Normalizing Flows

$$\mathbf{z}_m := \mathbf{f}_\theta^m \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) = \mathbf{f}_\theta^m(\mathbf{f}_\theta^{m-1}(\cdots(\mathbf{f}_\theta^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_\theta(\mathbf{z}_0)$$

Presented by
Ahmet Sarigun

Simple Prior to Complex Data Distributions

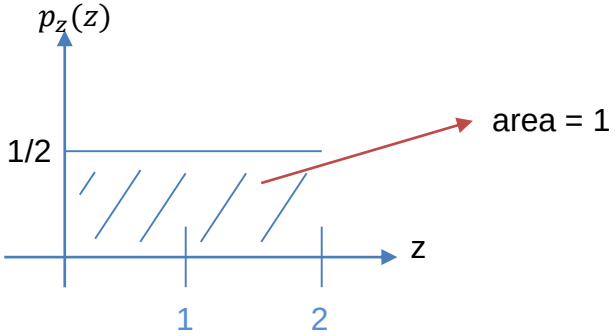
- Desirable properties of any model distribution:
  - Analytic density
  - Easy-to-sample
- Many simple distributions satisfy the above properties e.g., Gaussian, uniform distributions
- Unfortunately, data distributions could be much more complex (multi-modal)
- **Key idea**: Map simple distributions (easy to sample and evaluate densities) to complex distributions (learned via data) using **change of variables**.

Slide by Stefano Ermon, Aditya Grover

Change of Variables formula

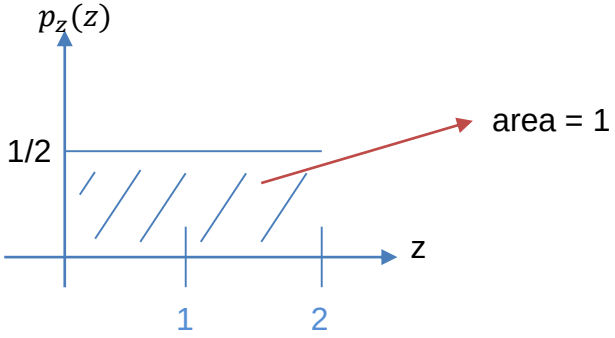
- Let Z be a uniform random variable $\mathcal{U}[0, 2]$ with density p_Z. What is $p_Z(1)$? $\frac{1}{2}$
- Let $X = 4Z$, and let p_X be its density. What is $p_X(4)$?
- $p_X(4) = p(X = 4) = p(4Z = 4) = p(Z = 1) = p_Z(1) = 1/2$ **No**
- Clearly, X is uniform in $[0, 8]$, so $p_X(4) = 1/8$

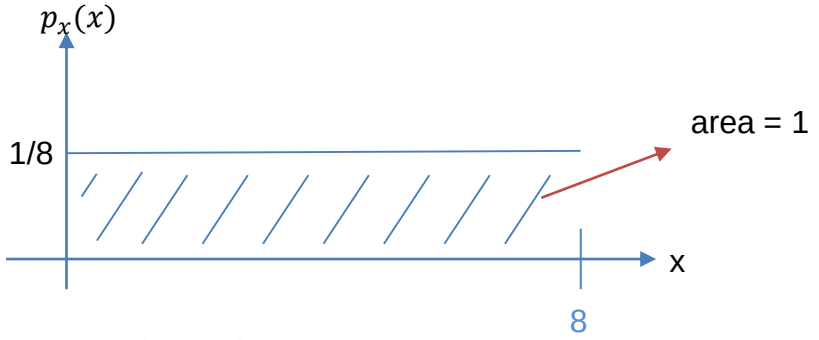
Change of Variables formula



- Let Z be a uniform random variable $\mathcal{U}[0, 2]$ with density p_Z. What is $p_Z(1)$? $\frac{1}{2}$
- Let $X = 4Z$, and let p_X be its density. What is $p_X(4)$?
- $p_X(4) = p(X = 4) = p(4Z = 4) = p(Z = 1) = p_Z(1) = 1/2$ **No**
- Clearly, X is uniform in $[0, 8]$, so $p_X(4) = 1/8$

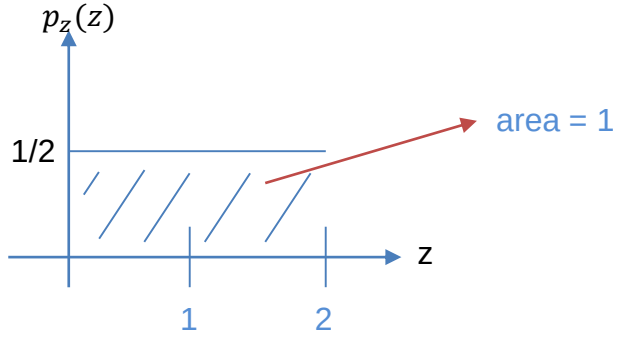
Change of Variables formula

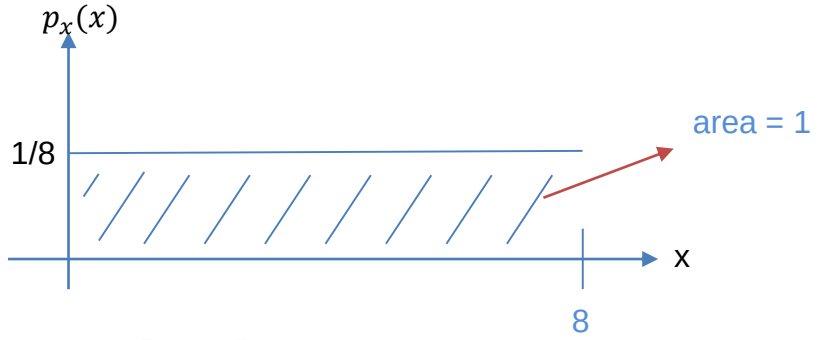




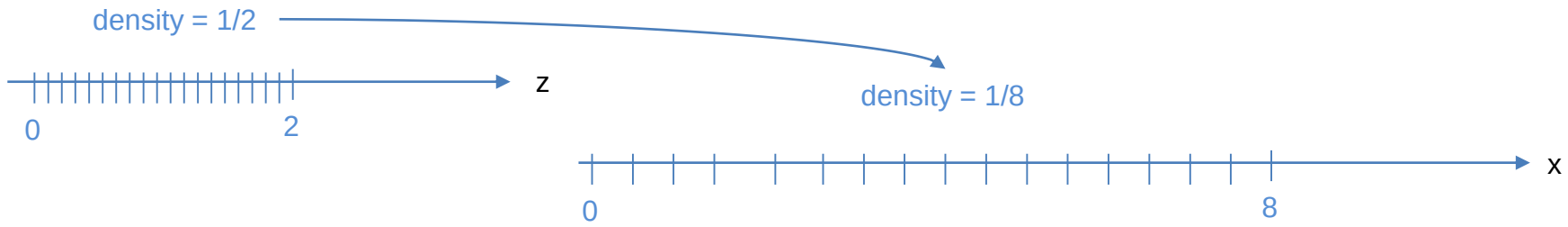
- Let Z be a uniform random variable $\mathcal{U}[0, 2]$ with density p_Z. What is $p_Z(1)$? $\frac{1}{2}$
- Let $X = 4Z$, and let p_X be its density. What is $p_X(4)$?
- $p_X(4) = p(X = 4) = p(4Z = 4) = p(Z = 1) = p_Z(1) = 1/2$ **No**
- Clearly, X is uniform in $[0, 8]$, so $p_X(4) = 1/8$

Change of Variables formula





- Let Z be a uniform random variable $\mathcal{U}[0, 2]$ with density p_Z. What is $p_Z(1)$? $\frac{1}{2}$
- Let $X = 4Z$, and let p_X be its density. What is $p_X(4)$?
- $p_X(4) = p(X = 4) = p(4Z = 4) = p(Z = 1) = p_Z(1) = 1/2$ **No**
- Clearly, X is uniform in $[0, 8]$, so $p_X(4) = 1/8$



Change of Variables formula

- **Change of variables (1D case)**: If $X = f(Z)$ and $f(\cdot)$ is monotone with inverse $Z = f^{-1}(X) = h(X)$, then:

$$p_X(x) = p_Z(h(x))|h'(x)|$$

- Previous example: If $X = 4Z$ and $Z \sim \mathcal{U}[0, 2]$, what is $p_X(4)$?
- Note that $h(X) = X/4$
- $p_X(4) = p_Z(1)h'(4) = 1/2 \times 1/4 = 1/8$

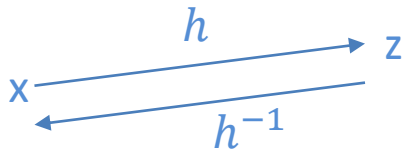
Change of Variables formula

- **Change of variables (1D case)**: If $X = f(Z)$ and $f(\cdot)$ is monotone with inverse $Z = f^{-1}(X) = h(X)$, then:

$$p_X(x) = p_Z(\underbrace{h(x)}_{z})|h'(x)|$$

- Previous example: If $X = 4Z$ and $Z \sim \mathcal{U}[0, 2]$, what is $p_X(4)$?
- Note that $h(X) = X/4$
- $p_X(4) = p_Z(1)h'(4) = 1/2 \times 1/4 = 1/8$

Change of Variables formula



- **Change of variables (1D case)**: If $X = f(Z)$ and $f(\cdot)$ is monotone with inverse $Z = f^{-1}(X) = h(X)$, then:

$$p_X(x) = p_Z(\underbrace{h(x)}_{z})|h'(x)|$$

- Previous example: If $X = 4Z$ and $Z \sim \mathcal{U}[0, 2]$, what is $p_X(4)$?
- Note that $h(X) = X/4$
- $p_X(4) = \underbrace{p_Z(1)}_{1/2}\underbrace{h'(4)}_{1/4} = 1/2 \times 1/4 = 1/8$

Geometry: Determinants and volumes

- Let Z be a uniform random vector in $[0, 1]^n$
- Let $X = AZ$ for a square invertible matrix A, with inverse $W = A^{-1}$. How is X distributed?
- Geometrically, the matrix A maps the unit hypercube $[0, 1]^n$ to a parallelotope
- Hypercube and parallelotope are generalizations of square/cube and parallelogram/parallelopiped to higher dimensions

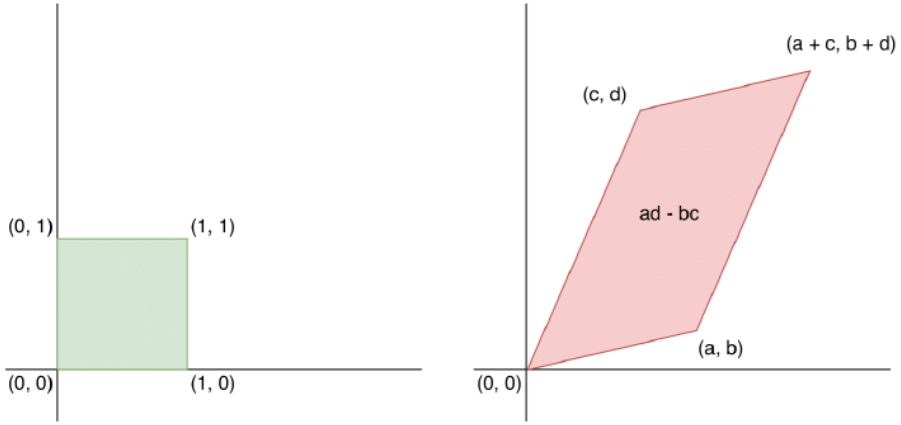


Figure: The matrix $A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ maps a unit square to a parallelogram

Slide by Stefano Ermon, Aditya Grover

Geometry: Determinants and volumes

- Let Z be a uniform random vector in $[0, 1]^n$
- Let $X = AZ$ for a square invertible matrix A, with inverse $W = A^{-1}$. How is X distributed?
- Geometrically, the matrix A maps the unit hypercube $[0, 1]^n$ to a parallelotope
- Hypercube and parallelotope are generalizations of square/cube and parallelogram/parallelopiped to higher dimensions

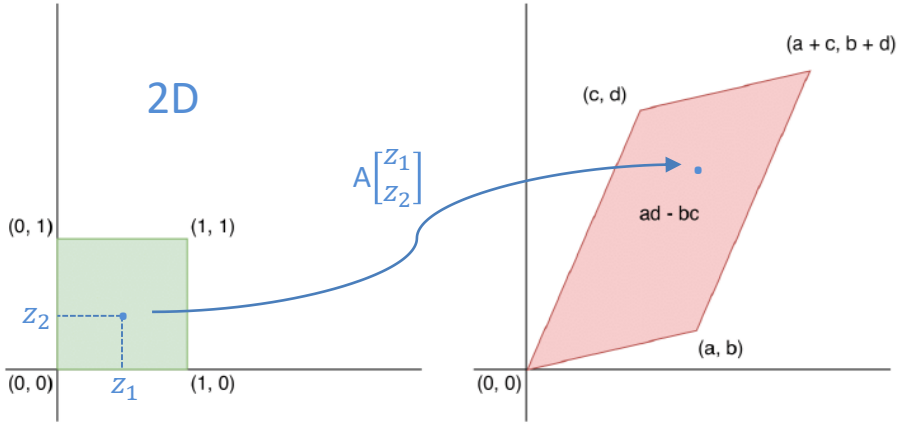
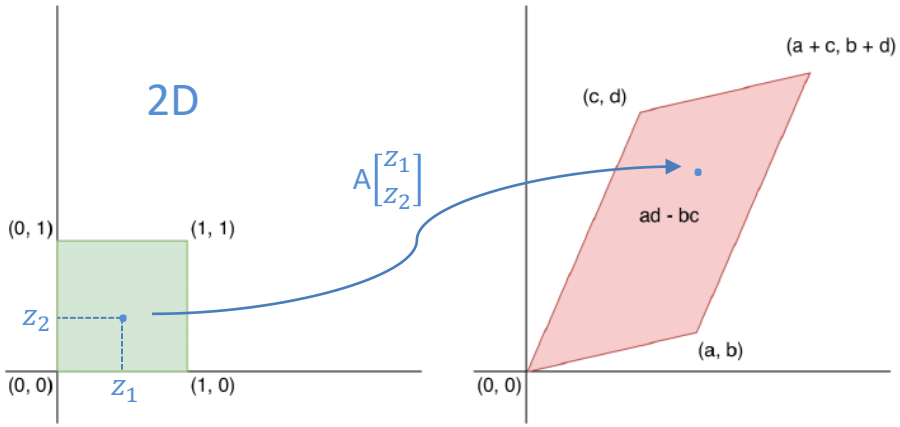


Figure: The matrix $A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ maps a unit square to a parallelogram

Slide by Stefano Ermon, Aditya Grover

Geometry: Determinants and volumes

- Let Z be a uniform random vector in $[0, 1]^n$
- Let $X = AZ$ for a square invertible matrix A, with inverse $W = A^{-1}$. How is X distributed?
- Geometrically, the matrix A maps the unit hypercube $[0, 1]^n$ to a parallelotope
- Hypercube and parallelotope are generalizations of square/cube and parallelogram/parallelopiped to higher dimensions



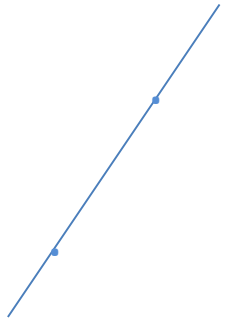


Figure: The matrix $A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ maps a unit square to a parallelogram

Geometry: Determinants and volumes

- The volume of the parallelotope is equal to the determinant of the transformation A

$$\det(A) = \det \begin{pmatrix} a & c \\ b & d \end{pmatrix} = ad - bc$$

- X is uniformly distributed over the parallelotope. Hence, we have

$$\begin{aligned} p_X(\mathbf{x}) &= p_Z(W\mathbf{x}) \left|\det(W)\right| \\ &= p_Z(W\mathbf{x}) \,/\, \left|\det(A)\right| \end{aligned}$$

Generalized change of variables

- For linear transformations specified via A, change in volume is given by the determinant of A
- For non-linear transformations $\mathbf{f}(\cdot)$, the *linearized* change in volume is given by the determinant of the Jacobian of $\mathbf{f}(\cdot)$.
- **Change of variables (General case)**: The mapping between Z and X, given by $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$, is invertible such that $X = \mathbf{f}(Z)$ and $Z = \mathbf{f}^{-1}(X)$.

$$p_X(\mathbf{x}) = p_Z\left(\mathbf{f}^{-1}(\mathbf{x})\right) \left| \det \left(\frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$

- Note 1: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension. For example, if $\mathbf{x} \in \mathbb{R}^n$ then $\mathbf{z} \in \mathbb{R}^n$
- Note 2: For any invertible matrix A, $det(A^{-1}) = det(A)^{-1}$

$$p_X(\mathbf{x}) = p_Z\left(\mathbf{z}\right) \left| \det \left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1}$$

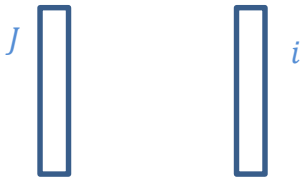
Slide by Stefano Ermon, Aditya Grover

Generalized change of variables

- For linear transformations specified via A, change in volume is given by the determinant of A
- For non-linear transformations $\mathbf{f}(\cdot)$, the *linearized* change in volume is given by the determinant of the Jacobian of $\mathbf{f}(\cdot)$.
- **Change of variables (General case)**: The mapping between Z and X, given by $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$, is invertible such that $X = \mathbf{f}(Z)$ and $Z = \mathbf{f}^{-1}(X)$.

f: x →z

$$p_X(\mathbf{x}) = p_Z\left(\mathbf{f}^{-1}(\mathbf{x})\right)\left|\det\left(\frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$



$$J_{ij} = \frac{\partial [f(z)]_i}{\partial z_j}$$

- Note 1: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension. For example, if $\mathbf{x} \in \mathbb{R}^n$ then $\mathbf{z} \in \mathbb{R}^n$
- Note 2: For any invertible matrix A, $det(A^{-1}) = det(A)^{-1}$

$$p_X(\mathbf{x}) = p_Z(\mathbf{z})\left|\det\left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}}\right)\right|^{-1}$$

Slide by Stefano Ermon, Aditya Grover

Generalized change of variables

- For linear transformations specified via A, change in volume is given by the determinant of A
- For non-linear transformations $\mathbf{f}(\cdot)$, the *linearized* change in volume is given by the determinant of the Jacobian of $\mathbf{f}(\cdot)$.
- **Change of variables (General case)**: The mapping between Z and X, given by $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$, is invertible such that $X = \mathbf{f}(Z)$ and $Z = \mathbf{f}^{-1}(X)$.

nxn matrix

f: x →z

$$p_X(\mathbf{x}) = p_Z\left(\mathbf{f}^{-1}(\mathbf{x})\right)\left|\det\left(\frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

J i

$$J_{ij} = \frac{\partial [f(z)]_i}{\partial z_j}$$

- Note 1: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension. For example, if $\mathbf{x} \in \mathbb{R}^n$ then $\mathbf{z} \in \mathbb{R}^n$
- Note 2: For any invertible matrix A, $det(A^{-1}) = det(A)^{-1}$

$$p_X(\mathbf{x}) = p_Z(\mathbf{z})\left|\det\left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}}\right)\right|^{-1}$$

Slide by Stefano Ermon, Aditya Grover

Generalized change of variables

- For linear transformations specified via A, change in volume is given by the determinant of A
- For non-linear transformations $\mathbf{f}(\cdot)$, the *linearized* change in volume is given by the determinant of the Jacobian of $\mathbf{f}(\cdot)$.
- **Change of variables (General case)**: The mapping between Z and X, given by $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$, is invertible such that $X = \mathbf{f}(Z)$ and $Z = \mathbf{f}^{-1}(X)$.

nxn matrix

f: x →z

$$p_X(\mathbf{x}) = p_Z\left(\mathbf{f}^{-1}(\mathbf{x})\right)\left|\det\left(\frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

J i

$$J_{ij} = \frac{\partial [f(z)]_i}{\partial z_j}$$

- Note 1: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension. For example, if $\mathbf{x} \in \mathbb{R}^n$ then $\mathbf{z} \in \mathbb{R}^n$
- Note 2: For any invertible matrix A, $det(A^{-1}) = det(A)^{-1}$

$$p_X(\mathbf{x}) = p_Z(\mathbf{z})\left|\det\left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}}\right)\right|^{-1}$$

Slide by Stefano Ermon, Aditya Grover

Two Dimensional Example

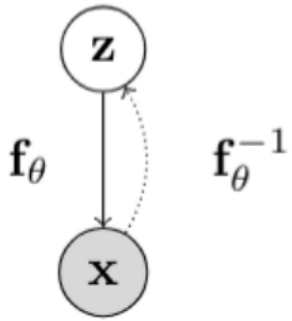
- Let Z_1 and Z_2 be continuous random variables with joint density p_{Z_1,Z_2}.
- Let $u = (u_1, u_2)$ be a transformation
- Let $v = (v_1, v_2)$ be the inverse transformation
- Let $X_1 = u_1(Z_1, Z_2)$ and $X_2 = u_2(Z_1, Z_2)$ Then, $Z_1 = v_1(X_1, X_2)$ and $Z_2 = v_2(X_1, X_2)$

$$
\begin{aligned}
& p_{X_1,X_2}(x_1, x_2) \\
&= p_{Z_1,Z_2}(v_1(x_1, x_2), v_2(x_1, x_2)) \left| \det \begin{pmatrix} \frac{\partial v_1(x_1,x_2)}{\partial x_1} & \frac{\partial v_1(x_1,x_2)}{\partial x_2} \\ \frac{\partial v_2(x_1,x_2)}{\partial x_1} & \frac{\partial v_2(x_1,x_2)}{\partial x_2} \end{pmatrix} \right| \text{(inverse)} \\
&= p_{Z_1,Z_2}(z_1, z_2) \left| \det \begin{pmatrix} \frac{\partial u_1(z_1,z_2)}{\partial z_1} & \frac{\partial u_1(z_1,z_2)}{\partial z_2} \\ \frac{\partial u_2(z_1,z_2)}{\partial z_1} & \frac{\partial u_2(z_1,z_2)}{\partial z_2} \end{pmatrix} \right|^{-1} \text{(forward)}
\end{aligned}
$$

Slide by Stefano Ermon, Aditya Grover

Normalizing flow models

- Consider a directed, latent-variable model over observed variables X and latent variables Z
- In a **normalizing flow model**, the mapping between Z and X, given by $\mathbf{f}_\theta : \mathbb{R}^n \mapsto \mathbb{R}^n$, is deterministic and invertible such that $X = \mathbf{f}_\theta(Z)$ and $Z = \mathbf{f}_\theta^{-1}(X)$



- Using change of variables, the marginal likelihood $p(\mathbf{x})$ is given by

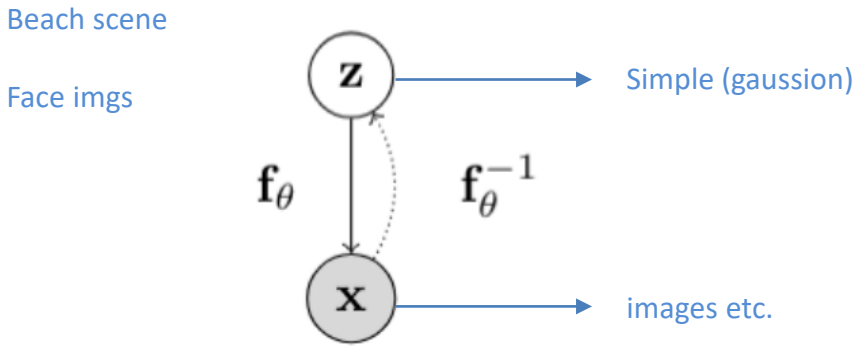
$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right)\left|\det\left(\frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

- Note: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension.

Slide by Stefano Ermon, Aditya Grover

Normalizing flow models

- Consider a directed, latent-variable model over observed variables X and latent variables Z
- In a **normalizing flow model**, the mapping between Z and X, given by $\mathbf{f}_\theta : \mathbb{R}^n \mapsto \mathbb{R}^n$, is deterministic and invertible such that $X = \mathbf{f}_\theta(Z)$ and $Z = \mathbf{f}_\theta^{-1}(X)$



- Using change of variables, the marginal likelihood $p(\mathbf{x})$ is given by

$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right)\left|\det\left(\frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

- Note: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension.

