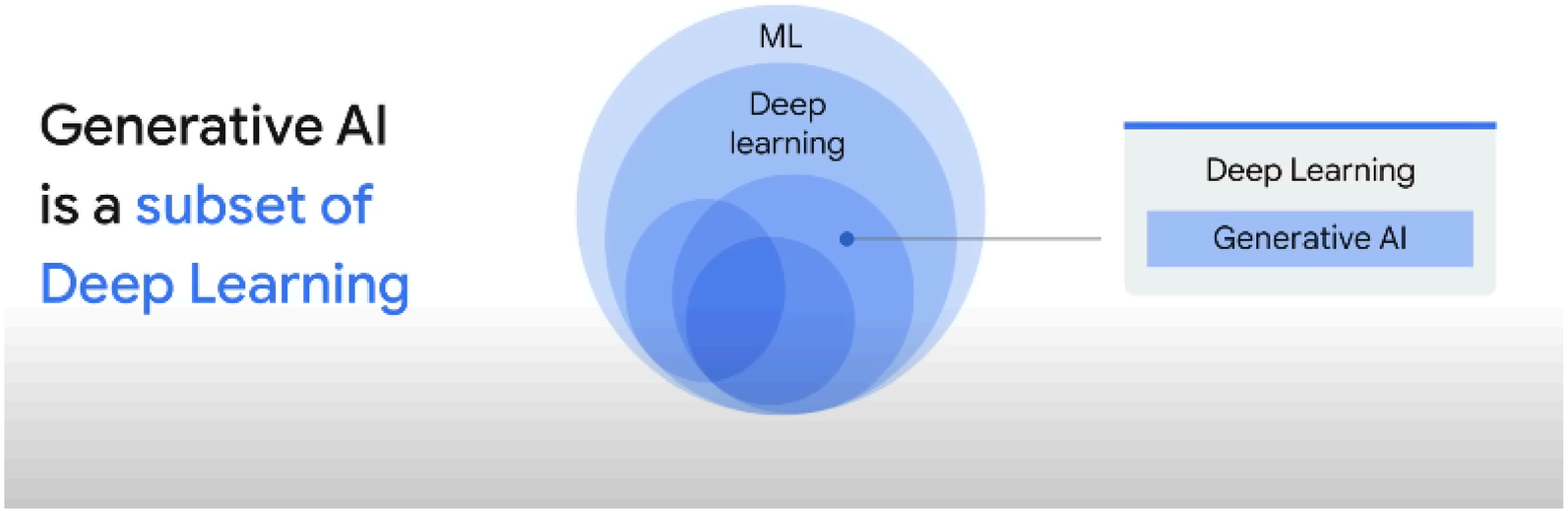
# GENERATIVE AI

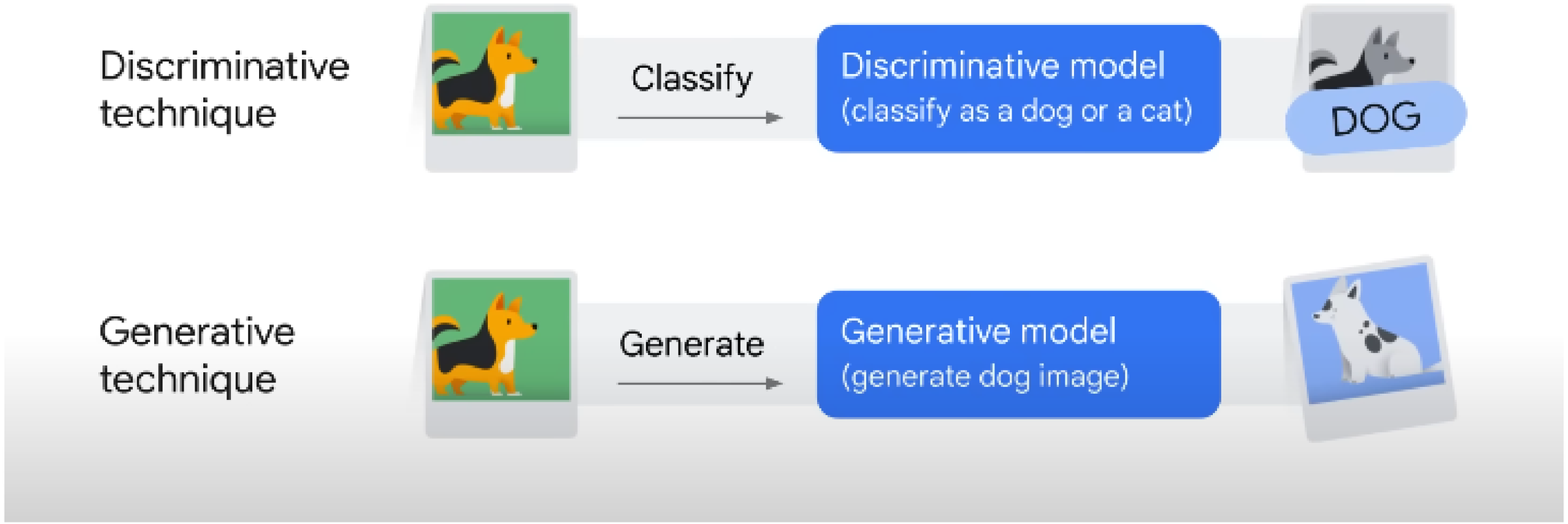
AI That Inspires, Designs, and Innovates.

# WHAT IS GEN AI?

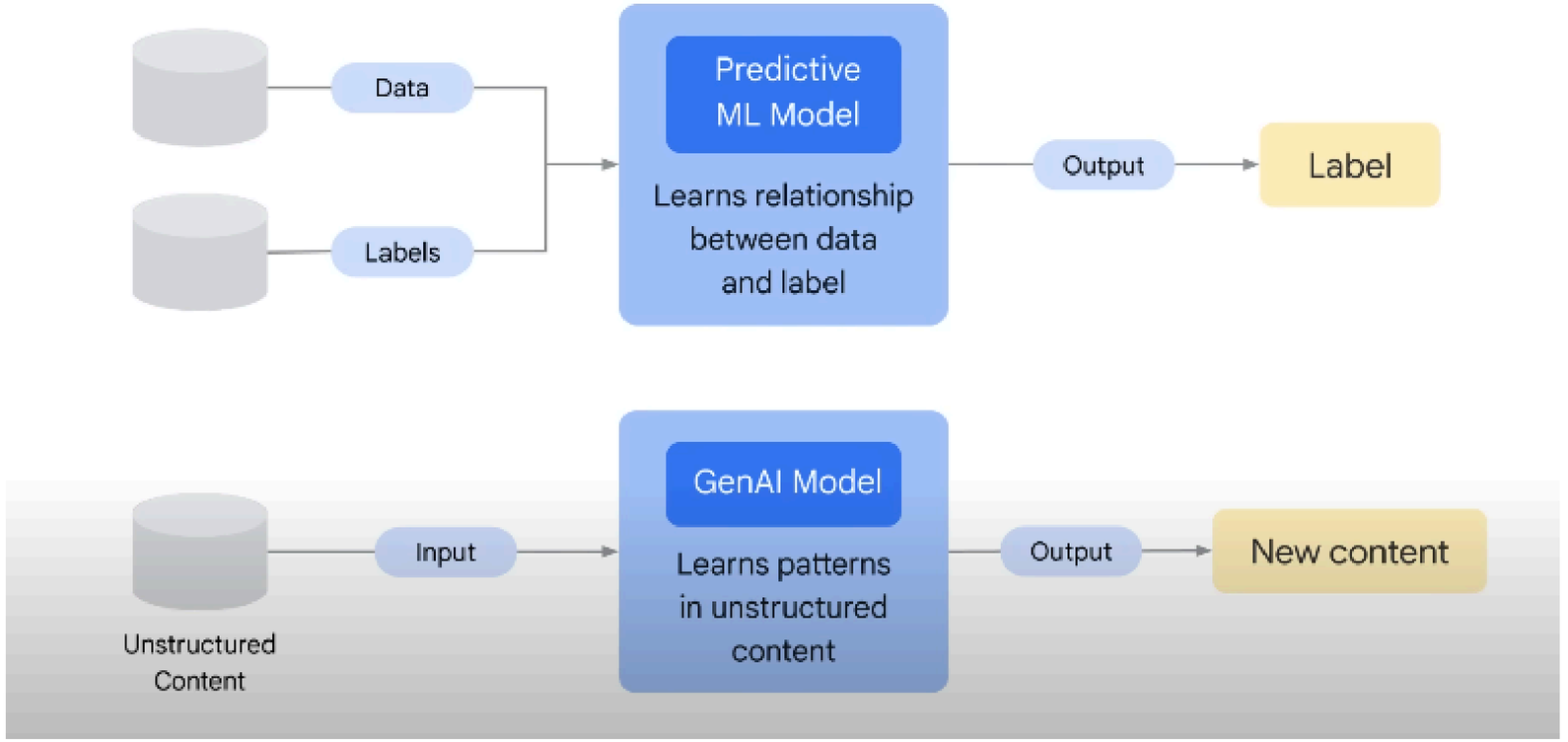
GENERATIVE AI REFERS TO A SUBSET OF ARTIFICIAL INTELLIGENCE MODELS DESIGNED TO CREATE NEW CONTENT, SUCH AS IMAGES, TEXT, MUSIC, AND EVEN CODE, BY LEARNING FROM EXISTING DATA.



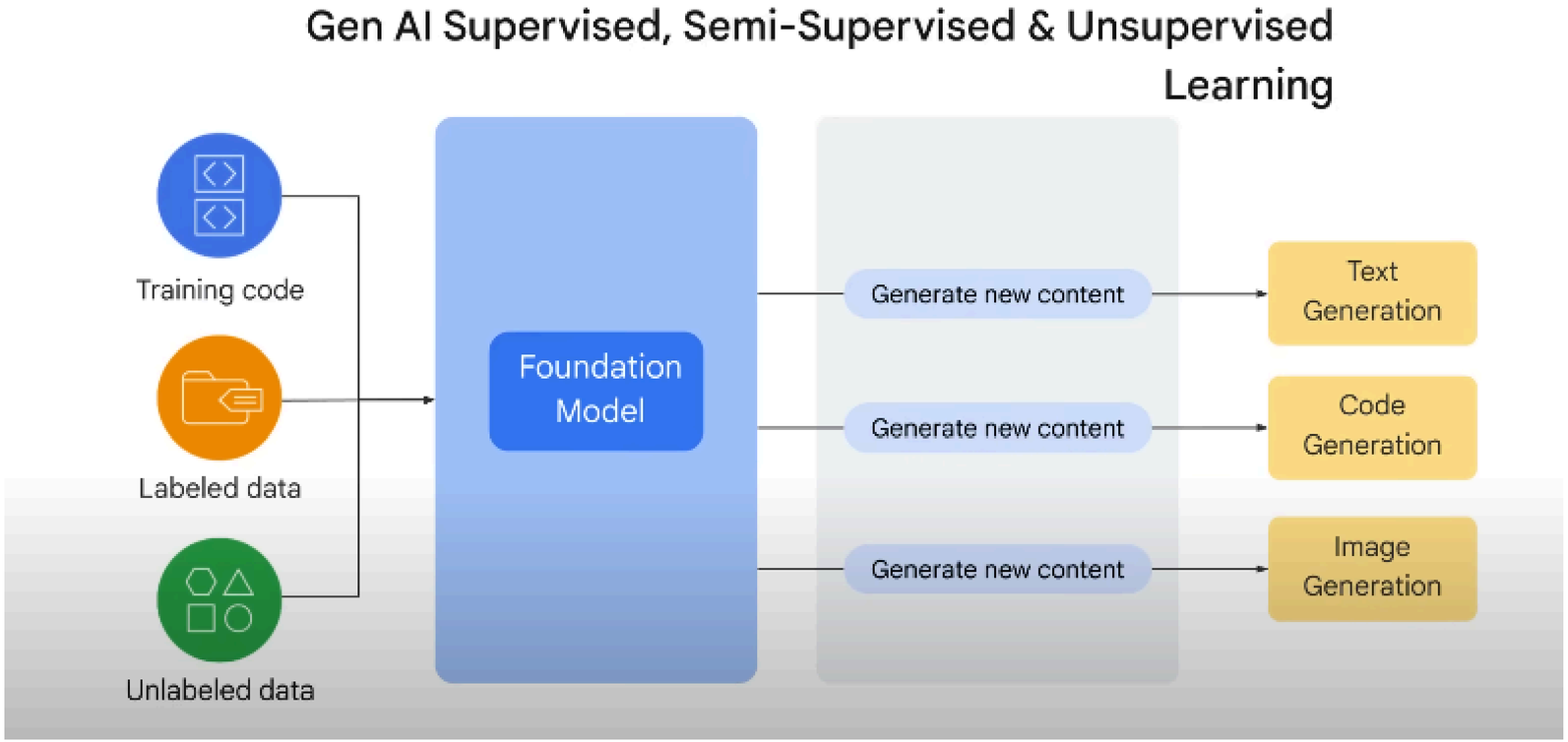
# HOW IS GEN AI DIFFERENT?



