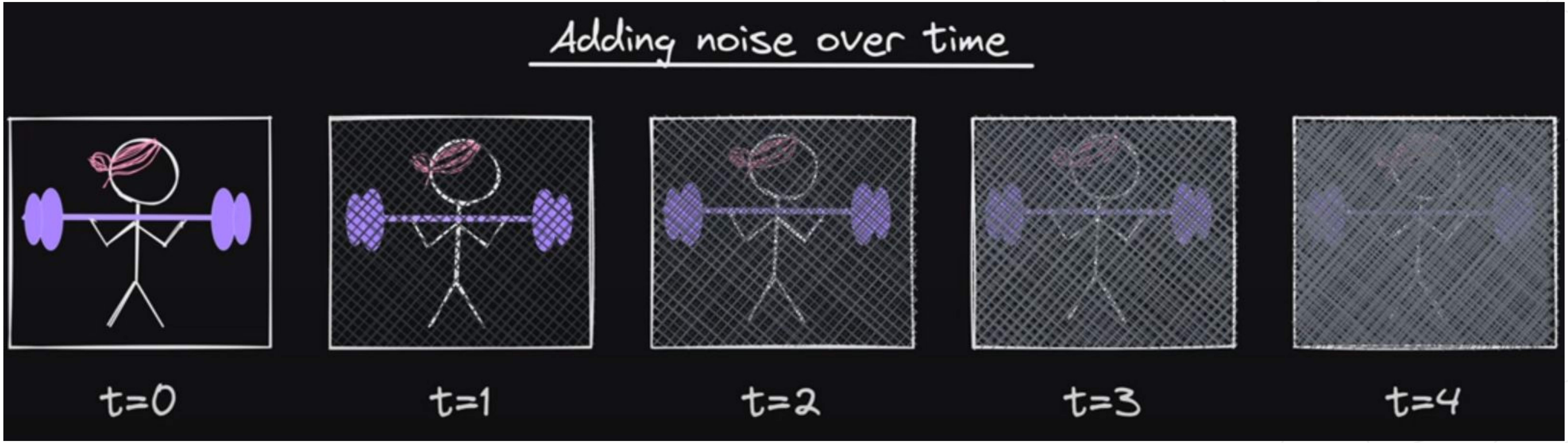
Main components:



source: https://www.youtube.com/watch?v=_JZPKbEp6gk

Stable Diffusion: Noise Scheduler

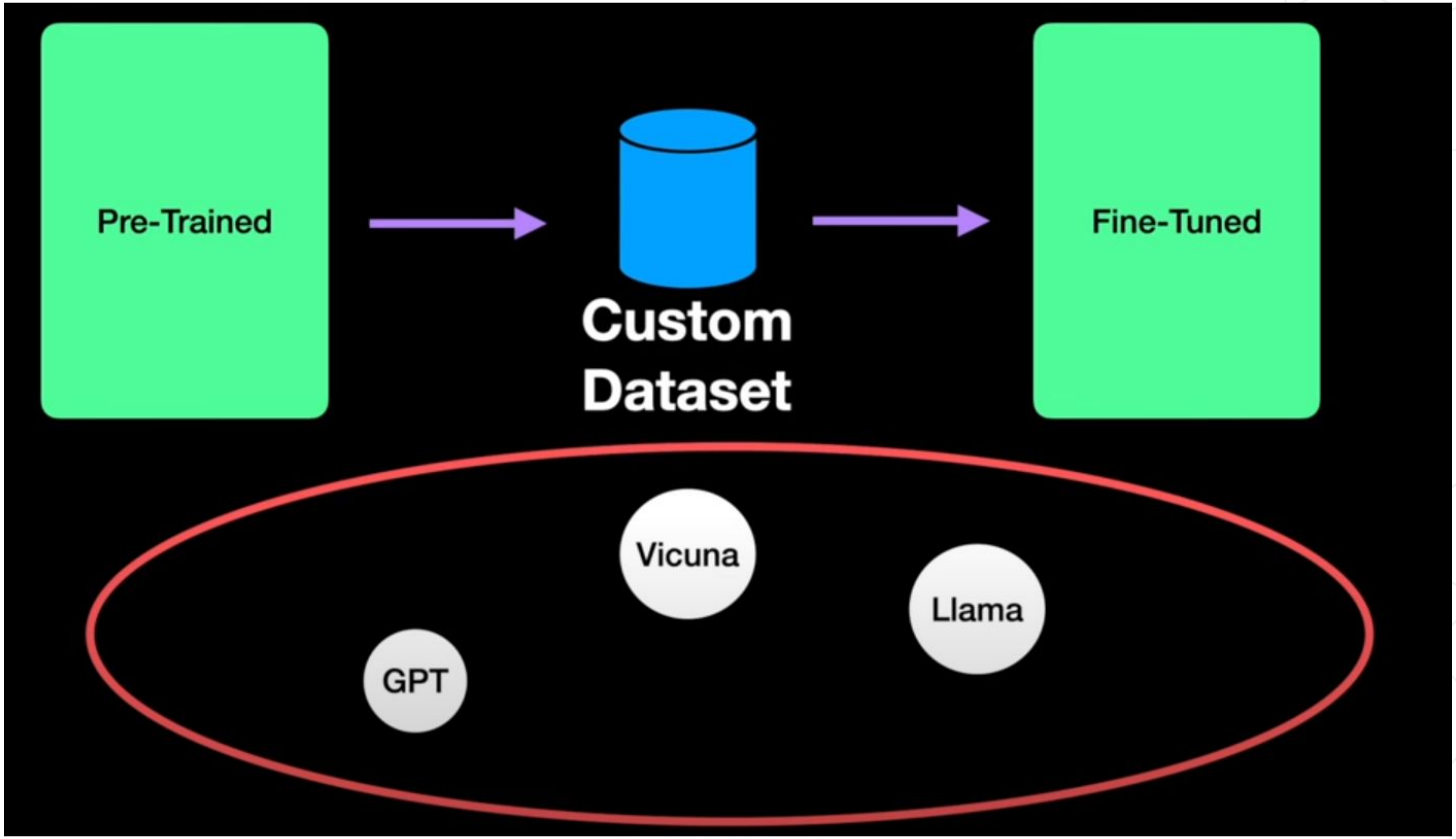
Generation of Synthetic Network Trace



source: https://www.youtube.com/watch?v=_JZPKbEp6gk

What is Low-Rank Adaptation (LoRa)?

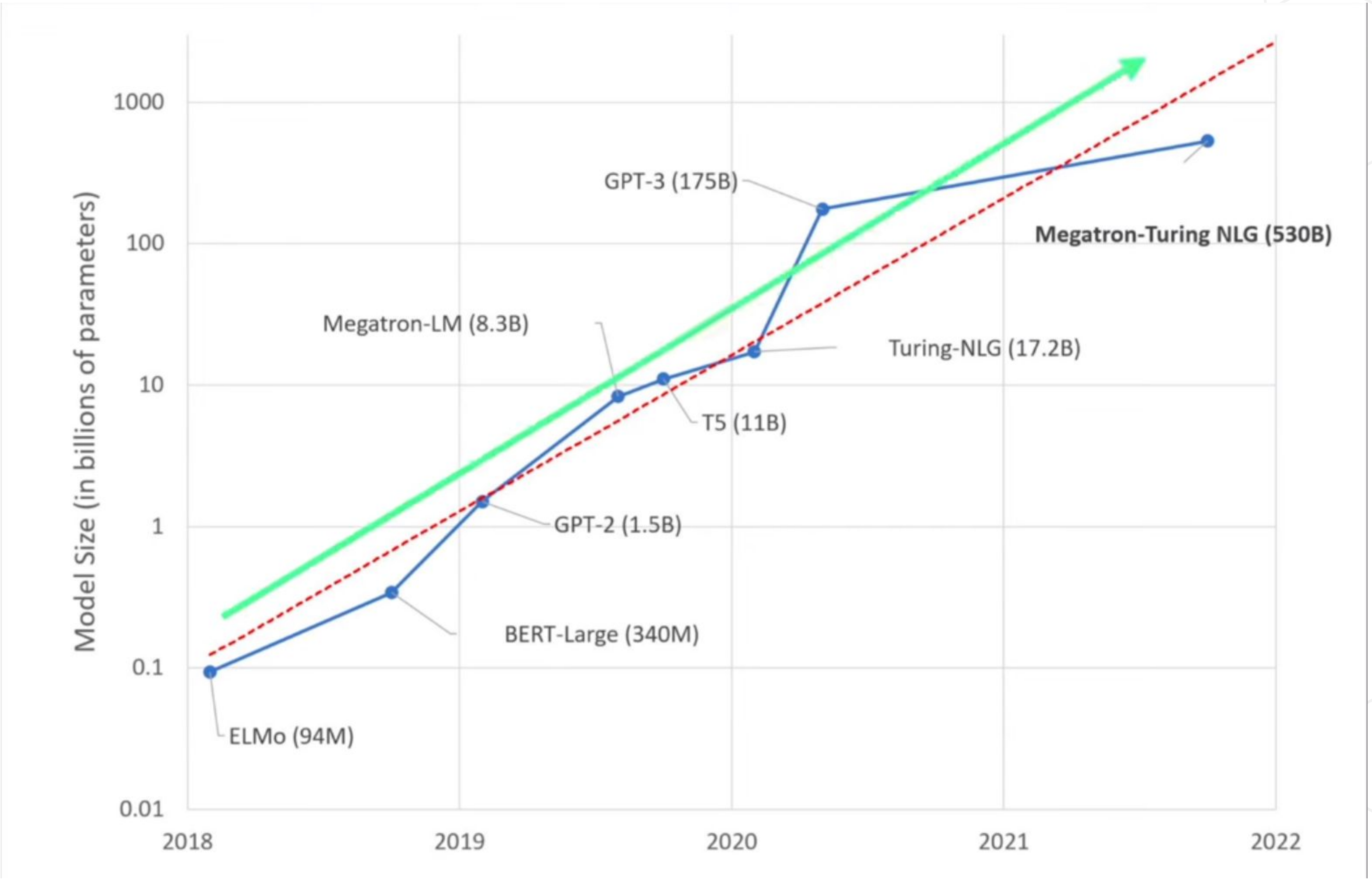
Generation of Synthetic Network Trace



source: https://www.youtube.com/watch?v=X4VvO3G6_vw

What is Low-Rank Adaptation (LoRa)?