Slide by Stefano Ermon, Aditya Grover

A Flow of Transformations

Normalizing: Change of variables gives a normalized density after applying an invertible transformation

Flow: Invertible transformations can be composed with each other

$$\mathbf{z}_m := \mathbf{f}_\theta^m \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) = \mathbf{f}_\theta^m(\mathbf{f}_\theta^{m-1}(\cdots(\mathbf{f}_\theta^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_\theta(\mathbf{z}_0)$$

$z_0 \quad f^1 \quad z_1 \quad f^2 \quad z_2 \qquad\qquad x$

- Start with a simple distribution for $\mathbf{z}_0$ (e.g., Gaussian)
- Apply a sequence of M invertible transformations $\mathbf{x} \triangleq \mathbf{z}_M$
- By change of variables

$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right) \prod_{m=1}^{M} \left|\det\left(\frac{\partial(\mathbf{f}_\theta^m)^{-1}(\mathbf{z}_m)}{\partial \mathbf{z}_m}\right)\right|$$

(Note: determininant of product equals product of determinants)

Slide by Stefano Ermon, Aditya Grover

$p_\theta(x)$

Figure from [link]

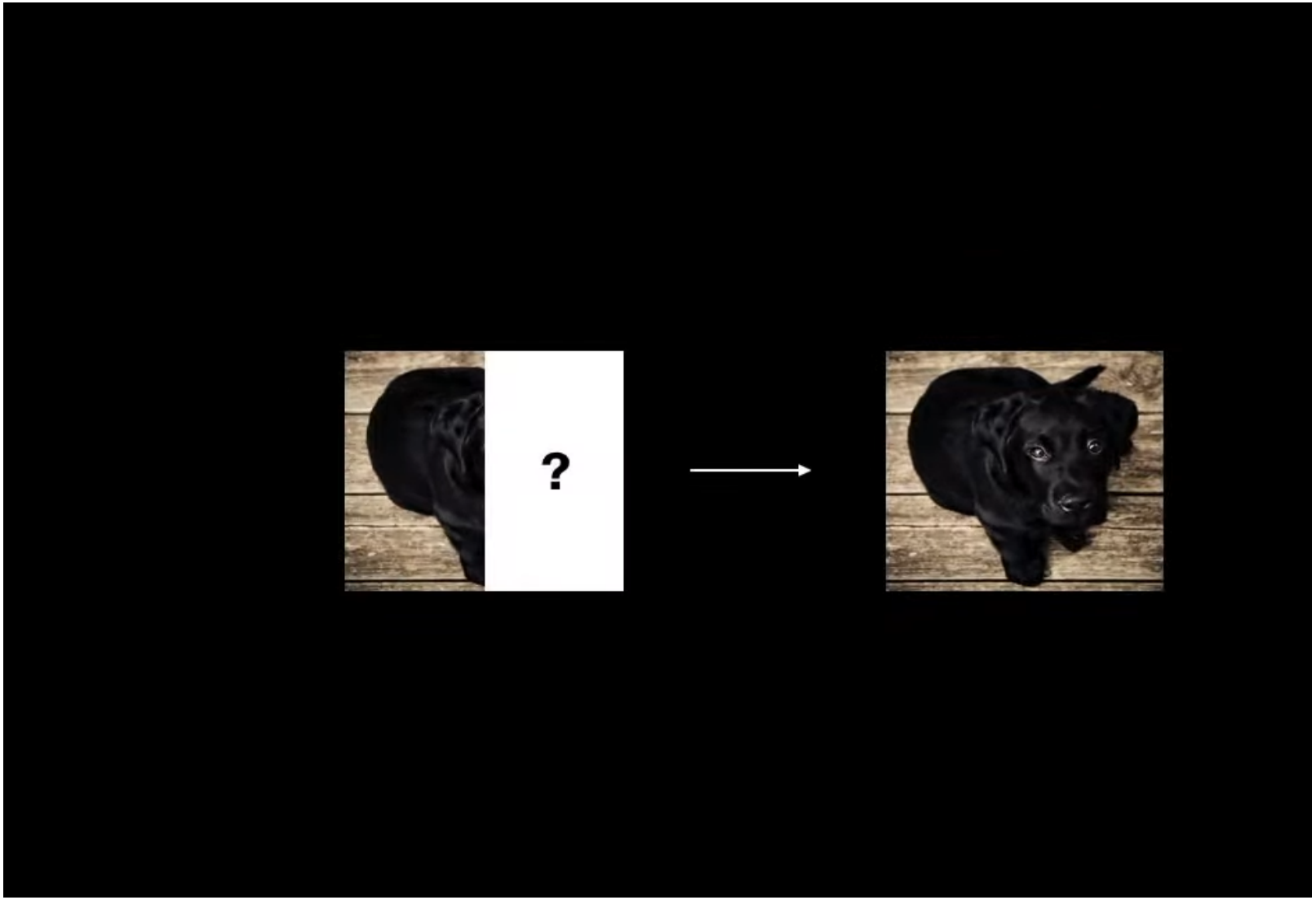
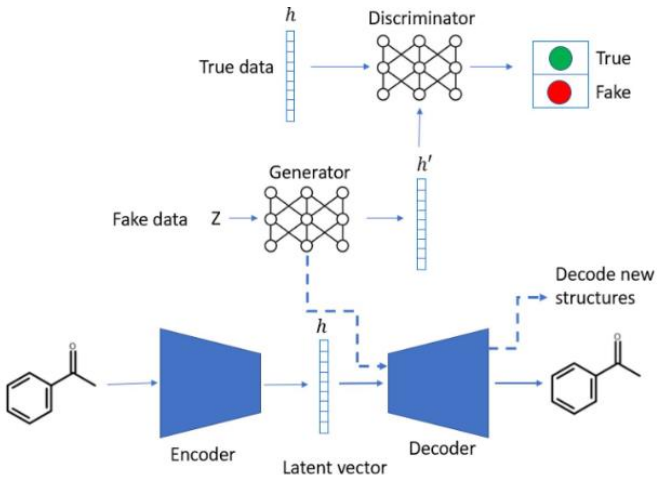


Figure from link

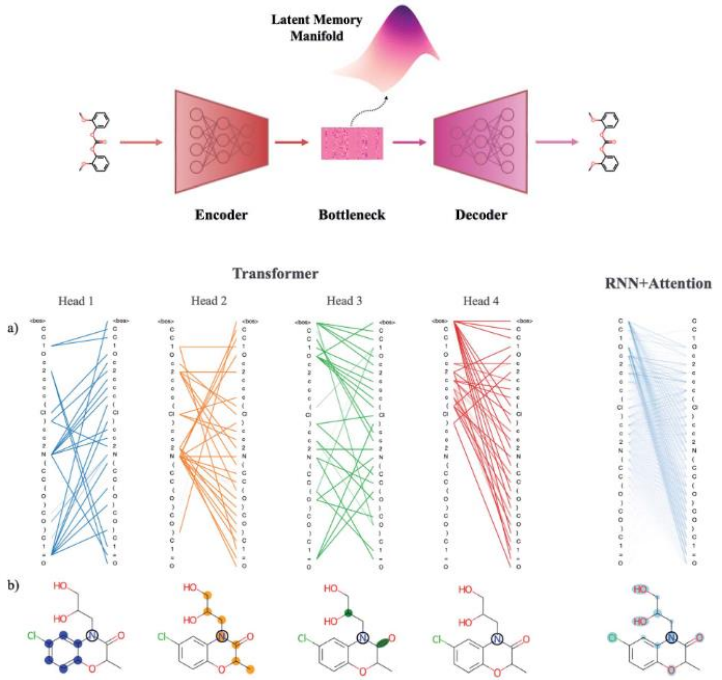
LatentGAN: A *de novo* molecular generation method using latent vector based generative adversarial network (2019)

Oleksii Prykhodko[1], Simon Viet Johansson[1], Panagiotis-Christos Kotsias[1], Josep Arús-Pous[1], Esben Jannik Bjerrum[1], Ola Engkvist[1], Hongming Chen[1]



TransVAE:Attention-based generative models for *de novo* molecular design (2021)

Orion Dollar[2], Nisarg Josh[2],David A. C. Beck[2], Jim Pfaendtner[2]



[1]Hit Discovery, Discovery Sciences, Biopharmaceutical R&D, AstraZeneca, Gothenburg, Sweden,
[2]Department of Chemical Engineering, University of Washington

Presented by Ahmet Sarigun

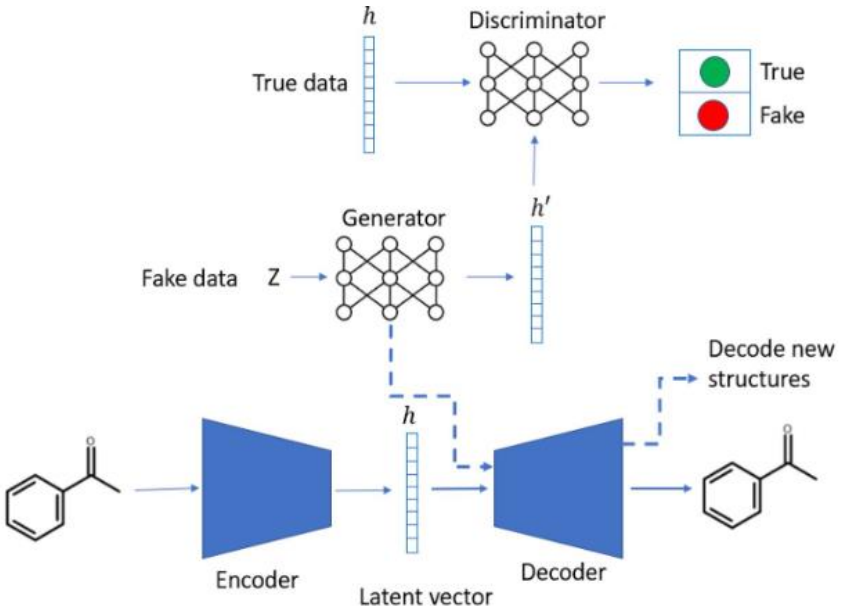
Figures from papers

Outline

- ✓ **LatentGAN**
  - ❖ **Introduction**
  - ❖ **Methods and materials**
    - Heteroencoder architecture
    - The GAN architecture
    - Workflow for training and sampling of the LatentGAN
    - Dataset and machine learning models for scoring
    - Related Works
  - ❖ **Results and discussion**
    - Training the heteroencoder
    - Training on the ChEMBL subset
    - Comparison with similar generative networks
    - On the properties of autoencoder latent spaces
  - ❖ **Conclusions**
- ✓ **TransVAE**
  - ❖ **Introduction**
  - ❖ **Results and discussion**
    - Variational autoencoder and the information bottleneck
    - Adding attention to the VAE
    - Impact of attention

LatentGAN: A de novo molecular generation method using latent vector based generative adversarial network (2019)

Oleksii Prykhodko[1], Simon Viet Johansson[1], Panagiotis-Christos Kotsias[1], Josep Arús-Pous[1], Esben Jannik Bjerrum[1], Ola Engkvist[1], Hongming Chen[1]



[1]Hit Discovery, Discovery Sciences, Biopharmaceutical R&D, AstraZeneca, Gothenburg, Sweden,

Figure from paper

Introduction

- Deep generative models learn how to generate molecules by generalizing the probability of the generation process of a large set of chemical structures (i.e., training set).

- Structure generation is basically a sampling process following the learned probability distribution (Segler et al., 2018; Olivecrona et al., 2017; Arús-Pous et al., 2019; Arús-Pous et al., 2019b). It is a data-driven process and requires very few predefined rules.

- Early attempted architectures were inspired by the deep learning methods used in natural language processing (NLP) (Segler et al., 2018; Voss 2015).

Segler et al., 2018; Generating focused molecule libraries for drug discovery with recurrent neural networks
Olivecrona et al., 2017; Molecular de-novo design through deep reinforcement learning
Arús-Pous et al., 2019; Exploring the GDB-13 chemical space using deep generative models
Arús-Pous et al., 2019b; Randomized SMILES strings improve the quality of molecular generative models
Voss 2015; Modeling molecules with recurrent neural networks

Introduction

- Later on, the REINVENT method was proposed, which combines RNNs with reinforcement learning to generate structures with desirable properties (Olivecrona et al., 2017).

- Another architecture, the variational autoencoder (VAE), was also shown to generate novel chemical space (Gómez-Bombarelli et al., 2018; Blaschke et al., 2018).

Olivecrona et al., 2017; Molecular de-novo design through deep reinforcement learning
Gómez-Bombarelli et al., 2018; Automatic chemical design using a data-driven continuous representation of molecules
Blaschke et al., 2018; Application of generative autoencoder in de novo molecular design

Introduction

- In this study, a new molecular generation strategy is described which combines an autoencoder and a GAN
- The difference between this method and previous GAN methods such as ORGANIC and RANC is that the generator and discriminator network do not use SMILES strings as input, but instead n-dimensional vectors derived from the code-layer of an autoencoder trained as a SMILES heteroencoder (Kotsias et al., 2019).
- This allows the model to focus on optimizing the sampling and not worry about SMILES syntax issues. The decoder part of a pretrained heteroencoder (Bjerrum et al., 2018) neural network was used to translate the generated n-dimensional vector into molecular structures.

Kotsias et al., 2019; Direct steering of de novo molecular generation using descriptor conditional recurrent neural networks (cRNNs)
Bjerrum et al., 2018; Improving chemical autoencoder latent space and molecular de novo generation diversity with heteroencoders

Introduction

- To learn the generator's distribution p_g over data x, firstly, input noise variables $p_z(z)$, defined.
- Representing a mapping to data space as $G(z;\theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g.
- Second multilayer perceptron $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

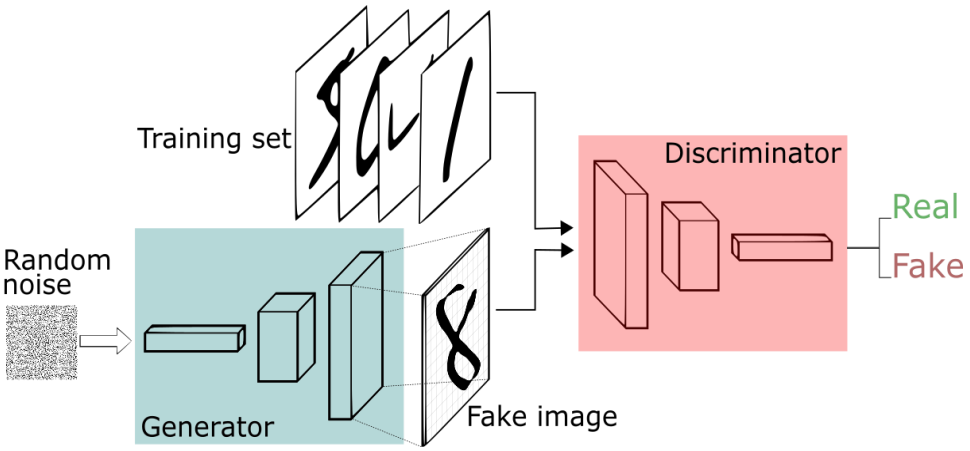


Figure from link

Introduction

$$\min_{G} \max_{D} V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

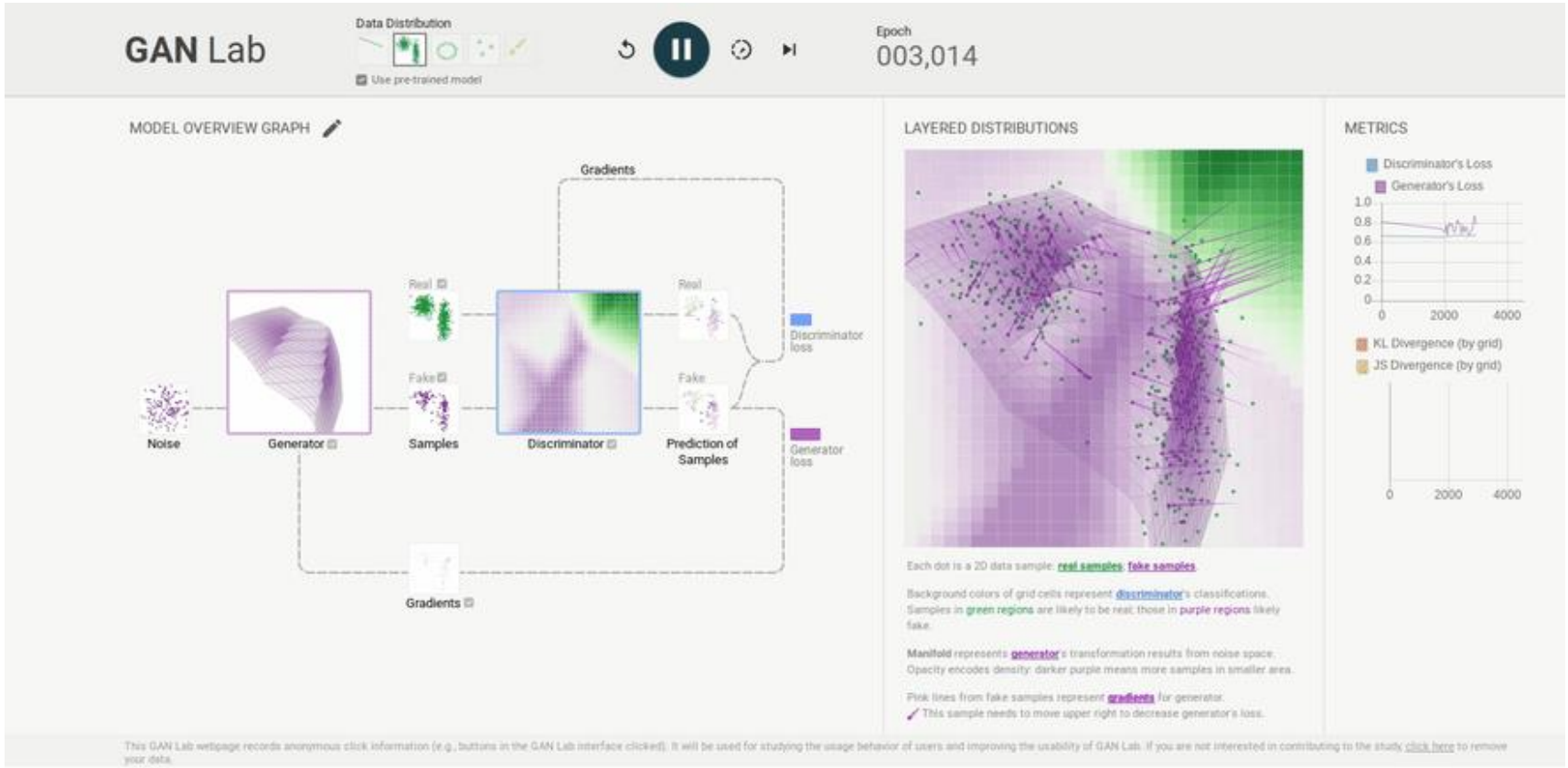


Figure from [source]

Methods and Materials

The GAN Architecture

- **Generative adversarial networks *(Goodfellow et al., 2014)*** **(individual samples)**

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

- **Wasserstein GANs *(Arjovsky et al., 2017)*** **(Distributed samples)**

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[D(x)] - \mathbb{E}_{z\sim p_z(z)}[D(G(z))]$$

(D: $\uparrow$, G: $\downarrow$)

- **Wasserstein GANs with Gradient Penalty *(Gulrajani et al., 2017)*** **(Fix Grad.)**

$$\mathcal{L} = \underbrace{\mathbb{E}_{x\sim p_{data}(x)}[D(x)] - \mathbb{E}_{z\sim p_z(z)}[D(G(z))]}_{\text{Original critic loss}} + \underbrace{\lambda\,\mathbb{E}_{\hat{x}\sim p_{data}(\hat{x})}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Gradient Penalty}}$$

$$f: X \to Y \text{ is } K-\text{Lipschitz if for distance } d_x \text{ and } d_y \text{ on } X \text{ and } Y \; d_y(f(x_1), f(x_2)) \le K d_x(x_1, x_2) \;\; (\text{bdd correl.})$$

Goodfellow et al., 2014; Generative Adversarial Networks
Arjovsky et al., 2017; Wasserstein GAN
Gulrajani et al., 2017; Wasserstein GANs with Gradient Penalty

Methods and Materials

Heteroencoder Architecture

- A heteroencoder is an autoencoder architecture trained on pairs of different representations of the same entity, i.e. different non-canonical SMILES of the same molecule.

- It consists of two neural networks, namely, the encoder and decoder, which are jointly trained as a transformation pipeline.

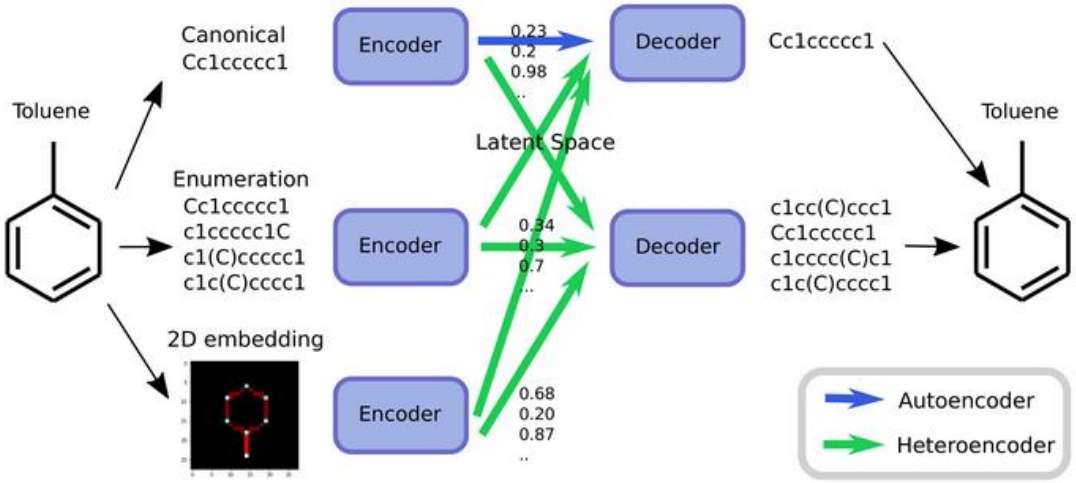


Figure from [link]

Methods and Materials

Heteroencoder Architecture

- The encoder is responsible for translating one-hot encoded SMILES strings into a numerical latent representation whereas the decoder accepts this latent representation and attempts to reconstruct one of the possible non-canonical SMILES string that it represents.

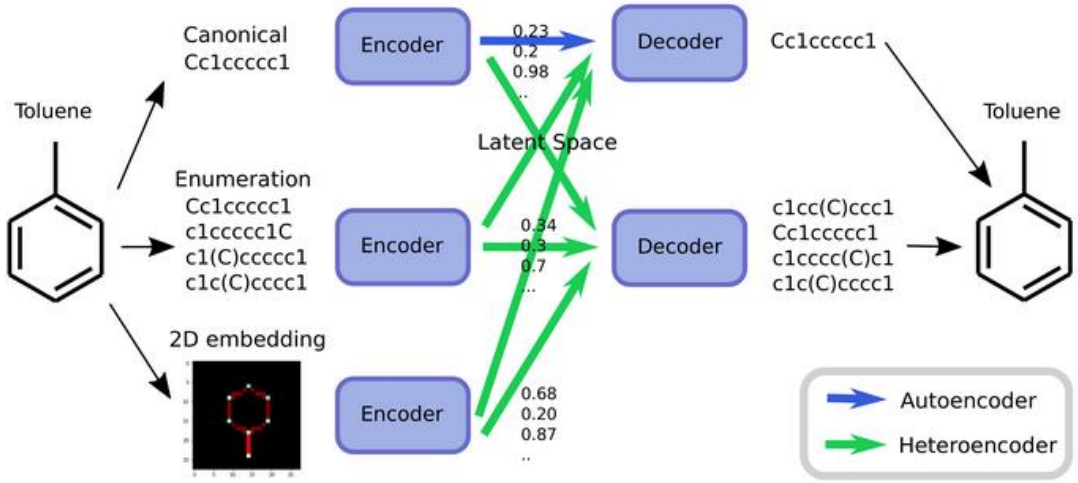


Figure from [link]

Methods and Materials

Heteroencoder Architecture

- Initially, the one-hot encoded SMILES string is propagated through a two-layer bidirectional encoder with 512 Long Short-Term Memory (De Cao & Kipf, 2018) units per layer, half of which are used for the forward and half for the backward direction.

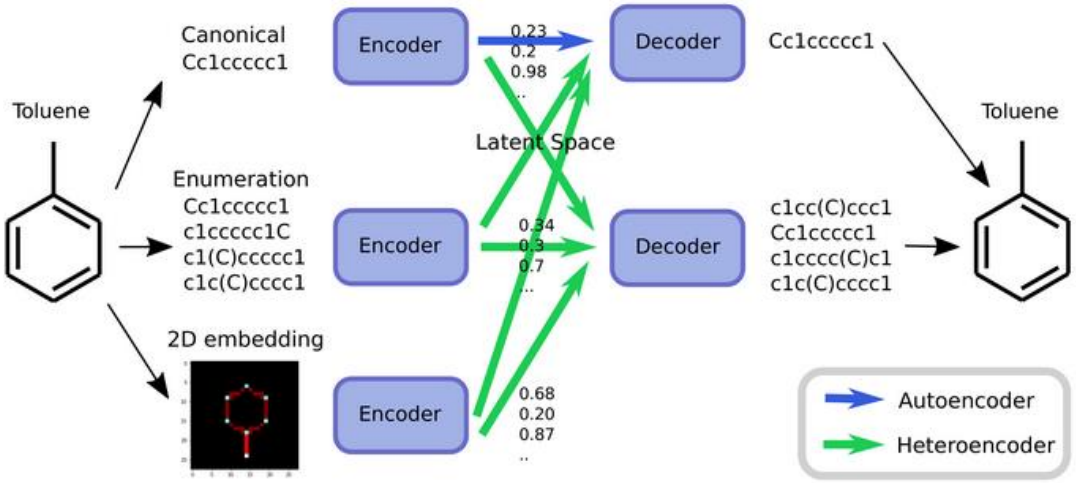


Figure from [link]

Methods and Materials

Heteroencoder Architecture

- The output of both directions is then concatenated and input to a feed-forward layer with 512 dimensions.