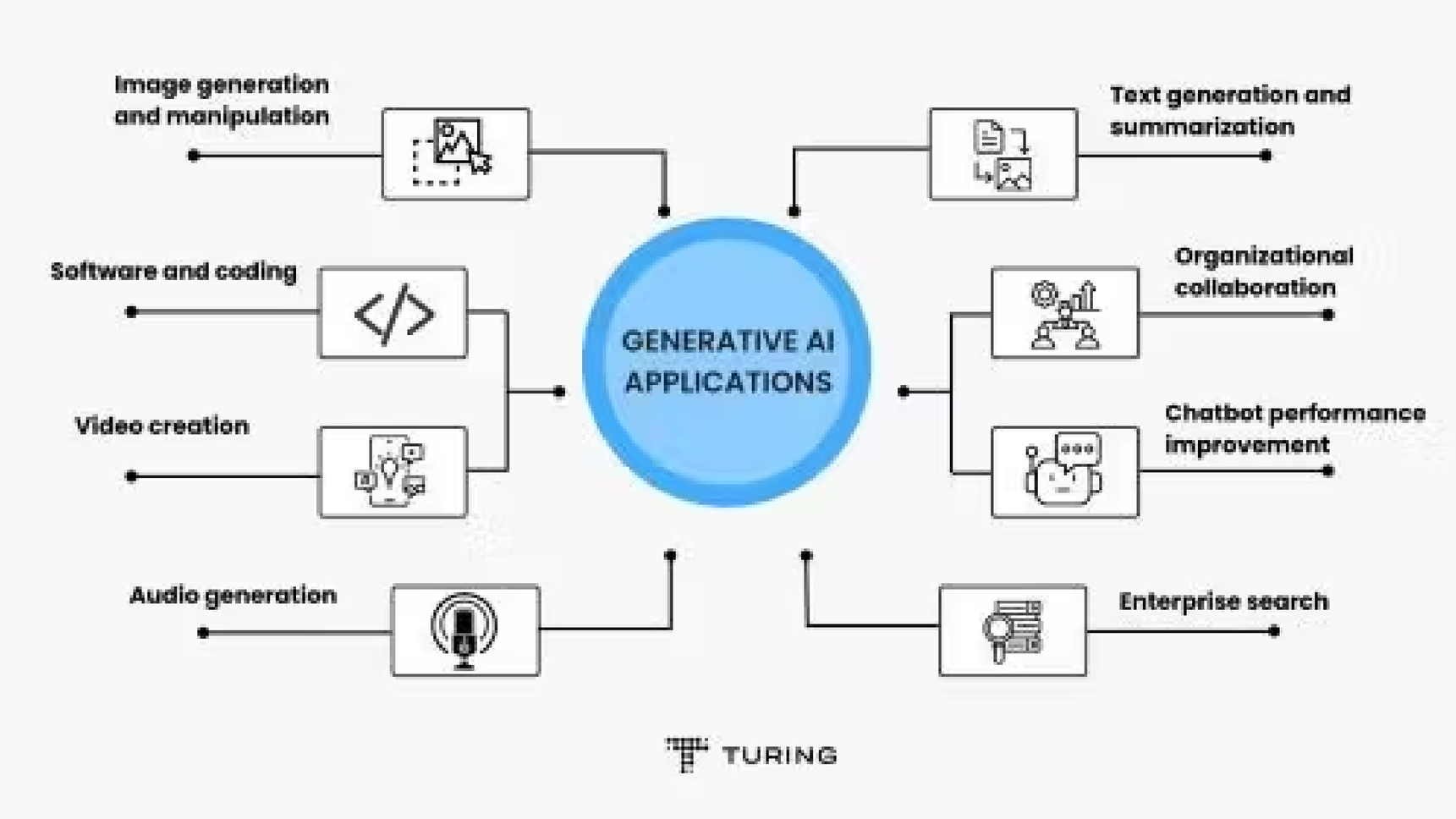
# HOW IS GEN AI DIFFERENT?



# HOW IS GEN AI DIFFERENT?



# USES OF GEN AI



# USES OF GEN AI

- **TEXT GENERATION**
- GENERATIVE AI CAN CREATE COHERENT, HUMAN-LIKE TEXT. TOOLS LIKE OPENAI'S GPT (GENERATIVE PRETRAINED TRANSFORMER) ARE EXAMPLES OF GENERATIVE TEXT MODELS, USEFUL FOR WRITING, SUMMARIZING, OR EVEN CODING.
- **IMAGE GENERATION**
- GENERATIVE AI CAN GENERATE REALISTIC OR STYLIZED IMAGES FROM SCRATCH, USED IN FIELDS SUCH AS ART, GAME DESIGN, AND MARKETING. GANS ARE OFTEN EMPLOYED FOR THIS PURPOSE.
- **MUSIC COMPOSITION**
- AI MODELS CAN CREATE ORIGINAL MUSIC COMPOSITIONS BY LEARNING FROM EXISTING TRACKS. THIS IS USEFUL IN ENTERTAINMENT, ADVERTISING, AND EVEN THERAPY.
- **CODE GENERATION**
- MODELS LIKE COPILOT USE GENERATIVE AI TO ASSIST DEVELOPERS BY WRITING CODE BASED ON GIVEN PROMPTS OR TASKS.

# BENEFITS AND CHALLENGES OF GENERATIVE AI

- ## BENEFITS

- **CREATIVITY ENHANCEMENT**: ASSISTS ARTISTS, WRITERS, AND CREATORS BY PRODUCING HIGH-QUALITY, ORIGINAL CONTENT.
- **AUTOMATION**: SPEEDS UP PROCESSES SUCH AS CONTENT CREATION, DESIGN, AND EVEN PRODUCT DEVELOPMENT.
- **PERSONALIZATION**: ENABLES TAILORED SOLUTIONS, SUCH AS PERSONALIZED ARTWORK OR CONTENT, BASED ON INDIVIDUAL PREFERENCES.

# BENEFITS AND CHALLENGES OF GENERATIVE AI

- ## CHALLENGES

- **ETHICAL CONCERNS**: THERE ARE CHALLENGES WITH COPYRIGHT INFRINGEMENT, AS GENERATIVE AI MODELS MAY REPLICATE PARTS OF THE DATA THEY ARE TRAINED ON..
- **QUALITY CONTROL**: OUTPUTS CAN SOMETIMES LACK ORIGINALITY OR BE PRONE TO BIASES DEPENDING ON THE TRAINING DATA.
- **DEEPFAKES AND MISUSE**: AI-GENERATED IMAGES OR VIDEOS (DEEPFAKES) CAN BE USED FOR MALICIOUS PURPOSES, SUCH AS MISINFORMATION OR FRAUD.

# GENERATIVE AI MODELS

## 01 - GANs

GANs consist of two neural networks: the generator and the discriminator. The generator creates new data (such as images), while the discriminator evaluates whether the data is real (from the training set) or fake (generated). These two networks compete with each other, improving the generator's ability to create realistic outputs over time.

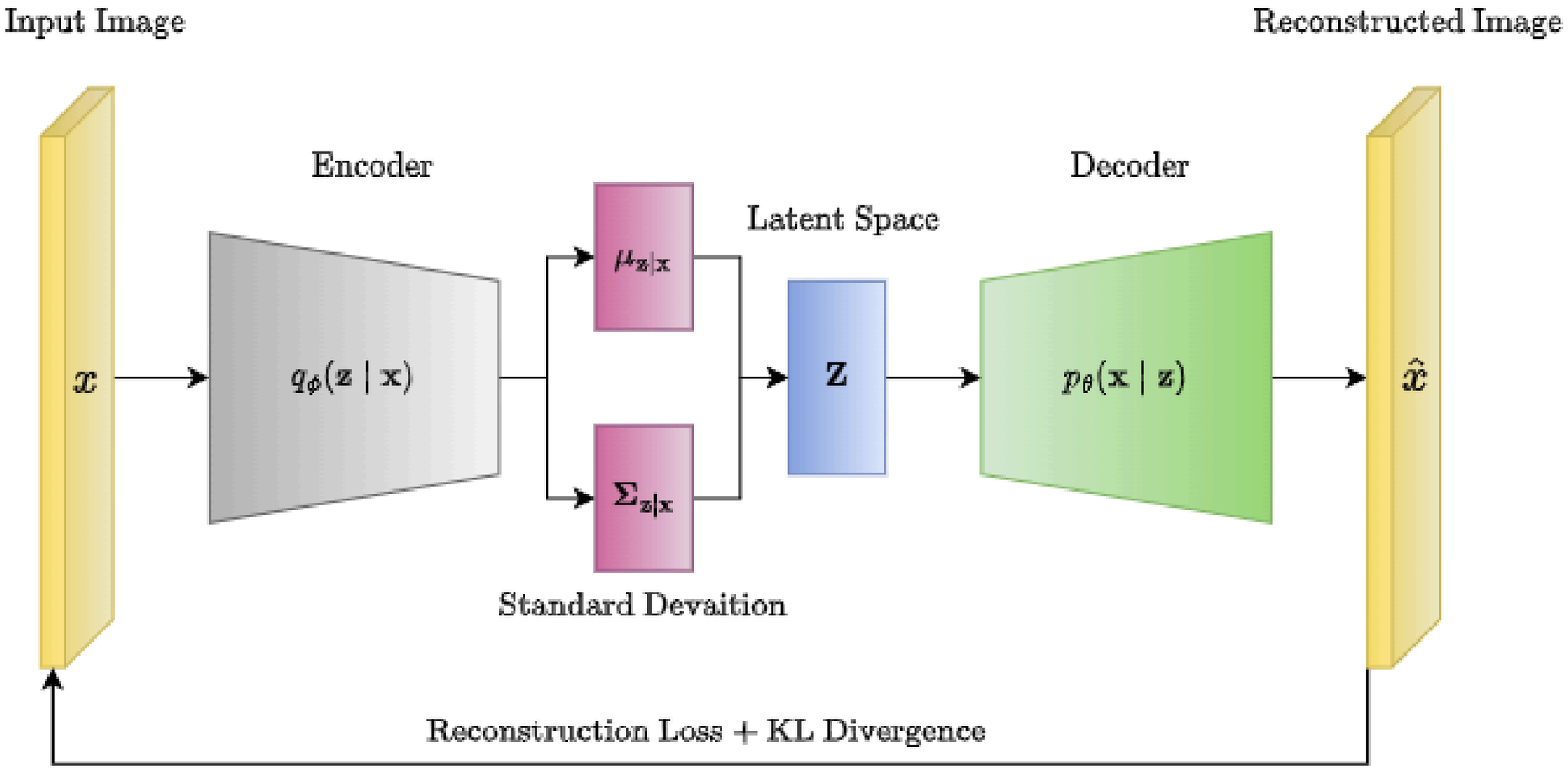
## 02 - VAEs

VAEs encode input data (such as images) into a compressed latent representation, then decode it back into a new version of the input. While the autoencoder compresses the data, the variational aspect adds a level of randomness, allowing it to generate new, similar outputs.

## 03 - DIFFUSION MODELS

Diffusion models generate data by gradually adding noise to training data and then learning to reverse the process to remove the noise. These models have gained attention due to their ability to generate highly detailed and realistic images.