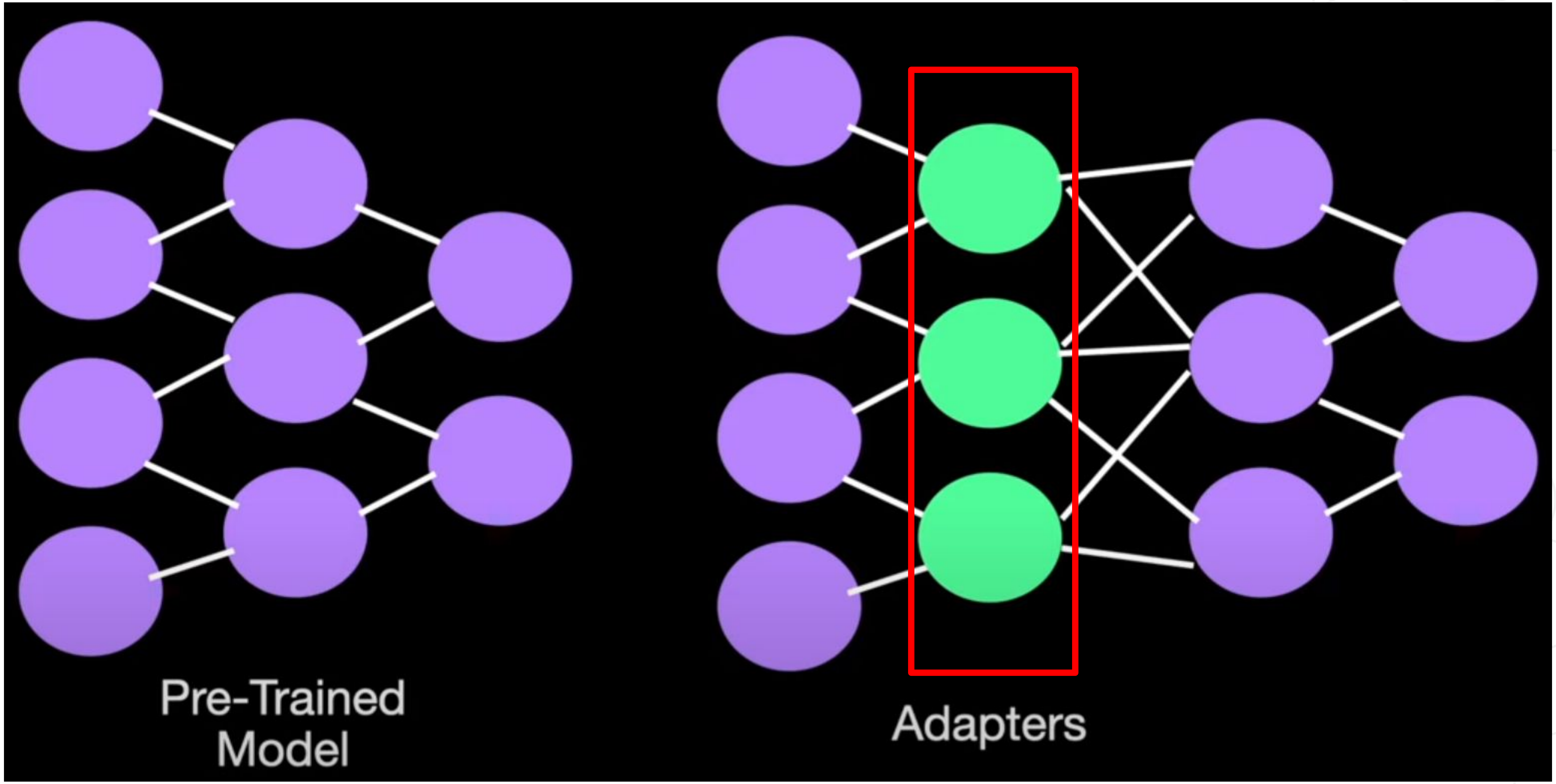
Generation of Synthetic Network Trace



source: https://huggingface.co/blog/large-language-models

LoRa is an adapter

Generation of Synthetic Network Trace



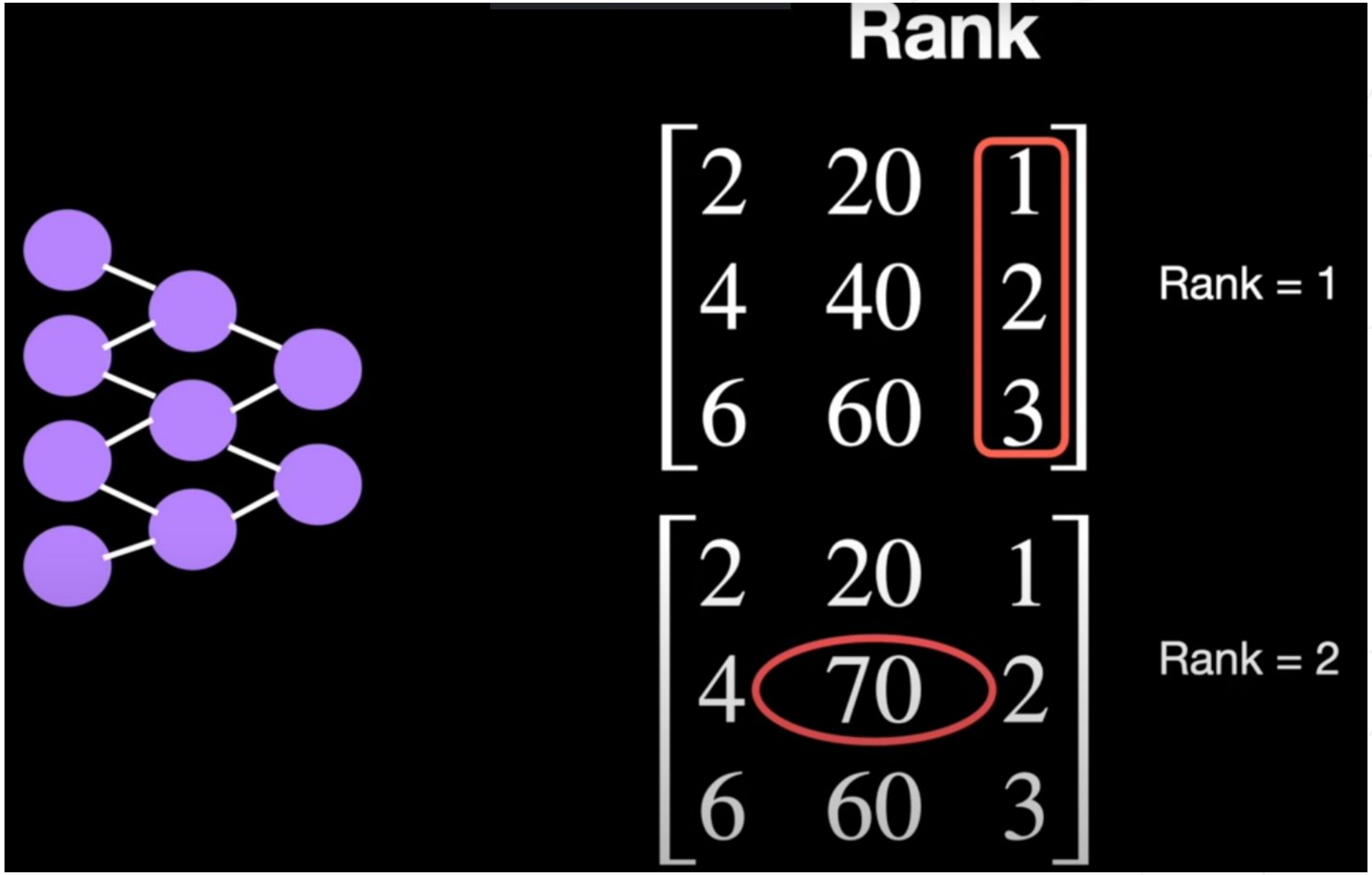
source: https://www.youtube.com/watch?v=X4VvO3G6_vw

LoRa leverages low-rank (of a matrix)

Generation of Synthetic Network Trace

For instance, an 3x3 eye-matrix has rank 3, since all columns are independent

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



LoRa leverages low-rank (of a matrix)