- The latent representation of the molecule is fed to a feed-forward layer, the output of which is copied and inserted as hidden and cell states to a four-layer unidirectional LSTM RNN decoder with the same specifications as the encoder.

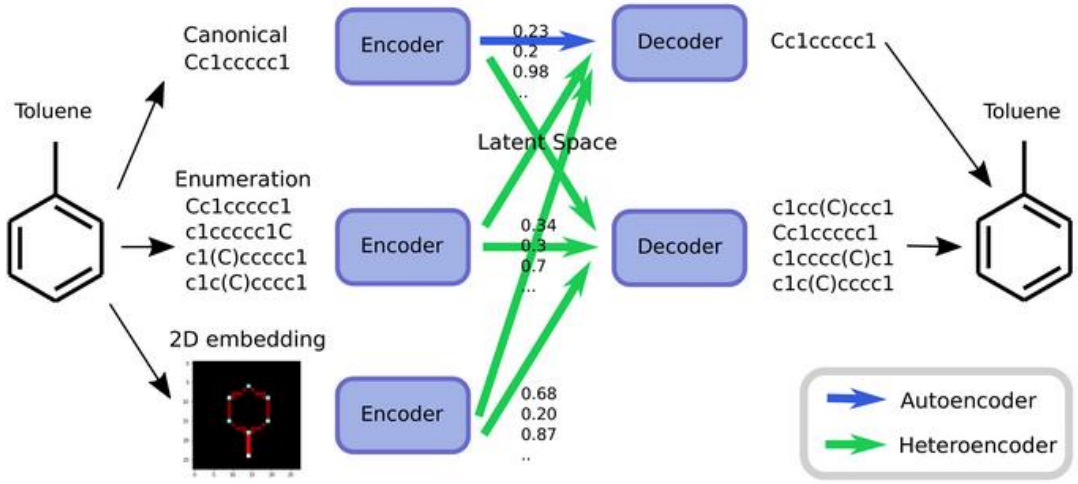


Figure from [link]

Methods and Materials

Heteroencoder Architecture

- The decoder was trained using the teacher's forcing method (Williams&Zipser, 2008). The model was trained using the decoding loss function of categorial cross entropy between the decoded and the training SMILES.

- After training the heteroencoder, the noise layer is deactivated, resulting in a deterministic encoding and decoding of the GAN training and sampled sets.

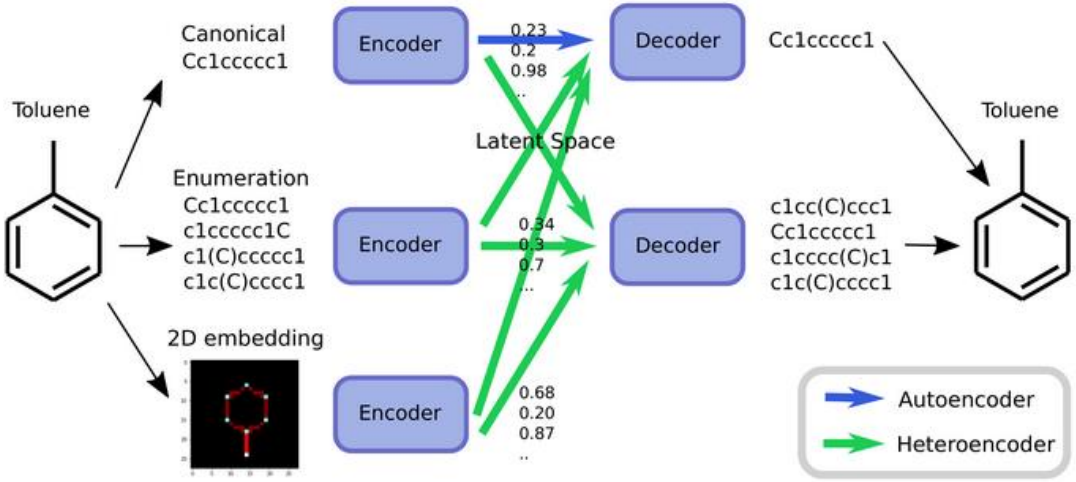


Figure from [link]

Williams&Zipser, 2008; A learning algorithm for continually running fully recurrent neural networks

Methods and Materials

The GAN Architecture

- A Wasserstein GAN with gradient penalty (WGAN-GP) (Gulrajani et al., 2017; Luo, 2018) was chosen as a GAN model. Every GAN consists of two neural networks, generator and discriminator that train simultaneously.

- First, the discriminator, usually called the critic in the context of WGANs, tries to distinguish between real data and fake data.

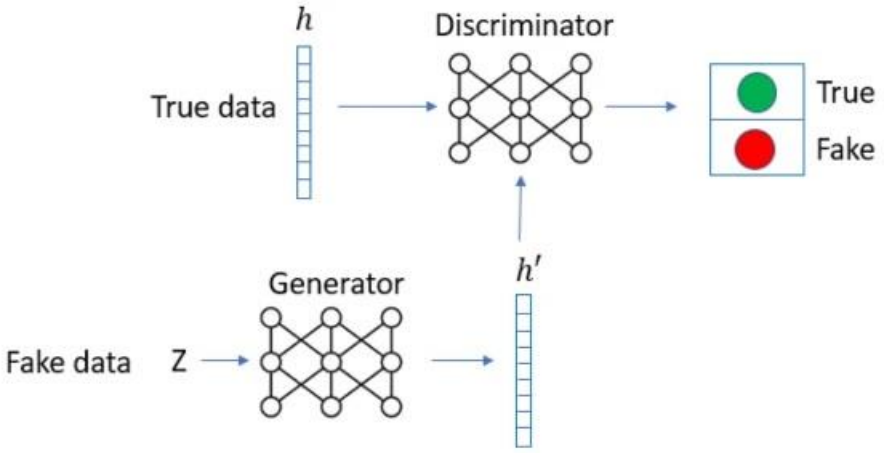


Figure from paper

Gulrajani et al., 2017; Improved training of wasserstein GANs
Luo, 2018; EEG data augmentation for emotion recognition using a conditional wasserstein GAN

Methods and Materials

The GAN Architecture

- It is formed by three feed-forward layers of 256 dimensions each with the leaky ReLU activation function between, except for the last layer where no activation function was used. Second, the generator consists of five feed-forward layers of 256 dimensions each with batch normalization and leaky ReLU activation function between each.

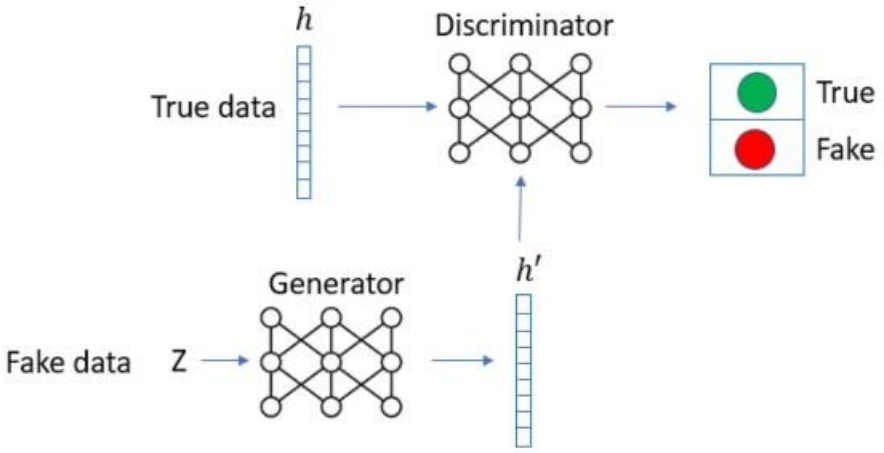


Figure from paper

Methods and Materials

Workflow for training and sampling of the LatentGAN

- The heteroencoder model was first pre-trained on the ChEMBL database for mapping structures to latent vectors.
- To train the full GAN model, first the latent vector *h* of the training set was generated using the encoder part of the heteroencoder.

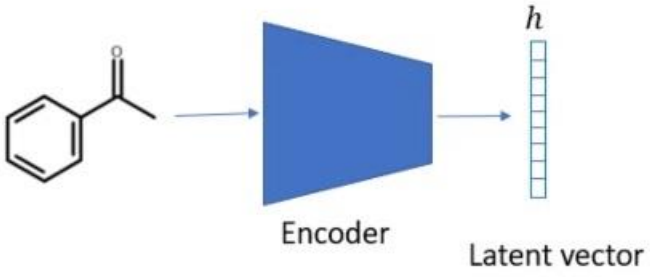


Figure from paper

Methods and Materials

Workflow for training and sampling of the LatentGAN

- Then, it was used as the true data input for the discriminator, while a set of random vectors sampled from a uniform distribution were taken as fake data input to the generator.

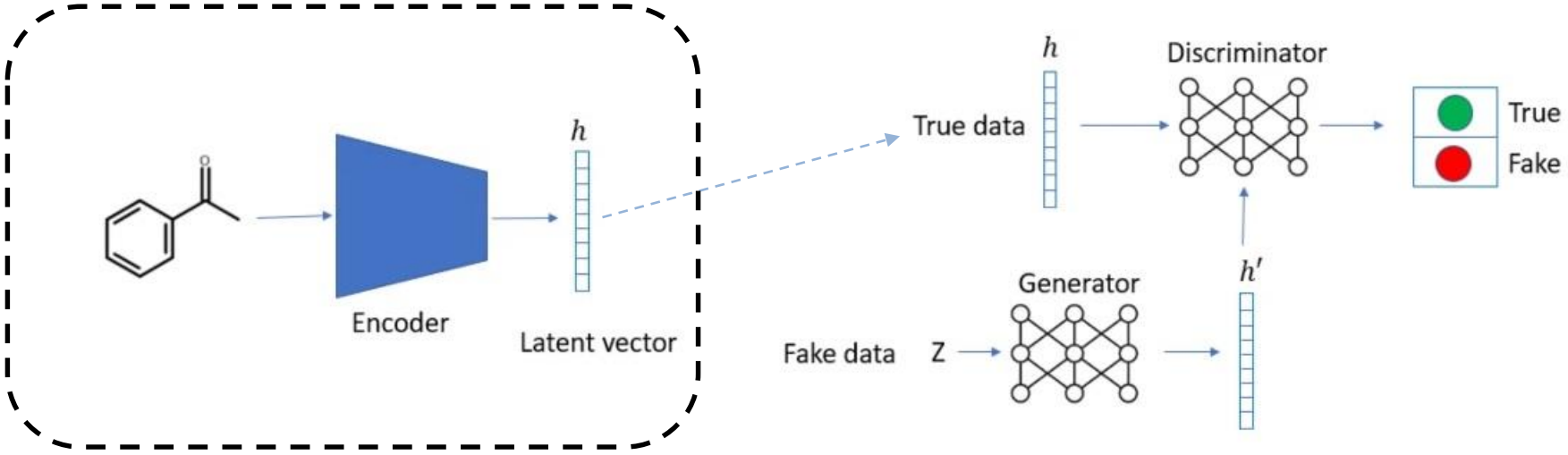


Figure from paper

Methods and Materials

Workflow for training and sampling of the LatentGAN

- For every five batches of training for the discriminator, one batch was assigned to train the generator, so that the critic is kept ahead while providing the generator with higher gradients.

- Once the GAN training was finished, the Generator was sampled multiple times and the resulting latent vectors were fed into the decoder to obtain the SMILES strings of the underlying molecules.

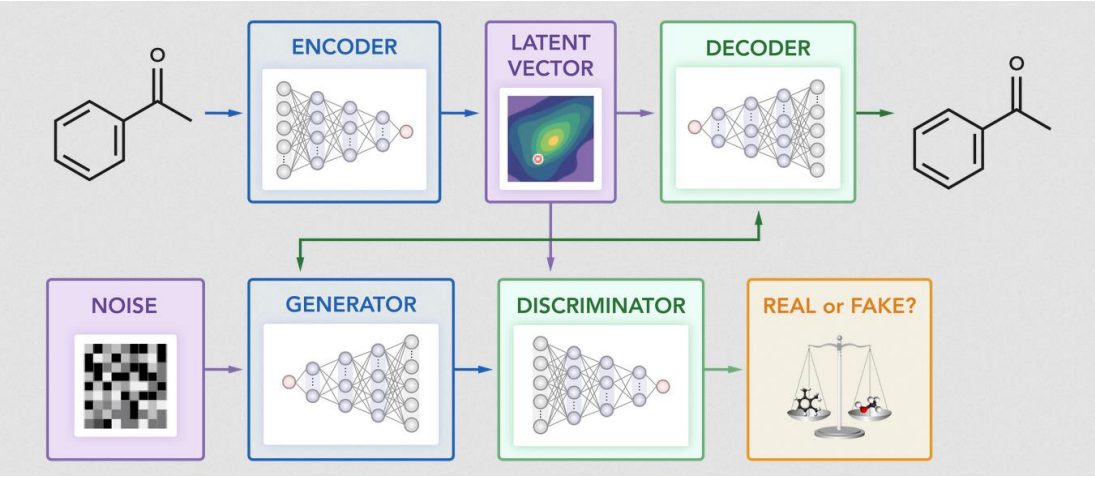


Figure from link

Methods and Materials

Dataset and machine learning models for scoring

- The heteroencoder was trained on 1,347,173 SMILES from the ChEMBL dataset.
- This is a subset of ChEMBL 25 without duplicates that has been standardized using the MolVS v0.1.1 package with respect to the fragment, charge, isotope, stereochemistry and tautomeric states.
- The set is limited to SMILES of containing only [H, C, N, O, S, Cl, Br] atoms and a total of 50 heavy atoms or less.
- Furthermore, molecules known to be active towards DRD2 were removed as part of an experiment for the heteroencoder (the process of which can be found at (Kotsias et al., 2019), which uses the same decoder model, but not the encoder). A set of randomly selected 100,000 ChEMBL compounds were later selected for training a general GAN model.

Methods and Materials

Dataset and machine learning models for scoring

- Moreover, three target datasets (corresponding to EGFR, S1PR1 and HTR1A) were extracted from ExCAPE-DB (Sun et al., 2017) for training target specific GANs.

- The ExCAPE-DB datasets were then clustered into training and test sets so that chemical series were assigned either to the training or to the test set

Table 1 Targeted data set and the performance of the SVM models

Target	Training set	Test set	SVM model	
			ROC-AUC	Kappa value
EGFR	2949	2326	0.850	0.56
HTR1A	48,283	23,048	0.993	0.90
S1PR1	49,381	23,745	0.995	0.91

Training set size (training set), test set size (test set), receiver operating characteristic area under the curve (ROC-AUC), kappa value

Table from paper

Sun et al., 2017; ExCAPE-DB: an integrated large scale dataset facilitating Big Data analysis in chemogenomics

Methods and Materials

Dataset and machine learning models for scoring

- To benchmark the performance of the targeted models, RNN based generative models for the three targets were also created by first training a prior RNN model on the same ChEMBL set used for training the heteroencoder model and then using transfer learning (Segler et al., 2018) on each focused target set.

- Target prediction models were calculated for each target using the Support vector machine learning (SVM) implementation in the Scikit-learn package and the 2048-length FCFP6 fingerprint were calculated using RDKit (Landrum, 2014).

Segler et al., 2018; Generating focused molecule libraries for drug discovery with recurrent neural networks
Landrum, 2014; RDKit: open-source cheminformatics

Methods and Materials

Related works

- A related architecture to the LatentGAN is the Adversarial Autoencoder (AAE) (Makhzani et al., 2015). The AAE uses a discriminator to introduce adversarial training to the autoencoder
  and is trained typically using a 3-step training scheme of (a) discriminator, (b) encoder, (c) encoder and decoder, compared to the LatentGANs 2-step training.

- The AAE have been used in generative modeling of molecules to sample molecular fingerprints using additional encoder training steps (Kadurin et al., 2017), as well as SMILES representations (Polykovskiy et al.,2018; Polykovskiy et al., 2018b).

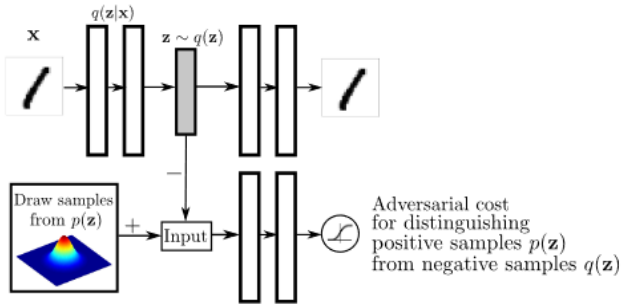


Figure 1: Architecture of an adversarial autoencoder. The top row is a standard autoencoder that reconstructs an image x from a latent code z. The bottom row diagrams a second network trained to discriminatively predict whether a sample arises from the hidden code of the autoencoder or from a sampled distribution specified by the user.

Figure from link

Makhzani et al., 2015; Adversarial autoencoders
Kadurin et al., 2017; The cornucopia of meaningful leads: applying deep adversarial autoencoders for new molecule development in oncology
Polykovskiy et al.,2018; Molecular sets (MOSES): a benchmarking platform for molecular generation models
Polykovskiy et al., 2018b; Entangled conditional adversarial autoencoder for de novo drug discovery

Methods and Materials

Related works

- Approaches that have utilized multiple discriminators have been used to combine conditional VAEs and conditional GANs to enforce constraints on the latent space (Engel et al., 2017) and thus increase the realism of the images.

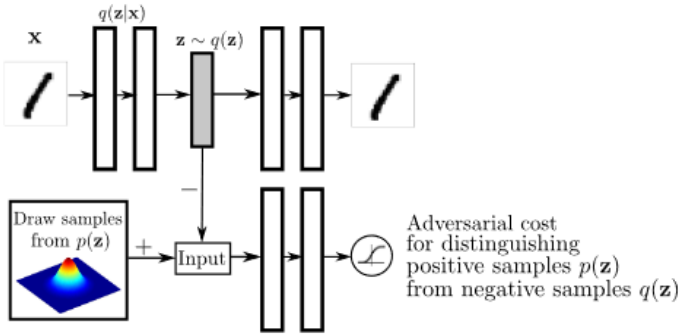


Figure 1: Architecture of an adversarial autoencoder. The top row is a standard autoencoder that reconstructs an image x from a latent code z. The bottom row diagrams a second network trained to discriminatively predict whether a sample arises from the hidden code of the autoencoder or from a sampled distribution specified by the user.

Figure from link

Engel et al., 2017; Latent constraints: learning to generate conditionally from unconditional generative models

Results and discussion

Training the heteroencoder

- The heteroencoder was trained on the 1,347,173 ChEMBL dataset compounds for 100 epochs.
- SMILES generated validity for the whole training set was 99% and 18% of the molecules were not reconstructed properly.
- Notice that the reconstruction error corresponds to decoding to a valid SMILES that belongs to a different compound; reconstruction to a different SMILES of the same molecule is not counted as an error.

Table 2 The performance of heteroencoder in both the training and test sets

Dataset	# compounds	Validity (%)	Reconstruction error (%)
Training set	974,105	99	18
Test set	10,823	98	20

Percent of valid SMILES strings generated by the decoder (validity), percent of molecules not reconstructed correctly from valid SMILES (reconstruction error)

Table from paper

Results and discussion

Training on the ChEMBL subset

- A LatentGAN was trained on a randomly selected 100,000 ChEMBL subset with the objective of obtaining drug-like compounds.
- The model was trained for 30,000 epochs until both discriminator and generator models had converged.
- Next, 200,000 compounds were generated from the LatentGAN model and were compared with the 100,000 ChEMBL training compounds to examine the coverage of the chemical space.

Results and discussion

Comparison with similar generative networks

- The LatentGAN was assessed using the MOSES benchmark platform (Polykovskiy et al., 2018), where several generative metrics are used to evaluate the properties of molecular generative networks on a sample of 30,000 SMILES after training on a canonical SMILES subset of the ZINC database of size 1,584,663.

- When compared to the similar structured networks of VAE, JTN-VAE (Jin et al., 2018) and AAE, it is noticeable that VAE model have an output distribution that has a significant overlap with the training set, as shown by the high scores of most test metrics (where the test set has a similar distribution to the training set) and the low novelty, indicating a mode collapse.

- When compared against the JTN-VAE and AAE models, the LatentGAN has shows comparable or better results in the Fréchet ChemNet Distance (FCD) (Preuer et al., 2018), Fragment (Frag) and Scaffold (Scaf) similarities, while producing slightly worse results in the cosine similarity to the nearest neighbor in the test set (SNN).

Polykovskiy et al., 2018; Molecular sets (MOSES): a benchmarking platform for molecular generation models
Jin et al., 2018; Junction tree variational autoencoder for molecular graph generation
Preuer et al., 2018; Fréchet ChemNet distance: a metric for generative models for molecules in drug discovery

Results and discussion

Comparison with similar generative networks

Table 1: Performance metrics for baseline models: fraction of valid molecules, fraction of unique molecules from 1,000 and 10,000 molecules. Reported (mean ± std) over three independent model initializations.

Model	Valid (↑)	Unique@1k (↑)	Unique@10k (↑)
Train	*1.0*	*1.0*	*1.0*
HMM	0.076 ± 0.0322	0.623 ± 0.1224	0.5671 ± 0.1424
NGram	0.2376 ± 0.0025	0.974 ± 0.0108	0.9217 ± 0.0019
Combinatorial	**1.0 ± 0.0**	0.9983 ± 0.0015	0.9909 ± 0.0009
CharRNN	0.975 ± 0.026	**1.0 ± 0.0**	0.999 ± 0.0
VAE	0.977 ± 0.001	**1.0 ± 0.0**	0.998 ± 0.001
AAE	0.937 ± 0.034	**1.0 ± 0.0**	0.997 ± 0.002
JTN-VAE	**1.0 ± 0.0**	**1.0 ± 0.0**	**0.9996 ± 0.0003**
LatentGAN	0.897 ± 0.002	**1.0 ± 0.0**	0.997 ± 0.005

Table from link

Results and discussion

Comparison with similar generative networks

Table 2: Performance metrics for baseline models: fraction of molecules passing filters (MCF, PAINS, ring sizes, charge, atom types), novelty, and internal diversity. Reported (mean ± std) over three independent model initializations.

Model	Filters (↑)	Novelty (↑)	$IntDiv_1$	$IntDiv_2$
Train	*1.0*	*0.0*	*0.857*	*0.851*
HMM	0.9024 ± 0.0489	**0.9994 ± 0.001**	0.8466 ± 0.0403	0.8104 ± 0.0507
NGram	0.9582 ± 0.001	0.9694 ± 0.001	**0.8738 ± 0.0002**	0.8644 ± 0.0002
Combinatorial	0.9557 ± 0.0018	0.9878 ± 0.0008	0.8732 ± 0.0002	**0.8666 ± 0.0002**
CharRNN	0.994 ± 0.003	0.842 ± 0.051	0.856 ± 0.0	0.85 ± 0.0
VAE	**0.997 ± 0.0**	0.695 ± 0.007	0.856 ± 0.0	0.85 ± 0.0
AAE	0.996 ± 0.001	0.793 ± 0.028	0.856 ± 0.003	0.85 ± 0.003
JTN-VAE	0.976 ± 0.0016	0.9143 ± 0.0058	0.8551 ± 0.0034	0.8493 ± 0.0035
LatentGAN	0.973 ± 0.001	0.949 ± 0.001	0.857 ± 0.0	0.85 ± 0.0

Table from link

Results and discussion

Comparison with similar generative networks

Table 3: Performance metrics for baseline models: Fréchet ChemNet Distance (FCD) and Similarity to a nearest neighbor (SNN); Reported (mean ± std) over three independent model initializations. Results for random test set (Test) and scaffold split test set (TestSF).