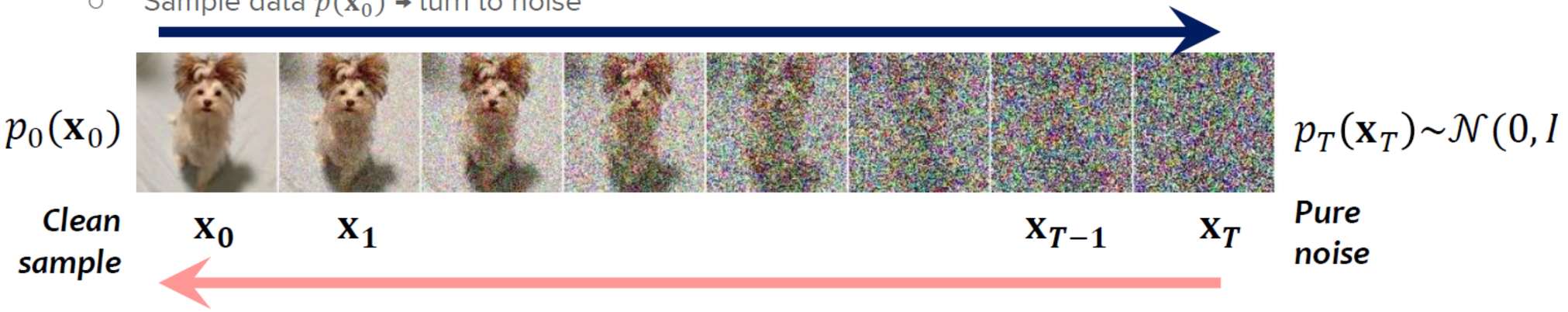
# VAE


Input Image
Reconstructed Image
Encoder
Decoder
Latent Space
$x$
$q_\phi(\mathbf{z} \mid \mathbf{x})$
$\mu_{\mathbf{z}|\mathbf{x}}$
$\Sigma_{\mathbf{z}|\mathbf{x}}$
$\mathbf{z}$
$p_\theta(\mathbf{x} \mid \mathbf{z})$
$\hat{x}$
Standard Devaition
Reconstruction Loss + KL Divergence

# DIFFUSION MODEL

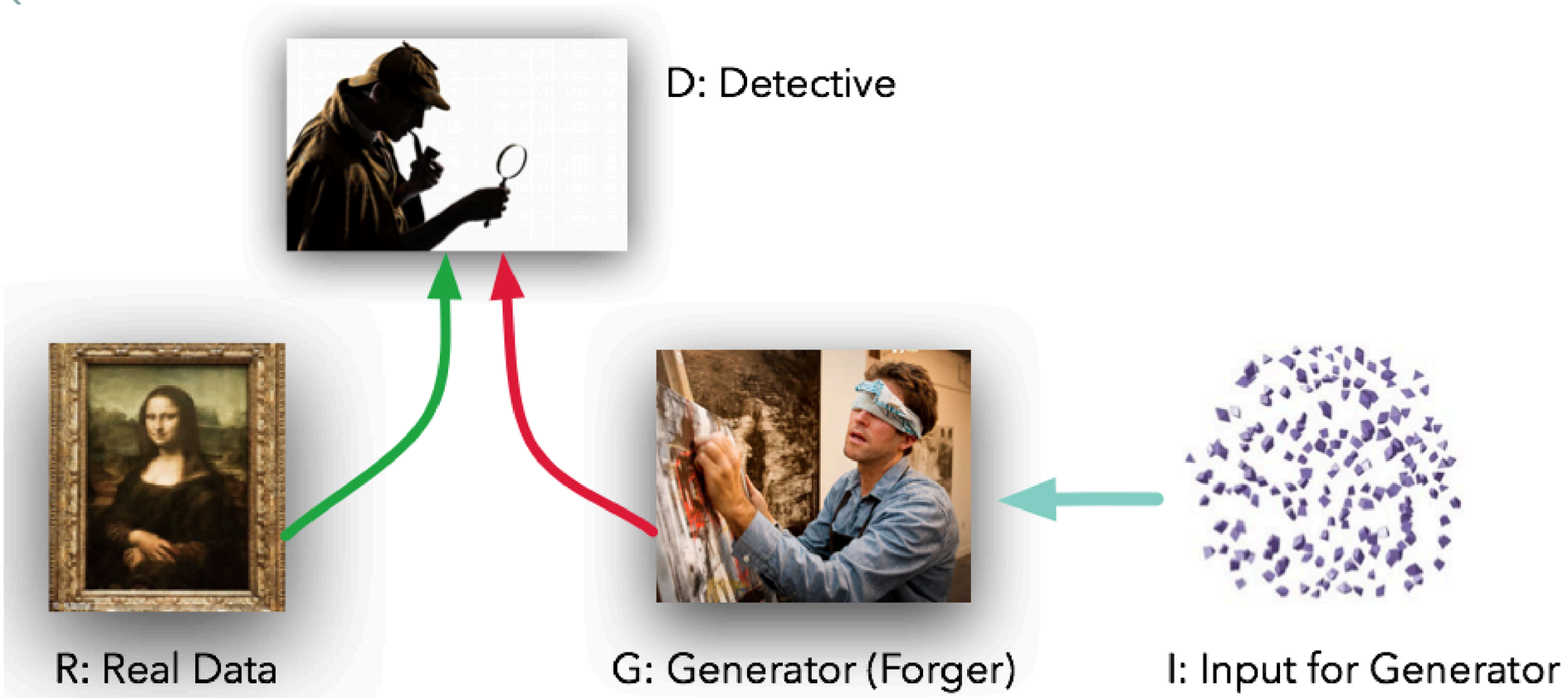
## Denoising diffusion models

- **Forward / noising process**
  - Sample data $p(\mathbf{x}_0)$ → turn to noise

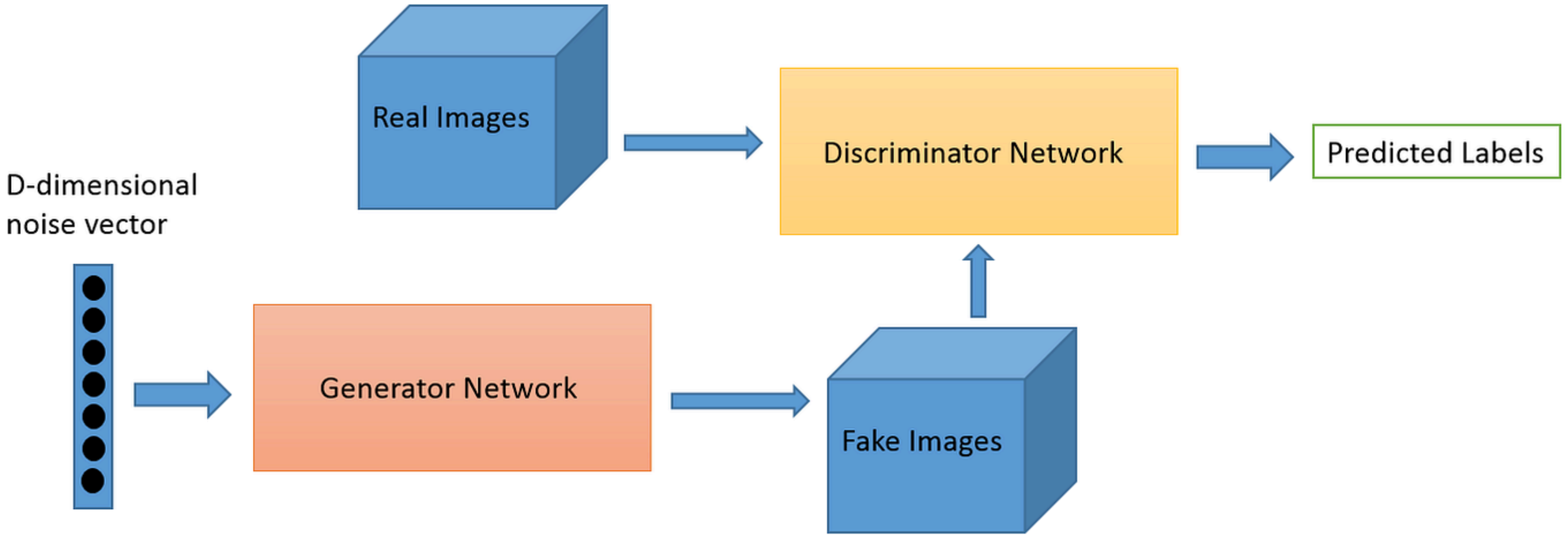



- **Reverse / denoising process**
  - Sample noise $p_T(\mathbf{x}_T)$ → turn into data

