8. GANs

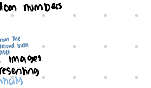
Applications
- Robots to street scenes
- Portrait to map
- Labels to Facade
- Day to night
- Edges to photo

Generative Adversarial Network GANs
- an unsupervised architecture
- uses 2 neural networks competing against eachother in order to generate new synthetic instances of data
- They can learn to mimic any distribution of data
- Can be used for
  - image generation
  - voice generation
  - video generation

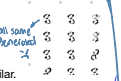
Steps
- The generator takes in random numbers and returns an image
- The discriminator is fed
  - this generated image
  - a stream of images taken from the actual/real world dataset
  - takes both real and fake images and returns probabilities representing
    - 1 → Prediction of authenticity
    - 0 → fake

$$\min_G \max_D V(D,G)$$

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



Mode Collapse Problem in GAN
- The mode collapse problem is a common problem in GAN models.
- The generator learns to map several different z values to the same generated data point x.
- Mode collapse usually happens in GAN when the distribution of training data, Pdata (x), has multiple modes.

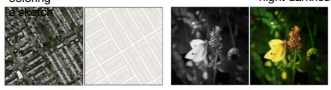


- a resolve the mode collapse problem
- When mode collapses, all images created look similar.
- Feed real images and generated images into the discriminator separately in different batches and compute the similarity of the image x with images in the same batch.
- append the similarity $o(x)$ in one of the dense layers in the discriminator to classify whether this image is real or generated.
- If the mode starts to collapse, the similarity of generated images increases.
- The discriminator can use this score to detect generated images and penalize the generator if mode is collapsing.

Alternative Strategies to Prevent Mode Collapse
- Unrolled GANs: They allow the generator to update itself using a copy of the discriminator from several steps ahead, preventing the generator from overfitting to the current discriminator.
- Experience Replay: This involves keeping a memory bank of past generated images and occasionally showing them to the discriminator, which helps maintain diversity in the generator's output.
- Modified Training Objectives: Alternative loss functions, like Wasserstein loss or least squares loss, provide more stable gradients and can help prevent mode collapse.
- Regularization: Adding noise to inputs or labels, or using dropout in the discriminator, can prevent it from making overly confident decisions, which in turn pressures the generator to be more diverse.
- Architecture Tweaks: Adjusting the network architecture, such as adding more layers or changing activation functions, can help the generator explore a wider range of outputs.
- Feature Matching: The generator is trained to match the statistical features of the real data in some intermediate layer of the discriminator, promoting diversity in the generated data.
- Penalizing the Discriminator: Introducing penalties for the discriminator when it gets too confident can prevent it from overpowering the generator, leading to a more varied generation.
- Two Time-Scale Update Rule (TTUR): Using different learning rates for the generator and the discriminator can help balance their training and prevent the generator from collapsing to a few modes.

Application: Image to Image Translation by GAN
- Interactive GAN (iGAN)
- PatchGAN
- CycleGAN (Cycle-Consistent Generative Adversarial Networks)
- Simulated GAN (SimGAN)
- DeepFaceDrawing

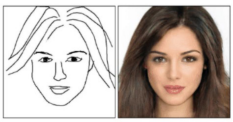
(a) coloring (a) PatchGAN (b) (b) changing daylight to night darkness in image



(c) changing an aerial image to a map (d) coloring a black-and-white image,

Transforming zebra to horse and vice versa

Cycle Gan
For unpaired image-to-image translation



Generating a facial image from a facial sketch.

DeepFace Drawing

Training Generative Adversarial Networks

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

$$= \int_x p_{data}(x) log(D(x)) dx + \int_z p_z(z) log(1 - D(G(z))) dz$$

$$x = G(z) \Rightarrow z = G^{-1}(x) \Rightarrow dz = (G^{-1})'(x) dx$$

$$\Rightarrow p_g(x) = p_z(G^{-1}(x))(G^{-1})'(x)$$

$$= \int_x p_{data}(x) log(D(x)) dx + \int_x p_z(G^{-1}(x)) log(1 - D(x))(G^{-1})'(x) dx$$

$$= \int_x p_{data}(x) log(D(x)) dx + \int_x p_g(x) log(1 - D(x)) dx$$

$$= \int_x p_{data}(x) log(D(x)) + p_g(x) log(1 - D(x)) dx$$

Understanding the objective function

$$\max_D V(D,G) = \max_D \int_x p_{data}(x) log(D(x)) + p_g(x) log(1 - D(x)) dx$$

$$\frac{\partial}{\partial D(x)}(p_{data}(x) log(D(x)) + p_g(x) log(1 - D(x))) = 0$$

$$\Rightarrow \frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} = 0$$

$$\Rightarrow D(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

$$C(G) = \max_D V(G,D)$$

$$= \int_x p_{data}(x) log(D(x)) + p_g(x) log(1 - D(x)) dx$$

$$= \int_x p_{data}(x) log(D_G^*(x)) + p_g(x) log(1 - D_G^*(x)) dx$$

$$= \int_x p_{data}(x) log(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}) + p_g(x) log(\frac{p_g(x)}{p_{data}(x) + p_g(x)}) dx$$