Model	FCD (↓)		SNN (↑)	
	Test	TestSF	Test	TestSF
Train	*0.008*	*0.476*	*0.642*	*0.586*
HMM	24.4661 ± 2.5251	25.4312 ± 2.5599	0.3876 ± 0.0107	0.3795 ± 0.0107
NGram	5.5069 ± 0.1027	6.2306 ± 0.0966	0.5209 ± 0.001	0.4997 ± 0.0005
Combinatorial	4.2375 ± 0.037	4.5113 ± 0.0274	0.4514 ± 0.0003	0.4388 ± 0.0002
CharRNN	**0.073 ± 0.025**	**0.52 ± 0.038**	0.601 ± 0.021	0.565 ± 0.014
VAE	0.099 ± 0.013	0.567 ± 0.034	**0.626 ± 0.0**	**0.578 ± 0.001**
AAE	0.556 ± 0.203	1.057 ± 0.237	0.608 ± 0.004	0.568 ± 0.005
JTN-VAE	0.3954 ± 0.0234	0.9382 ± 0.0531	0.5477 ± 0.0076	0.5194 ± 0.007
LatentGAN	0.296 ± 0.021	0.824 ± 0.030	0.538 ± 0.001	0.514 ± 0.009

Table from link

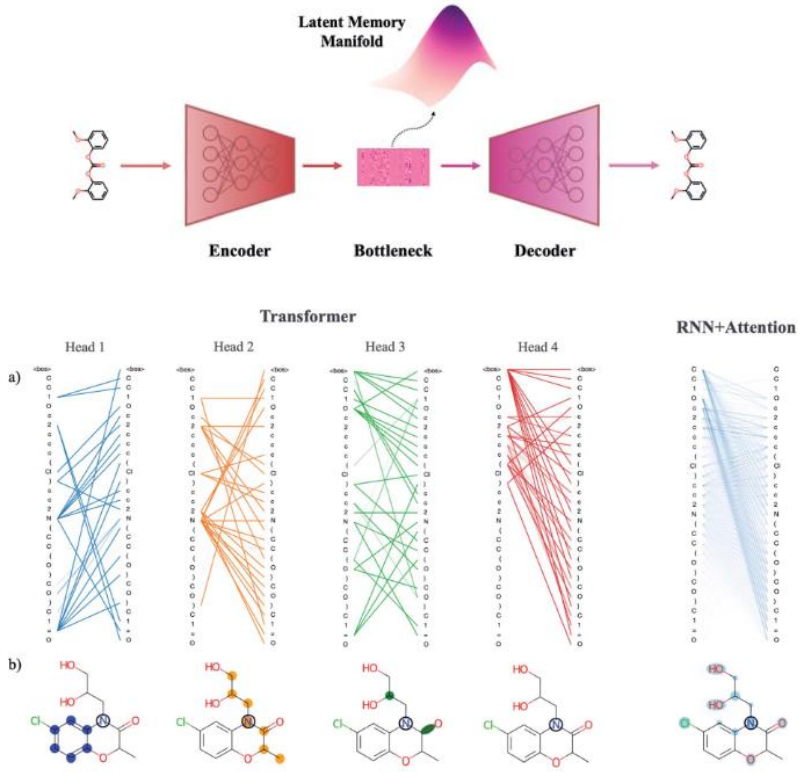
Results and discussion

Conclusions

- Combining a heteroencoder and a generative adversarial network.
- The pretrained autoencoder was used to map the molecular structure to latent vector and the GAN was trained using latent vectors as input as well as output, all in separate steps.
- After training on a subset of ChEMBL compounds, the LatentGAN was able to generate similar drug-like compounds.
- applied the method on three target biased datasets (EGFR, HTR1A and S1PR1) to investigate the capability of the LatentGAN to generate biased compounds.
- Encouragingly, our results show that most of the sampled compounds from the trained model are predicted to be active to the target which it was trained against, with a substantial portion of the sampled compounds being novel with respect to the training set.

TransVAE:Attention-based generative models for *de novo* molecular design (2021)

Orion Dollar[1], Nisarg Josh[1],David A. C. Beck[1], Jim Pfaendtner[1]



[1]Department of Chemical Engineering, University of Washington

Figure from paper

Introduction

- A VAE is capable of broadcasting a machine-interpretable representation of molecular structure (e.g. a SMILES string (Weininger, 1988), SELFIES string (Krenn et al., 2020) or molecular graph (Jin et al., 2018)) to a dense, continuous latent space or “model memory”. This memory has several unique features that make VAEs promising for inverse design:

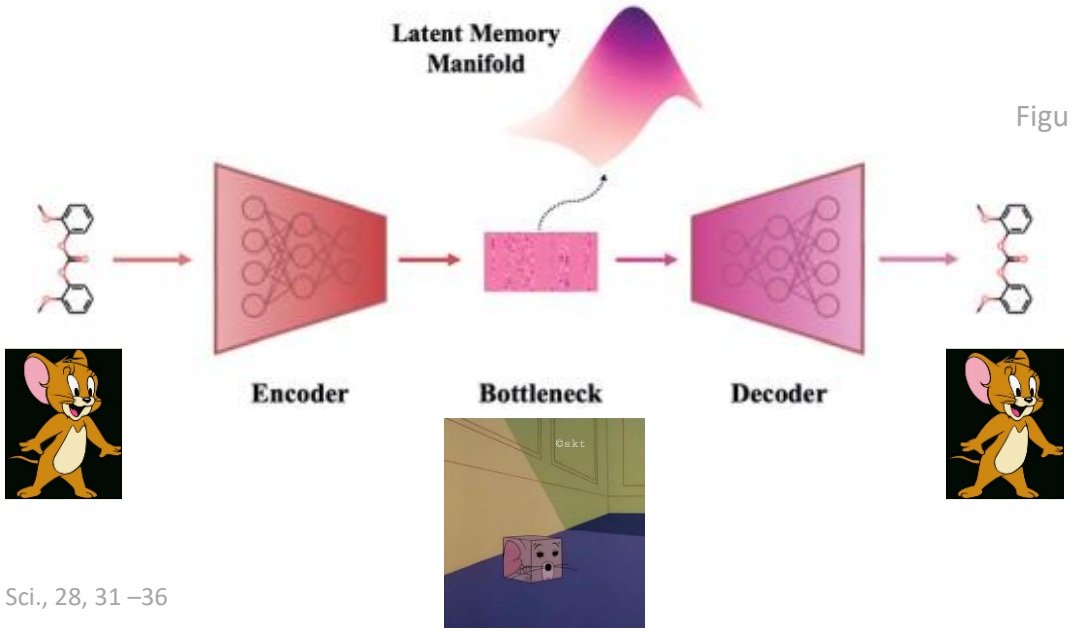


Figure from paper (except Jerry)

Weininger, 1988; J. Chem. Inf. Comput. Sci., 28, 31 –36

Krenn et al., 2020; Machine Learning: Science and Technology,045024

Jin et al., 2018; arXiv:1802.04364 [cs.LG]

Introduction

i. it can be embedded with a property and thus serve as an approximation of the joint probability distribution of molecular structure and chemical property.

ii. During training, it will organize itself meaningfully so that similar molecules are near each other in phase space.

iii. Due to its mapping from discrete to continuous data, it can be navigated with gradient-based optimization methods (G´omez-Bombarelli et al., 2018).

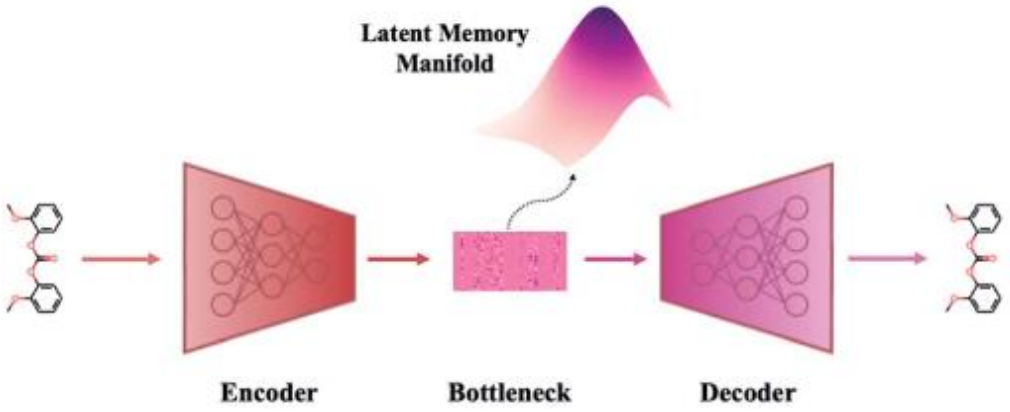


Figure from paper

G´omez-Bombarelli et al., 2018; ACS Cent. Sci., 4, 268 –276

Introduction

- In spite of these benefits, generative VAE models suffer from a set of complicating issues that have been the focus of much recent work.

- Although more robust than their adversarial counterparts, VAEs are still subject to experiencing posterior collapse in which the decoder learns to ignore the latent memory altogether and reconstruct a fuzzy approximation of the input distribution (Goyal et al., 2017).

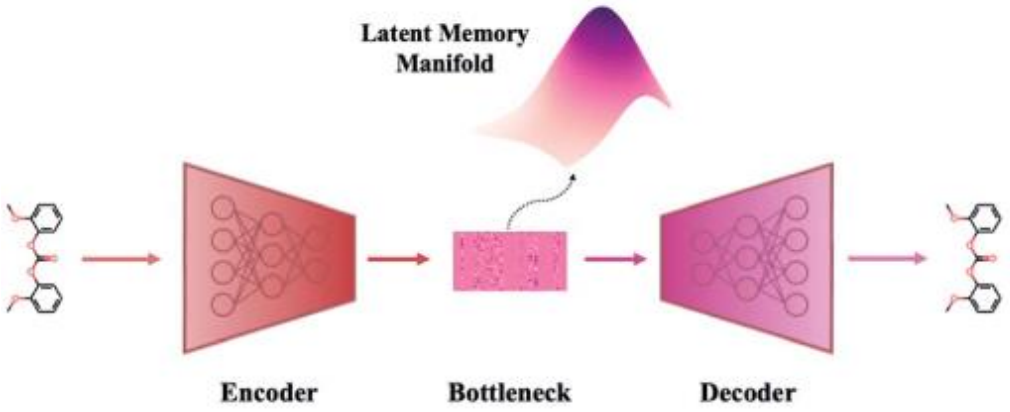


Figure from paper

Goyal et al., 2017; arXiv:1711.05411 [stat.ML]

Introduction

- On the other hand, even with a meaningful posterior there are often pockets of phase space within the latent memory that do not map to any valid chemical structures.

- Many recent innovations in architecture, featurization and hyperparameter selection have centered around these problems and have proven quite successful at improving reconstruction accuracy and sampling validity (Jin et al., 2018; Mohammadi et al., 2019; Yan et al., 2019).

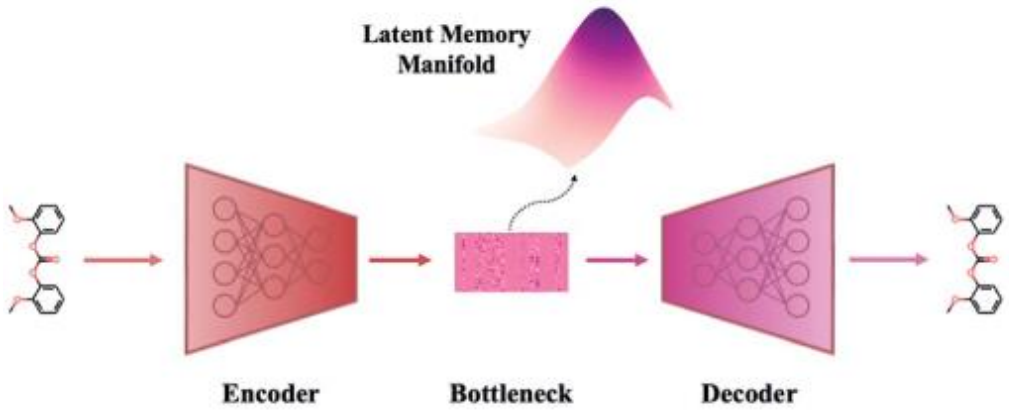


Figure from paper

Jin et al., 2018; arXiv:1802.04364 [cs.LG]
Mohammadi et al., 2019; DOI: 10.26434/chemrxiv.7977131.v2
Yan et al., 2019; arXiv, 20,1–7.

Introduction

- However, we lack a holistic view of the effect of these improvements on the practical utility of a model's latent memory. For instance, metrics to examine the diversity and novelty of sampled molecules are not well-defined (Coley, 2020).

- These traits are arguably as important as validity, if not more so. Generating samples is orders of magnitude faster than training and a model that can generalize to regions of chemical phase space far outside the training set is valuable for exploration.

- Although fewer studies have evaluated generative VAE models in this way, the results reported in the Moses benchmarking platform indicate that there is still significant room for improvement.

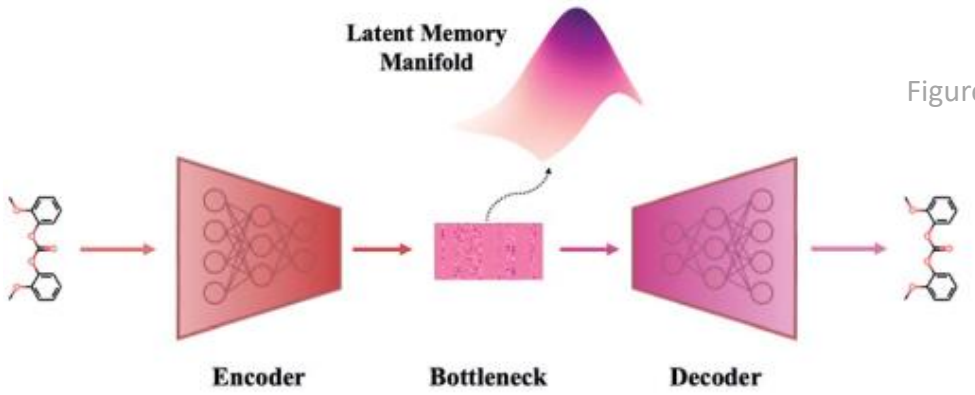


Figure from paper

Coley, 2020; Trends Chem., 3(2), 133–145.

Introduction

- Despite the overwhelming number of similarities between model architectures used for molecular generation and those used for NLP, the state-of-the-art in the former lags notably behind that of the latter.

- While attention mechanisms have been used in the field of chemistry for tasks like graph-based analyses of chemical structure (Payne et al., 2020), atom-mapping (Schwaller et al., 2021) and organic reaction predictions (Schwaller et al., 2019), they have not yet been incorporated into any context-independent generative algorithms.

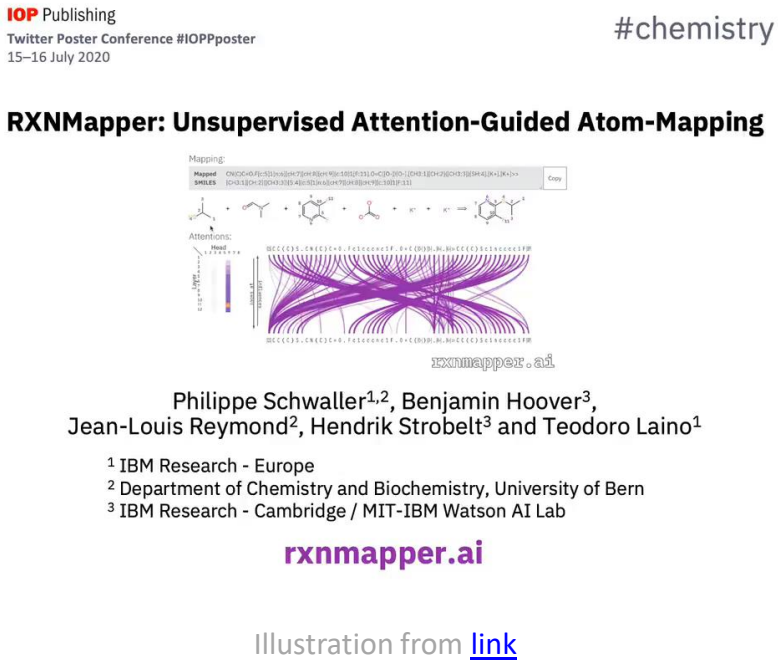


Illustration from link

Payne et al., 2020; arXiv:2007.16012 [q-bio.BM]
Schwaller et al., 2021; Sci. Adv., 7, eabe4166
Schwaller et al., 2019; ACS Cent. Sci., 5, 1572–1583

Results and discussion

Variational autoencoder and the information bottleneck

- A VAE consists of an encoder that takes a sequence as input, i.e., a SMILES string, and a decoder that attempts to reconstruct the input as accurately as possible.
- Prior to decoding, the encoder transforms the input, x, into an intermediate latent representation, z, that serves as the "model memory."
- Information is bottlenecked between the encoder and decoder such that $d_{latent} \ll d_{input}$ where d is the dimensionality of a given layer.

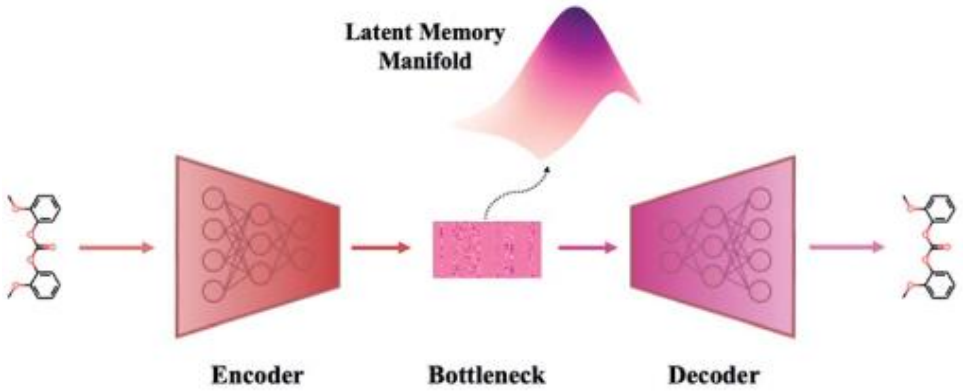


Figure from paper

Results and discussion

Variational autoencoder and the information bottleneck

- A VAE can be thought of as a compression algorithm that produces compact, information dense representations of molecular structures.

- The encoder learns how to compress the input data and the decoder learns how to reconstruct the full sequence from the compressed representation

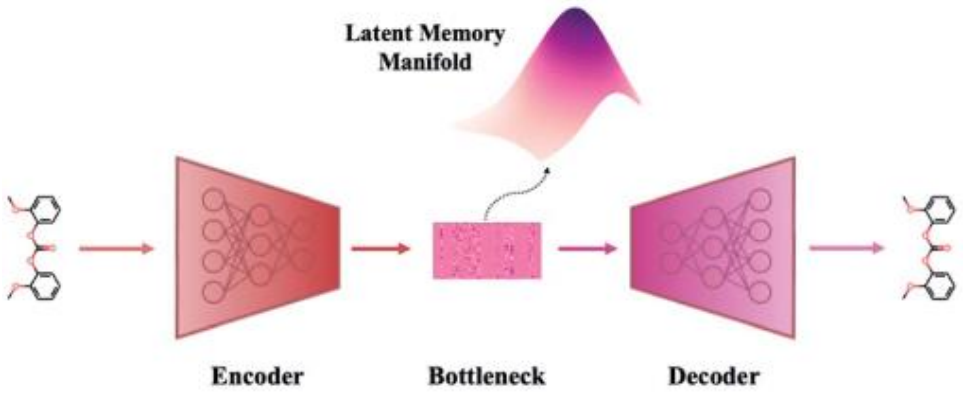


Figure from paper

Results and discussion

Variational autoencoder and the information bottleneck

- The training objective seeks to minimize the reconstruction loss between the input and output while simultaneously learning the ground truth probability distribution of the training data.

- The latter half of this objective is especially important to the generative capacity of the model. Knowledge of the marginal likelihood, $p(x|z)$, allows us to directly sample new data points by first querying from the model's memory, z, and then decoding.

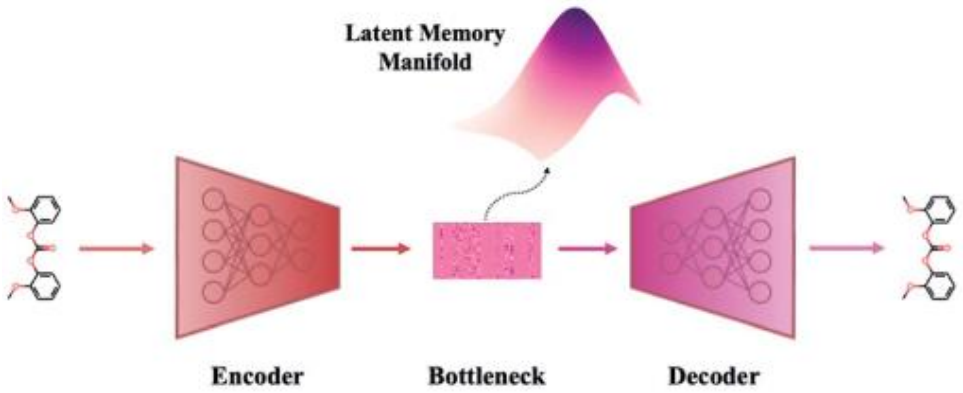


Figure from paper

Results and discussion

Variational autoencoder and the information bottleneck

- To achieve this, we assume the true posterior can be adequately approximated by a set of Gaussians.

- The Kullbach–Leibler divergence (KLD) (Kullbach&Leibler, 1951) between z and the standard normal distribution $N(0,1)$ is minimized alongside the reconstruction loss and thus the full objective function can be formalized according to the variational lower bound as

$$\log p_\theta(x|z) \geq \mathcal{L}(\theta, \emptyset; x, z) = \underbrace{\mathbb{E}_{q_\emptyset(z|x)}[log p_\theta(x|z)]}_{\text{Reconstruction loss of decoder } \boldsymbol{p_\theta(x|z)}} - \underbrace{\beta D_{KL}(q_\emptyset(z|x) \parallel p(z))}_{\text{KLD loss between the encoder output } \boldsymbol{q_\emptyset(z|x)}\text{, and the standard normal distribution } \boldsymbol{p(z)}}$$

- The KLD loss is scaled by a Lagrange multiplier, β, that controls the relative magnitude of the two terms. This architecture is known as a β -VAE and is a more general form of VAE (β = 1) (Alemi et al., 2016)

Kullbach&Leibler, 1951; Ann. Math. Stat., 22, 79 –86
Alemi et al., 2016; arXiv:1612.00410 [cs.LG].

Results and discussion

Variational autoencoder and the information bottleneck

```python
import torch
from torch.nn import functional as F


def kl_divergence_loss(means, logvars):
    """
    Calculate KL divergence given means and logvars of the Gaussians.
    Might want to refer to the original paper:
    Kingma and Welling. Auto-Encoding Variational Bayes. ICLR, 2014
    """
    kl_divergence_loss = -0.5 * torch.sum(1 + logvars - means.pow(2) - logvars.exp())

    return kl_divergence_loss


def mse_reconstruction_loss(x_recons, x):
    """ Calculate the standard MSE reconstruction loss between images and reconstructions. """
    mse_reconstruction_loss = F.mse_loss(x_recons, x.view(-1, 784))
    #mse_reconstruction_loss = mse_reconstruction_loss(x_recons, x)
    return mse_reconstruction_loss


def vae_loss(x, x_recons, means, logvars):
    return mse_reconstruction_loss(x, x_recons) + kl_divergence_loss(means, logvars)

```

$$\log p_\theta(x|z) \geq \mathcal{L}(\theta, \emptyset; x, z) = \underbrace{\mathbb{E}_{q_\emptyset(Z|X)}[log p_\theta(x|z)]}_{\text{Reconstruction loss of decoder } p_\theta(x|z)} - \underbrace{\beta D_{KL}(q_\emptyset(z|x) \parallel p(z))}_{\text{KLD loss between the encoder output } q_\emptyset(z|x), \text{ and the standard normal distribution } p(z)}$$

Introduction

Adding attention to the VAE

- In standard RNNs, the first recurrent cell takes the first element of the sequence and outputs a hidden state.
- That hidden state is then propagated down the sequence with each subsequent recurrent cell taking the previous cell's hidden output and the next sequence element as inputs until the entire sequence has been traversed.
- The final hidden state is the “contextual embedding” of the sequence.