# GAN ARCHITECTURE

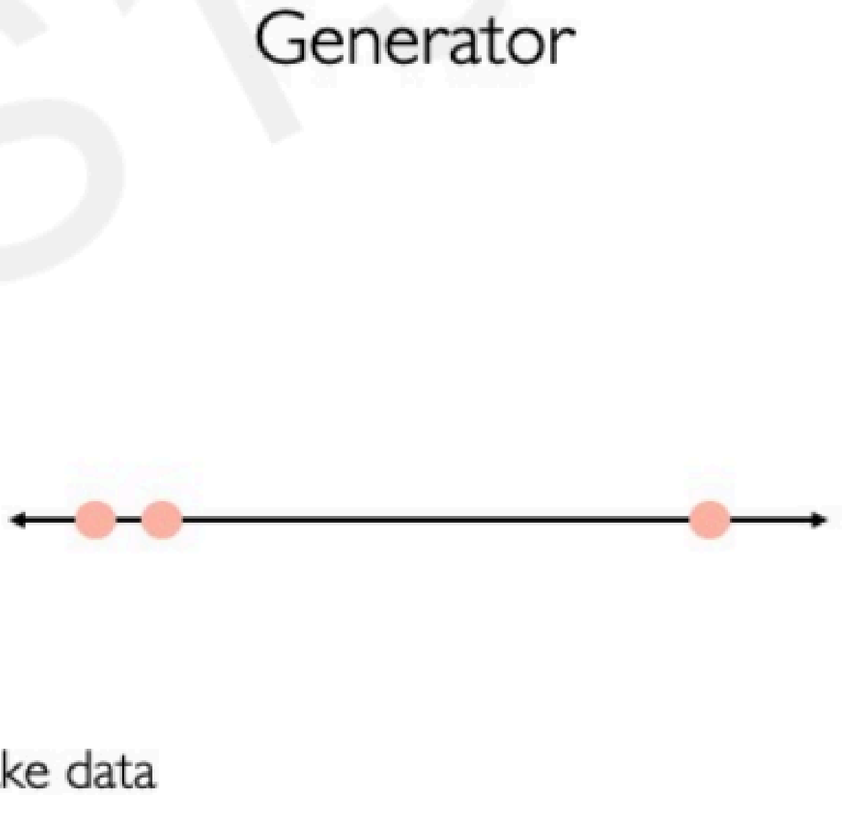


# GAN ARCHITECTURE



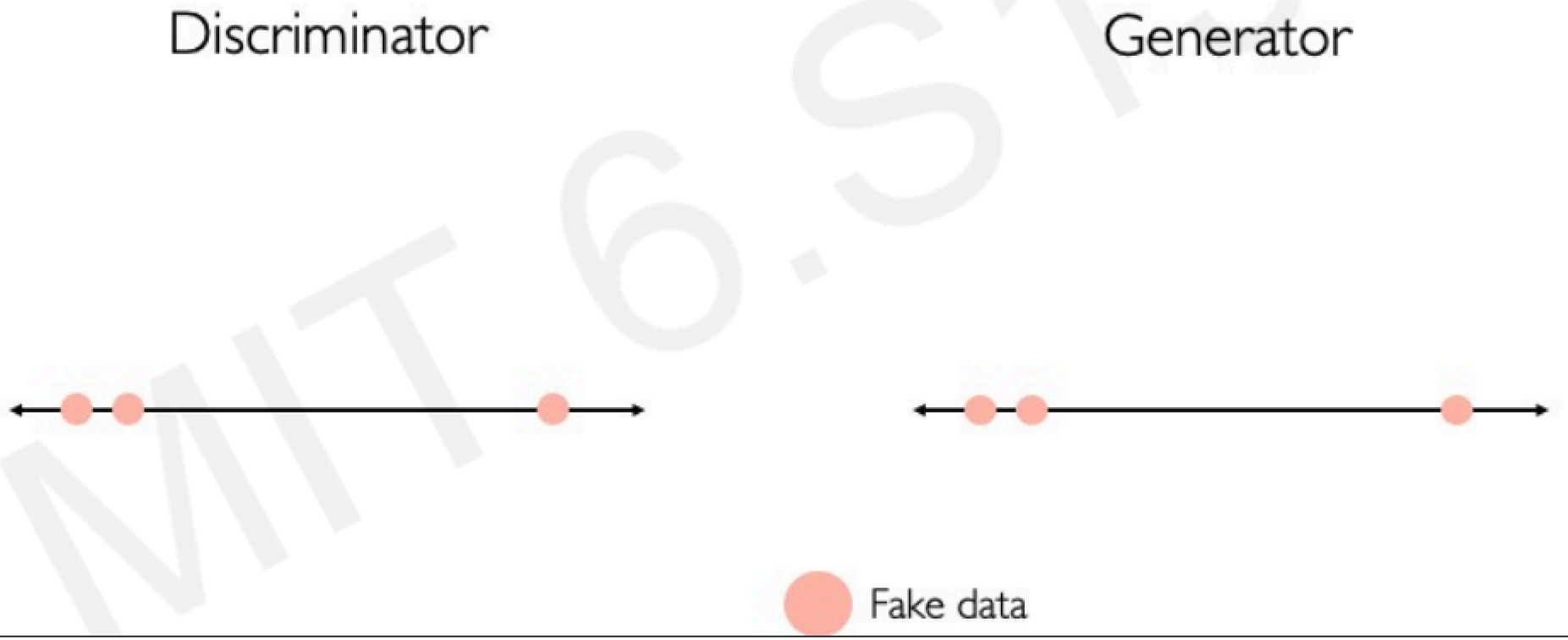
# Intuition behind GANs

**Generator** starts from noise to try to create an imitation of the data.



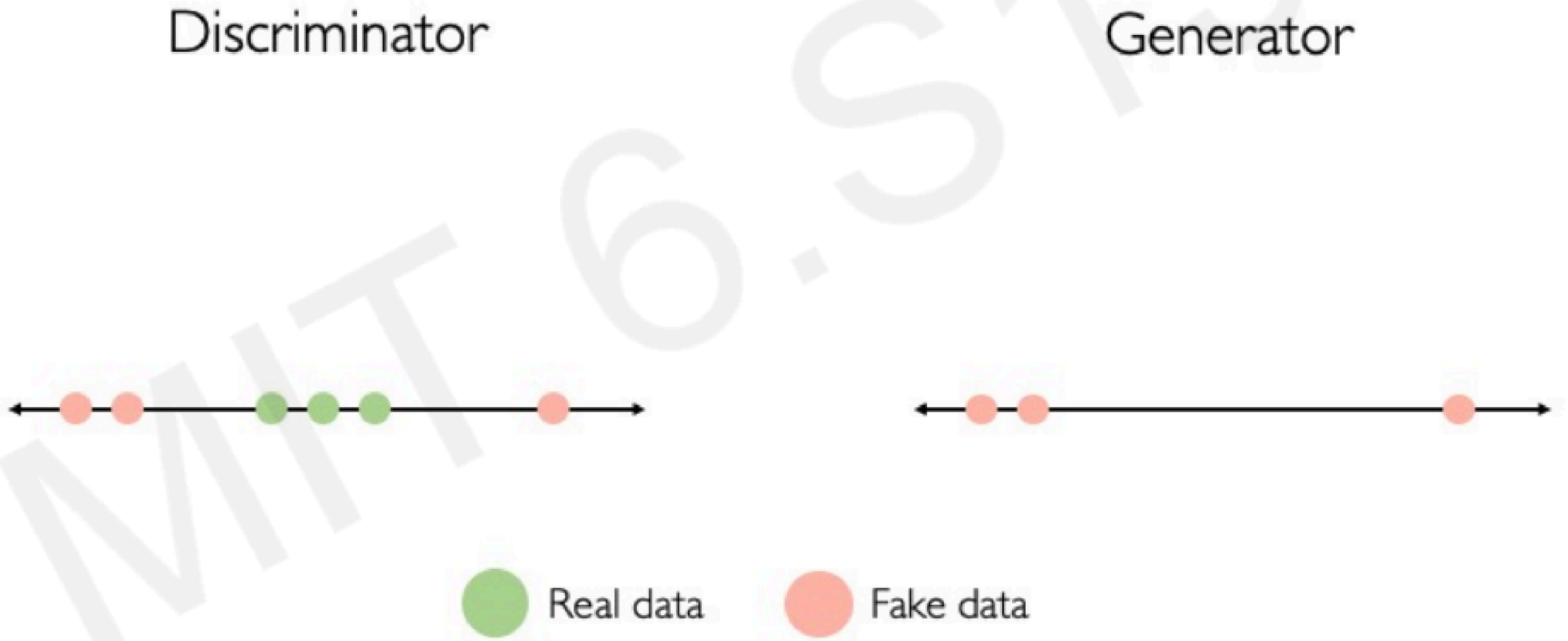
# Intuition behind GANs

**Discriminator** looks at both real data and fake data created by the generator.



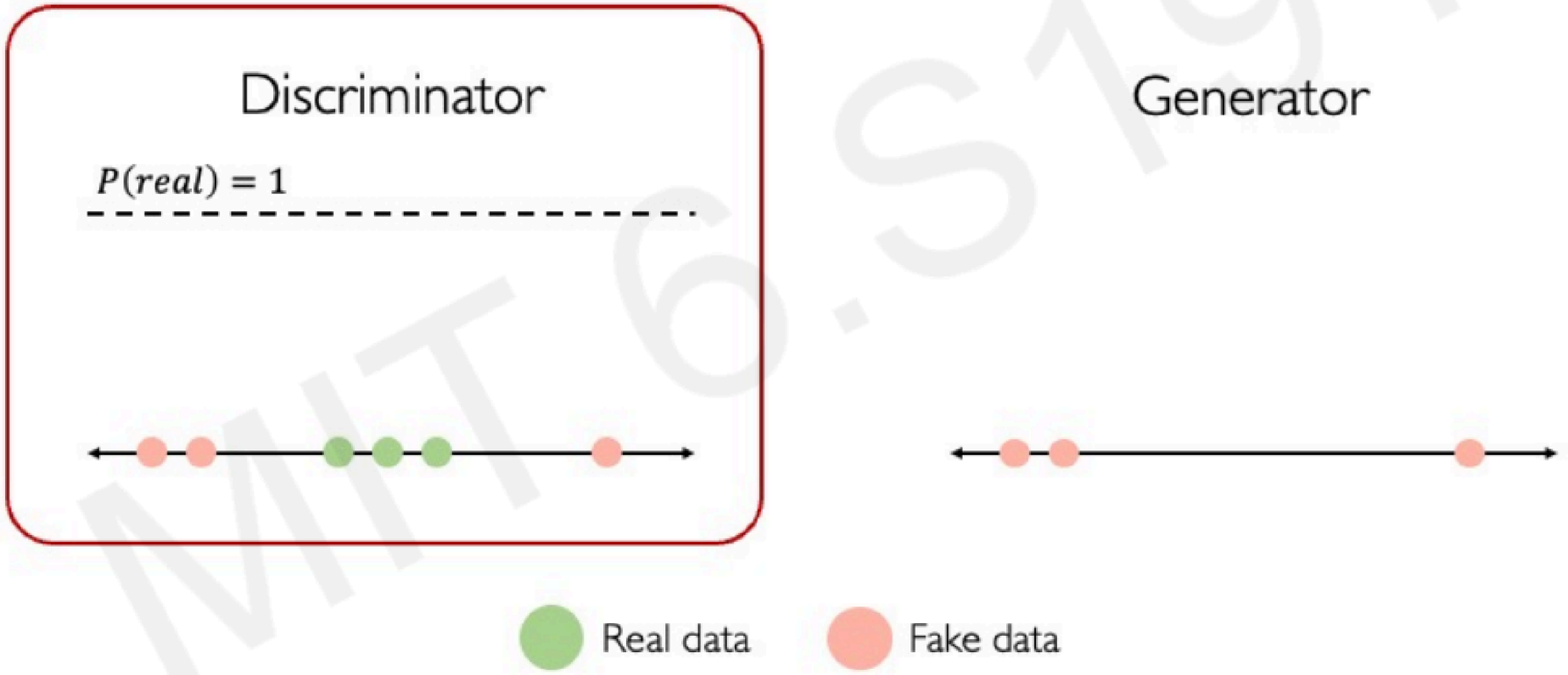
# Intuition behind GANs

**Discriminator** looks at both real data and fake data created by the generator.



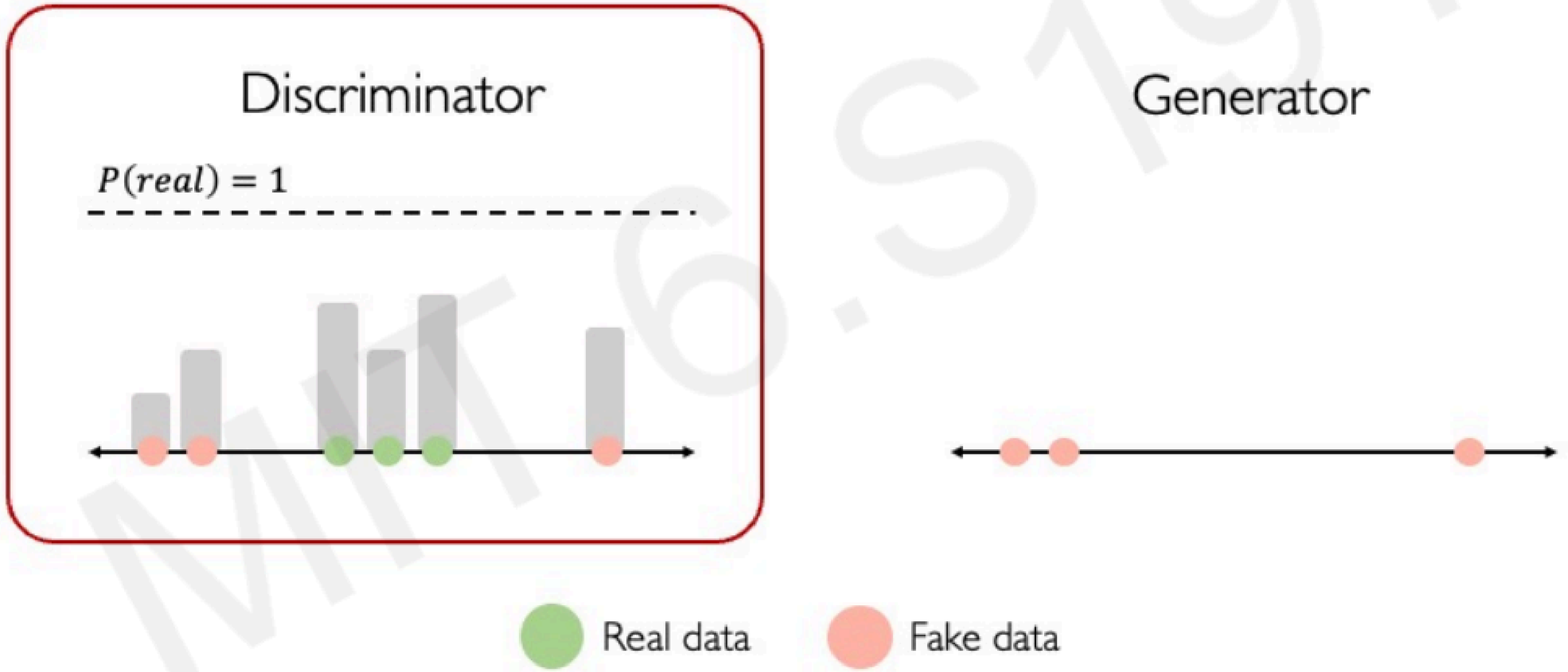
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