Generation of Synthetic Network Trace

SMARTNESS

Rank Decomposition

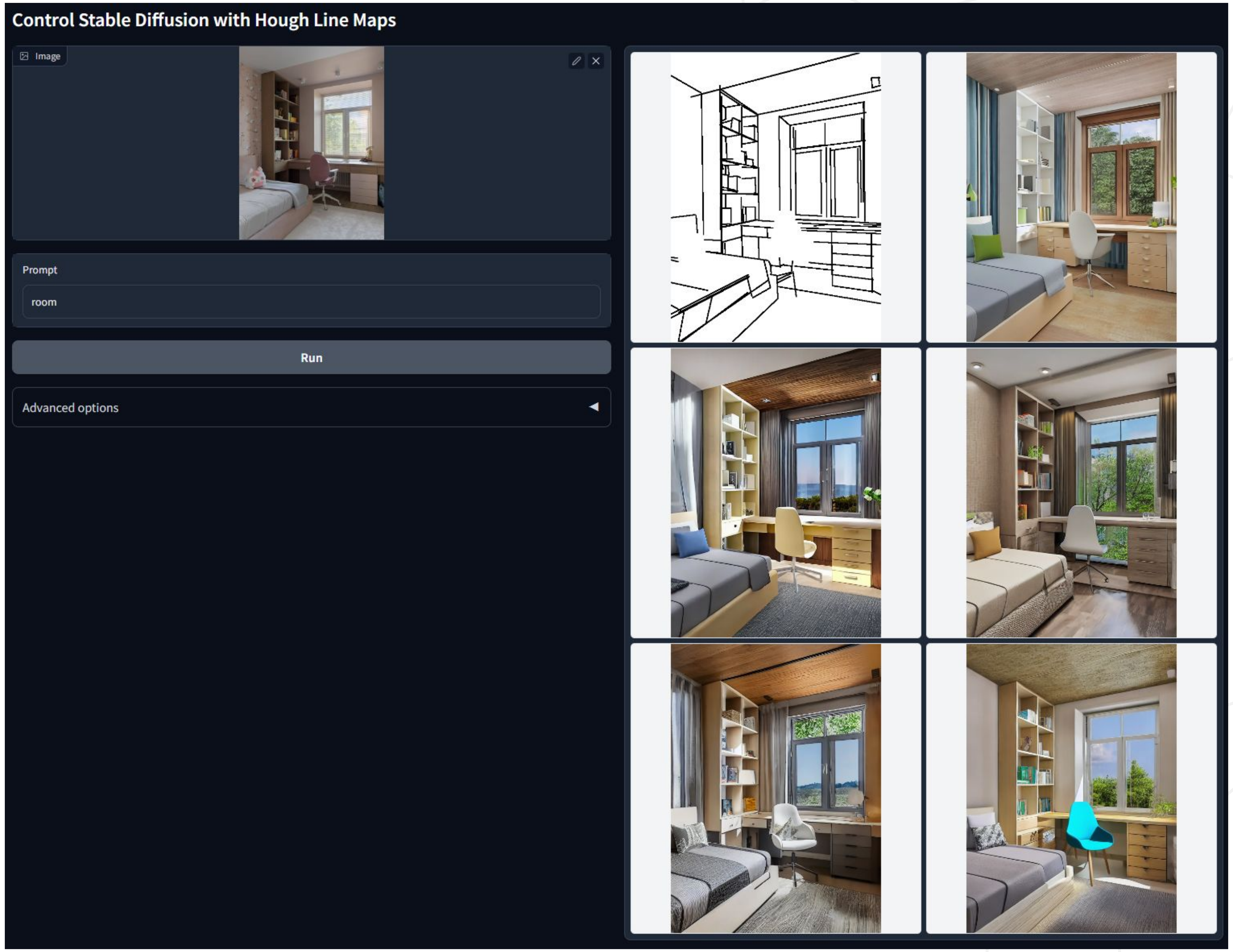
$$\underset{3\times 3}{\begin{bmatrix} 2 & 20 & 1 \\ 4 & 40 & 2 \\ 6 & 60 & 3 \end{bmatrix}} = \underset{3\times 1}{\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}} \times \underset{1\times 3}{\begin{bmatrix} 2 & 20 & 30 \end{bmatrix}}$$

What is ControlNet?

Generation of Synthetic Network Trace

prompt: “room”