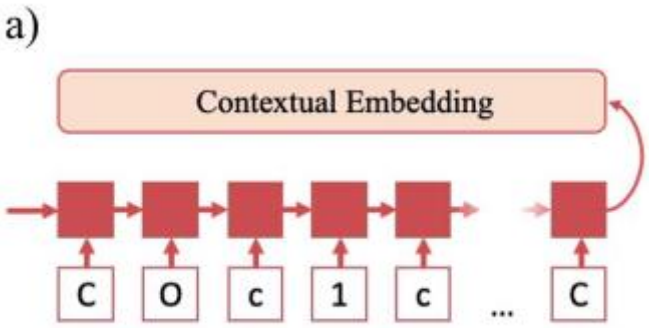
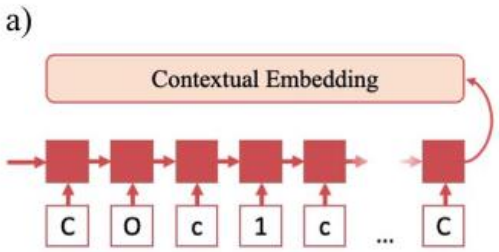


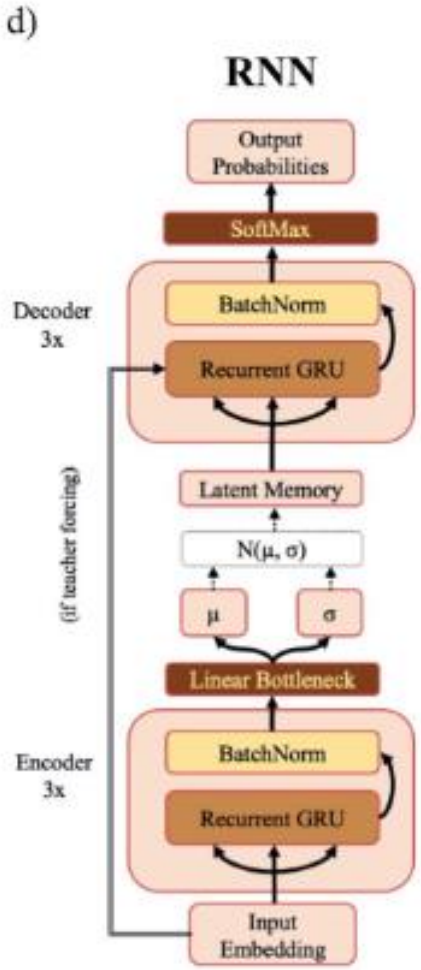
Figure from paper

Introduction

Adding attention to the VAE

- In some architectures the contextual embedding and the latent memory may be the same size.
- However, oftentimes there will be an additional set of linear bottleneck layers that further compress the output of the encoder GRU layers $d_{encoder} \longrightarrow d_{latent}$





Figures from paper

Introduction

Adding attention to the VAE

- In attention-based recurrent models (RNNAttn), the flow of information proceeds similarly to a standard RNN.
- However rather than only using the final hidden output state, a weighted combination of all the hidden states along the sequence is used as the contextual embedding

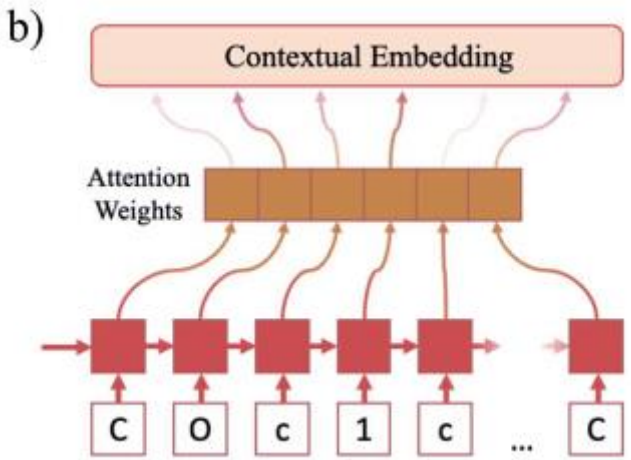
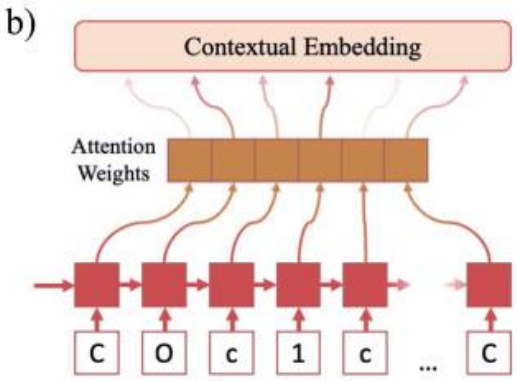


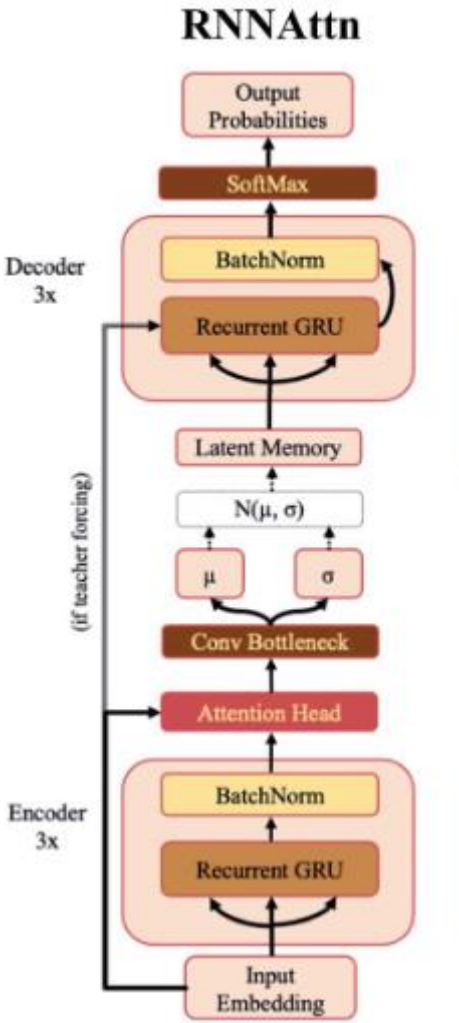
Figure from paper

Introduction

Adding attention to the VAE

- The attention weights are learned during training by letting the input sequence “attend” to its own hidden state matrix.
- This allows the model to eschew the linearity imposed by the RNN architecture and learn long-range dependencies between sequence elements.



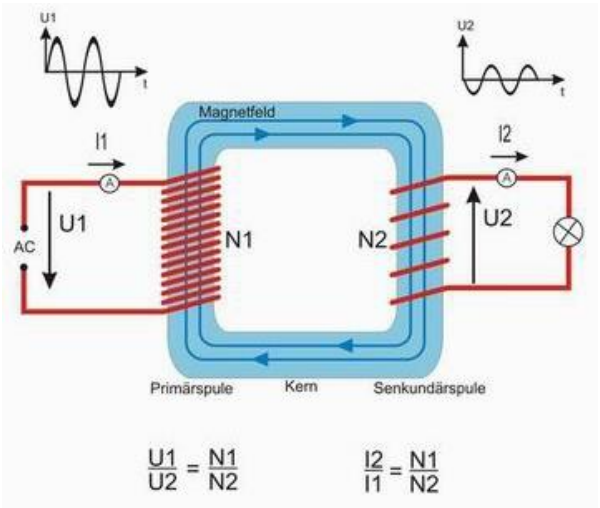


Figures from paper

Results and discussion

Adding attention to the VAE

Transformers when I was/am at:



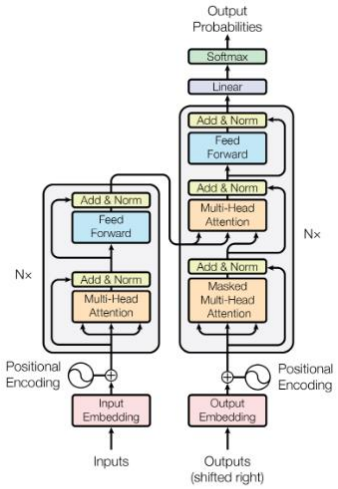


Figure 1: The Transformer - model architecture.

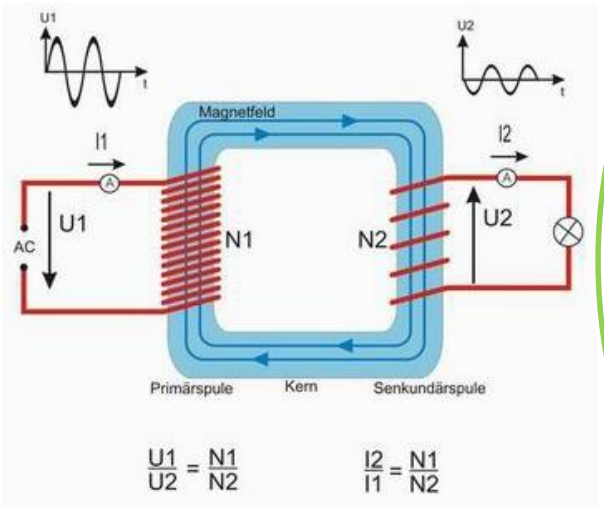
Primary School **High School** **University**

Results and discussion

Adding attention to the VAE

Time to play with you ☺

Transformers when I was/am at:



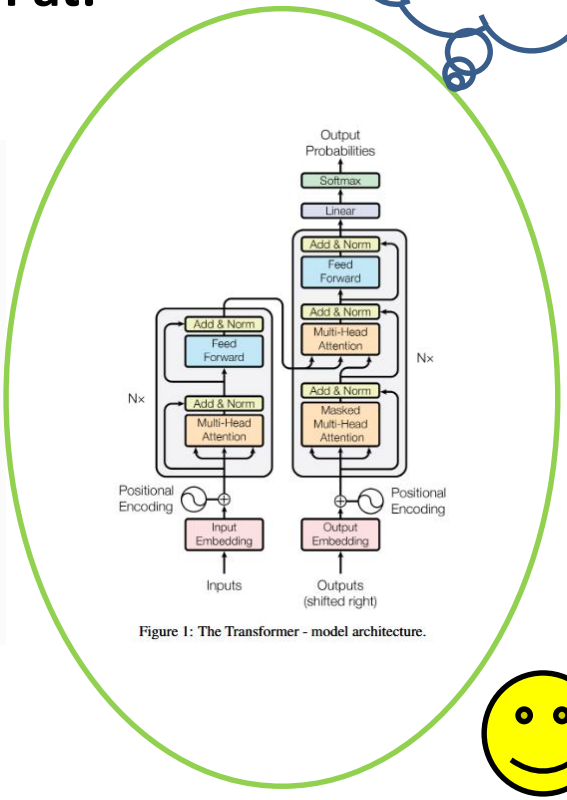


Figure 1: The Transformer - model architecture.

☺

Primary School **High School** **University**

Results and discussion

Adding attention to the VAE

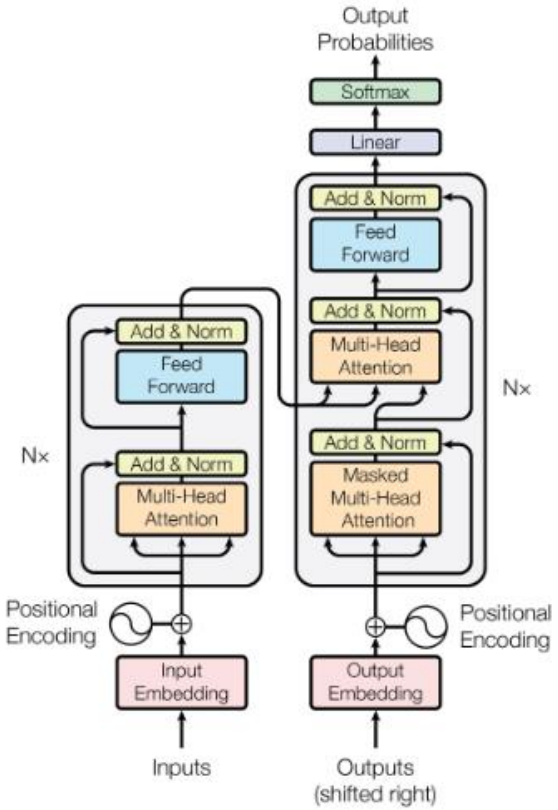


Figure 1: The Transformer - model architecture.

Vaswani et al., 2017; Attention is All You Need

Figure from link

Results and discussion

Adding attention to the VAE

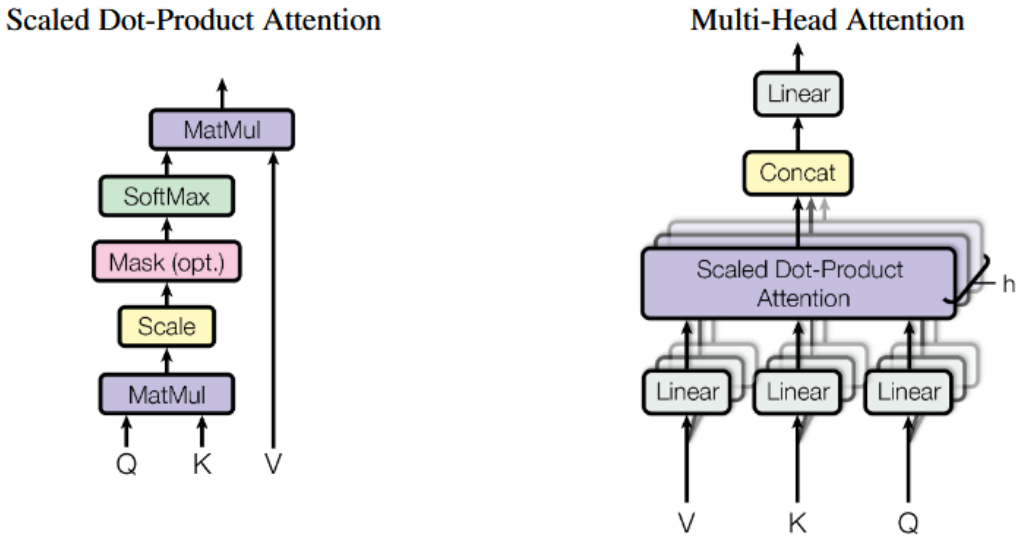


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Vaswani et al., 2017; Attention is All You Need

Figure from link

Results and discussion

Adding attention to the VAE

- Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$MultiHead(Q, K, V) = Concat(head_1, .., head_h)W^o$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$

- Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_k}$ and $W_i^O \in \mathbb{R}^{d_{model} \times d_k}$.

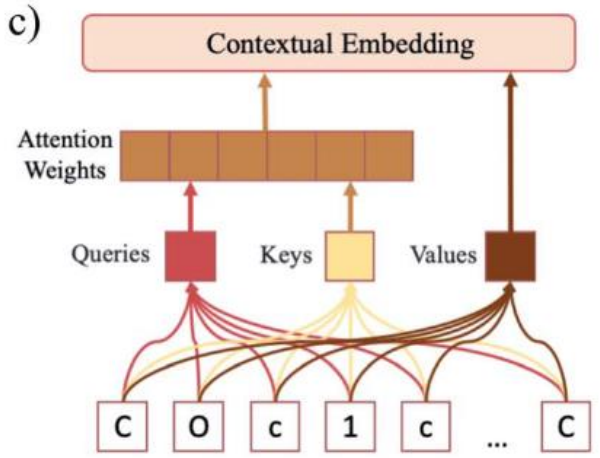
Vaswani et al., 2017; Attention is All You Need

Results and discussion

Adding attention to the VAE

- Transformer (Trans) models remove recurrence altogether and exclusively use attention head layers (Vaswani et al., 2017).

- The inputs are a set of keys, values and queries transformed from the initial input sequence that are sent through a series of matrix multiplications to calculate the attention weights and the contextual embedding.



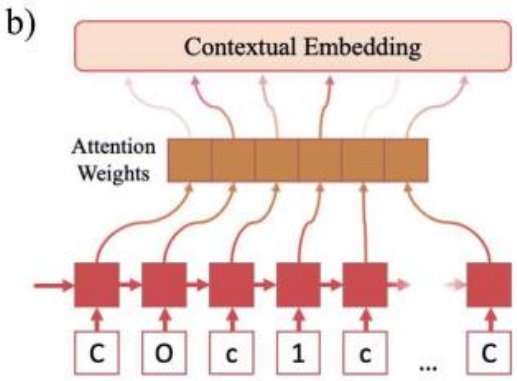
Vaswani et al., 2017; Attention is All You Need

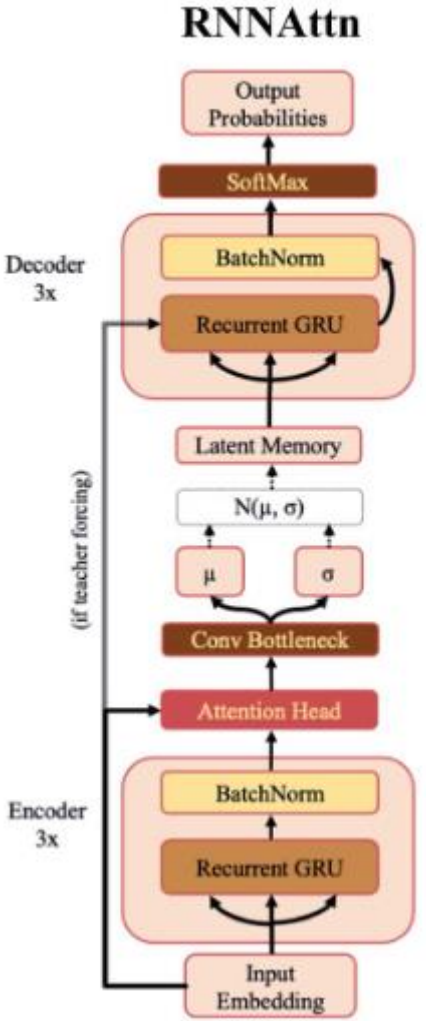
Figure from paper

Results and discussion

Adding attention to the VAE

- The attention weights are learned during training by letting the input sequence “attend” to its own hidden state matrix.
- This allows the model to eschew the linearity imposed by the RNN architecture and learn long-range dependencies between sequence elements.





Figures from paper

Results and discussion

Adding attention to the VAE

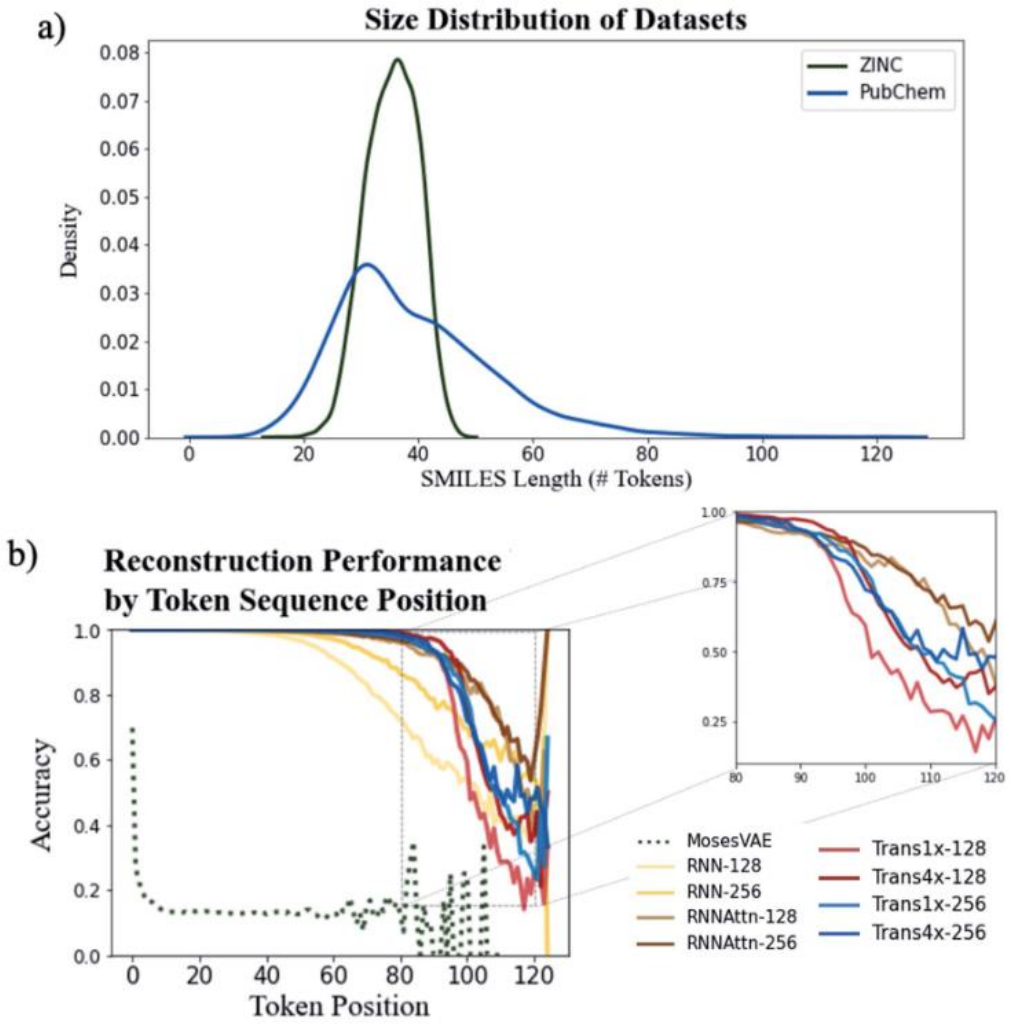
Table 1 Model architectures. The dimensionality of the model (d_{model}) is defined as the size of the sequential layers. Recurrent model names are written as ModelType-{d_{model}}. Transformer model names are written as Trans{$d_{feedforward}/d_{model}$}$x$ − {d_{model}}. All models used in this study have a latent dimensionality of size 128

Model type	d_{model}	d_{latent}	$d_{feedfoward}$
RNN-128	128	128	n/a
RNN-256	256	128	n/a
RNNAttn-128	128	128	n/a
RNNAttn-256	256	128	n/a
Trans1x-128	128	128	128
Trans4x-128	128	128	512
Trans1x-256	256	128	256
Trans4x-256	256	128	1024

Table from paper

Results and discussion

Impact of attention



Assessing model reconstruction performance on the PubChem dataset (trained for 60 epochs). Input data molecular size distributions (a) and reconstruction accuracies for all model types as a function of the token position (b). Zoomed comparison of attention-based models (inset).

Figure from paper

Results and discussion

Table 2 Comparison of generative metrics for all models with a random sampling scheme. Reconstruction accuracy is calculated based on the models ability to predict every token within a single SMILES string with 100% accuracy

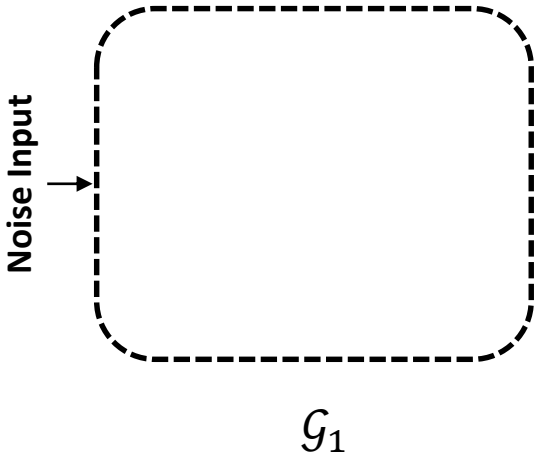
Model type	Entropy (nats)	% Reconstruction accuracy (ZINC)	% Validity	% Novelty	Cross diversity
MosesVAE	127.4	0.000	**0.976**	0.696	0.213
RNN-128	453.9	0.996	0.475	0.996	0.516
RNN-256	458.7	0.996	0.846	0.988	0.459
RNNAttn-128	393.4	0.996	0.672	**0.999**	**0.548**
RNNAttn-256	383.2	0.995	0.851	0.995	0.492
Trans1x-128	**576.3**	**0.998**	0.227	0.998	0.538
Trans4x-128	546.4	**0.998**	0.365	0.998	0.530
Trans1x-256	556.6	**0.998**	0.424	0.995	0.502
Trans4x-256	529.5	**0.998**	0.567	0.996	0.503

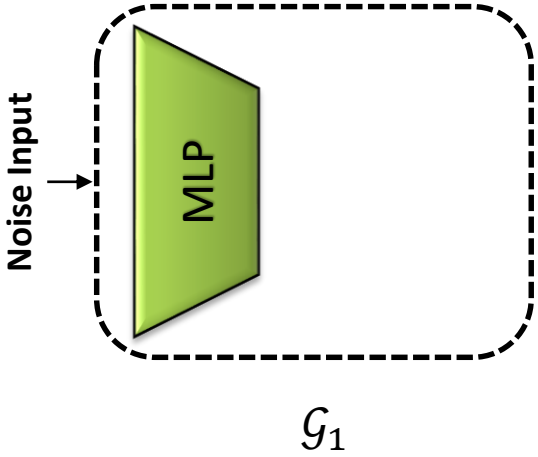
Table from paper

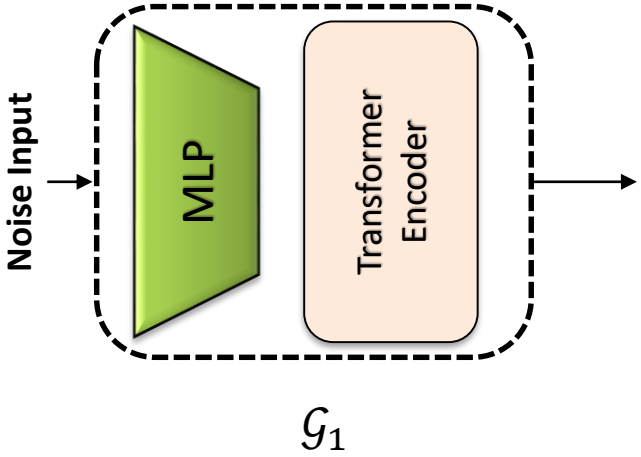
MolecularTransGAN: Model Architecture of DrugGEN