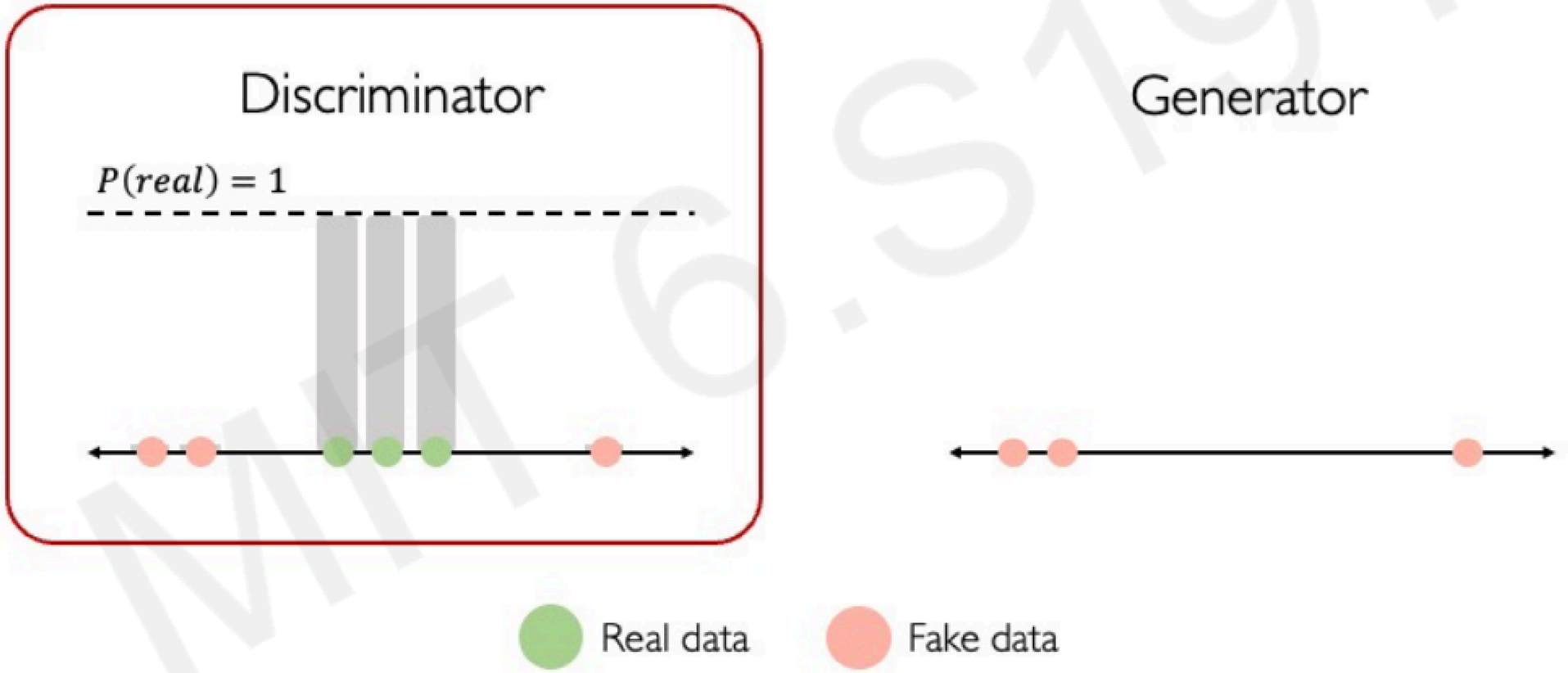
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



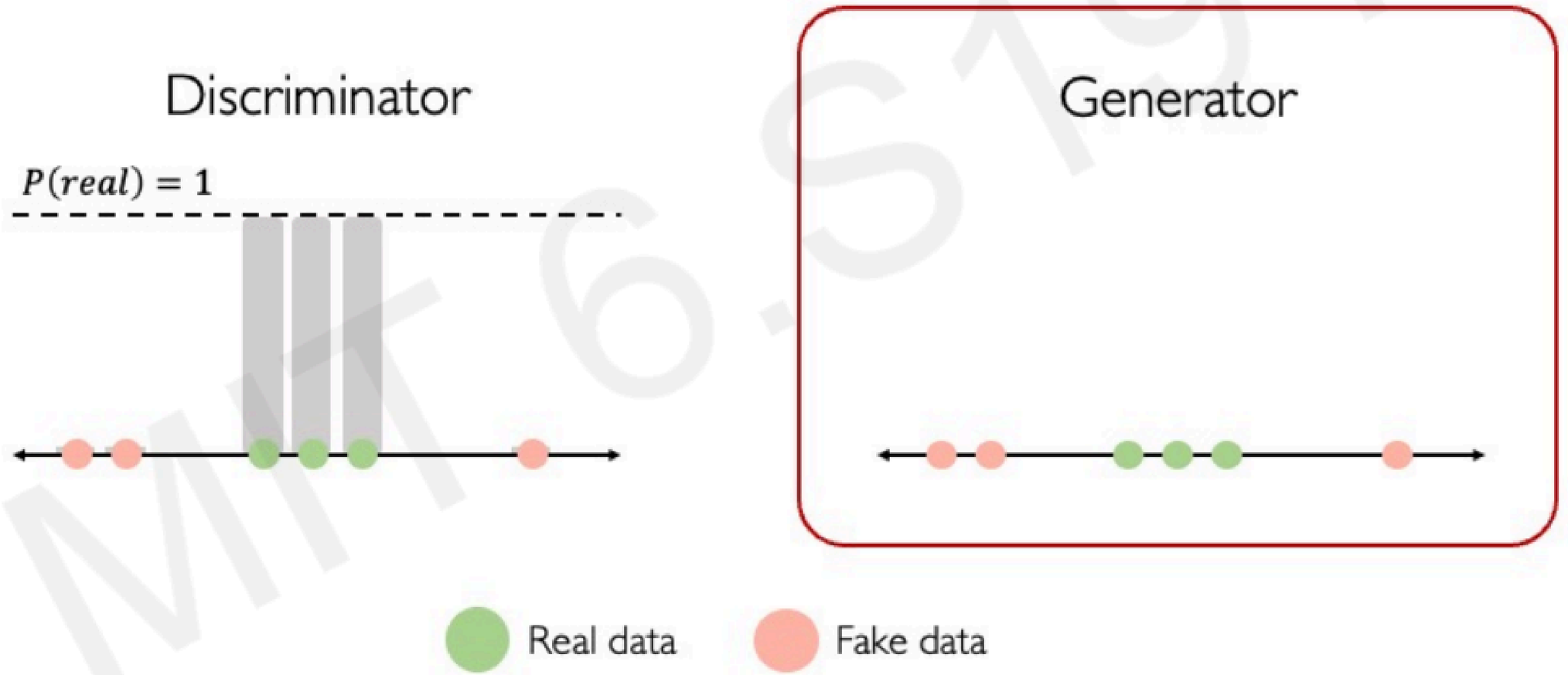
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