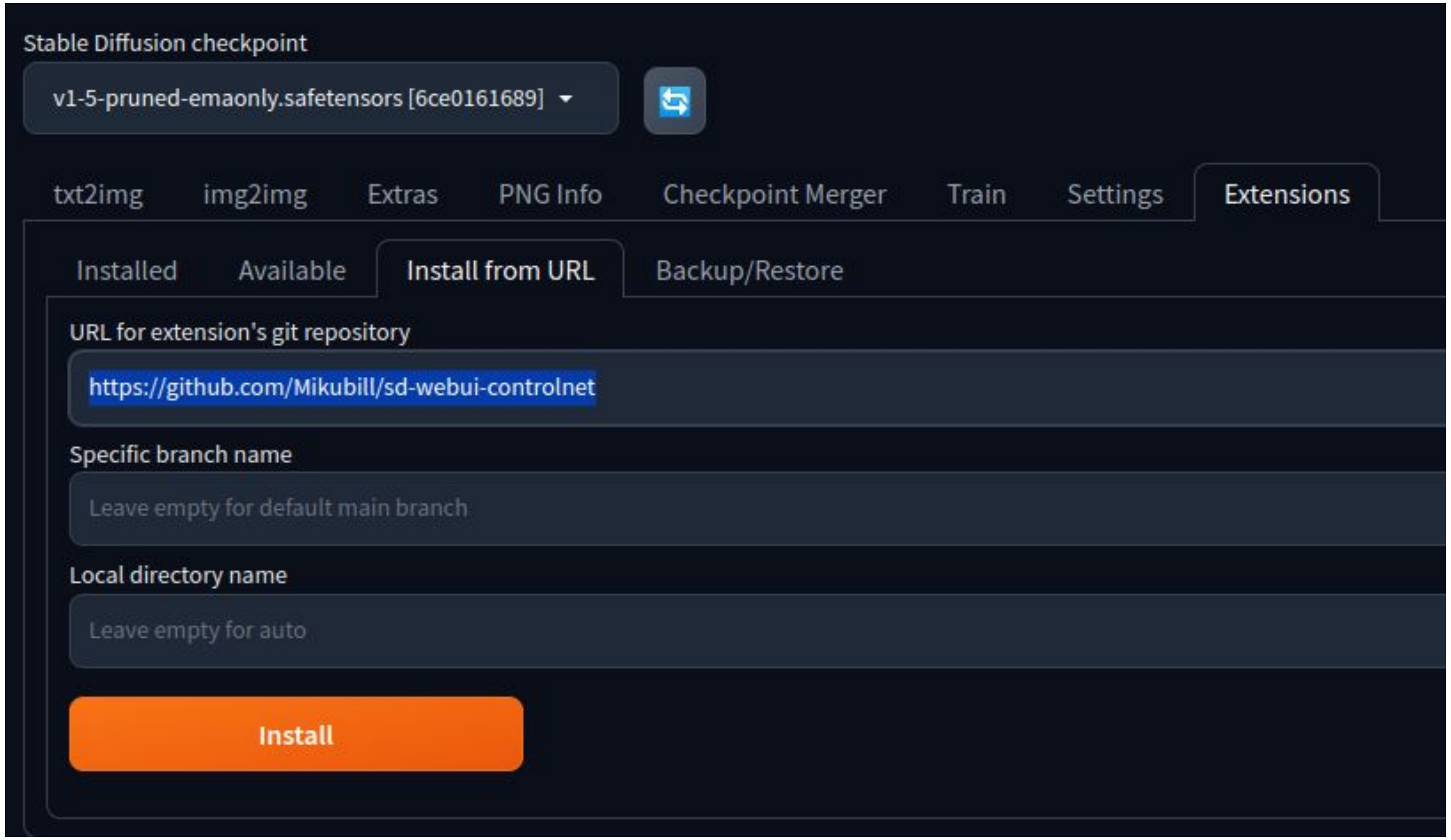
ControlNet with M-LSD Lines



NetDiffusion: Main Steps

Generation of Synthetic Network Trace

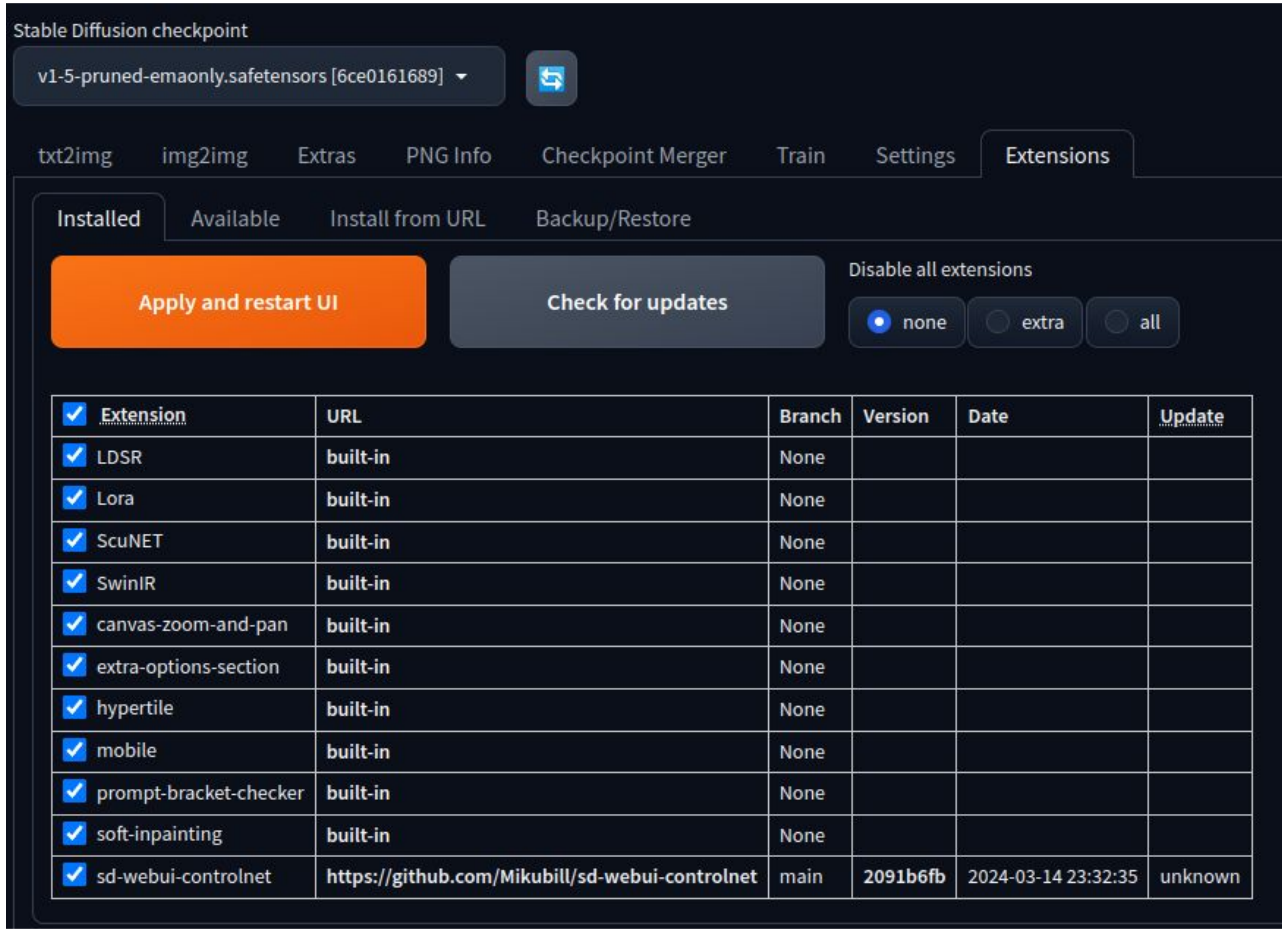
7. Installing ControlNet extension



NetDiffusion: Main Steps

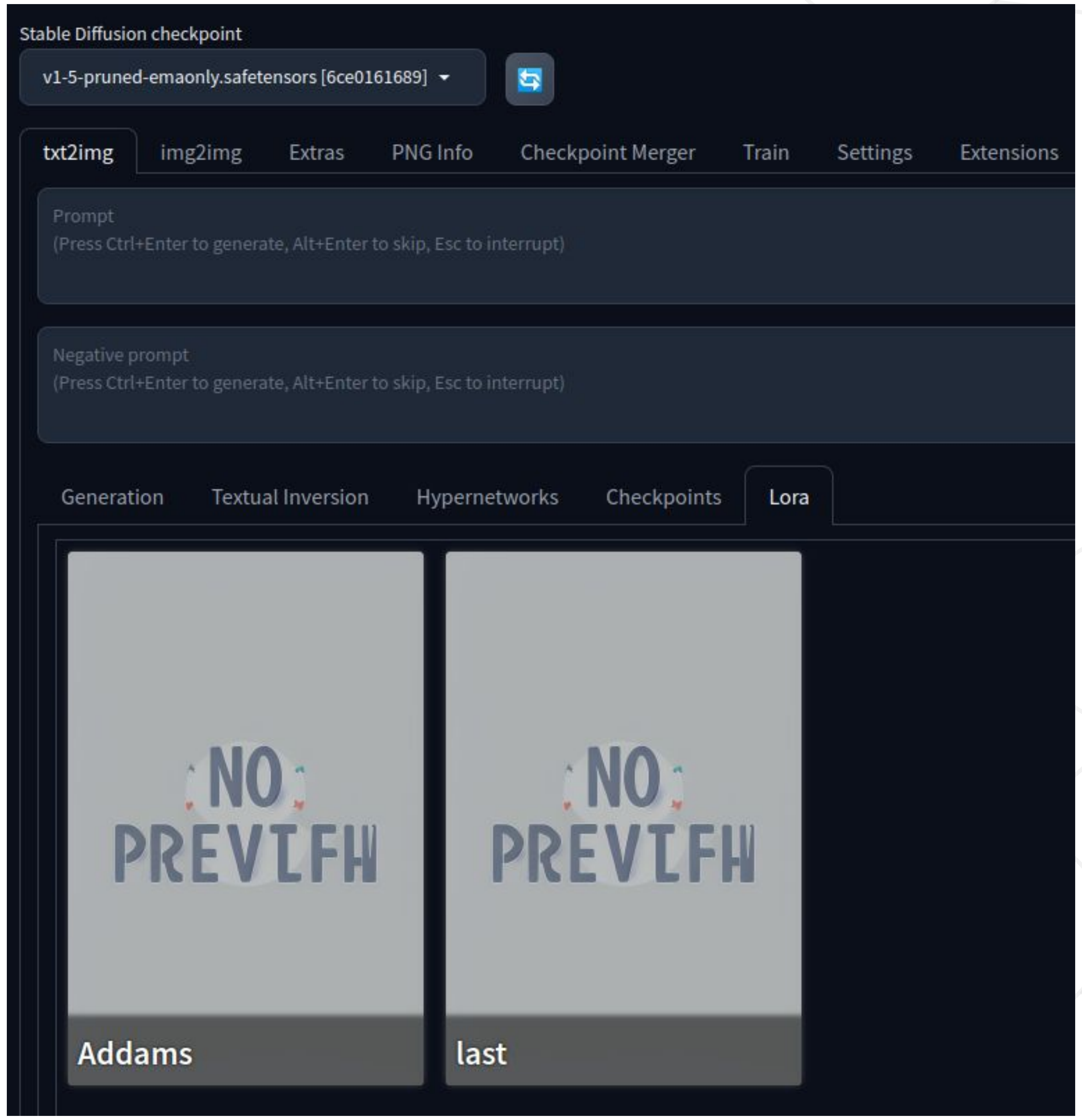
Generation of Synthetic Network Trace

8. Restarting WEB-UI and showing installed ControlNet extension



Extension	URL	Branch	Version	Date	Update
LDSR	built-in	None			
Lora	built-in	None			
ScuNET	built-in	None			
SwinIR	built-in	None			
canvas-zoom-and-pan	built-in	None			
extra-options-section	built-in	None			
hypertile	built-in	None			
mobile	built-in	None			
prompt-bracket-checker	built-in	None			
soft-inpainting	built-in	None			
sd-webui-controlnet	https://github.com/Mikubill/sd-webui-controlnet	main	2091b6fb	2024-03-14 23:32:35	unknown