$$= \int_x p_{data}(x) log(\frac{p_{data}(x)}{\frac{p_{data}(x)+p_g(x)}{2}}) + p_g(x) log(\frac{p_g(x)}{\frac{p_{data}(x)+p_g(x)}{2}}) dx - log(4)$$

$$= KL[p_{data}(x) || \frac{p_{data}(x) + p_g(x)}{2}] + KL[p_g(x) || \frac{p_{data}(x) + p_g(x)}{2}] - log(4)$$

$$C(G) = KL[p_{data}(x) || \frac{p_{data}(x)+p_g(x)}{2}] + KL[p_g(x) || \frac{p_{data}(x)+p_g(x)}{2}] - log(4)$$

$$\geq 0 \qquad \geq 0$$

$$\min_G C(G) = 0 + 0 - log(4) = -log(4)$$

$$KL[p_{data}(x) || \frac{p_{data}(x) + p_g(x)}{2}] = 0$$

when $p_{data}(x) = \frac{p_{data}(x)+p_g(x)}{2}$

$$\Rightarrow p_{data}(x) = p_g(x)$$

KL(Kullback - Leibler) divergence

$$JSD(P||Q) = \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M),$$

$$M = \frac{1}{2}(P + Q)$$

Conditional GAN
- Assume that the dataset with which GAN is trained has c number of classes.
- The original GAN generates points from any class, and we do not have control to generate a point from a specific class.
- Conditional GAN, also called the conditional adversarial network, gives the user the opportunity to choose the class of generation of points.

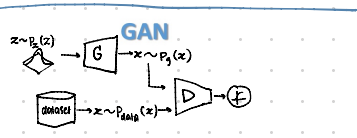
Implementation of Conditional GAN
- Concatenate the one-hot encoded label y to the point x for the input to the discriminator.
- Concatenate the one-hot encoded label y to the noise z for the input to the generator.
- For these, the input layers of discriminator and generator are enlarged to accept the concatenated inputs

For the dataset $\{x_i \in \mathbb{R}^d\}_{i=1}^n$, let the one-hot encoded class labels be $\{y_i \in \mathbb{R}^c\}_{i=1}^n$
In conditional GAN, we can use the following loss function instead:

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

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

the discriminator and generator are both conditioned on the labels



Inference from Conditional GAN
- In the inference phase, the user choose the desired class label and the generator generates a new point from that class.

Application: Text to Image Generation
- Image is generated from some descriptive

This bird is red and brown in color, with a stubby beak.

This small bird has a white breast, light grey head, and black wings and tail.

StackGAN
text-to-image generation.

Application: Mixing Image Characteristics
- FineGAN
- An unsupervised GAN model which disentangles the features of the generated image to background, shape, and color/texture.
- FineGAN generates an image hierarchically. It starts with generating the background.

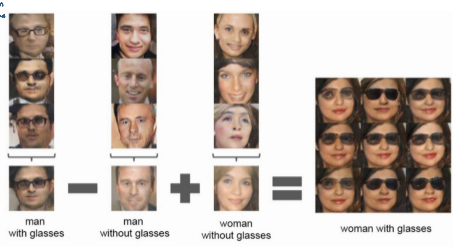
MixNMatch
- built upon FineGAN
- It gives the user the opportunity to choose the background, shape, and colour/texture from three pictures and it generates an image with the chosen characteristics.

generating an image by borrowing its characteristics from three images using MixNMatch

MixNMatch, FineGAN
Designed to combine attributes (like background, texture, shape) from different images or labels.

generating an image by borrowing its characteristics from different domains using improved MixNMatch.

Other Applications
- In some structures of GAN, learned latent space is meaningful and we can do vector arithmetic in the latent space.
- For example in Deep Convolutional GAN (DCGAN)
- DCGAN uses an all-convolutional network for both generator and discriminator.
- Inpainting
- GAN learns to inpaint the lost part based on the available pixels in the image.
- Medical application
- Generating histopathology images which can give insight into cancer diagnosis from pathology whole slide images
- NLP
- Speech processing
- Network embedding
- Logic
- Sketch retrieval

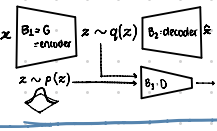


Vector arithmetic in the latent space on images by DCGAN.

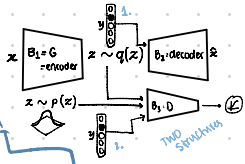
Adversarial Autoencoder (AAE)

- In contrast to variational autoencoder which uses KL divergence and evidence lower bound, AAE uses adversarial learning for imposing a specific distribution on the latent variable in its coding layer.