Presented by
Ahmet Sarigun

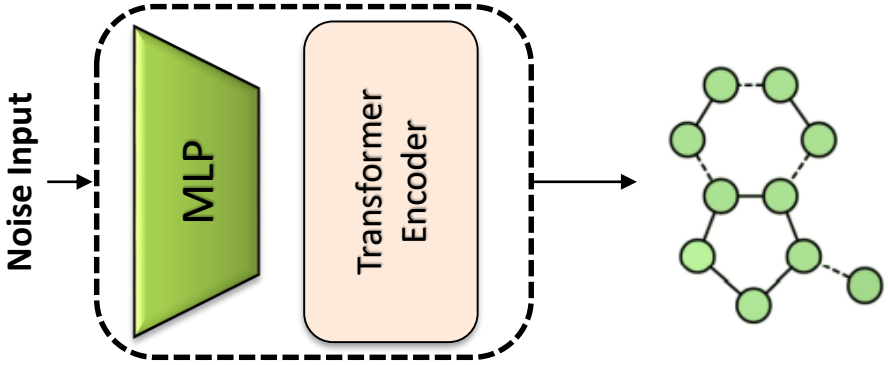
Noise Input


Noise Input
$\mathcal{G}_1$

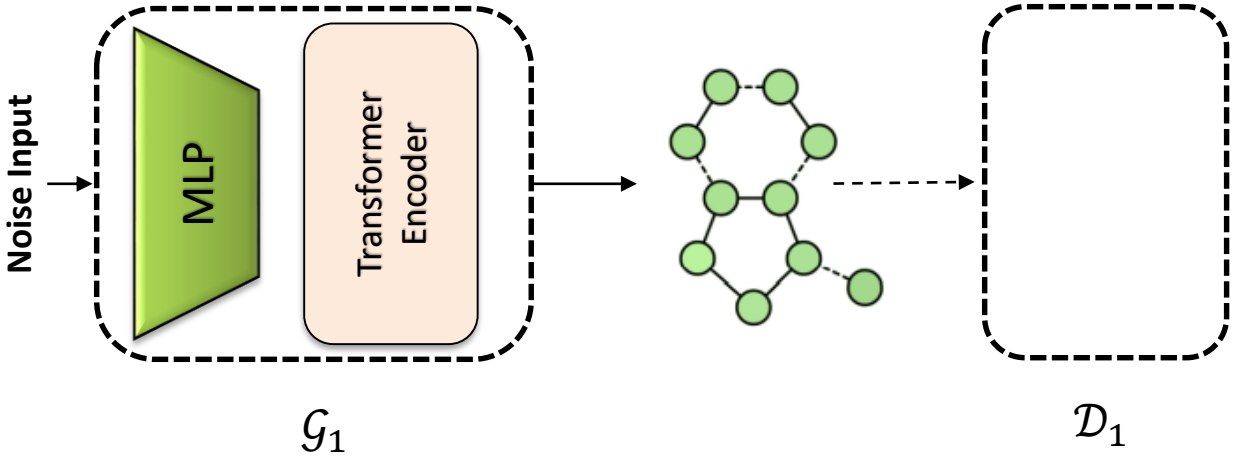

Noise Input
MLP
$\mathcal{G}_1$

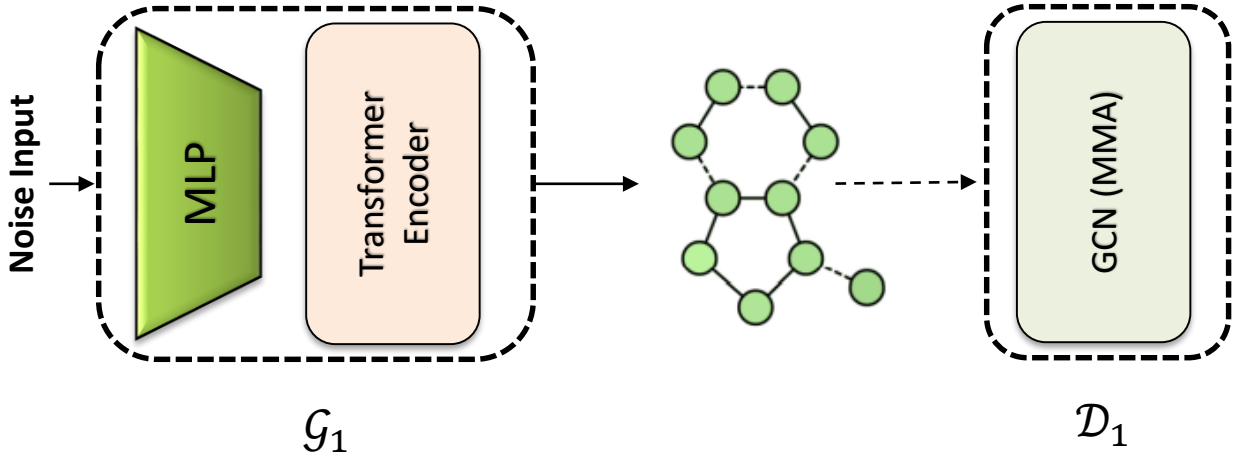


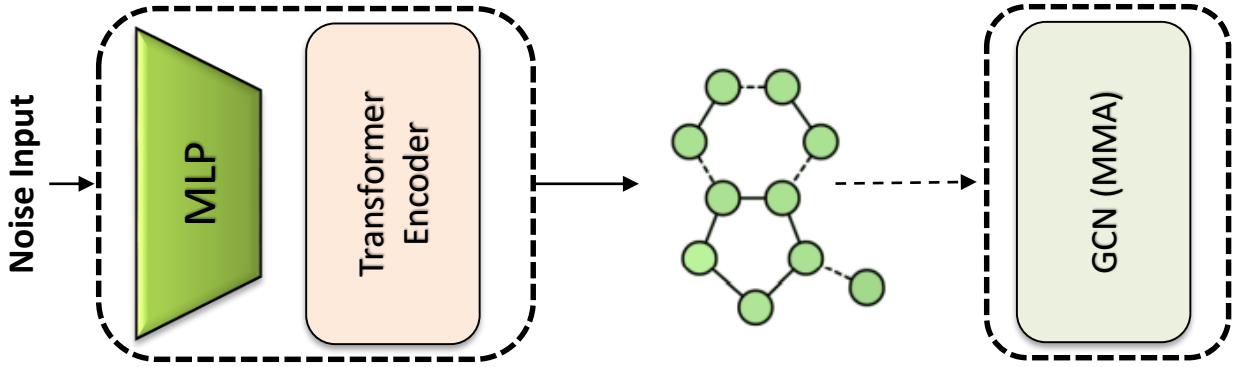
$\mathcal{G}_1$



$\mathcal{G}_1$

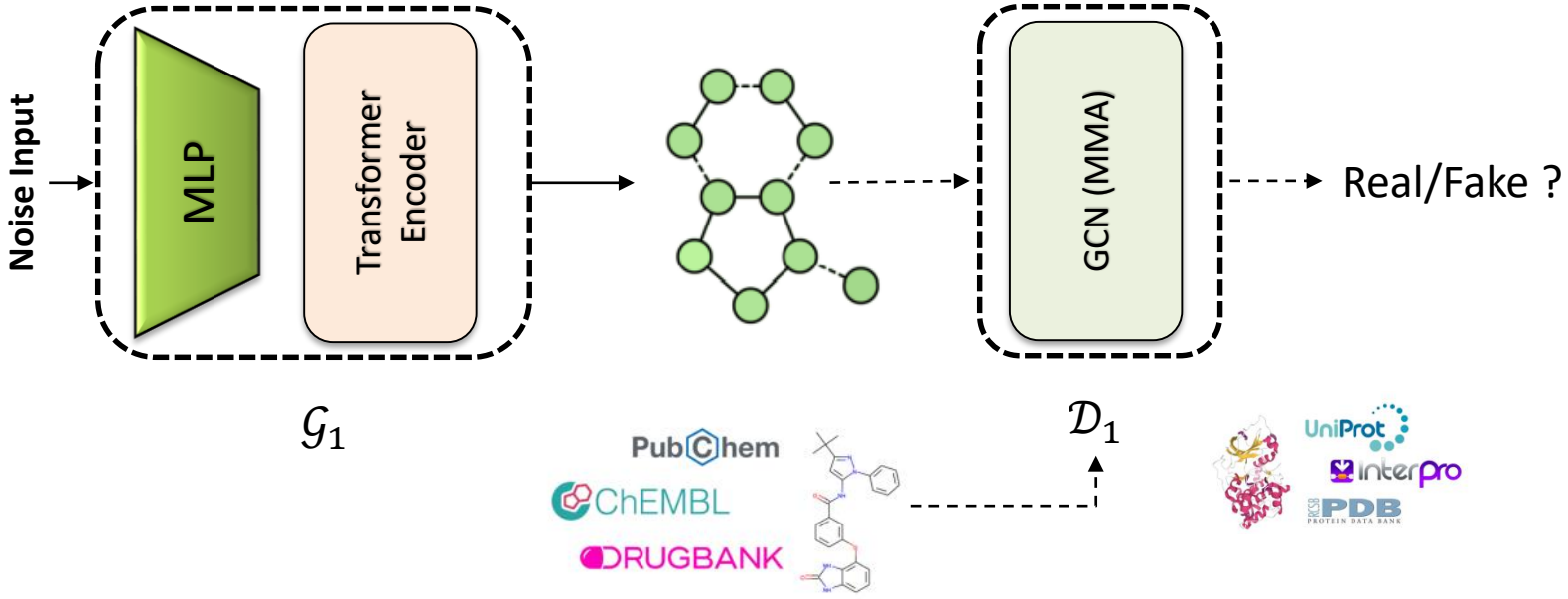

Noise Input
MLP
Transformer Encoder
$\mathcal{G}_1$
$\mathcal{D}_1$

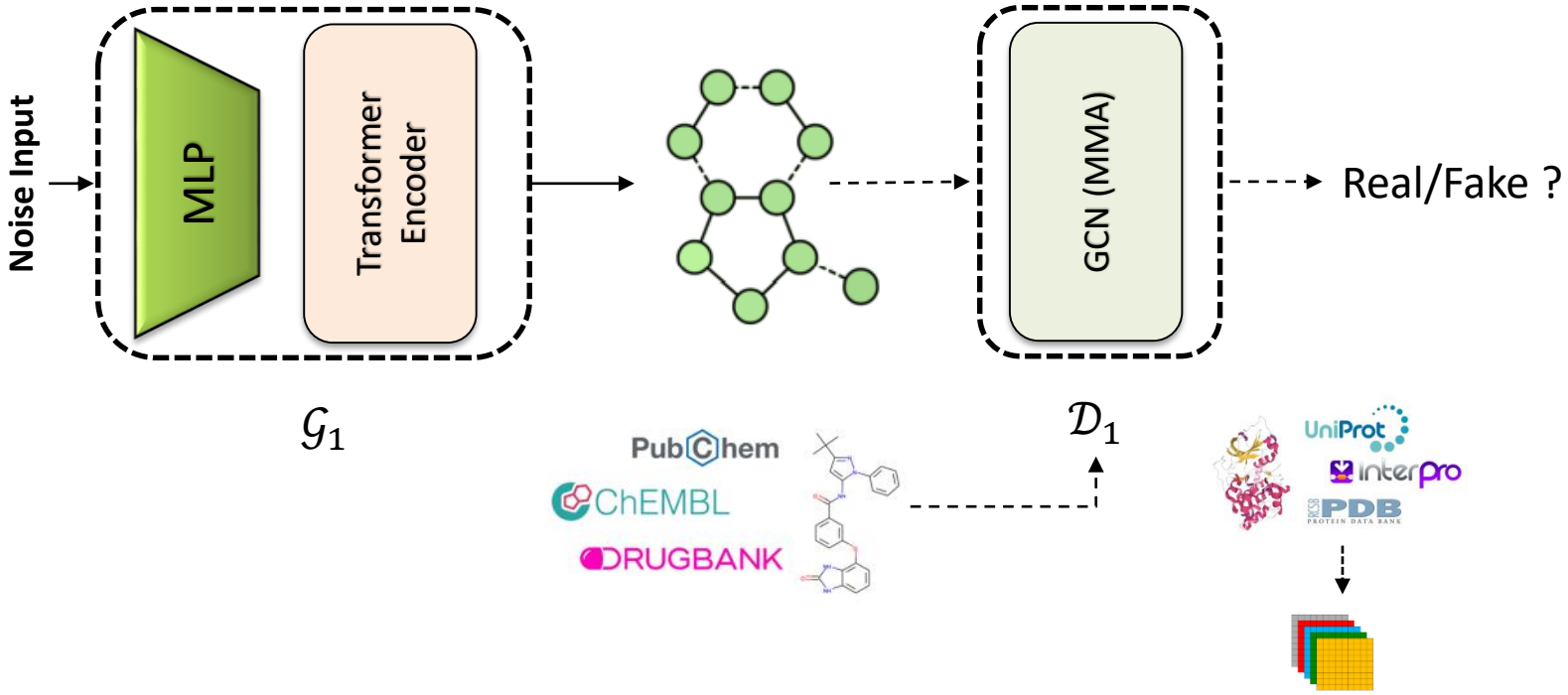

Noise Input
MLP
Transformer Encoder
GCN (MMA)
$\mathcal{G}_1$
$\mathcal{D}_1$

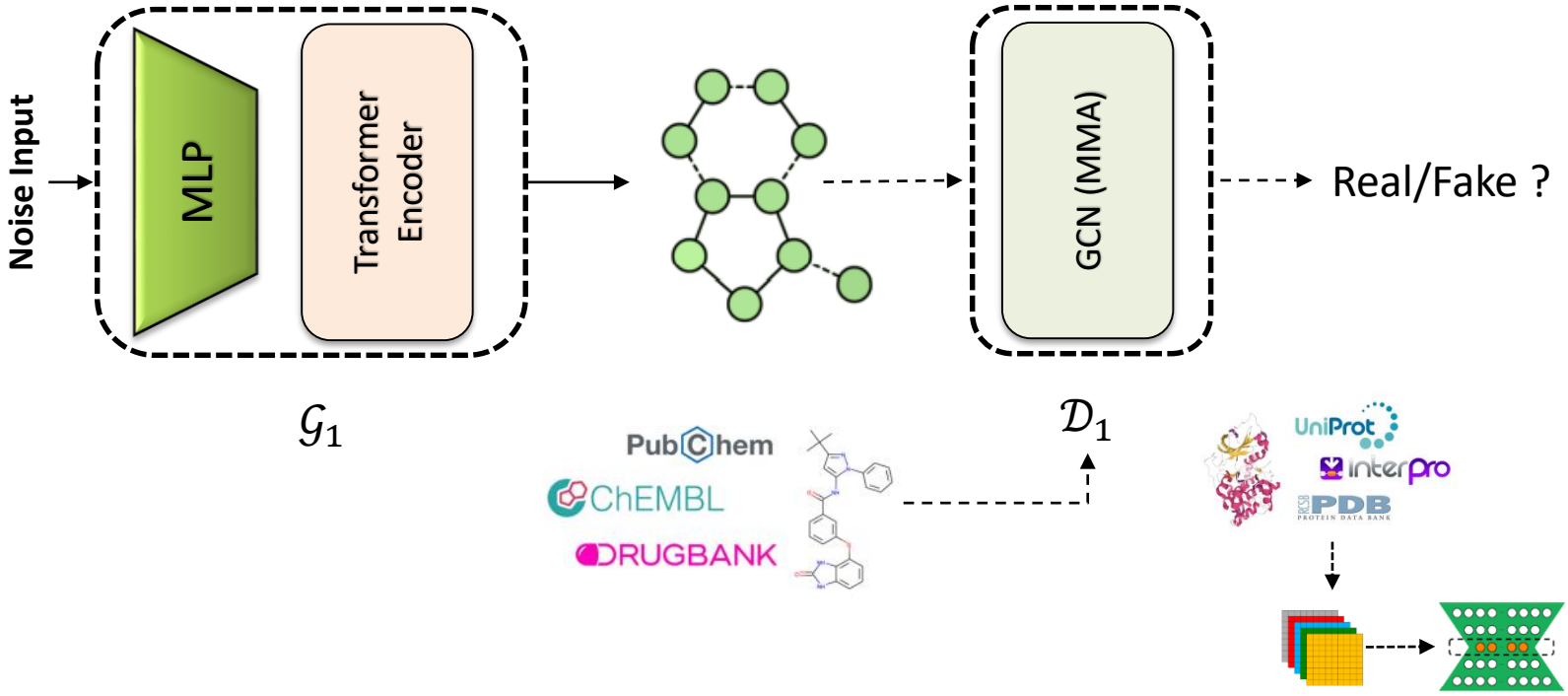


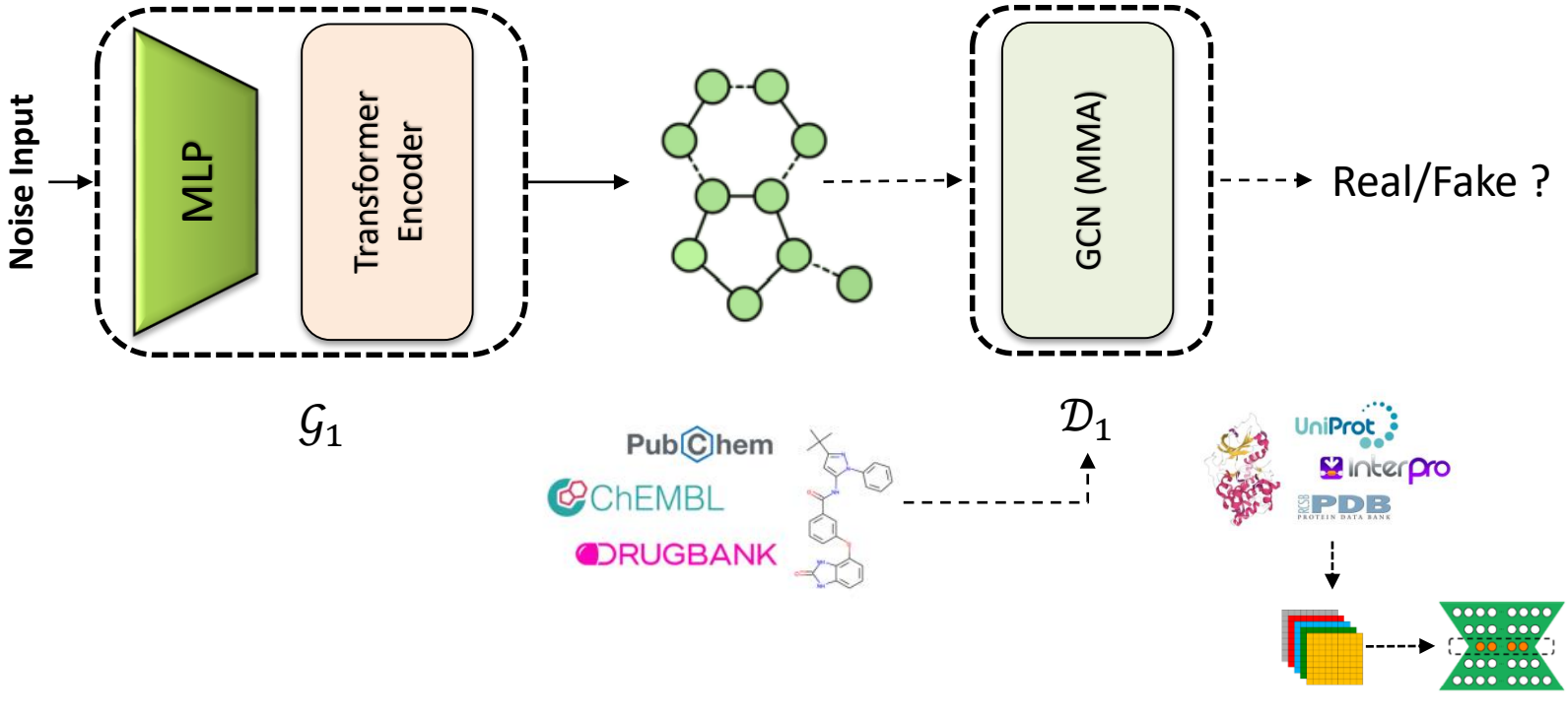
$\mathcal{G}_1$

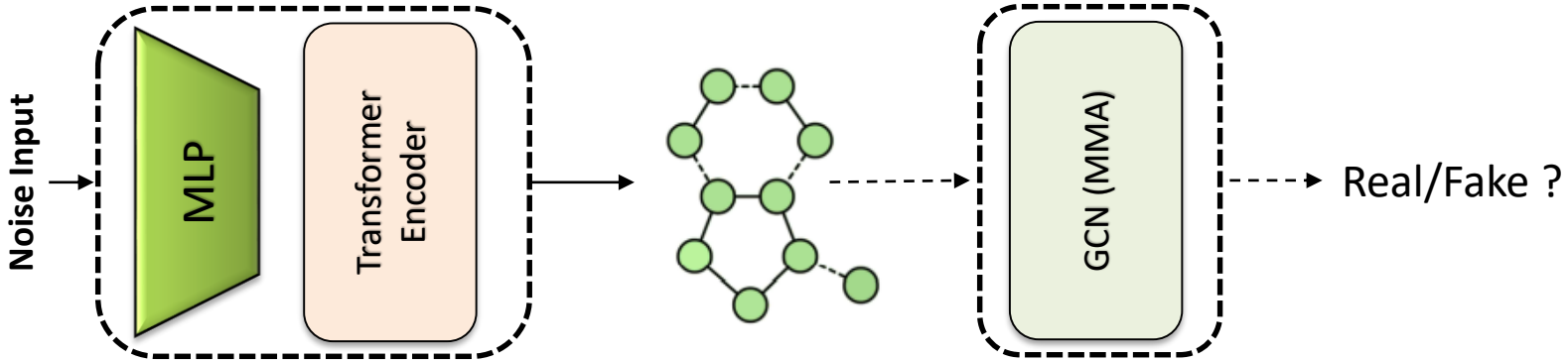
$\mathcal{D}_1$


Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB


Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB

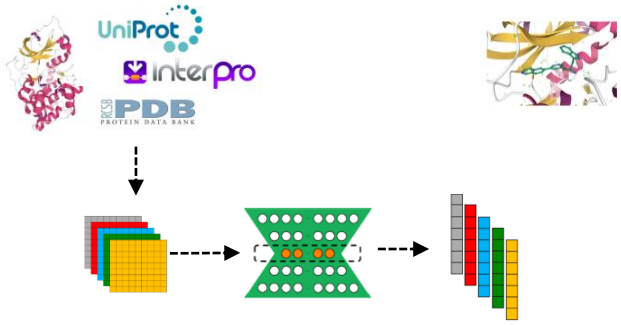

Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB

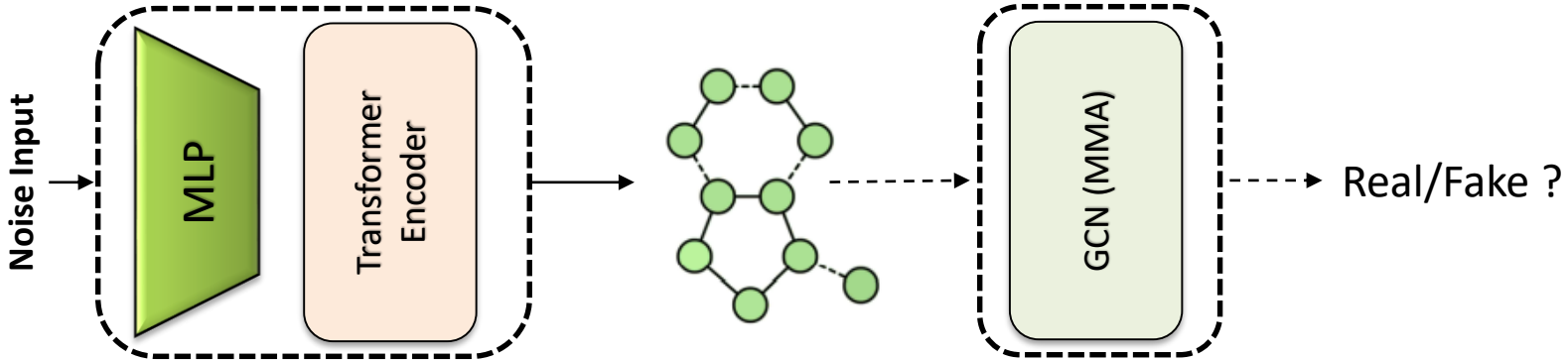

Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB



$\mathcal{G}_1$

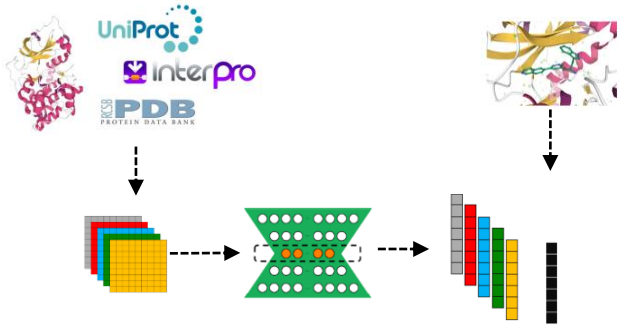
$\mathcal{D}_1$

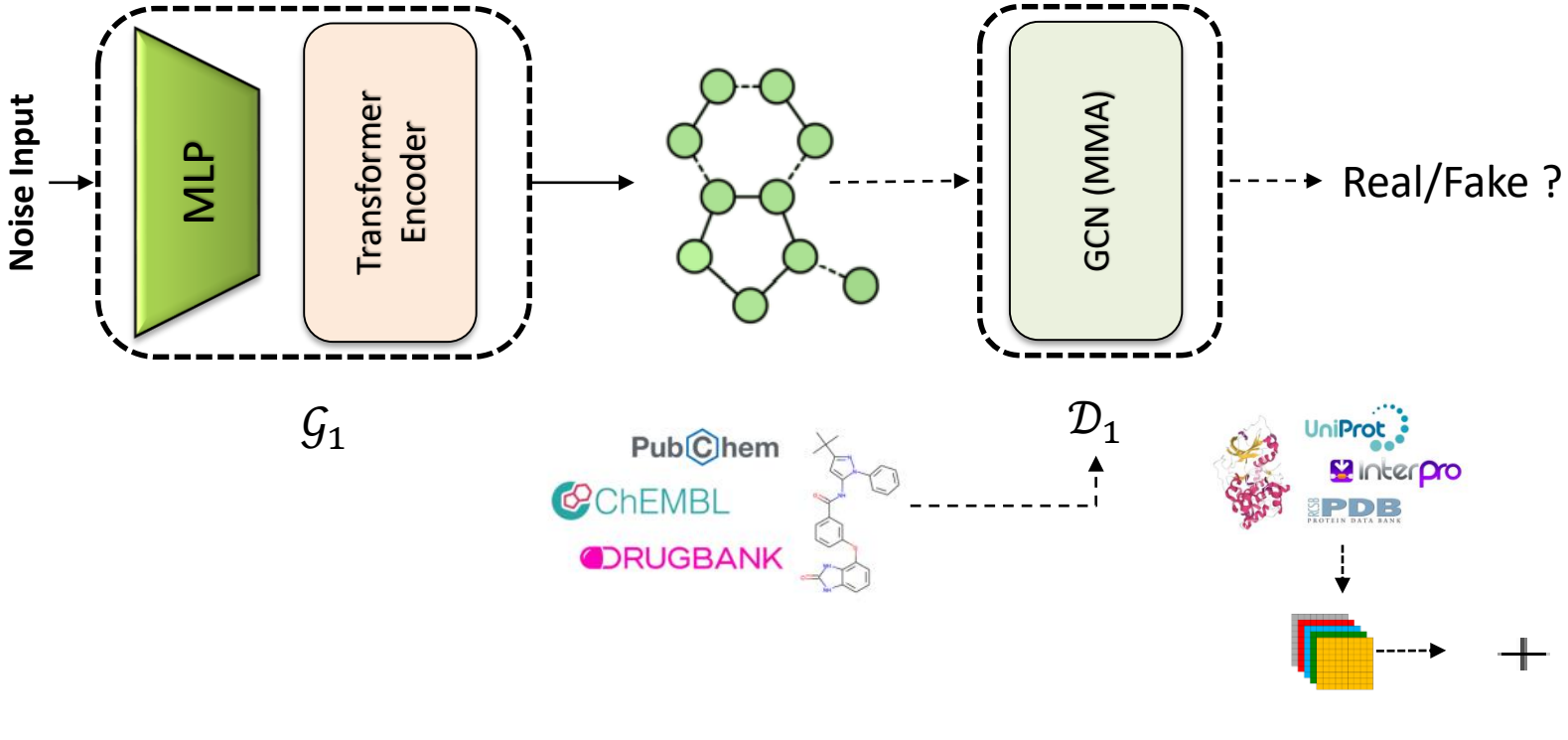


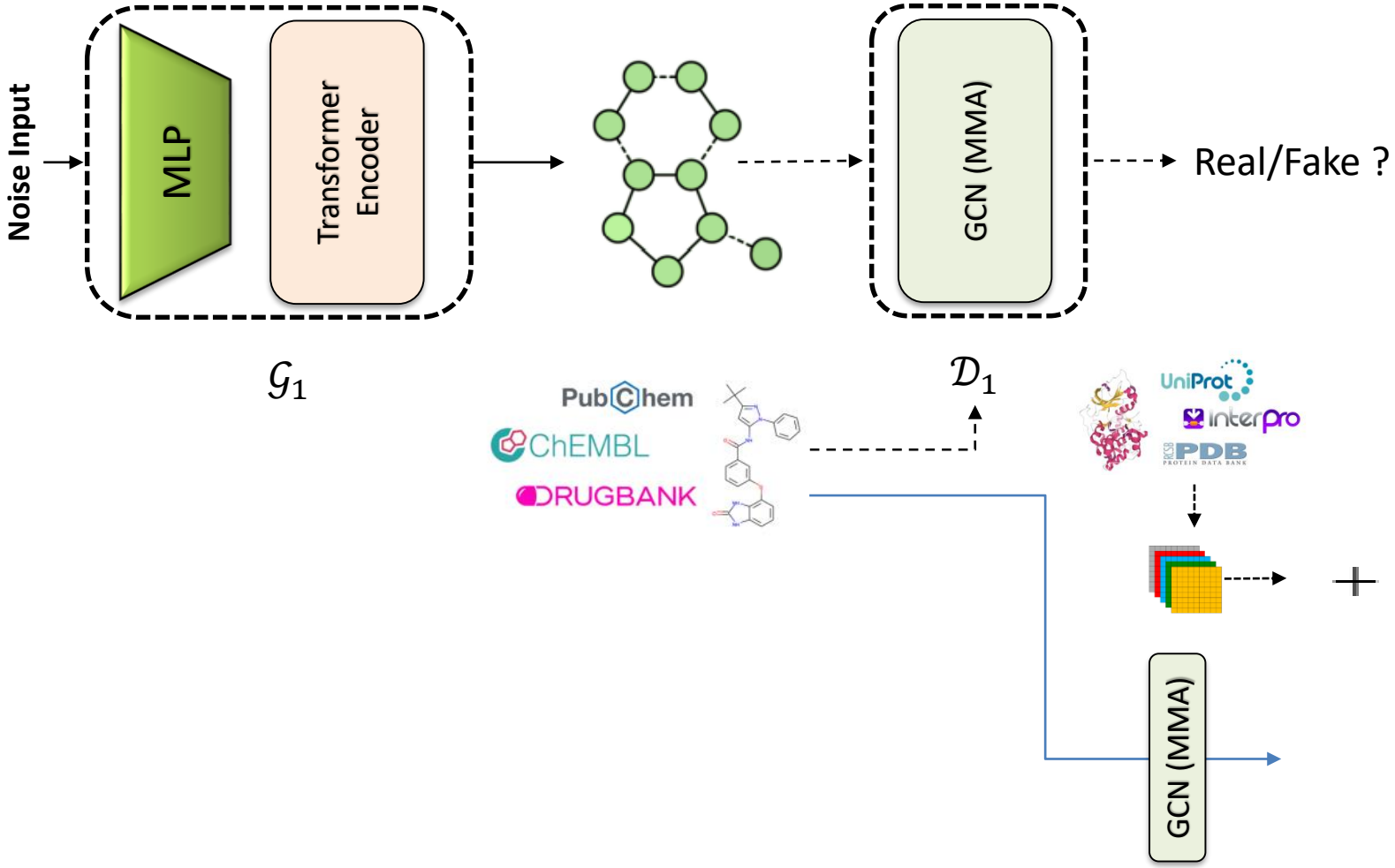


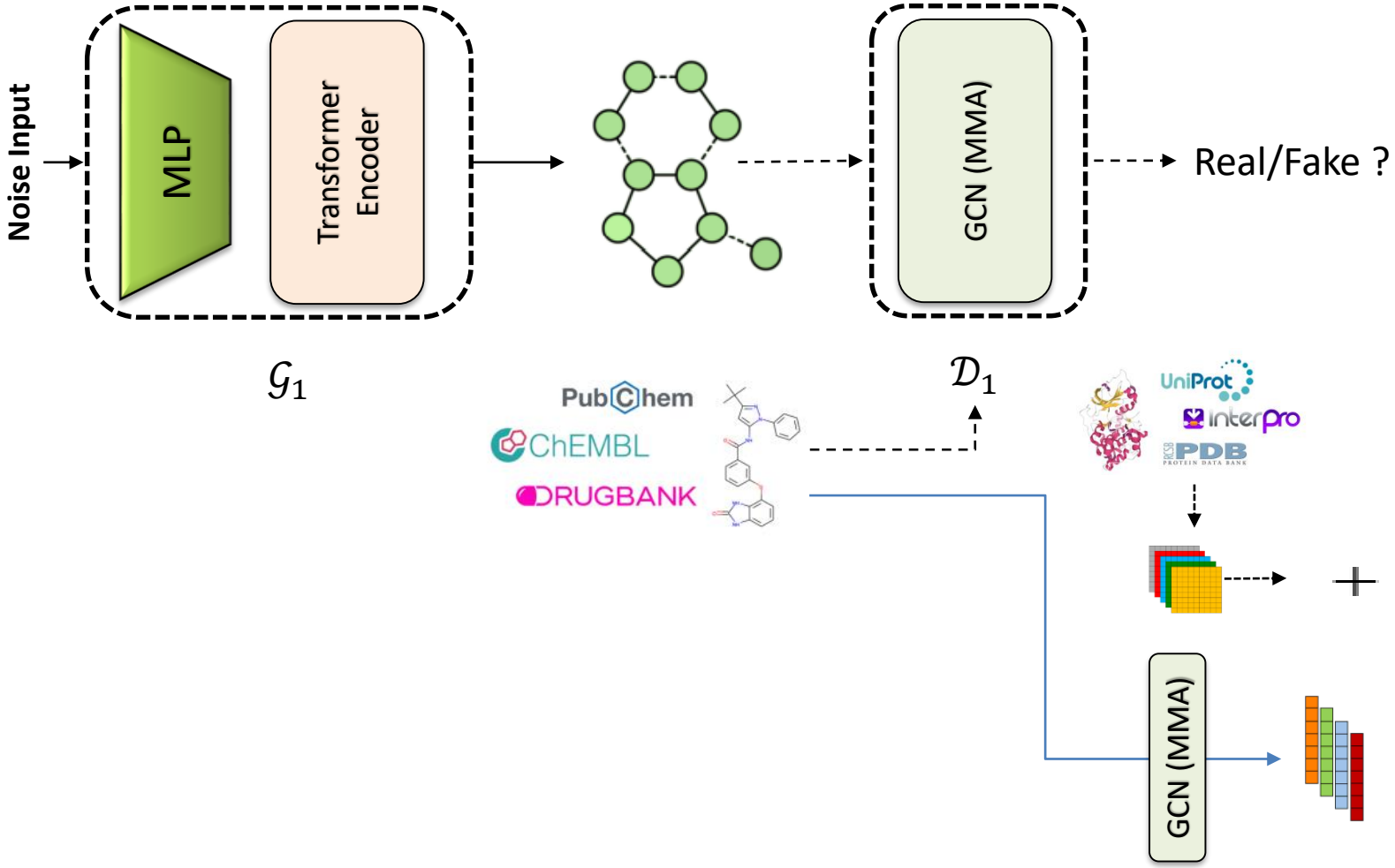
$\mathcal{G}_1$

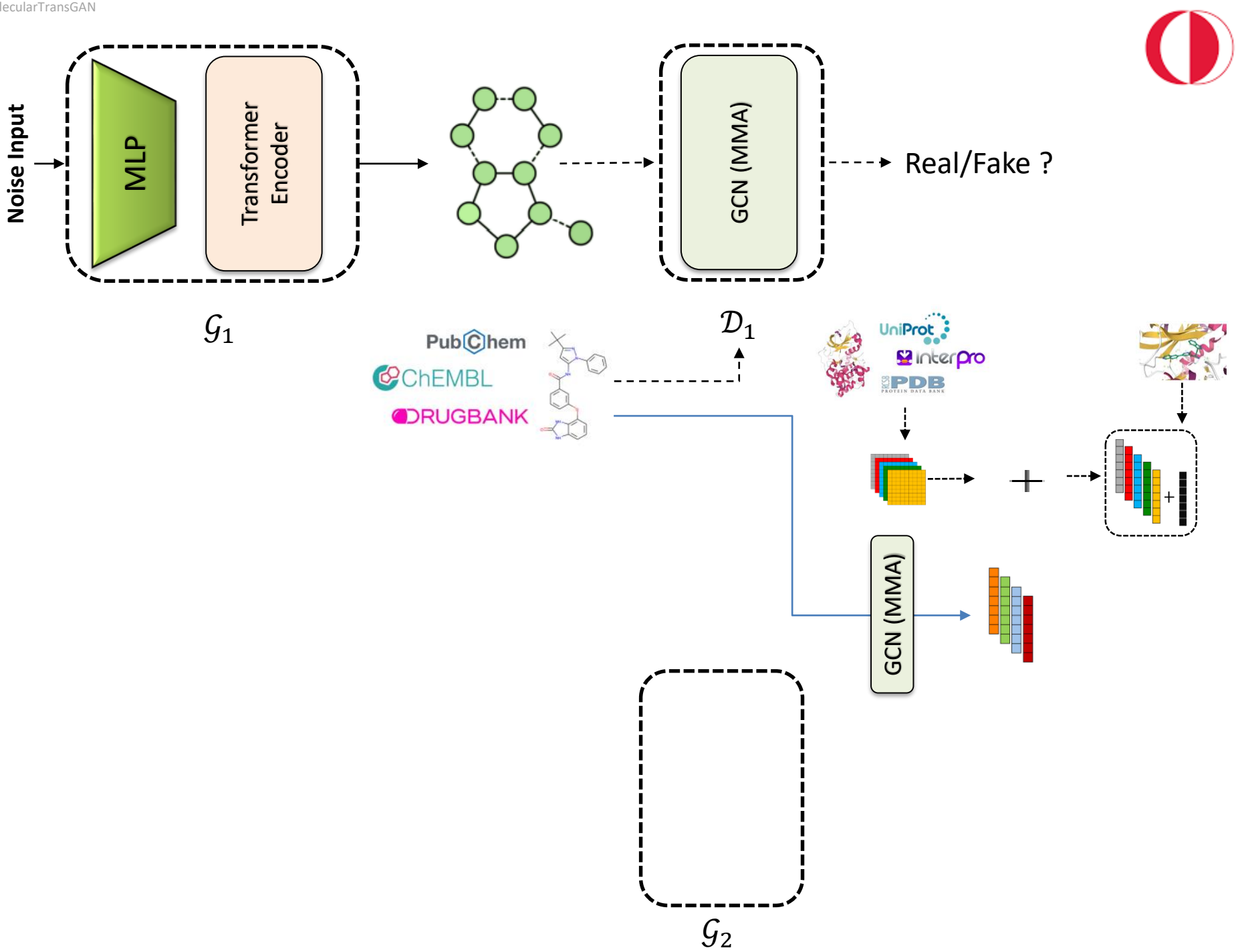
$\mathcal{D}_1$

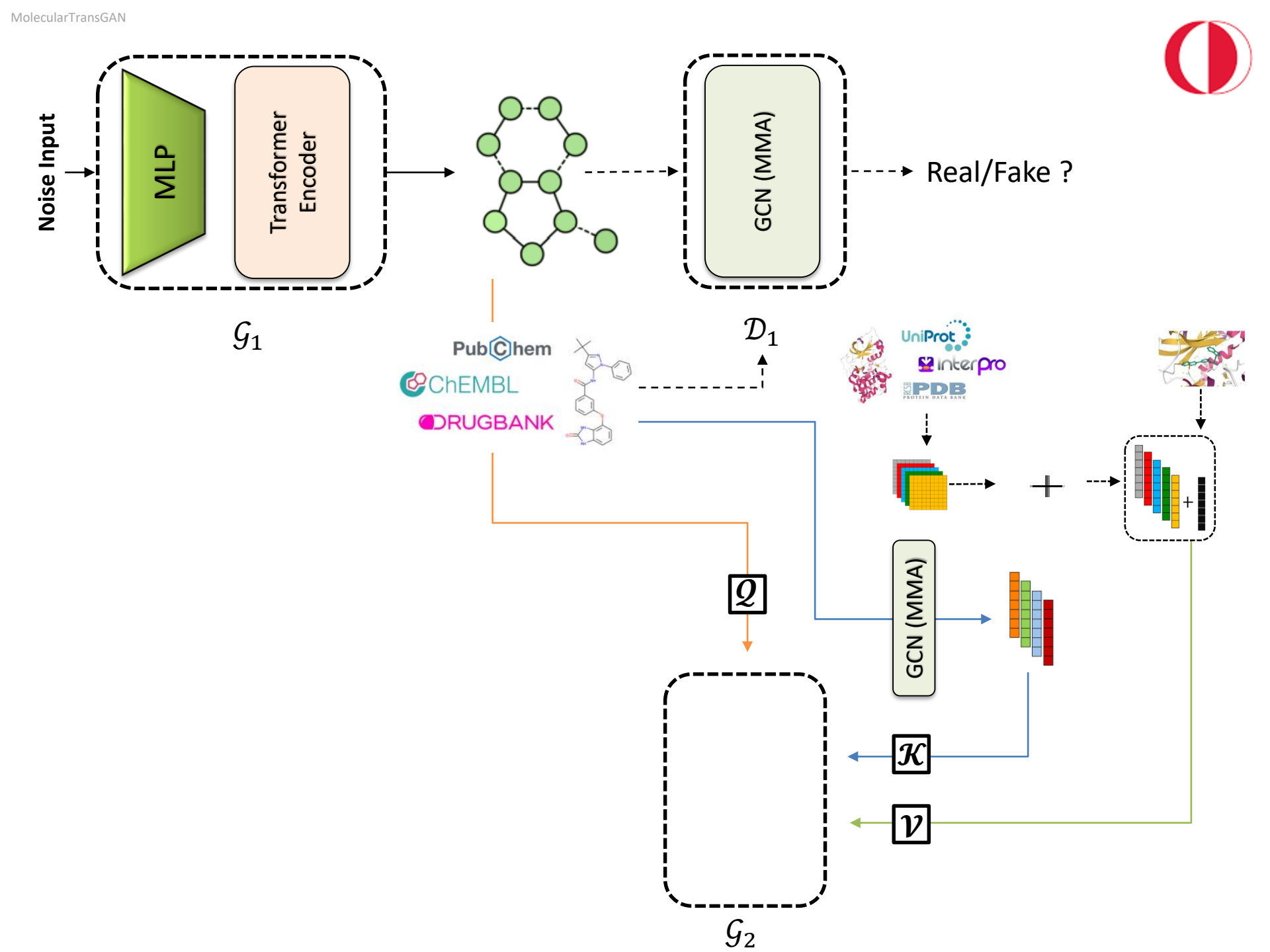


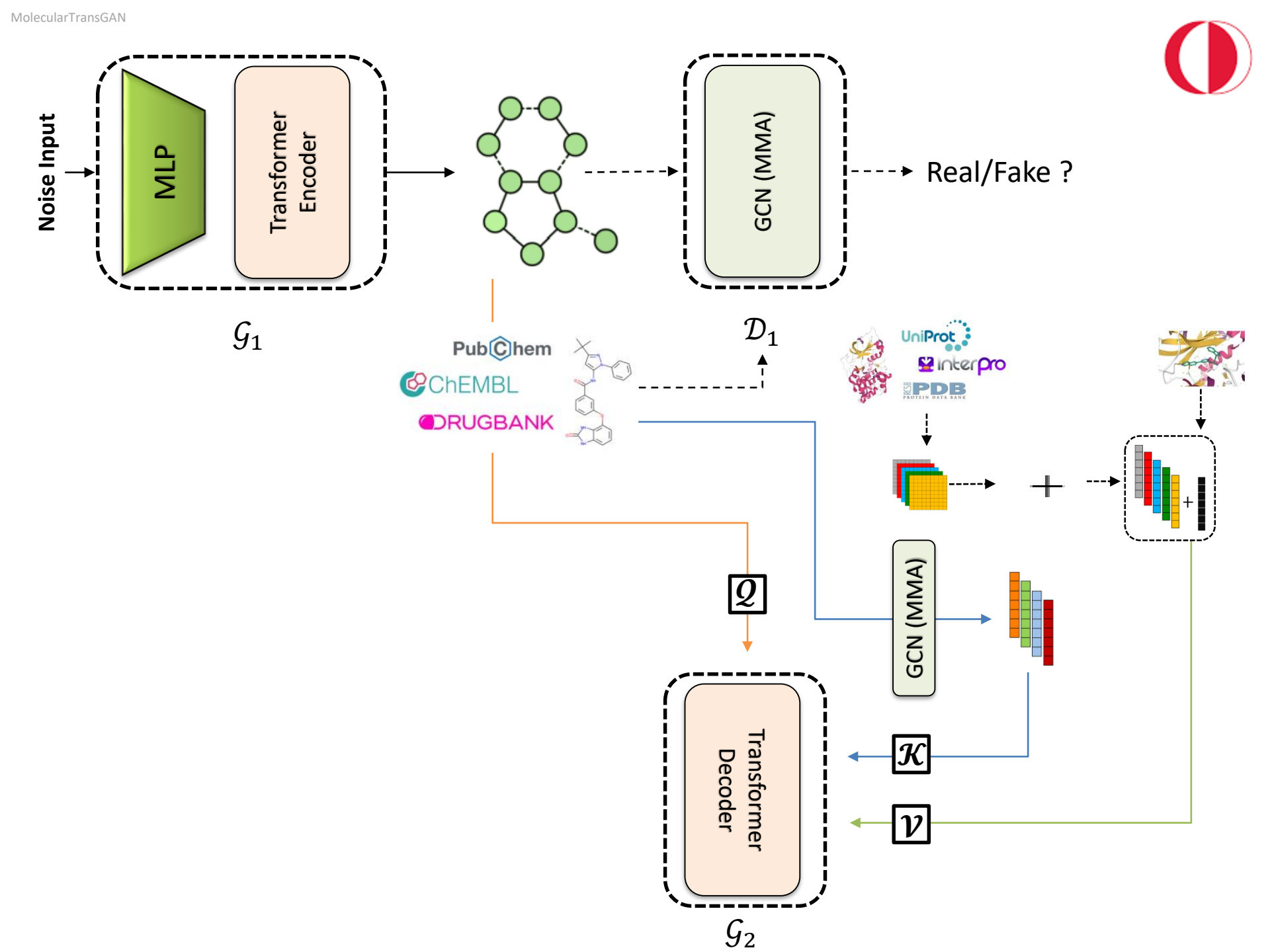

Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB

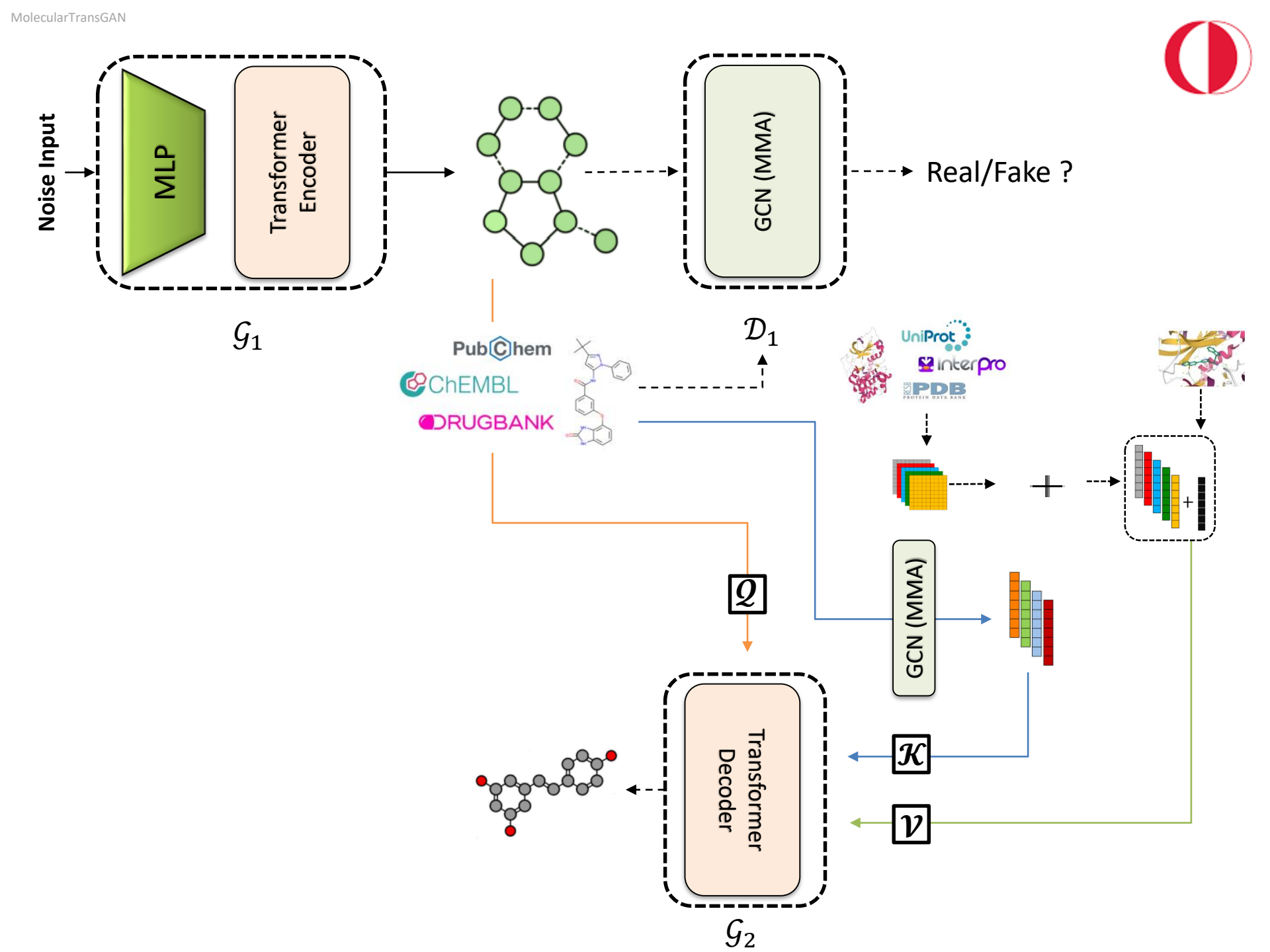

Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB
GCN (MMA)