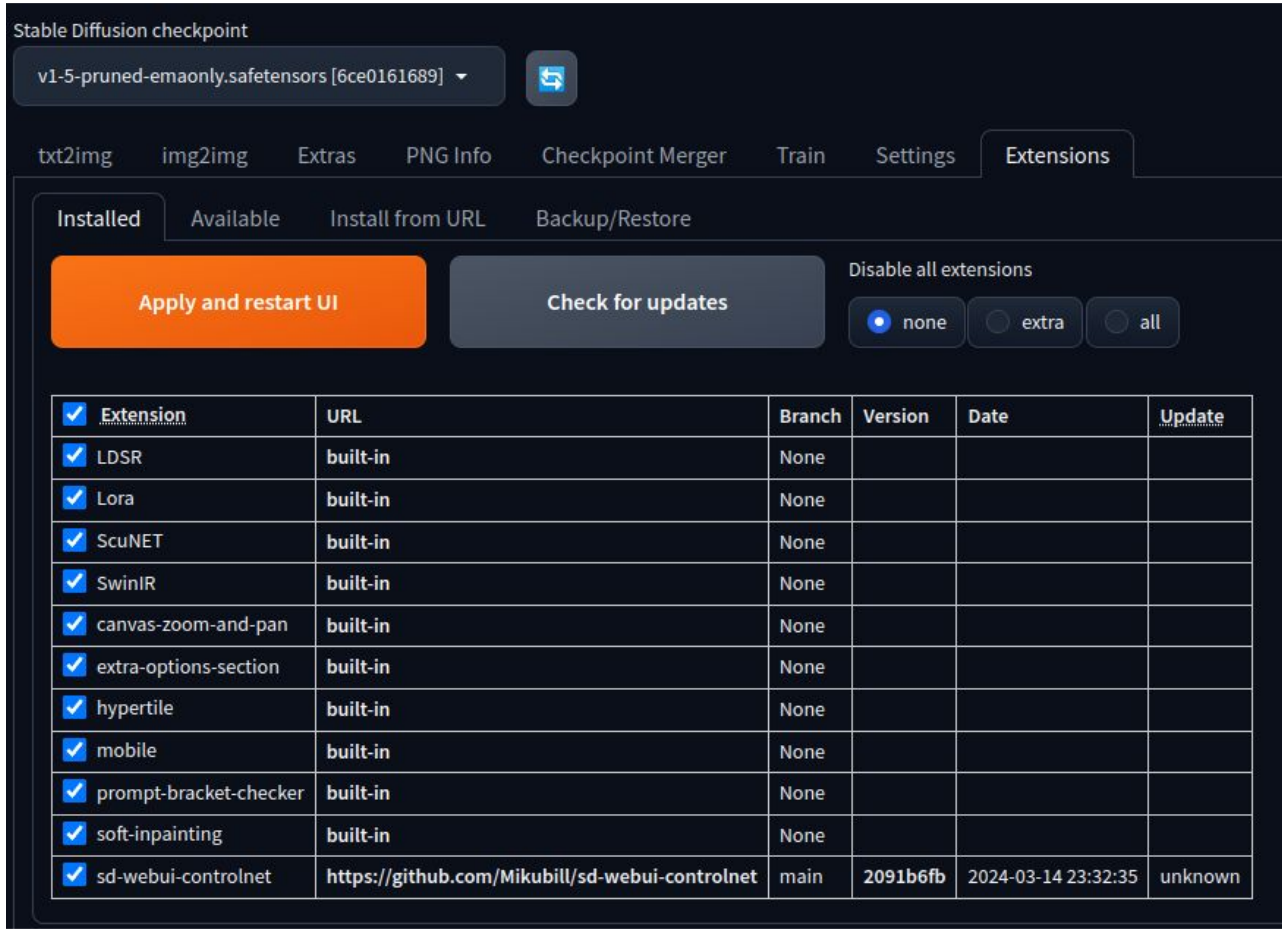
9. Showing the LoRA models



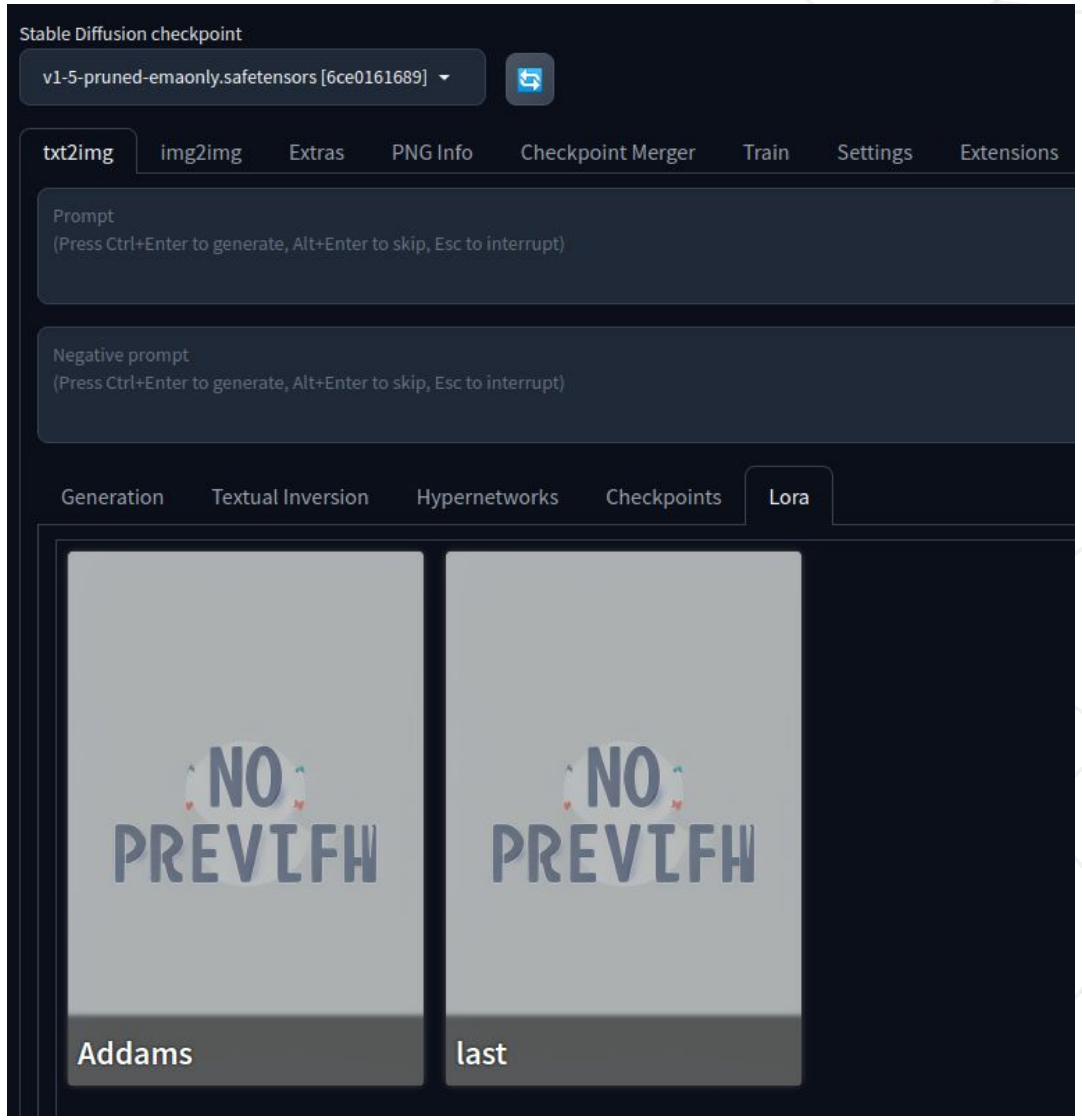
NetDiffusion: Main Steps

Generation of Synthetic Network Trace

8. Restarting WEB-UI and showing installed ControlNet extension



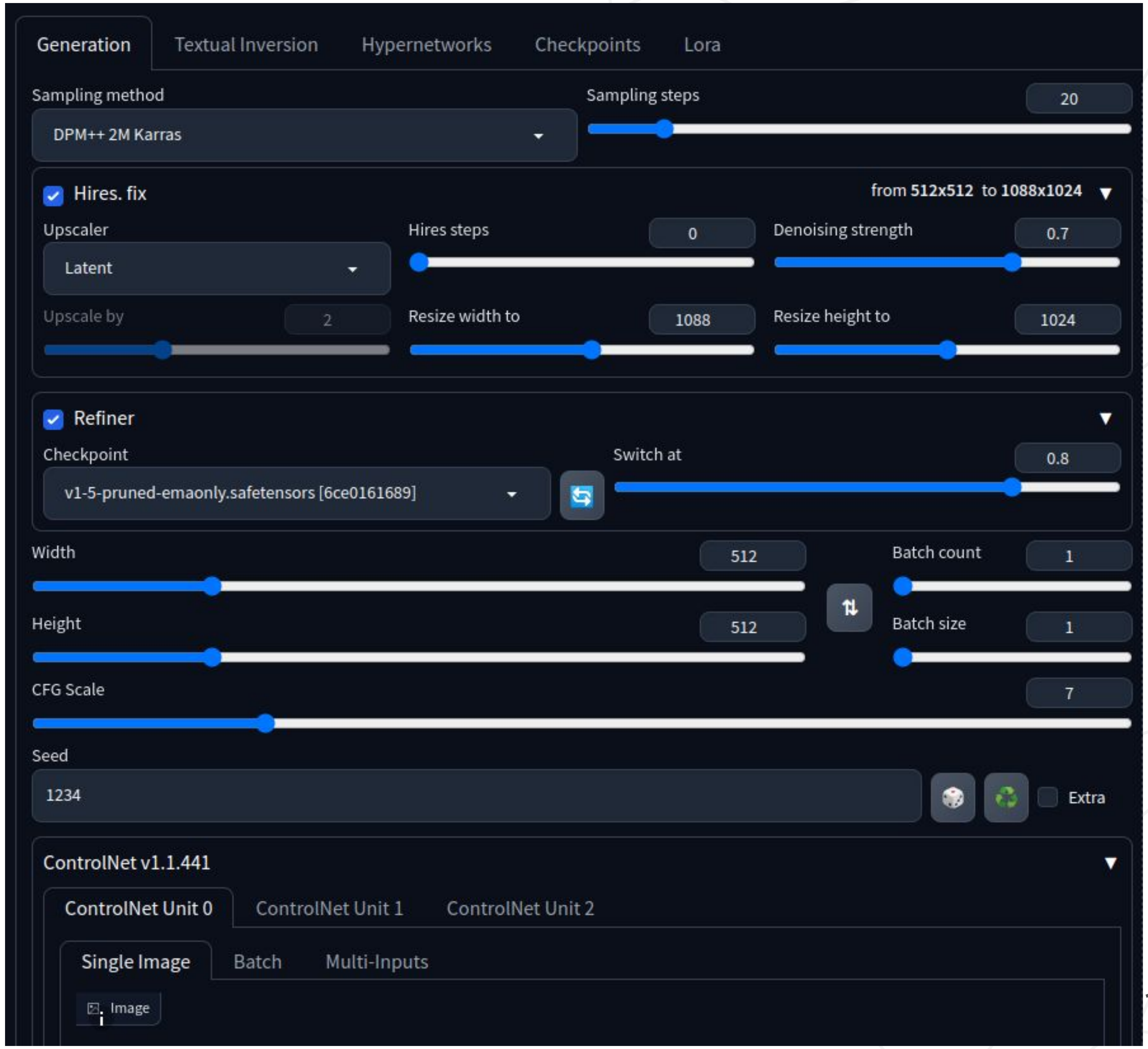
9. Showing the LoRA models



NetDiffusion: Main Steps

Generation of Synthetic Network Trace

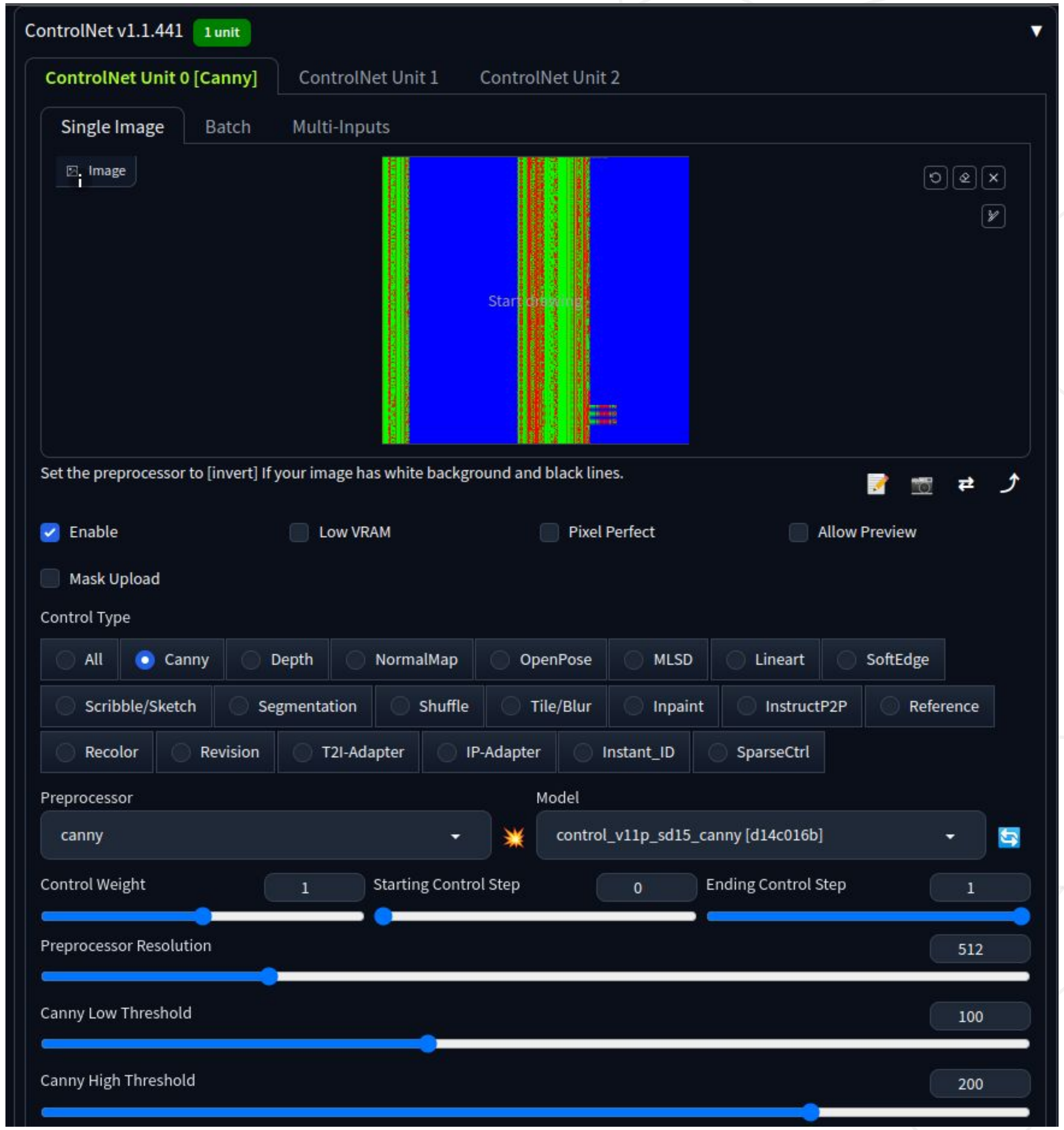
10. Upscaling the final image from 816x768 to 1088x1024



NetDiffusion: Main Steps

Generation of Synthetic Network Trace

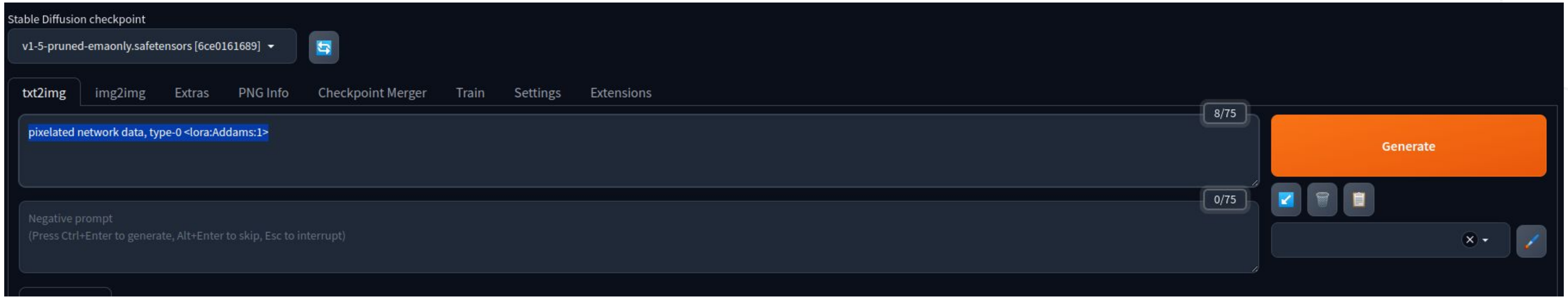
11. Setting up ControlNet with Canny filter



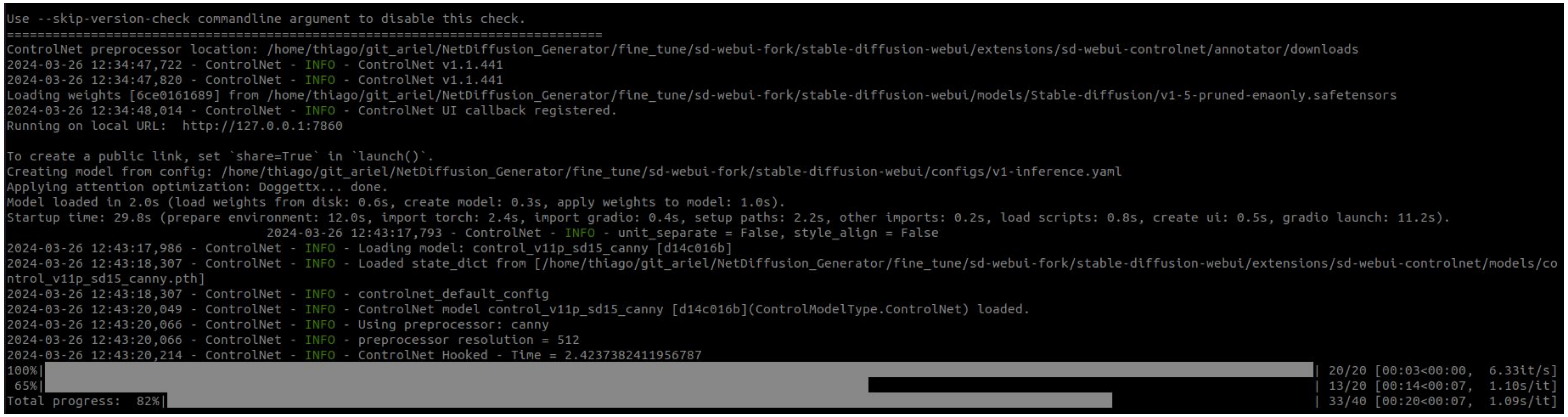
NetDiffusion: Main Steps

Generation of Synthetic Network Trace

12. Setting the final prompt



13. Generating the final image

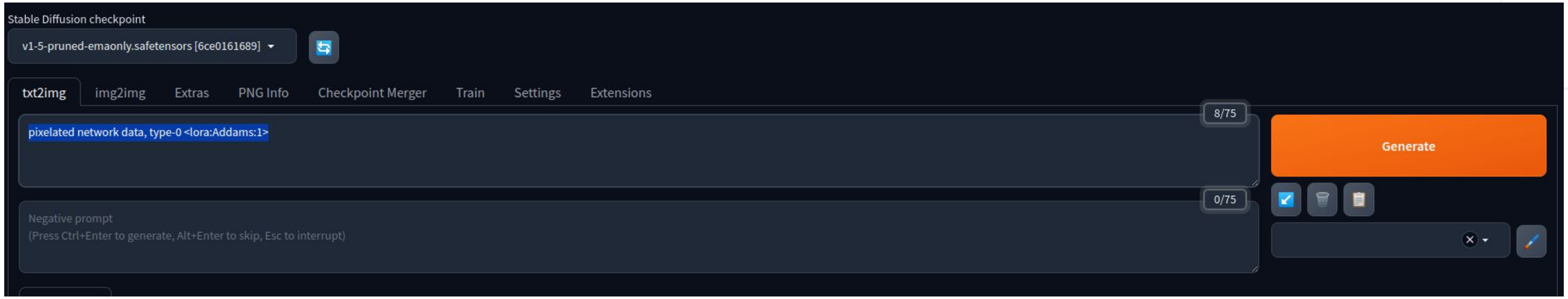


```
Use --skip-version-check commandline argument to disable this check.
==============================================================================
ControlNet preprocessor location: /home/thiago/git_ariel/NetDiffusion_Generator/fine_tune/sd-webui-fork/stable-diffusion-webui/extensions/sd-webui-controlnet/annotator/downloads
2024-03-26 12:34:47,722 - ControlNet - INFO - ControlNet v1.1.441
2024-03-26 12:34:47,820 - ControlNet - INFO - ControlNet v1.1.441
Loading weights [6ce0161689] from /home/thiago/git_ariel/NetDiffusion_Generator/fine_tune/sd-webui-fork/stable-diffusion-webui/models/Stable-diffusion/v1-5-pruned-emaonly.safetensors
2024-03-26 12:34:48,014 - ControlNet - INFO - ControlNet UI callback registered.
Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