Unsupervised AAE



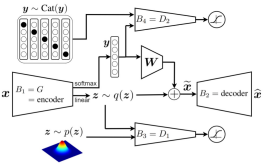
Supervised AAE



Dimensionality Reduction with AAE

The low-dimensional representation of

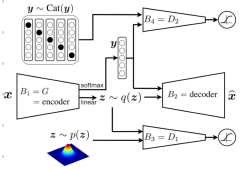
$$\tilde{x} = W^T y + z,$$



Semi-Supervised AAE

- Consider a partially labelled dataset.
- The labelled part of data has c number of classes.
- AAE can be used for semi-supervised learning with the partially labelled dataset.
- We can use the same structure for Semi-Supervised.
- But rather than the clusters, we assume we have c number of classes.
- We have a partially labelled part of the dataset.

Semi-Supervised AAE

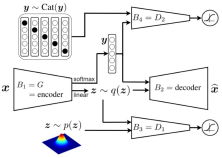


- If the point x has a label use it. Labels are one-hot vectors.
- If the point x does not have any label, randomly sample a label y € R^c from a categorical distribution, i.e., y ~ Cat (y)
- This categorical distribution gives a one-hot encoded vector where the prior probability of every class is estimated by the proportion of the class's population to the total number of labelled points.

Clustering with AAE

- Assume we have c number of clusters.
- All points are unlabeled.
- The cluster indices are sampled randomly by the categorical distribution.
- The cluster labels and the latent code are both trained.

Clustering with AAE



COMPARISIONS

GAN:
Think of a forger (G) and a detective (D).
G tries to fool D with fake images.
No explicit understanding of how G makes data — it just learns to trick D.

VAE:
Learns a smooth, interpretable "latent space" (compressed version of data).
Generates new samples by decoding latent vectors.
Loss is based on how well it reconstructs and how close latent vectors are to a normal distribution.
Tends to produce blurrier images but better control.

AAE:
Combines VAE's structure (encoder-decoder) with GAN's loss in the latent space.
Instead of using KL divergence (like VAE), uses a GAN to make sure latent codes look like the prior (e.g., normal distribution).
Better at clustering and generating structured latent representations.

Conditional GAN:
Like GAN but with labels added as input.
For example, "generate a 7" instead of any digit.
Helps solve the lack of control in vanilla GAN.

Model	Pros	Cons
GAN	Realistic generation, no labels needed	Training instability, mode collapse, no inference
AAE	Generative + encoding, flexible latent priors	Less sharp images, adversarial tuning needed
Supervised AAE	Combines generation & classification	Needs labeled data, may overfit
Unsupervised AAE	No labels, useful for clustering	No semantic control, weaker generation quality
Conditional GAN	Controlled generation by labels	Needs labels, can overfit, GAN issues remain
Semi-Supervised AAE	Uses limited labels effectively	Complex to train, balancing losses is tricky
Clustering AAE	Finds groups without labels, useful latent representations	May need to predefine cluster count, cluster quality depends on prior

When to use

1. GANs (Generative Adversarial Networks)
Use when: You want to generate realistic data (e.g., images, speech) from noise.
Input: Random noise z
Output: Synthetic sample
Labels needed? ❌ No
Use cases: Image generation, super-resolution, style transfer, image inpainting

🔷 **2. AAE (Adversarial Autoencoder)**
Use when: You want to impose a prior distribution on latent space and reconstruct input data.
Combines: Autoencoder + GAN
Input: Data sample x
Output: Reconstructed x + latent vector z
Labels needed? ❌ No
Use cases: Dimensionality reduction, clustering, anomaly detection

🔷 **3. Supervised AAE**
Use when: You have labeled data and want the latent space to reflect class structure.
Labels needed? ✅ Yes
Learns: Both reconstruction and class-discriminative latent codes
Use cases: Semi-supervised learning, structured latent spaces

🔷 **4. Unsupervised AAE**
Use when: You want to learn structure in data without labels
Labels needed? ❌ No
Learns: Prior matching of latent space + reconstruction
Use cases: Clustering, representation learning, generative modeling

🔷 **5. Conditional GAN (cGAN)**
Use when: You want to generate data conditioned on labels or attributes
Input: Noise z + condition y (e.g., class label)
Output: Sample matching the label
Labels needed? ✅ Yes (for training)
Use cases: Text-to-image, image-to-image translation, label-controlled generation

🔷 **6. Semi-Supervised AAE**
Use when: You have both labeled and unlabeled data
Uses: A small amount of labeled data to guide training; unlabeled data to improve generalization
Labels needed? ✅ Partially
Use cases: When labeled data is scarce but unlabeled data is abundant (e.g., medical imaging, fraud detection)

🔷 **7. Clustering with AAE**
Use when: You want to automatically cluster data in latent space
Labels needed? ❌ No
Technique: Sample latent codes from a mixture of distributions (e.g., Gaussian + Categorical)
Use cases: Discovering groups in data, unsupervised classification, data exploration

B) Assume that we want to classify images in 3 categories of **A**pple, **O**range and **B**anana for which the prior probabilities are **P(A) = 0.3, P(O) = 0.5, P(B) = 0.2**. We see a new image $\mathbf{x}$ from which we extract a feature vector **x**. The likelihoods are:

P(x | A) = 0.24
P(x | O) = 0.15
P(x | B) = 0.38

What should image $\mathbf{x}$ be classified as? An apple, an orange, or a banana? Why? Write down the complete derivations. **(7 points)**

$$P(A \mid x) = p(x \mid A)\,p(A) / p(x) = 0.24 * 0.3 / p(x) = 0.72 / p(x)$$
$$P(O \mid x) = p(x \mid O)\,p(O) / p(x) = 0.15 * 0.5 / p(x) = 0.75 / p(x)$$
$$P(B \mid x) = p(x \mid B)\,p(B) / p(x) = 0.38 * 0.2 / p(x) = 0.76 / p(x)$$

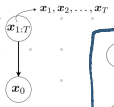
$$\Rightarrow P(B \mid x) > P(O \mid x) > P(A \mid x)$$

⇒ It has to be classified as **Banana**

7. Diffusion Models

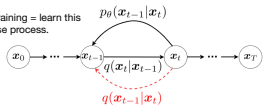
Diffusion Models

Diffusion models are generative models that:
Gradually add noise to an image (forward process).
Then learn to reverse the process and remove the noise to generate a clean image (reverse process).
• Diffusion models use a Markov chain to do it



Denoising Diffusion Probabilistic Models (DDPMs)

So, Training = learn this reverse process.



Joint Distribution over Latent States in DDPMs

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

all latent states from time 1 to T

initial state

output mean variance

$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$

Forward Pass

Data → Noise

Backward Pass

$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t,t), \beta_t^2 I)$

output mean

to variance

$x_0 \longrightarrow x_1 \longrightarrow \cdots \longrightarrow x_T$

$x_0 \longleftarrow x_1 \longleftarrow \cdots \longleftarrow x_T$

#identity matrix is used to simplify calculation

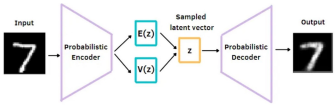
$x_n \begin{bmatrix} \sigma & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & \sigma \end{bmatrix}$ ← covariance

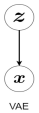
x_1 x_2 x_3

How It Works DDPMS

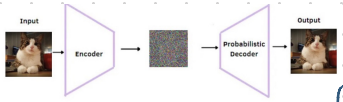
1. Forward Diffusion Process
Start with a real image x_0.
At each timestep t, add a small amount of Gaussian noise:
$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon$
where ε ~ N(0, I) and β_t is a small noise variance.
After many steps (say 1000), the image becomes nearly random noise.
2. Reverse Process (Generation)
Train a neural network (usually a U-Net) to predict the noise ε added at each step, given the noisy image x_t and the timestep t.
The model learns to denoise the image one step at a time, using the learned noise estimate: $x_{t-1} \approx \frac{x_t - \sqrt{\beta_t}\,(\text{predicted noise})}{\sqrt{1-\beta_t}}$
•Starting from pure noise x_T, apply the denoising steps iteratively to get a new, realistic sample x_0.

Variational Autoencoder





VAEs (Variational Autoencoders) are a type of generative model in machine learning used to learn the underlying structure of data and generate new data that is similar to the original. They are especially popular in unsupervised learning and are used in applications like image generation, anomaly detection, and representation learning.



how it works VAE
Input to Encoder: An image (or other data) is fed into the encoder.
Latent Distribution: Instead of producing a single vector, the encoder outputs two vectors:
Mean vector (μ)
Standard deviation vector (σ)
These define a normal distribution: z ~ N(μ, σ²).
Sampling (Reparameterization Trick): A latent vector z is sampled from this distribution. To make it differentiable (for backpropagation), a trick is used:
$z = \mu + \sigma \cdot \epsilon$
4. Decoder Reconstructs Input: The decoder takes z and tries to reconstruct the original input.
Loss Function:
Reconstruction Loss: Measures how close the output is to the original input (e.g., using MSE or binary cross-entropy).
•KL Divergence: Encourages the latent distribution to be close to a standard normal distribution (N(0,1)), helping regularize the latent space.
Total Loss = Reconstruction loss + KL Divergence

Diffusion Model VS VAEs

	In VAE	In DDPM
	• Observed variable x • Hidden Variable z Latent variable z, direct mapping	• Observed variable xo • Hidden variable X1, X2, ... XT Multiple noisy steps — a full sequence, not just one latent variable.
Objective Functions	$p_\theta(x) = \int p_\theta(x,z)\,dz$	$p_\theta(x_0) = \int p_\theta(x_0, x_{1:T})\,dx_{1:T}$
Variational Lower Bounds	$p_\theta(x) \geq \mathbb{E}_{q(z\|x)}[\log p_\theta(x\|z)] - D_{KL}(q(z\|x)\,\|\,p_\theta(z))$	$p_\theta(x_0) \geq \mathbb{E}_{q(x_{1:T}\|x_0)}[\log p_\theta(x_0\|x_{1:T})] - D_{KL}(q(x_{1:T}\|x_0)\,\|\,p_\theta(x_{1:T}))$
	YES	Yes but optimized differently
Loss	KL divergence + reconstruction	Predict the noise added
Target	maximize ELBO	minimise noise prediction loss
ELBO	ELBO ensures that latent variable z follows a desired distribution (like Gaussian).	Similar principle, but over many time steps. Loss is decomposed for each timestep and simplified
Encoder	Maps n→2	Forward Pass Adds noise: $x_0 \to x_T$
Decoder	Maps z→n	Backward Pass Learns denoising: $x_T \to x_0$
Process Type	1 step	Multistep (Markov chain)