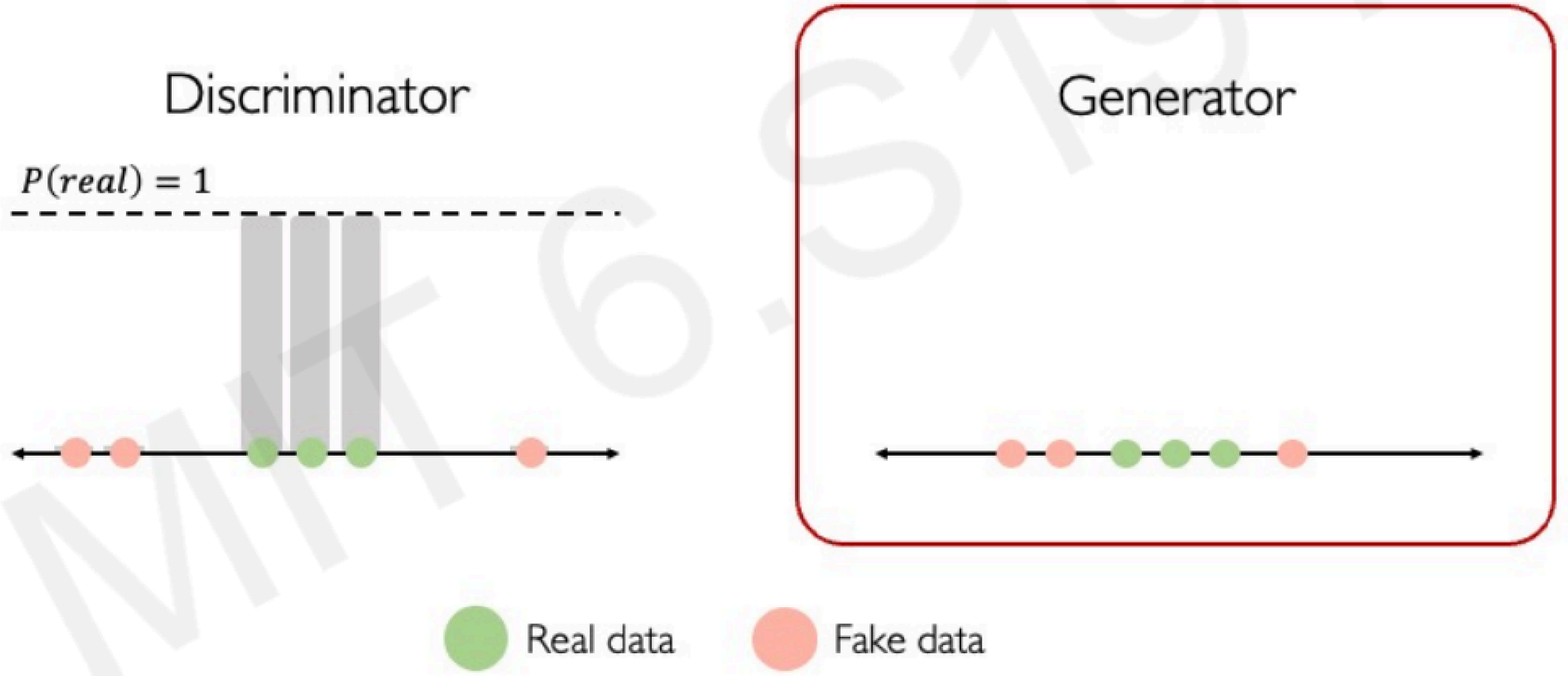
# Intuition behind GANs

**Generator** tries to improve its imitation of the data.



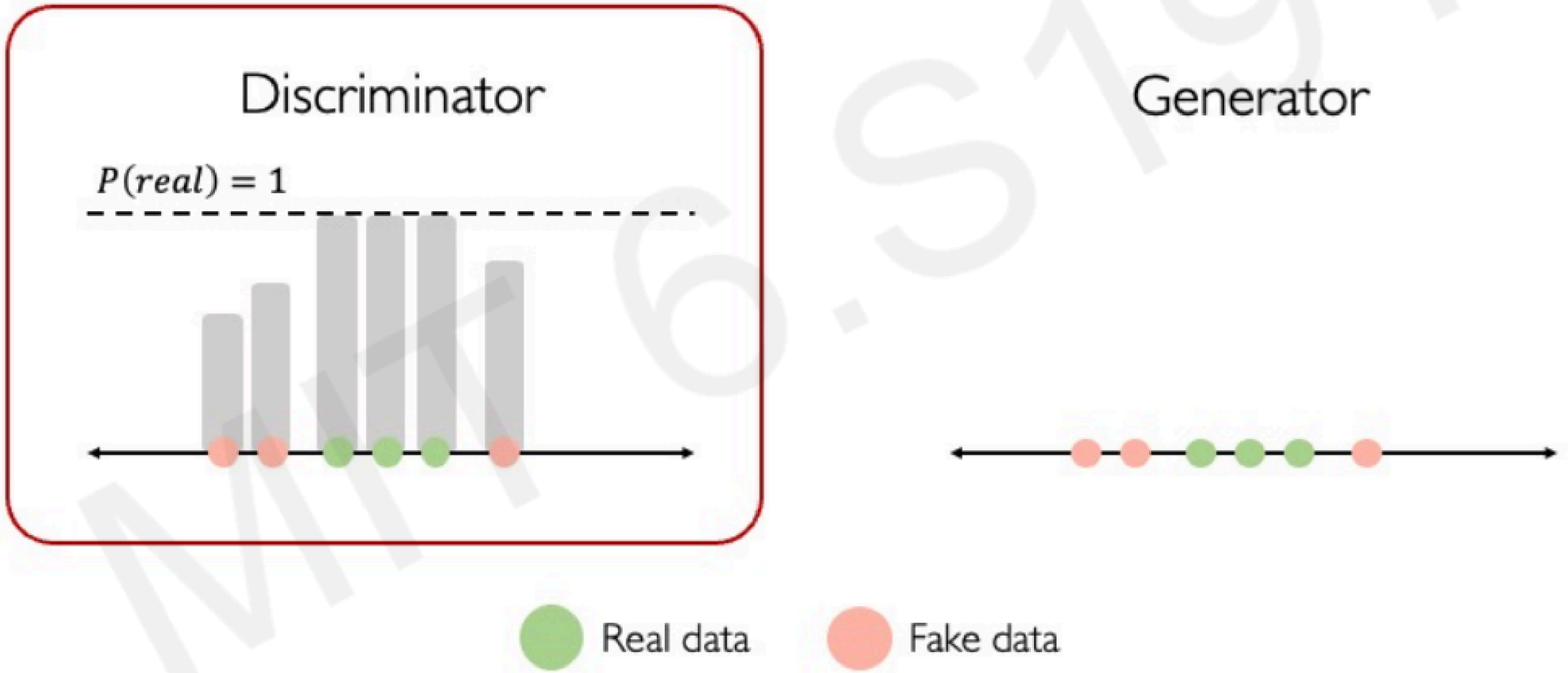
# Intuition behind GANs

**Generator** tries to improve its imitation of the data.



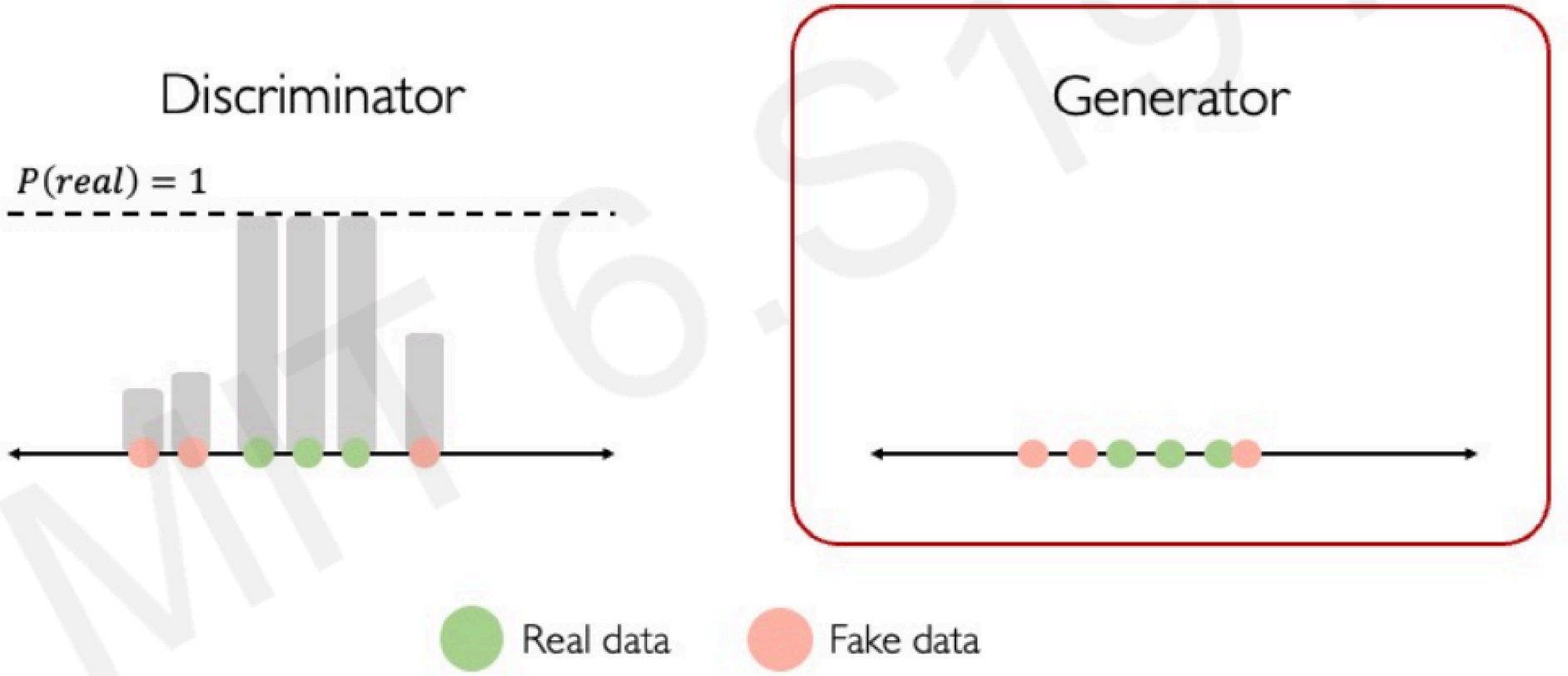
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