Creating model from config: /home/thiago/git_ariel/NetDiffusion_Generator/fine_tune/sd-webui-fork/stable-diffusion-webui/configs/v1-inference.yaml
Applying attention optimization: Doggettx... done.
Model loaded in 2.0s (load weights from disk: 0.6s, create model: 0.3s, apply weights to model: 1.0s).
Startup time: 29.8s (prepare environment: 12.0s, import torch: 2.4s, import gradio: 0.4s, setup paths: 2.2s, other imports: 0.2s, load scripts: 0.8s, create ui: 0.5s, gradio launch: 11.2s).
                                  2024-03-26 12:43:17,793 - ControlNet - INFO - unit_separate = False, style_align = False
2024-03-26 12:43:17,986 - ControlNet - INFO - Loading model: control_v11p_sd15_canny [d14c016b]
2024-03-26 12:43:18,307 - ControlNet - INFO - Loaded state_dict from [/home/thiago/git_ariel/NetDiffusion_Generator/fine_tune/sd-webui-fork/stable-diffusion-webui/extensions/sd-webui-controlnet/models/co
ntrol_v11p_sd15_canny.pth]
2024-03-26 12:43:18,307 - ControlNet - INFO - controlnet_default_config
2024-03-26 12:43:20,049 - ControlNet - INFO - ControlNet model control_v11p_sd15_canny [d14c016b](ControlModelType.ControlNet) loaded.
2024-03-26 12:43:20,066 - ControlNet - INFO - Using preprocessor: canny
2024-03-26 12:43:20,066 - ControlNet - INFO - preprocessor resolution = 512
2024-03-26 12:43:20,214 - ControlNet - INFO - ControlNet Hooked - Time = 2.4237382411956787
100%|                                                                      | 20/20 [00:03<00:00,  6.33it/s]
 65%|                                                                      | 13/20 [00:14<00:07,  1.10s/it]
Total progress:  82%|                                                        | 33/40 [00:20<00:07,  1.09s/it]
```