VAE sampling
a) μ: 2, σ: 0.5, ε: 0.3
$z = \mu + \sigma \cdot \epsilon$
$= 2 + 0.5(0.3)$
$= 2.15$

Diversion b/w Gaussians
a) μ: 1, σ: 1

$KL(\mathcal{N}(\mu,\sigma)) \,\|\, \mathcal{N}(0,1)) = -\frac{1}{2}(1 + \log(\sigma^2) - \mu^2 - \sigma^2)$

$\therefore \sigma^2 = 1$
$\therefore \log(1) = 0$

So $KL = \frac{-1}{2}(1+0-1-1)$
$= 0.5$

Diffusion Forward Process calculation
a) $\alpha_t = 0.9$, $x_{t-1} = 0.8$, $\epsilon = 0.5$

$x_t = \sqrt{\alpha_t}(x_{t-1}) + \sqrt{1-\alpha_t}(\epsilon)$

$= \sqrt{0.9}(0.8) + \sqrt{1-0.9}(0.5)$
$= 0.9171$

Diffusion Backward Process calculation
a) $x_t = 2$, $\alpha_t = 0.9$
$\sigma_t = 0.1$

1. What is the key difference between a standard autoencoder and a VAE?
A standard autoencoder learns a deterministic mapping (input → compressed code → output).
A VAE learns a probabilistic mapping: it maps input to a distribution (usually Gaussian)
, samples from it, and then decodes.

2. Why do we need the KL divergence term in the VAE loss function?
It ensures that the latent distribution stays close to a standard normal distribution (N(0,1)).
Without this, sampling from the latent space would generate unrealistic or untrained outputs.

3. What is the reparameterization trick, and why is it used in VAEs?
Trick: Instead of sampling $z \sim \mathcal{N}(\mu, \sigma^2)$ we write, $z = \mu + \sigma \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0,1)$
It's used to make sampling differentiable, so we can backpropagate through it.

4. In a VAE, what are μ and σ used for?
They define the mean and standard deviation of the latent distribution for a given input.
They're used to sample a latent vector z that gets decoded.

5. What does it mean to "sample" from the latent space in a VAE?
It means generating a value z from the learned Gaussian distribution N(μ,σ2), rather than using a fixed code.

6. Can a VAE generate new data without any input image? Explain how.
Yes! You can randomly sample z~N(0,1) and feed it into the decoder to generate a new image.

7. Why do diffusion models add noise to data during training?
To simulate a process where clean data slowly becomes noise.
This helps the model learn how to denoise during generation.

8. What is the difference between the forward process and reverse process in diffusion models?
Forward: Gradually add Gaussian noise: x0⟶x1⟶⋯⋯⟶xT
Reverse: Learn to remove noise: xT⟶xT−1⟶⋯⋯⟶x0

9. In DDPMs, what does the model learn during training?
The model learns to predict the noise added at each step, or equivalently, how to denoise the image.

10. How are DDPMs different from VAEs in terms of latent representation?
VAEs use one latent variable z.
DDPMs use a sequence of noisy variables x1,x2,...,xT
DDPM's latent space is temporal, evolving step by step.

11. Why do Diffusion Models produce better images than VAEs?
VAEs optimize a tradeoff between reconstruction and regularization → can produce blurry results.
Diffusion models focus on denoising at each step, allowing sharper and more high-fidelity images
, at the cost of slower sampling.

1. ELBO (Evidence Lower Bound)
ELBO is the function VAEs maximize during training.
It is a lower bound on the log-likelihood of the data (log p(x)).
VAEs can't directly maximize log p(x) (too complex), so they optimize ELBO instead.