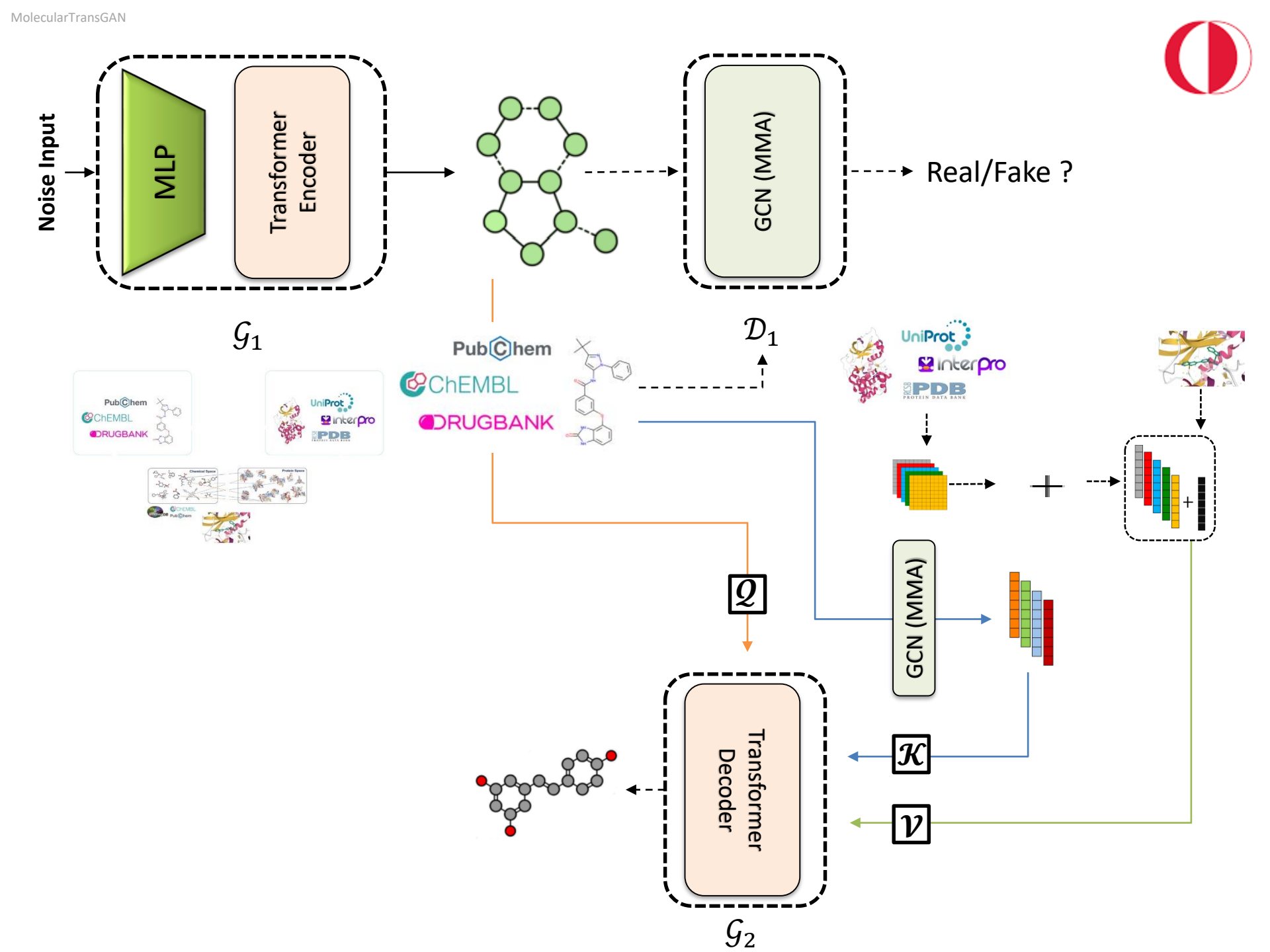

Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB
GCN (MMA)

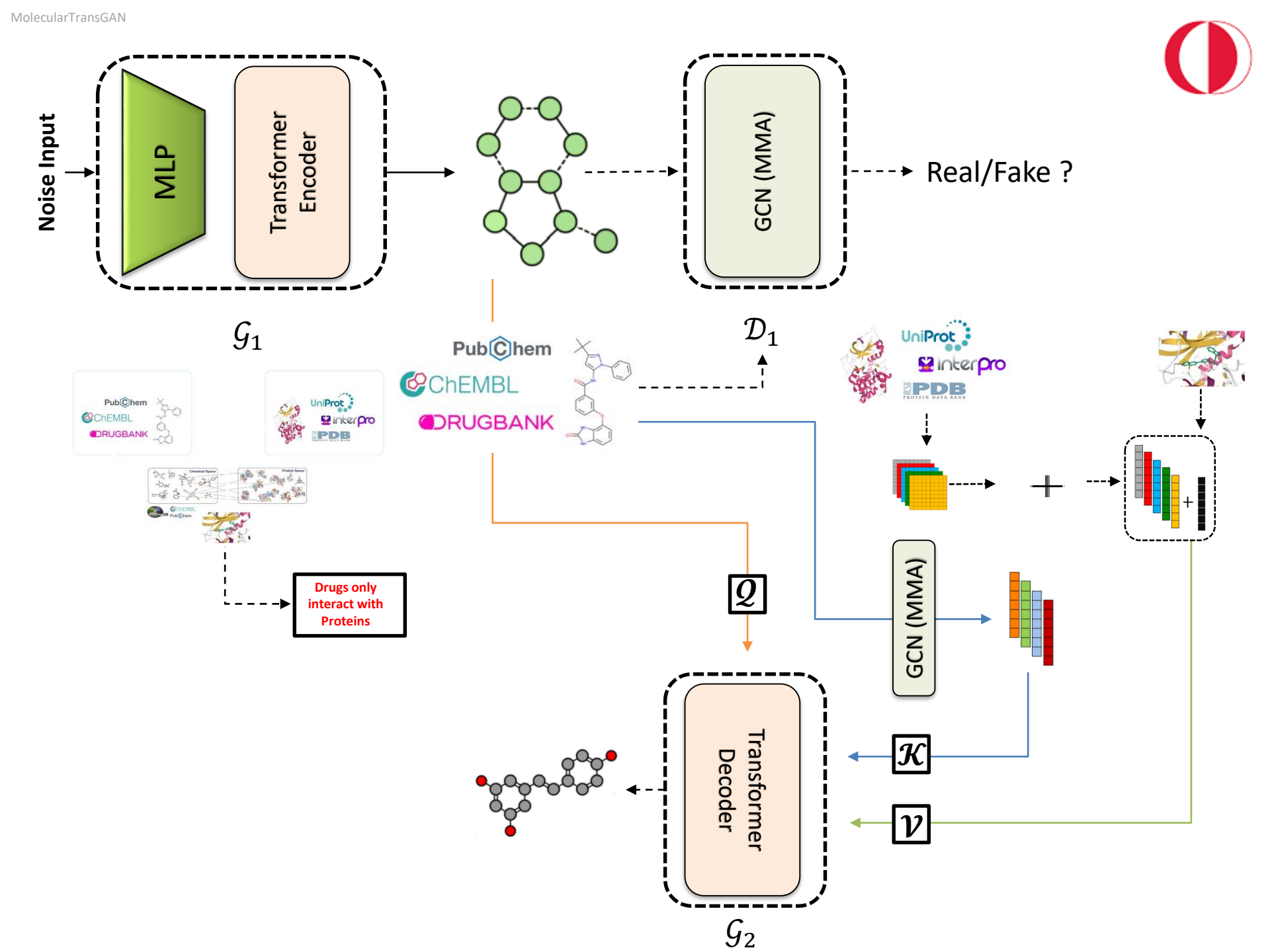

Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB
GCN (MMA)
$\mathcal{G}_2$

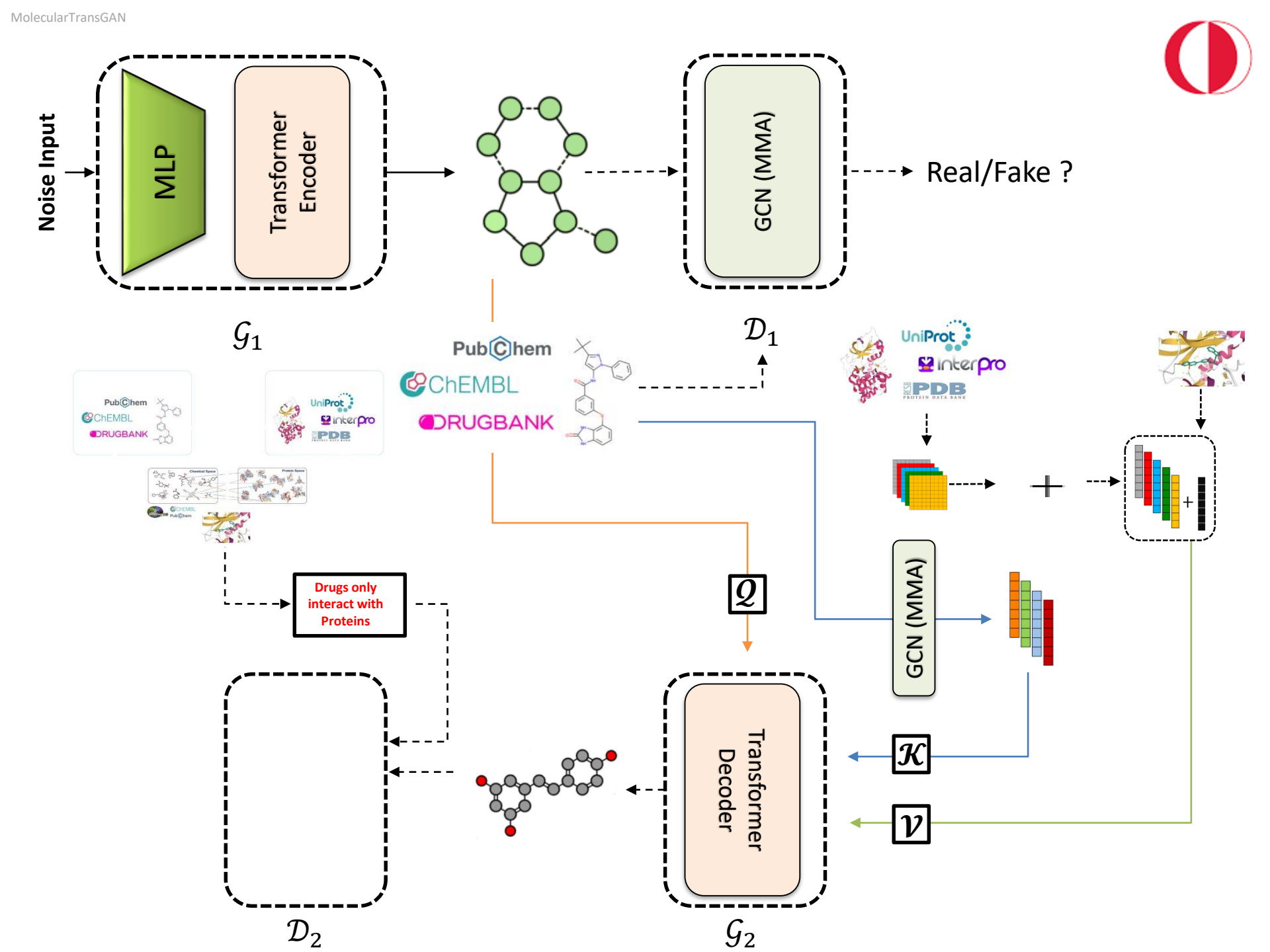

Noise Input
MLP
Transformer Encoder
$\mathcal{G}_1$
GCN (MMA)
Real/Fake ?
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB
Q
K
V
GCN (MMA)
$\mathcal{G}_2$

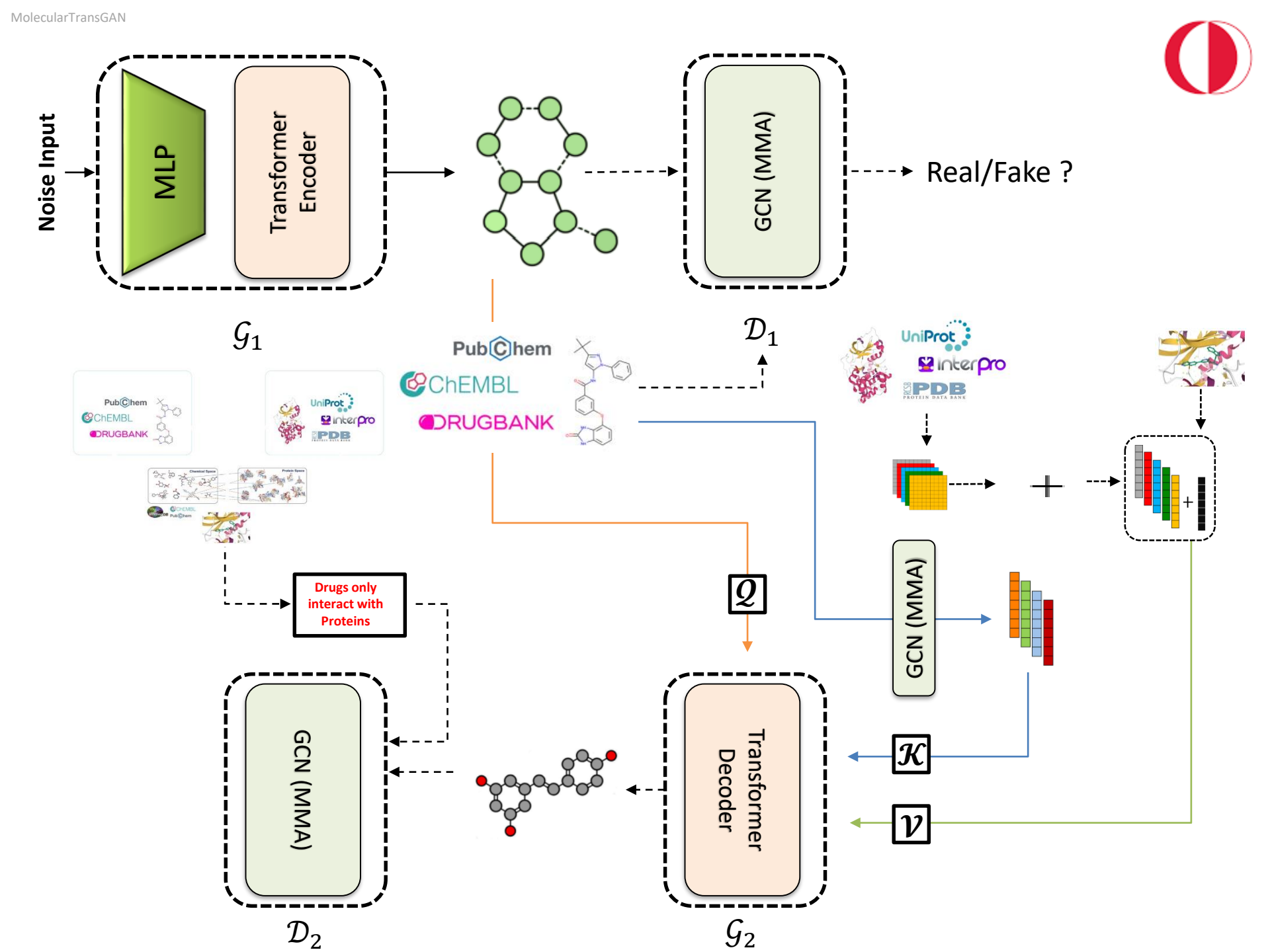

Noise Input
MLP
Transformer Encoder
$\mathcal{G}_1$
GCN (MMA)
Real/Fake ?
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB
GCN (MMA)
$\mathcal{Q}$
$\mathcal{K}$
$\mathcal{V}$
Transformer Decoder
$\mathcal{G}_2$

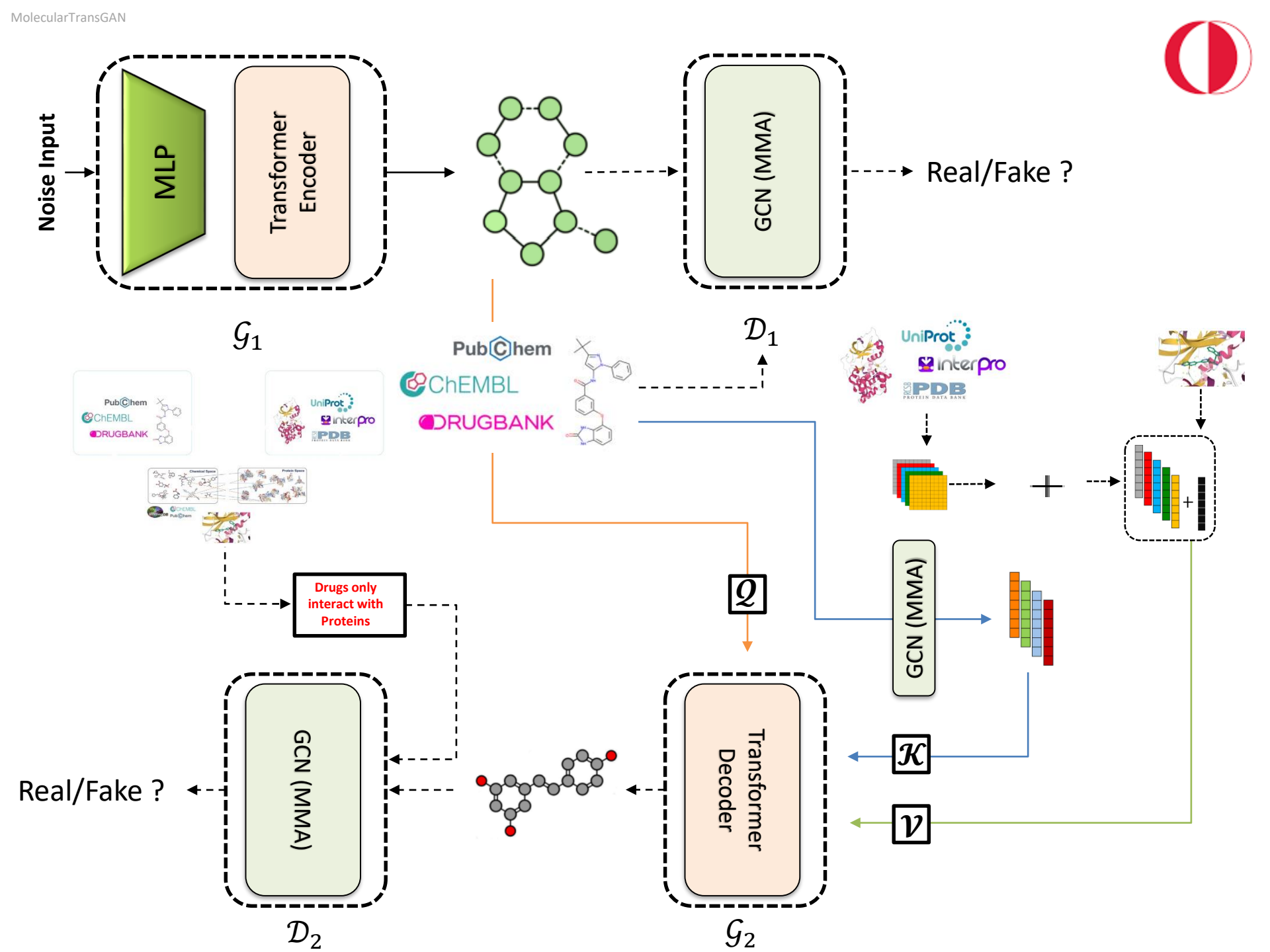

Noise Input
MLP
Transformer Encoder
$\mathcal{G}_1$
GCN (MMA)
Real/Fake ?
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
InterPro
PDB
GCN (MMA)
Q
$\mathcal{K}$
$\mathcal{V}$
Transformer Decoder
$\mathcal{G}_2$


Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
interpro
PDB
Q
$\mathcal{K}$
$\mathcal{V}$
GCN (MMA)
Transformer Decoder
$\mathcal{G}_2$


Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
interpro
PDB
Drugs only interact with Proteins
Q
$\mathcal{K}$
$\mathcal{V}$
GCN (MMA)
Transformer Decoder
$\mathcal{G}_2$


Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
interpro
PDB
Drugs only interact with Proteins
Q
$\mathcal{K}$
$\mathcal{V}$
GCN (MMA)
Transformer Decoder
$\mathcal{D}_2$
$\mathcal{G}_2$


Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
interpro
PDB
Drugs only interact with Proteins
Q
$\mathcal{K}$
$\mathcal{V}$
GCN (MMA)
Transformer Decoder
GCN (MMA)
$\mathcal{D}_2$
$\mathcal{G}_2$

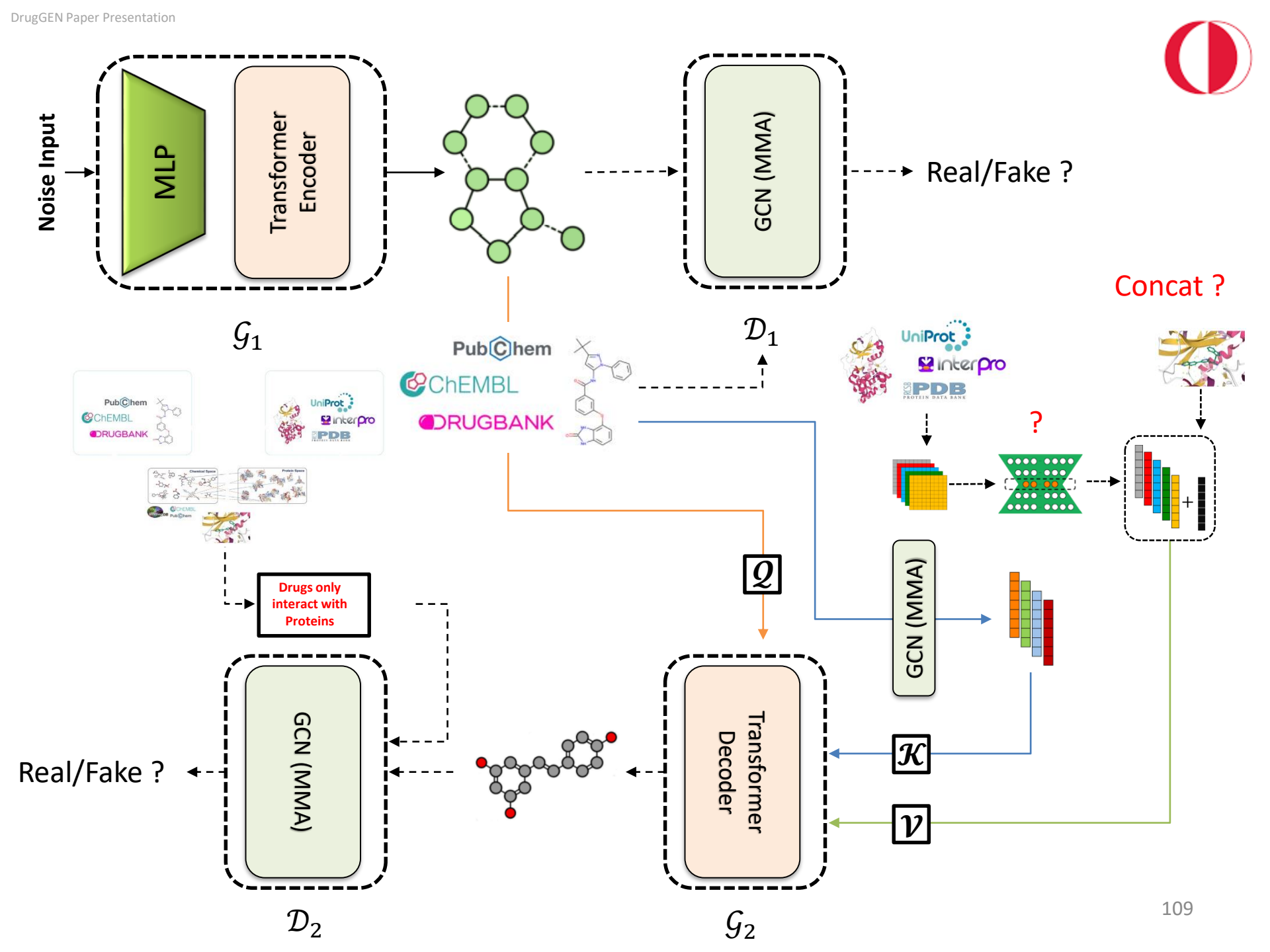

Noise Input
MLP
Transformer Encoder
GCN (MMA)
Real/Fake ?
$\mathcal{G}_1$
$\mathcal{D}_1$
PubChem
ChEMBL
DRUGBANK
UniProt
interpro
PDB
Drugs only interact with Proteins
Q
$\mathcal{K}$
$\mathcal{V}$
GCN (MMA)
Transformer Decoder
Real/Fake ?
$\mathcal{D}_2$
$\mathcal{G}_2$

The GAN Architecture

$$\min \max V(D, G) = \mathbb{E}_{x \sim p_{data}(x), \tilde{x} \sim p_{data}(\tilde{x})}[\mathcal{D}_1(x) + \mathcal{D}_2(\tilde{x})] - \mathbb{E}_{z \sim p_z(z)}[\mathcal{D}_1(\mathcal{G}_1(z)) + \mathcal{D}_2(\mathcal{G}_2(Q, K, V))]$$

G ***D*** ↑ ↓

- x ; PubChem, ChEMBL, DrugBank
- $\tilde{x}$; Drugs only interact with Proteins
- z ; Noise Input
- Q; $\mathcal{G}_1(z)$
- K; GCN(PubChem, ChEMBL, DrugBank) (Compount properties)
- V; $\tilde{x}$ + Autoencoders(UniProt, interPro, PDB) (Target Protein Properties & Interaction Value)


Noise Input
MLP
Transformer Encoder
$\mathcal{G}_1$
GCN (MMA)
Real/Fake ?
$\mathcal{D}_1$
Concat ?
PubChem
ChEMBL
DRUGBANK
UniProt
interpro
PDB
?
Drugs only interact with Proteins
Q
GCN (MMA)
K
V
Transformer Decoder
Real/Fake ?
GCN (MMA)
$\mathcal{D}_2$
$\mathcal{G}_2$