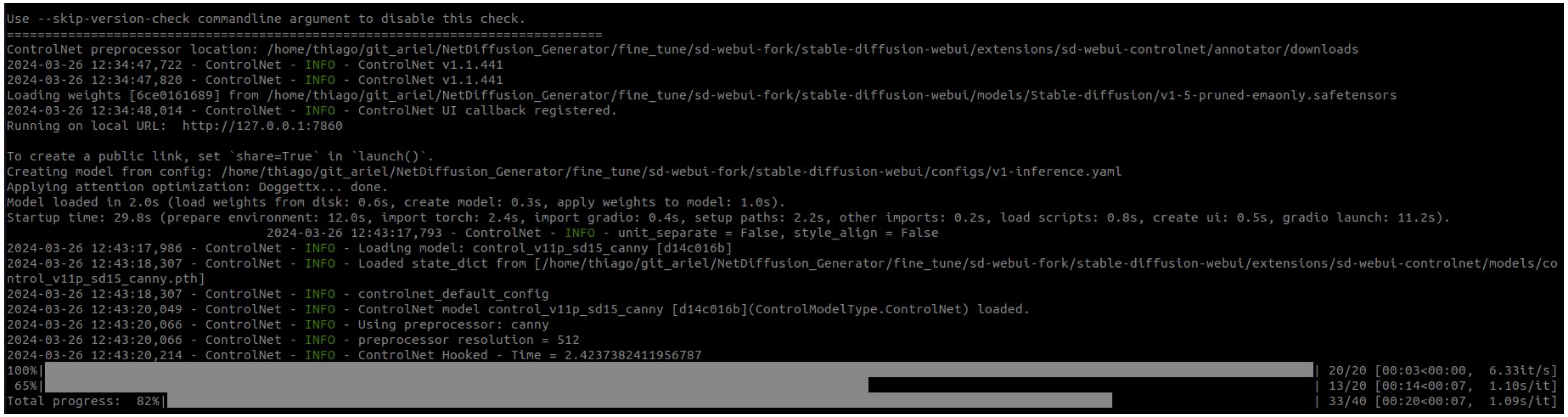
NetDiffusion: Main Steps

Generation of Synthetic Network Trace

12. Setting the final prompt



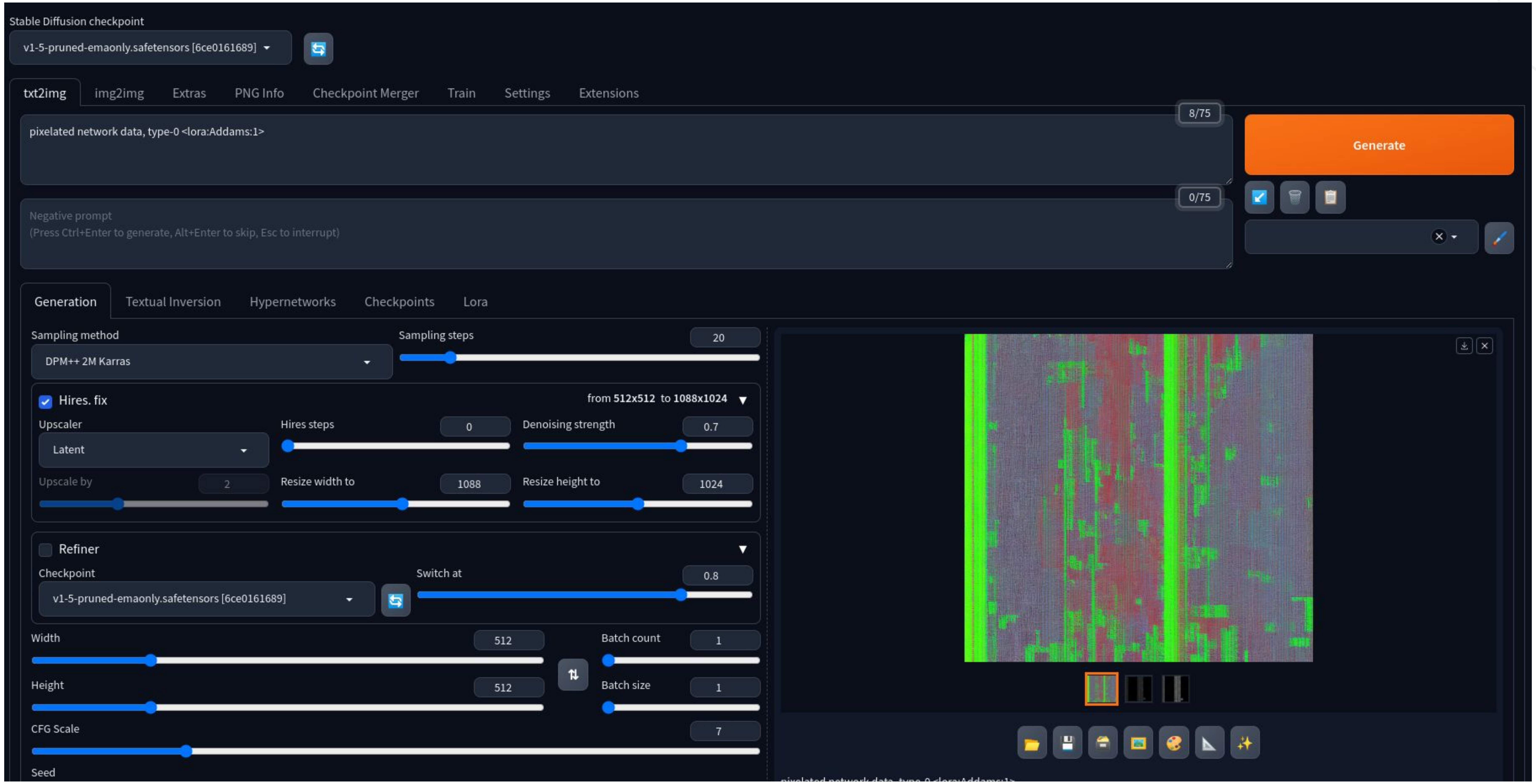
13. Generating the final image



NetDiffusion: Main Steps

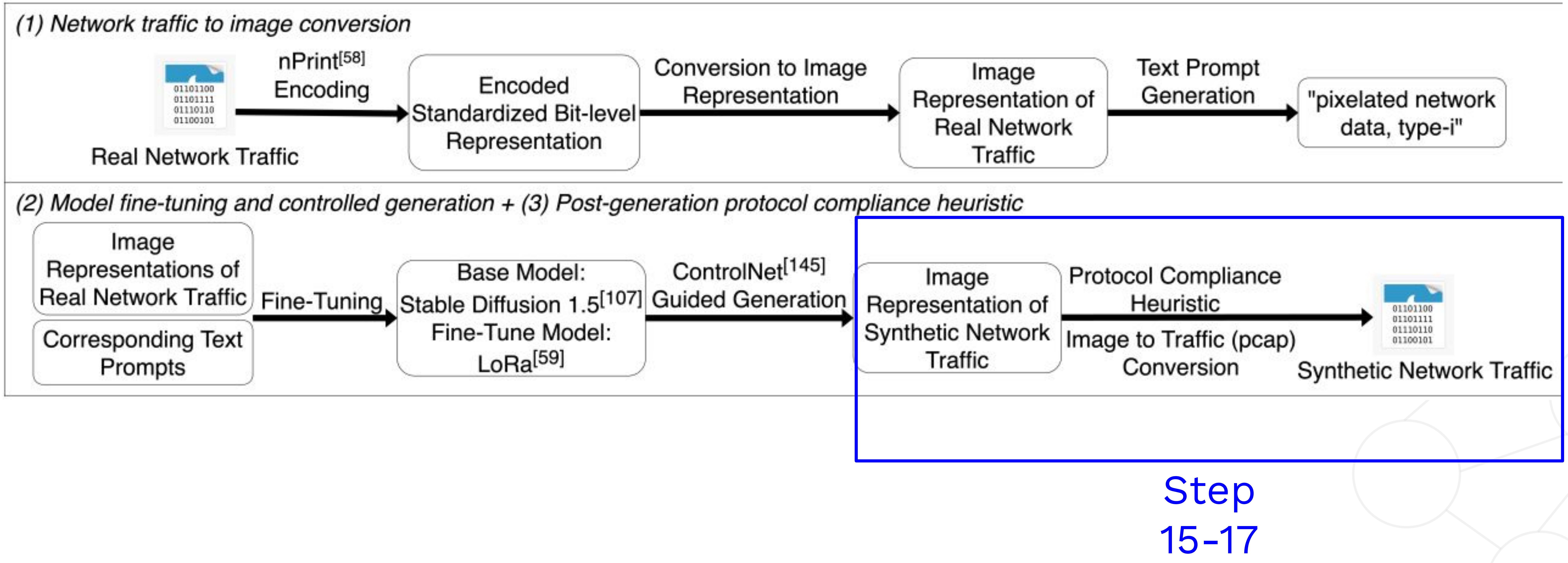
Generation of Synthetic Network Trace

14. Visualizing the final image



NetDiffusion: Workflow

Generation of Synthetic Network Trace



NetDiffusion: Main Steps

Generation of Synthetic Network Trace

15. Listing the final generated image

```
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/fine_tune/sd-webui-fork/stable-diffusion-webui/output/txt2img-images/2024-03-26$ ls
00000-1234.png
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/fine_tune/sd-webui-fork/stable-diffusion-webui/output/txt2img-images/2024-03-26$
```

16. Post-processing the final image

```
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/post-generation$ python3 color_processor.py && python3 img_to_nprint.py && python3 mass_reconstruction.py
1088
1024
1088
1024
1088
1024
../data/generated_nprint/00000-1234.nprint
../data/replayable_generated_pcaps/00000-1234.pcap
/home/thiago/git_ariel/NetDiffusion_Generator/post-generation/reconstruction.py:74: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future vers
s. Value '199.180.150.116' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.
  generated_nprint.at[idx, 'src_ip'] = implementing_src_ip
tcp
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: more Inconsistent linktypes detected! The resulting file might contain invalid packets.
../data/generated_nprint/00008-1234.nprint
../data/replayable_generated_pcaps/00008-1234.pcap
/home/thiago/git_ariel/NetDiffusion_Generator/post-generation/reconstruction.py:74: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future vers
s. Value '24.43.115.154' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.
  generated_nprint.at[idx, 'src_ip'] = implementing_src_ip
udp
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: more Inconsistent linktypes detected! The resulting file might contain invalid packets.
../data/generated_nprint/netflix_5.nprint
../data/replayable_generated_pcaps/netflix_5.pcap
/home/thiago/git_ariel/NetDiffusion_Generator/post-generation/reconstruction.py:74: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future vers
s. Value '27.14.20.98' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.
  generated_nprint.at[idx, 'src_ip'] = implementing_src_ip
tcp
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: more Inconsistent linktypes detected! The resulting file might contain invalid packets.
```

NetDiffusion: Main Steps

Generation of Synthetic Network Trace

15. Listing the final generated image

```
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/fine_tune/sd-webui-fork/stable-diffusion-webui/output/txt2img-images/2024-03-26$ ls
00000-1234.png
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/fine_tune/sd-webui-fork/stable-diffusion-webui/output/txt2img-images/2024-03-26$
```

16. Post-processing the final image

```
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/post-generation$ python3 color_processor.py && python3 img_to_nprint.py && python3 mass_reconstruction.py
1088
1024
1088
1024
1088
1024
../data/generated_nprint/00000-1234.nprint
../data/replayable_generated_pcaps/00000-1234.pcap
/home/thiago/git_ariel/NetDiffusion_Generator/post-generation/reconstruction.py:74: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future vers
s. Value '199.180.150.116' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.
  generated_nprint.at[idx, 'src_ip'] = implementing_src_ip
tcp
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: more Inconsistent linktypes detected! The resulting file might contain invalid packets.
../data/generated_nprint/00008-1234.nprint
../data/replayable_generated_pcaps/00008-1234.pcap
/home/thiago/git_ariel/NetDiffusion_Generator/post-generation/reconstruction.py:74: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future vers
s. Value '24.43.115.154' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.
  generated_nprint.at[idx, 'src_ip'] = implementing_src_ip
udp
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: more Inconsistent linktypes detected! The resulting file might contain invalid packets.
../data/generated_nprint/netflix_5.nprint
../data/replayable_generated_pcaps/netflix_5.pcap
/home/thiago/git_ariel/NetDiffusion_Generator/post-generation/reconstruction.py:74: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future vers