Variational Lower Bound

$$\mathbb{E}_{q(x_{1:T}|x_0)}[\log p_\theta(x_0|x_{1:T})] - D_{KL}(q(x_{1:T}|x_0)||p_\theta(x_{1:T}))$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)}[\log p_\theta(x_0|x_{1:T})] - \int q(x_{1:T}|x_0) \log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{1:T})} dx_{1:T}$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)}[\log p_\theta(x_0|x_{1:T})] - \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{1:T})}\right]$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log p_\theta(x_0|x_{1:T}) - \log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{1:T})}\right]$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log p_\theta(x_0|x_{1:T}) + \log \frac{p_\theta(x_{1:T})}{q(x_{1:T}|x_0)}\right]$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_0|x_{1:T})p_\theta(x_{1:T})}{q(x_{1:T}|x_0)}\right] = \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_0,x_{1:T})}{q(x_{1:T}|x_0)}\right]$$

$$\mathbb{E}_q\left[\log \frac{p_\theta(x_0,x_T)}{q(x_{1:T}|x_0)}\right]$$

$$= \mathbb{E}_q\left[\log \frac{p(x_T)\prod_{t=1}^T p_\theta(x_{t-1}|x_t)}{q(x_{1:T}|x_0)}\right]$$

$$= \mathbb{E}_q[\log p(x_T)] + \sum_{t=1}^T \mathbb{E}_q[\log p_\theta(x_{t-1}|x_t)] - \mathbb{E}_q[\log q(x_{1:T}|x_0)]$$

$$= \mathbb{E}_q[\log p(x_T)] + \sum_{t=1}^T \mathbb{E}_q[\log p_\theta(x_{t-1}|x_t)] - \mathbb{E}_q\left[\log \prod_{t=1}^T q(x_t|x_{t-1})\right]$$

$$= \mathbb{E}_q[\log p(x_T)] + \sum_{t=1}^T \mathbb{E}_q[\log p_\theta(x_{t-1}|x_t)] - \sum_{t=1}^T \mathbb{E}_q[\log q(x_t|x_{t-1})]$$

$$= \mathbb{E}_q[\log p(x_T)] + \sum_{t=1}^T \mathbb{E}_q\left[\log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}\right]$$

2. Decomposing the ELBO
ELBO has two main parts:
ELBO = Reconstruction Term - KL Divergence
Reconstruction Term: Measures how well the decoder reconstructs input x from latent code z. It encourages accurate generation.
KL Divergence: Regularizes the latent distribution to be close to a standard normal (N(0,1)), preventing overfitting.

Decomposing the ELBO

- $\mathbb{E}_q[\log p(x_T)]$: Validates the model's calibration to the true noise at the last diffusion step, ensuring the reverse process starts accurately.
- $\sum_{t=1}^T \mathbb{E}_q\left[\log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}\right]$: Acts as a regularizer by comparing the model's backward predictions to the known forward diffusion, guiding the model to correct any discrepancies.

$$\mathcal{L}(\boldsymbol{x}_0) = \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log p(\boldsymbol{x}_T) + \sum_{t=1}^T \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})}\right]$$

By the Markov property:

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) = q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_0),$$

and by Bayes' rule:

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_0) = \frac{q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0)}.$$

Plugging this equation into the ELBO, we get

$$\ell(\boldsymbol{x}_0) = \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log p(\boldsymbol{x}_T) + \sum_{t=2}^T \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)}{q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0)} + \underbrace{\sum_{t=2}^T \log \frac{q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)}}_{*} + \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)}{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}\right]$$

$$* = -\log q(\boldsymbol{x}_T \mid \boldsymbol{x}_0) + \log q(\boldsymbol{x}_1 \mid \boldsymbol{x}_0)$$

Hence the negative ELBO (variational upper bound) becomes

$$= -\mathbb{E}_q\left[\log \frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T \mid \boldsymbol{x}_0)} + \sum_{t=2}^T \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)}{q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0)} + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0 \mid \boldsymbol{x}_1)\right]$$

$$= \underbrace{D_{\mathrm{KL}}(q(\boldsymbol{x}_T \mid \boldsymbol{x}_0) \| p(\boldsymbol{x}_T))}_{L_T(\boldsymbol{x}_0)} + \sum_{t=2}^T \mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} \underbrace{D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) \| p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t))}_{L_{t-1}(\boldsymbol{x}_0)} - \underbrace{\mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0 \mid \boldsymbol{x}_1)}_{L_0(\boldsymbol{x}_0)}$$

3. Reverse Process (in Diffusion Models)
Starts with pure noise and gradually denoises it using a trained model.
At each step, the model predicts the noise and removes it to recover the original data.
It's the generation phase of diffusion models.

Revers Process

We need to learn a neural network to approximate the conditioned probability distributions in the reverse diffusion process,

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)).$$

$$q(\mathbf{x}_t \mid \mathbf{x}_0)$$

Recall that the forward process is defined by the equation:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t \mid \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$$

We can sample $\mathbf{x}_t$ at any arbitrary time step t in a closed form.
Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

4. Mean (μ) and Variance (σ²) in VAEs
The encoder outputs two vectors: mean (μ) and log-variance (log σ²).
They define a Gaussian distribution in latent space:
z ~ N(μ, σ²)
Sampling from this gives diverse and continuous latent representations.