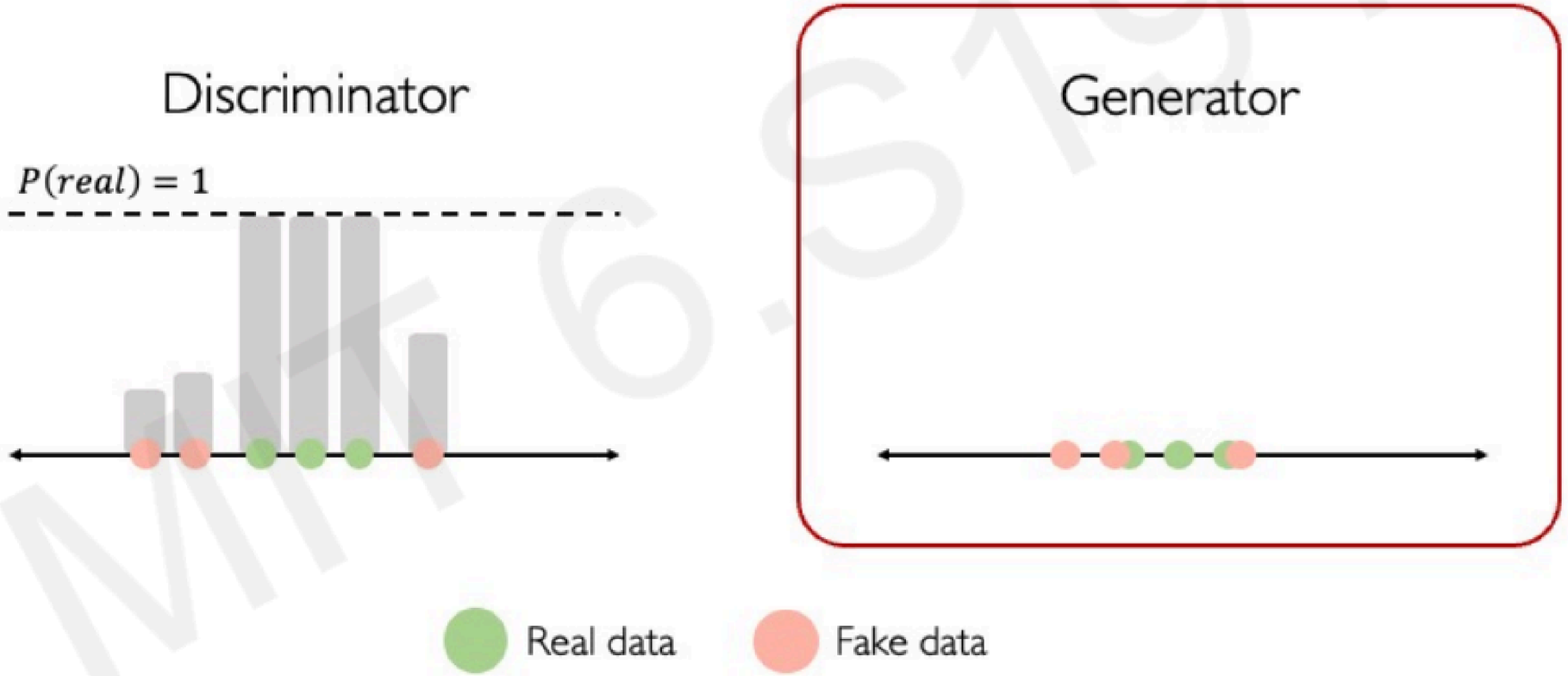
# Intuition behind GANs

**Generator** tries to improve its imitation of the data.



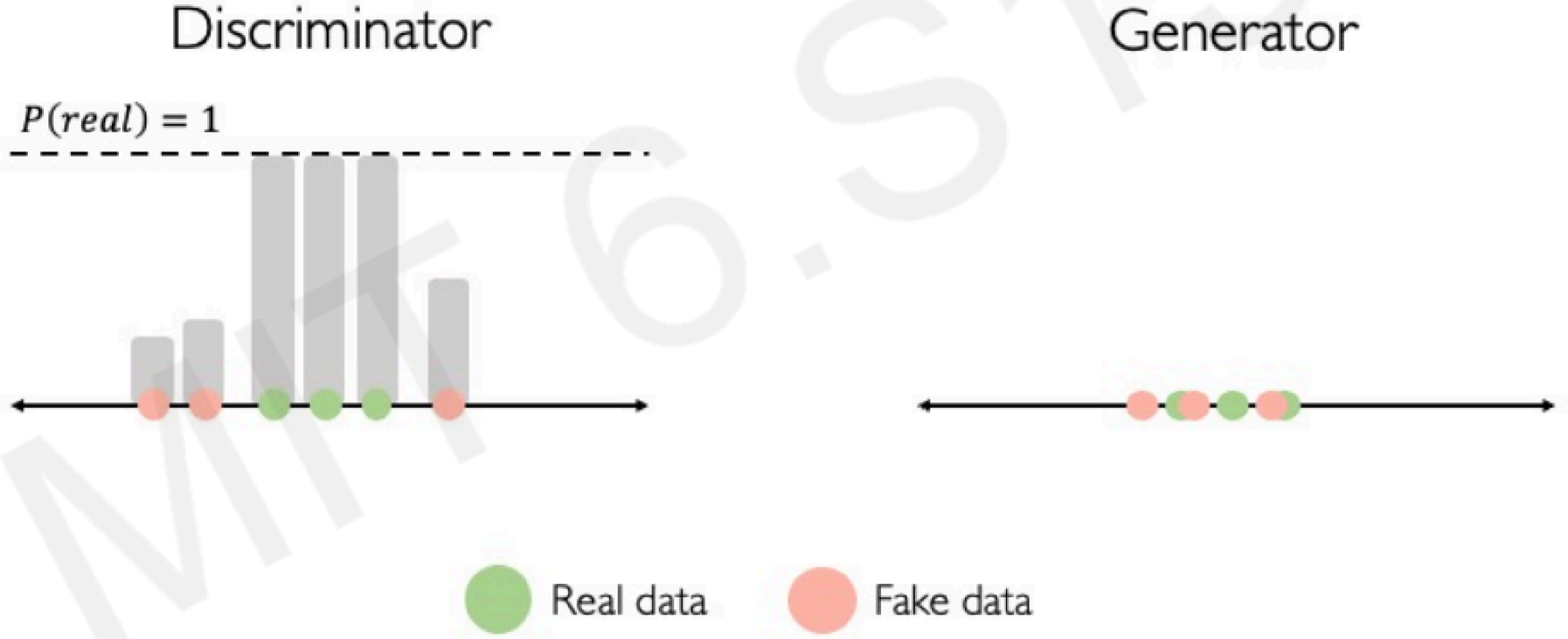
# Intuition behind GANs

**Generator** tries to improve its imitation of the data.

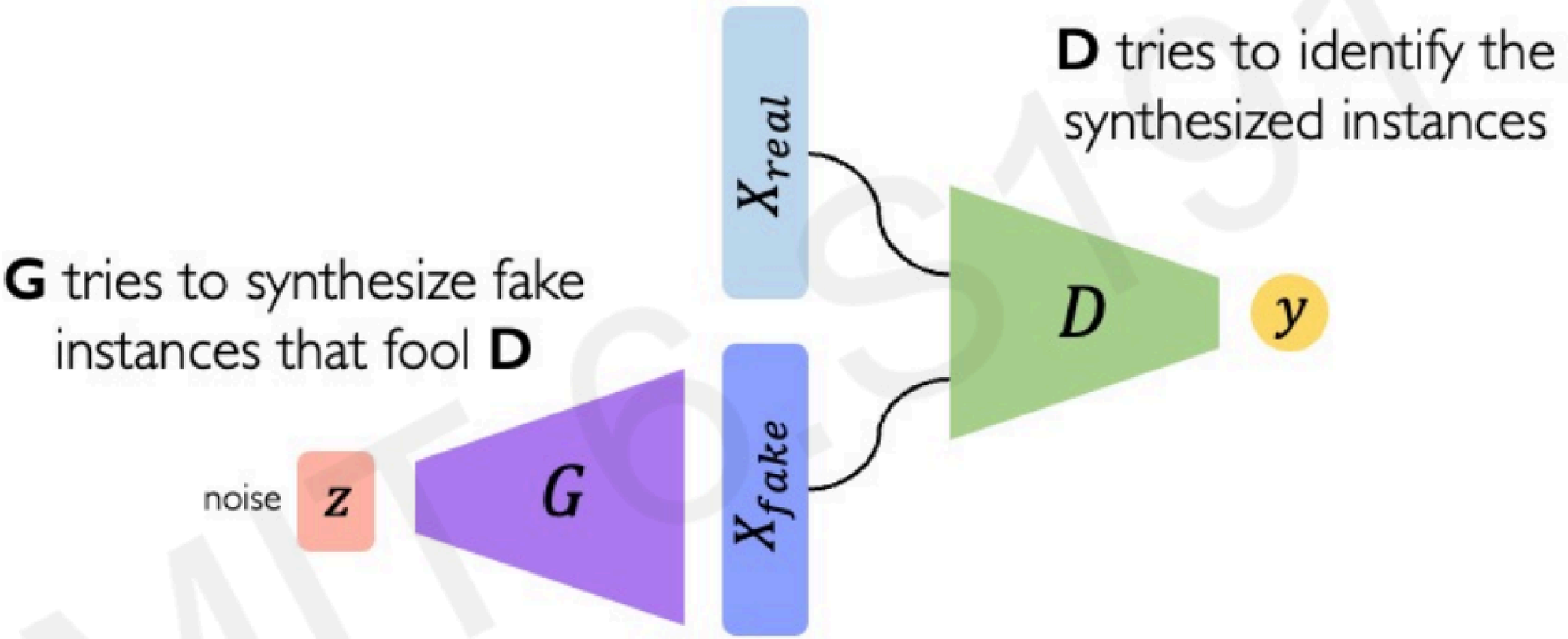


# Intuition behind GANs

**Discriminator** tries to identify real data from fakes created by the generator.
**Generator** tries to create imitations of data to trick the discriminator.



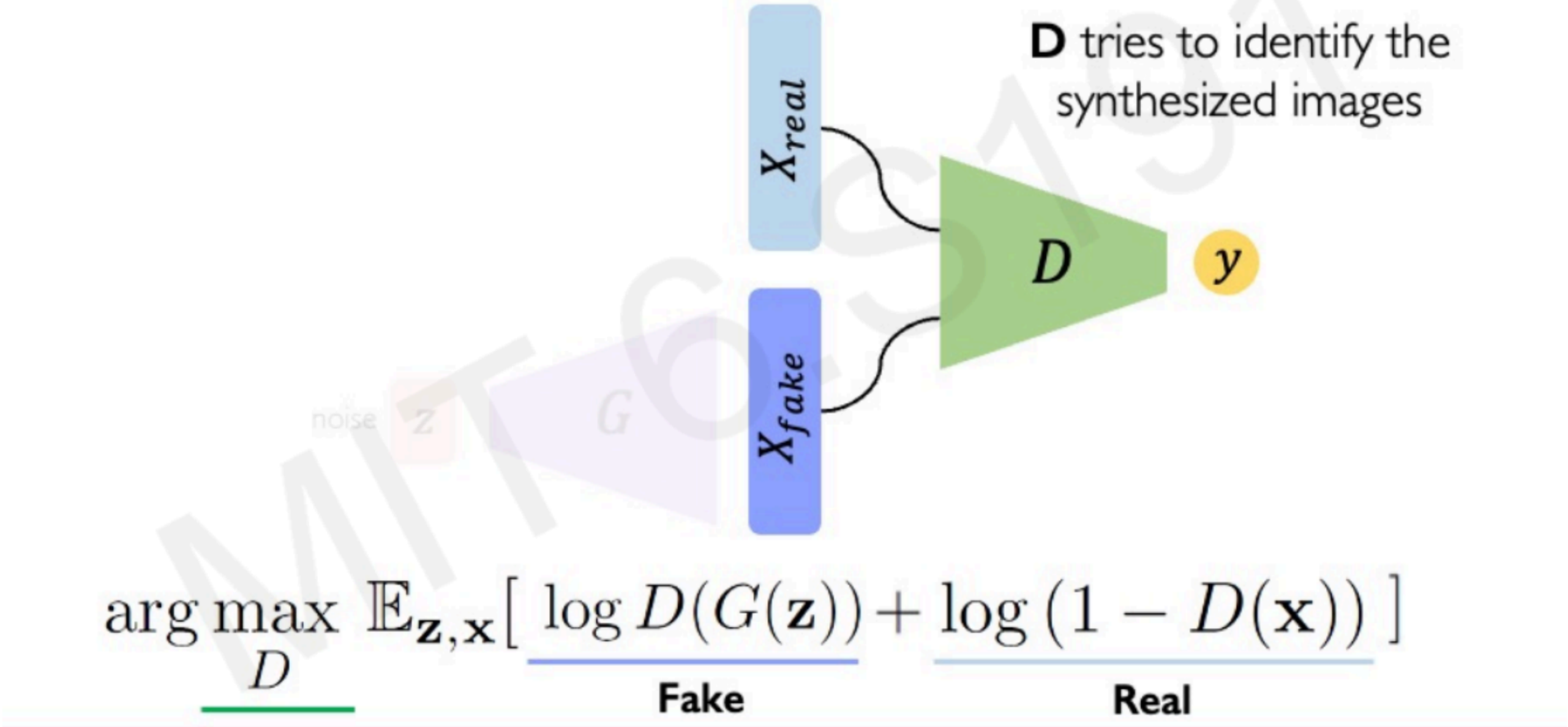
# Training GANs




**Training:** adversarial objectives for **D** and **G**

**Global optimum: G** reproduces the true data distribution

# LOSS FUNCTION

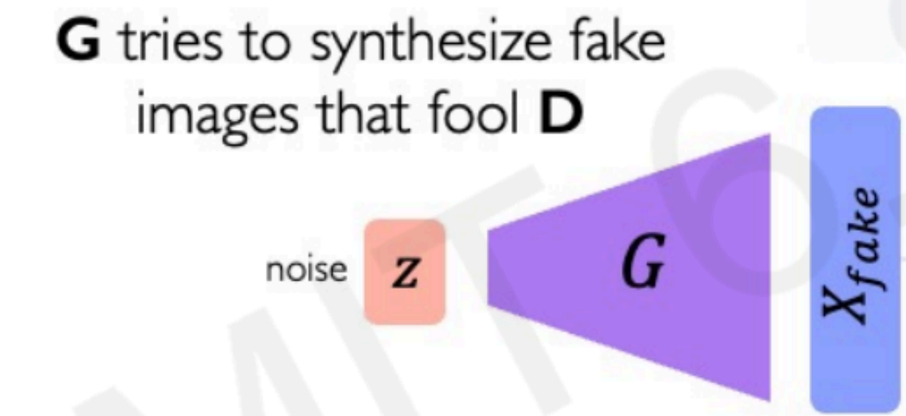



$$\arg\max_{D} \mathbb{E}_{\mathbf{z},\mathbf{x}}[\underbrace{\log D(G(\mathbf{z}))}_{\text{Fake}} + \underbrace{\log(1 - D(\mathbf{x}))}_{\text{Real}}]$$

D(x) is the probability of predicting real data as fake

So Discriminator tries to increase the probability of getting answers correct

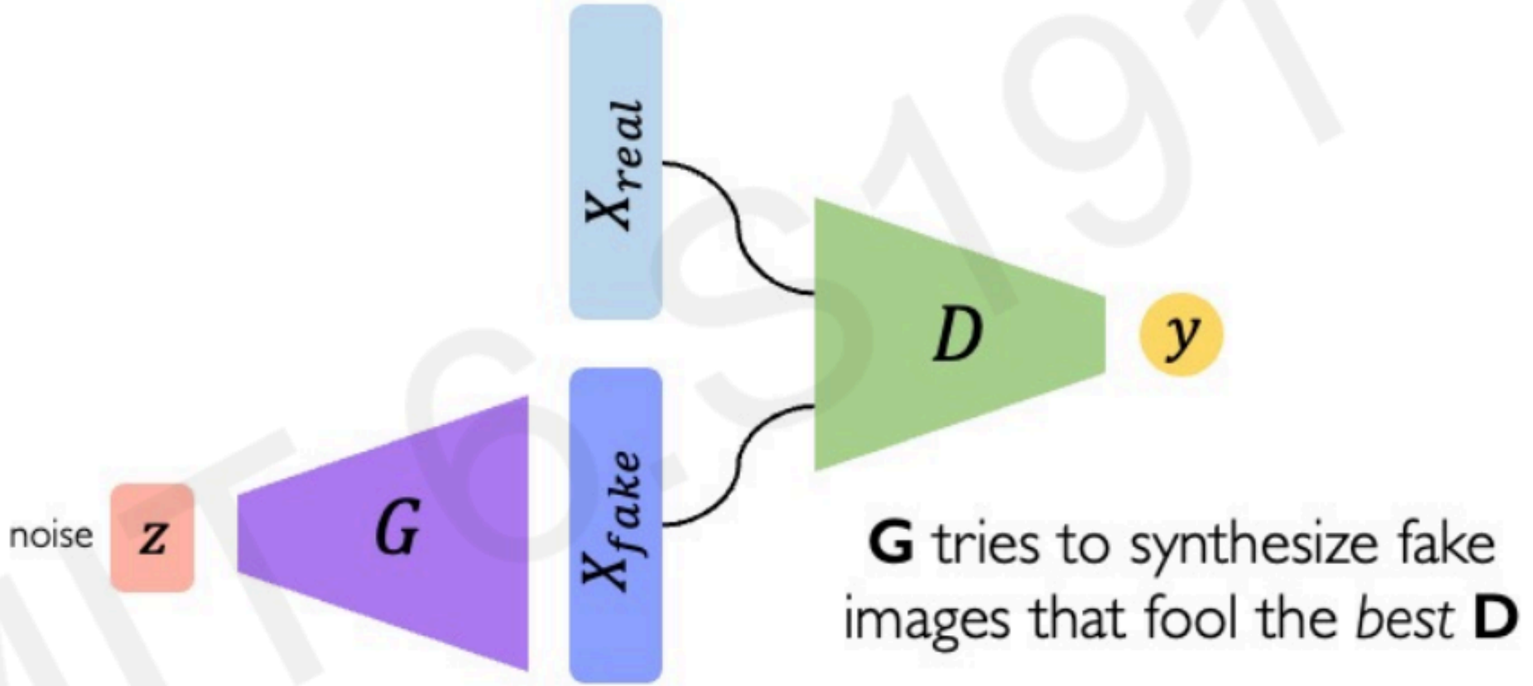
# LOSS FUNCTION




$$\arg\min_{G} \mathbb{E}_{\mathbf{z},\mathbf{x}}[\log D(G(\mathbf{z})) + \log(1 - D(\mathbf{x}))]$$