s. Value '27.14.20.98' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.
  generated_nprint.at[idx, 'src_ip'] = implementing_src_ip
tcp
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: Inconsistent linktypes detected! The resulting file might contain invalid packets.
WARNING: more Inconsistent linktypes detected! The resulting file might contain invalid packets.
```

NetDiffusion: Main Steps

Generation of Synthetic Network Trace

17. Testing the replayable PCAPs with tcpreplay

```
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [R.], seq 61:113, ack 0, win 608, options [sack 1 {1358143278:1389648685},unknown-36,sackOK[len 3],[bad opt]>
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [SR.], seq 2315281366:2315281430, ack 2938939350, win 1003, options [sack 1 {1358255916:1423268654},unknown-52,[bad opt]>
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [SRP.], seq 2315281430:2315281434, ack 2938939350, win 530, options [sack 1 {1358180911:1389647915},eol], length 4
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [SRP.], seq 2315281434:2315281494, ack 2938939350, win 547, options [sack 1 {1899314735:1389648171},eol], length 60
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [SR.], seq 2315281494:2315281554, ack 2938939350, win 768, options [sack 1 {1358255660:1372871470},sack 0 ,[bad opt]>
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [SR.], seq 2315281554, ack 2938939350, win 512, options [[bad opt]
3:00:00.000000 IP 199.180.150.116.44373 > 67.79.64.168.28917: Flags [R.], seq 45, ack 0, win 0, options [sack 1 {2522978159:1996660334},mss 416,eol], length 0
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [SR.], seq 2315281554, ack 2938939394, win 40, options [sack 1 {1362441987:1389648127},eol], length 0
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [.], ack 1, win 0, options [sack 1 {1899312896:1423268355},[bad opt]>
3:00:00.000000 IP 199.180.150.116.44373 > 67.79.64.168.28917: Flags [P.], seq 1:61, ack 0, win 640, options [sack 1 {2522978671:2013567724},eol], length 60
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [R.U], seq 0:64, ack 1, win 0, urg 0, options [[bad opt]
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [SRP.UEW], seq 2315281618:2315281658, ack 2938939394, win 256, urg 0, options [[bad opt]
3:00:00.000000 IP 199.180.150.116.44373 > 67.79.64.168.28917: Flags [SRP.], seq 2938939454, ack 2315281618, win 256, options [sack 1 {2520881712:2046925615},eol], length 0
3:00:00.000000 IP 67.79.64.168.28917 > 199.180.150.116.44373: Flags [SR.], seq 2315281658, ack 2938939454, win 256, options [[bad opt]
3:00:00.000000 IP 199.180.150.116.44373 > 67.79.64.168.28917: Flags [SRP.], seq 2938939454:2938939514, ack 2315281658, win 120, options [sack 1 {2519832843:2013305607},eol], length 60
^C User interrupt...
sendpacket_abort
Actual: 2014 packets (246652 bytes) sent in 0.317695 seconds
Rated: 776379.8 Bps, 6.21 Mbps, 6339.41 pps
Flows: 2 flows, 6.29 fps, 4028 flow packets, 0 non-flow
Statistics for network device: eno1
        Successful packets:        2013
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/data/replayable_generated_pcaps$ ^C
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/data/replayable_generated_pcaps$ sudo tcpreplay --loop=0 --verbose -i eno1 00000-1234.pcap
```

Conclusions and future perspectives

Conclusions and future perspectives

- ***Role of Generative AIs:*** Simulate complex network environments and generate high-fidelity synthetic data, enhancing training for RL algorithms and network management.
- ***Evolution and Application:*** Development from basic generative models to advanced GANs capable of producing realistic network traffic such as PCAP files.
- ***Practical Use:*** Generating synthetic time series data for network telemetry and training ML models, particularly valuable in privacy-sensitive applications.
- ***Future Trends:*** Integration of GANs with network management tasks, promising innovative solutions for dynamic, complex systems.
- ***Research Opportunities:*** Challenges in synthetic data generation, network simulation, and AI integration suggest significant potential for advancing network systems.

Thanks!

- ***Thiago Caproni Tavares***
  - ***thiago.caproni@ifsuldeminas.edu.br***
- ***Ariel Góes de Castro***
  - ***a272319@dac.unicamp.br***
- ***Leandro C. de Almeida***
  - ***leandro.almeida@ifpb.edu.br***
- ***Washington Rodrigo Dias da Silva***
  - ***washingtonrds@estudante.ufscar.br***
- ***Christian Esteve Rothenberg***
  - ***chesteve@unicamp.br***
- ***Luciano Verdi***
  - ***verdi@ufscar.br***