Mean of $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$

The reverse conditional probability is tractable when conditioned on $\mathbf{x}_0$:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}\right)$$

Using Bayes' rule, we have:

$$\begin{aligned} q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) &= q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \\ &\propto \exp\left(-\frac{1}{2}\left(\frac{(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1})^2}{\beta_t} + \frac{(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0)^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t}\right)\right) \\ &= \exp\left(-\frac{1}{2}\left(\frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1} + \alpha_t\mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\mathbf{x}_{t-1} + \bar{\alpha}_{t-1}\mathbf{x}_0^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t}\right)\right) \\ &= \exp\left(-\frac{1}{2}\left(\left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}\right)\mathbf{x}_{t-1}^2 - \left(\frac{2\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0\right)\mathbf{x}_{t-1} + C(\mathbf{x}_t, \mathbf{x}_0)\right)\right) \end{aligned}$$

where $C(\mathbf{x}_t, \mathbf{x}_0)$ is some function not involving $\mathbf{x}_{t-1}$.

We can express the mean and variance in the following manner: (recall that $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$):

$$\begin{aligned} \tilde{\beta}_t &= 1/\left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}\right) = 1/\left(\frac{\alpha_t - \bar{\alpha}_t + \beta_t}{\beta_t(1-\bar{\alpha}_{t-1})}\right) = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \\ \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \left(\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0\right) / \left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}\right) \\ &= \left(\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0\right) \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \\ &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 \end{aligned}$$

Recall $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t\right)$

$$\begin{aligned} \tilde{\boldsymbol{\mu}}_t &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t\right) \\ &= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right) \end{aligned}$$

5. KL Divergence
Measures how much one distribution (e.g., q(z|x)) differs from another (e.g., N(0,1)).
In VAEs, it keeps the latent space well-behaved and prevents overfitting.
Lower KL = closer to normal distribution.

KL for two Guassian

$$D_{KL}\left(q\left(x_{t-1} \mid x_t, x_0\right) \| p_{\boldsymbol{\theta}}\left(x_{t-1} \mid x_t\right)\right)$$

$$q(x) = \mathcal{N}(x; \mu_q, \Sigma_q) \quad \text{and} \quad p(x) = \mathcal{N}(x; \mu_p, \Sigma_p),$$

$$D_{KL}(q\|p) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_p^{-1}\Sigma_q) + (\mu_p - \mu_q)^T\Sigma_p^{-1}(\mu_p - \mu_q) - k + \ln\left(\frac{\det\Sigma_p}{\det\Sigma_q}\right)\right)$$

for q

$$\tilde{\boldsymbol{\mu}}_t(\boldsymbol{x}_t, \boldsymbol{x}_0) = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\boldsymbol{x}_t + \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}\right)$$

for p

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)\right)$$

6. VAE Loss Function
Total loss = Reconstruction Loss + KL Divergence
Reconstruction Loss ensures the output matches the input.
KL Divergence regularizes the latent space.

Loss fnction

$$\begin{aligned} L_t &= E_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{1}{2\,\|\boldsymbol{\Sigma}_\theta(\mathbf{x}_t,t)\|_2^2}\|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t,\mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t,t)\|^2\right] \\ &= E_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{1}{2\,\|\boldsymbol{\Sigma}_\theta\|_2^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right) - \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\right)\right\|^2\right] \\ &= E_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\|^2\right] \\ &= E_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t\right)\|_2^2\right] \end{aligned}$$

- Ho et al. (2020) discovered empirically that the diffusion model produces higher-quality images when using a simplified objective, omiting the weighting term

$$\begin{aligned} L_{\text{simple}} &= E_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\|^2\right] \\ &= E_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\left\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t\right)\right\|^2\right] \end{aligned}$$

9. Attention

↳ a technique that allows input word information to be passed all the way to the decoders

What is Attention?

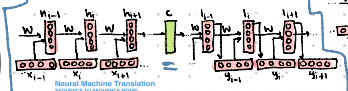
A weighted average of input states.
Each output word is generated by giving different importance (weights) to input words.
It helps the model focus on important parts of the input when producing each output:

Each output word is influenced by all input words — but not equally.
The attention weights decide how much each input word contributes.

Traditional Sequence-to-Sequence

What it did:
Used an encoder to process the entire input into one vector.
The decoder then used this vector to generate output.

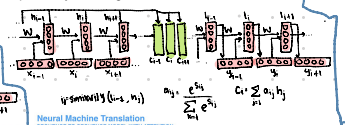
Problems:
Long sentences were hard to remember →
Long-range dependencies
No focus mechanism → Equal importance to every word
Couldn't parallelize well



Sequence-to-Sequence with Attention

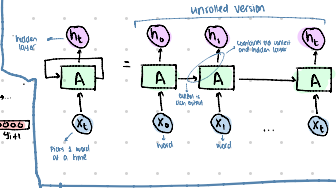
What it did:
Used an encoder to process the input sequence into a series of hidden states (not just one vector).
The decoder generated each output step by using attention to focus on relevant parts of the input.

Improvements:
Attention mechanism: Let the decoder dynamically focus on specific input words at each time step.
Handled long sentences better by accessing all encoder outputs.
Improved translation quality and long-range dependency modeling.