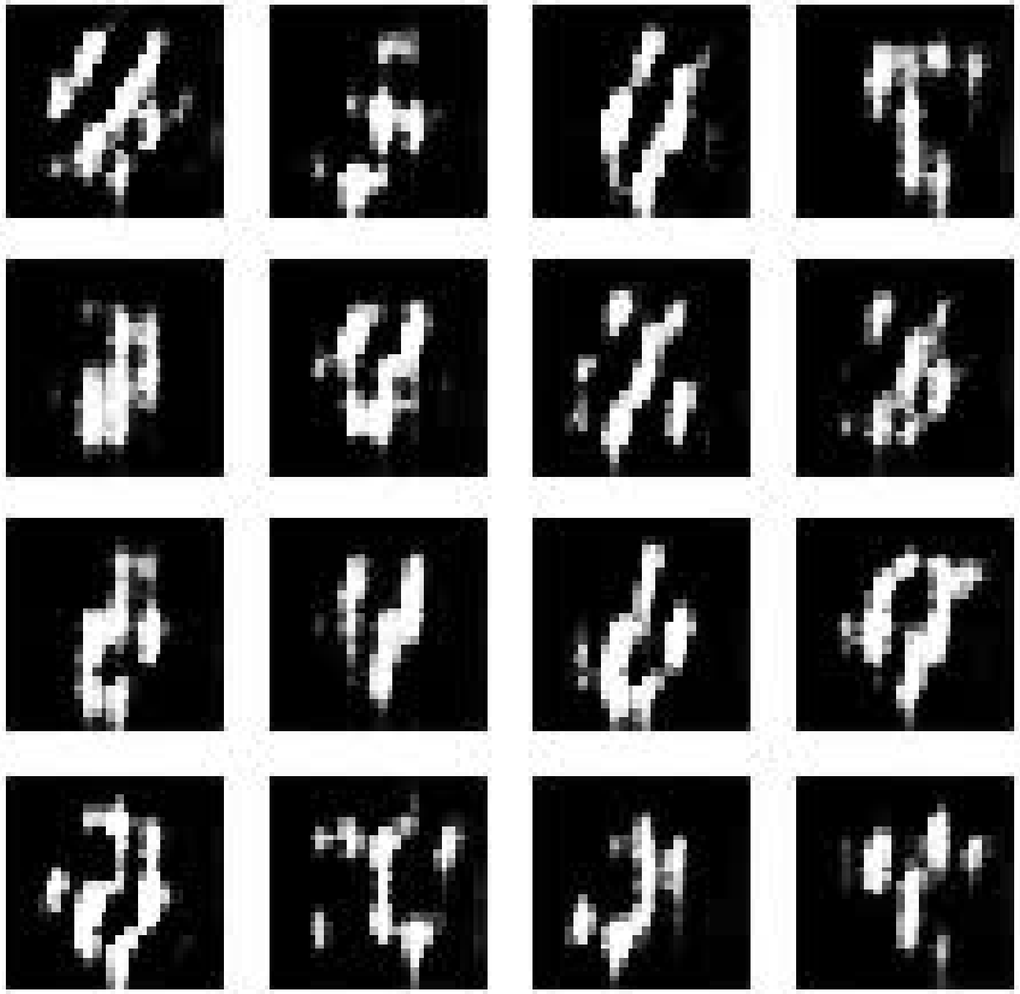
Generator tries to decrease the probability of getting answers correct

# LOSS FUNCTION




$$\arg\min_{G}\max_{D} \mathbb{E}_{\mathbf{z},\mathbf{x}}[\log D(G(\mathbf{z})) + \log(1 - D(\mathbf{x}))]$$

# GAN ON MNIST DATASET



# HOW CAN WE USE GEN AI MODELS ?

# API (APPLICATION PROGRAMMING INTERFACE)

# API KEYS

- Definition: An API key is a unique identifier used to authenticate a client or application when accessing an API. It helps to ensure that the person or system making a request to the API has permission to do so.
- Purpose: Security and authentication. It identifies who is making the request and often limits the number of requests or provides access to specific features.
- Example: When integrating with services like Google Maps API, OpenWeatherMap, or other third-party platforms, an API key is required to verify and track usage.
- Usage: Sent with API requests as part of the header, URL, or other mechanisms to verify the identity of the requester.

# INFERENCE API

- Definition: An inference API is typically used in the context of machine learning models. It allows users to send data (such as text, images, etc.) to a machine learning model hosted on a server, and in return, the model processes the data and sends back the prediction or result (inference).
- Purpose: Allows users to perform inference (prediction) using pre-trained models without needing to run the models locally. This is common with services like OpenAI's GPT models or Hugging Face's model hub.
- Example: A developer might send an image to an inference API that hosts a model for object detection. The API would return a response with the objects identified in the image.
- Usage: Inference APIs are used in AI/ML workflows where a pre-trained model performs a task such as text generation, image classification, or speech recognition.

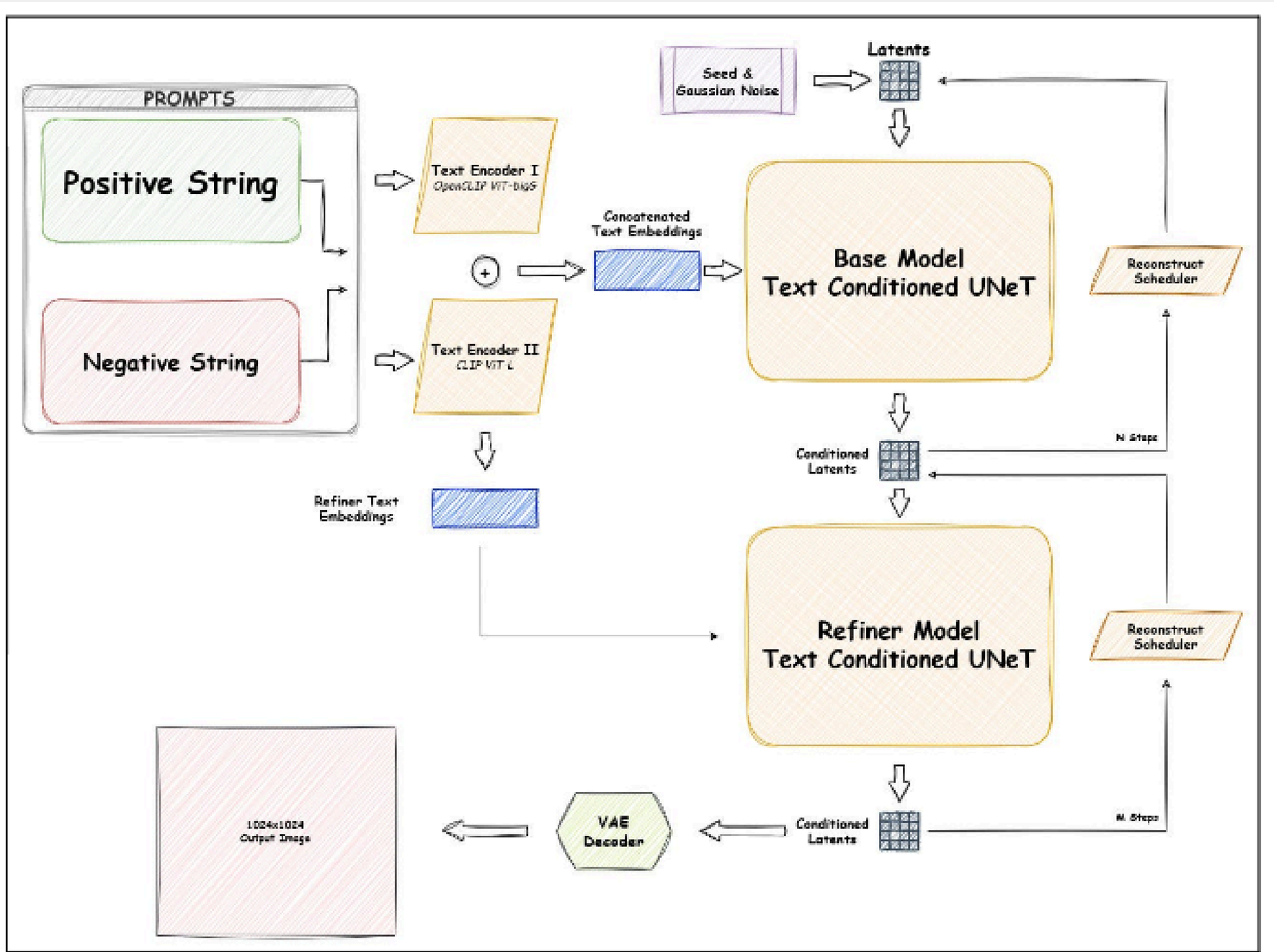
# GEMINI PRO

- Likely focuses on offering large, proprietary language models trained by Google, similar to ChatGPT or Bard. These models are fine-tuned for various tasks, such as general conversation, reasoning, search, and more.
- May offer advanced multimodal capabilities, integrating text with images, code, and potentially other types of data.
- Mostly proprietary models with access through Google's ecosystem (similar to how OpenAI integrates into Azure).

# HUGGING FACE

- Offers thousands of models, including open-source and community-developed ones. Hugging Face hosts not only large transformer-based models (like GPT, BERT, BLOOM, and T5) but also models specialized for tasks like sentiment analysis, translation, summarization, and more.
- Many of the models on Hugging Face are open-source and can be fine-tuned, customized, and deployed as needed.
- A diverse range of both small and large models, allowing for greater flexibility in choosing models based on performance needs and resource availability.

# STABLE DIFFUSION XL



**Prompts**: Inputs to the model consisting of a "Positive String" and a "Negative String", which guide the model on what to include or avoid in the generated image.

Text Encoder I & II: These are different encoders (e.g., OpenCLIP-VT/G and CLIP-VT/L) used to transform the text inputs into embeddings. The embeddings from these encoders are then concatenated to form a rich representation of the text input.

Refiner Text Embeddings: A refinement step that processes the concatenated embeddings to optimize or enhance the information they contain, making them more suitable for generating conditioned latents.

Seed & Gaussian Noise: Initial random inputs that, along with text embeddings, help in generating the initial latent representations of the image.

Base Model Text Conditioned UNeT: A UNeT-based model (a type of convolutional neural network that follows a U-shaped architecture, with an encoder for downsampling to capture context and a decoder for upsampling to precisely localize features) that takes the initial latents and the refined text embeddings to generate an initial version of the conditioned latents. This step includes a "Reconstruct Scheduler" that determines how the latent space is iteratively refined across several steps.

Refiner Model Text Conditioned UNeT: An additional refinement stage using a UNeT model that further processes the conditioned latents to enhance the final image output, involving multiple iterations as governed by another "Reconstruct Scheduler".

VAE Decoder: A variational autoencoder (VAE) decoder that converts the final conditioned latents into the pixel space, resulting in the generation of the final output image, typically at a high resolution like 1024x1024.

1. No Automatic Permanent Storage:
By default, most large language models (like GPT) are stateless. This means the prompts and responses you provide are processed in real-time, but they are not stored after the session unless there's a specific mechanism in place (like logging, monitoring, or session history).
unless specified, your prompts are not stored beyond conversation. The model doesn't inherently remember prompts across different sessions.
2. Temporary Retention (Session-Based):
During a session, prompts may be retained temporarily to maintain the flow of conversation(contextual token retention). This allows models to provide context-aware responses based on the earlier part of the conversation.
Once the session ends, the context is typically discarded.

1. Vector Database Integration:
What is a Vector Database? A vector database stores embeddings (vector representations) of text data, which allows for efficient similarity searches. Popular vector databases include Pinecone, Weaviate, Milvus, and FAISS.
Storing Conversations: You can store each message in the conversation as a vector embedding, making it searchable later based on semantic similarity. To store conversations in a vector database, follow these steps:
Convert text to vectors: Use an embedding model (like OpenAI's text-embedding-ada-002, Sentence-BERT, etc.) to convert messages into vector representations.
Store vectors in the database: Push these embeddings to a vector database along with any metadata, such as the timestamp, user ID, or the context of the conversation.
Search and retrieve: Later, you can search the database for semantically similar conversations or specific topics by querying with vector embeddings of new input.
2. Local Storage vs. Cloud:
Local Storage: You can use an open-source vector database like FAISS or Milvus on your local machine. This gives you full control over your data without any external dependencies but may limit scalability or require more manual setup.
Cloud Storage: Cloud-based solutions like Pinecone or Weaviate offer managed services that scale easily and handle infrastructure, allowing you to focus on building features without worrying about database management.
3. How to Implement:
Preprocessing:
Use an embedding model to generate vector representations of text.
Database Storage:
Choose your vector database (FAISS, Milvus, Pinecone, etc.) and set up connection drivers for your local or cloud instance.
Search:
When a new query comes in, convert it into an embedding and search against stored embeddings in the vector database to retrieve relevant responses or conversation history.
Example Workflow:
User Input → 2. Convert Input to Vector → 3. Store in Vector Database (Local or Cloud) → 4. Search → 5. Respond Based on Similar Conversations.

# RESOURCES

- MIT Deep learning course lecture 4-GANs and VAEs
- Google AI Developer- Gemini API key, prompt playground
- Render Realm Videos
- Hugging face-model cards, APi key ,interference API
- Krish Naik - Introduction to Hugging face open source models
- Project IDX-Image analyzer

# THANK YOU