Recurrent Neural Networks (RNN)

↳ uses loops to allow info to persist b/w inputs
↳ short term dependency solved by attention
↳ each word is an input, which is encoded as the output of the hidden layer
↳ which is fed to the next layer
↳ along with another input word in the sequence



Exact Match

Key	Value
Alice Johnson	555-1234
Bob Smith	555-5678
Carol Davis	555-8765
Dave Martin	555-4321
Eve Clark	555-3456

Example Query 1:
- **Query:** Find the phone number for Bob Smith.
- **Result:** 555-5678

Approximate Match

Database of Box Sizes and Weights

Key (Box Size in cubic inches)	Value (Weight in pounds)
300	30.0
400	40.0

Query: Box Size "360" cubic inches

Calculate Similarity Scores:
- Similarity of size "300": $\frac{1}{|360-300|} = \frac{1}{60}$
- Similarity of size "400": $\frac{1}{|360-400|} = \frac{1}{40}$

Calculate Coefficients:
- $a_1 = \frac{\frac{1}{60}}{\frac{1}{60}+\frac{1}{40}} = \frac{2}{5}$
- $a_2 = \frac{\frac{1}{40}}{\frac{1}{60}+\frac{1}{40}} = \frac{3}{5}$

Calculate Weighted Average Weight:

Weighted Average Weight $= (a_1 \cdot 30.0) + (a_2 \cdot 40.0) = 12.0 + 24.0 = 36.0$

Estimation: The estimated weight of a box with a size of "360" cubic inches is "36.0" pounds.

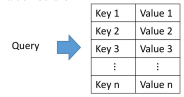
What's the difference in Self-Attention

Instead of one sequence attending to another (like input → output),
Self-Attention allows each word in a sentence to attend to all other words.
This is the core idea behind Transformer
Example Sentence: "The early bird catches the worm."
"early" attends to "bird"
"catches" attends to "worm"
Helps understand relations in a sentence — not just individual word meanings.
Process:
• Attention scores calculated between word pairs.
• Multiple layers help form complex concept representations.

Self-attention: Q, K, V all come from the same sequence.
Cross-attention: Q comes from one sequence, K and V from another.

Generalized definition

• To calculate the Attention of a target word with respect to the input word
• Use the Query of the target and the Key of the input
• Calculate a matching score
• These matching scores act as the weights of the Value vectors.

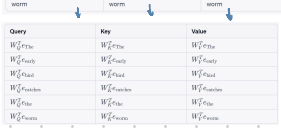


Key 1	Value 1
Key 2	Value 2
Key 3	Value 3
⋮	⋮
Key n	Value n

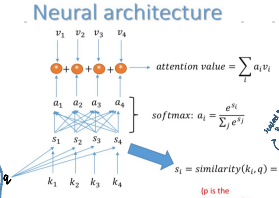
$attention(q, \boldsymbol{k}, \boldsymbol{v}) = \sum_i similarity(q, k_i) \times v_i$

Mimics the retrieval of a value vi for a query q based on key ki in database.

Query	Key	Value
The	The	The
early	early	early
bird	bird	bird
catches	catches	catches
the	the	the
worm	worm	worm



Neural architecture



Matrix Forms:
• Words in sequence: X
• Queries: Q
• Keys: K
• Values: V

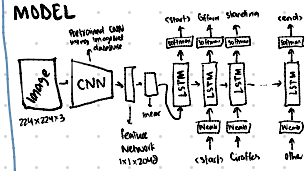
Projection into Subspaces (Post Definitions):
• Queries: qi = $W_q x_i$
• Keys: ki = $W_k x_i$
• Values: Vi = $W_v x_i$

Similarity Measures:
• Inner product: $q_1^T k_1 = x_1^T W_q^T W_k x_1$
• Acts like a kernel matrix, measuring similarity.

Attention Computation:
• Z := attention(Q,K, V) = Vsoftmax$\left(\frac{1}{\sqrt{D}} Q^T K\right)$

Attention in computer vision

• Image Captioning



Annotation vector
Spatial information of an image is captured by lower convolutional layer of a CNN.

$v = \{v_1, \ldots, v_t\}, v_i \in R^d$

$s_{ti} = f_{att}(v_i, h_{t-1})$

$a_{ti} = \frac{e^{s_{ti}}}{\sum_j e^{s_{tj}}}$

$z_t = \Phi(\{v_i\}, \{a_i\})$

B) Calculating Attention for the Word "bird"

Objective:
• Calculate the attention weights for the word "bird" within the sentence
Procedure:
• Focus on the word "bird" as our point of interest.
Attention Calculation:
• Calculate a similarity between the query vector of "bird" and the key vectors of other words in the sentence.

word "The", "early", "bird", "catches", "the", "worm".
Embedding [1,0] [0,1] [1,1] [0,2] [1,2] [1,2]
(repeated)

2. Define weight Matrices: W_Q: [0 1 / 1 0] (identity matrix), W_K: [1 2 / 0 1]
3. Calculate Query Vector for Bird: $q_2 = W_Q \cdot e_{bird} = [0\ 1]\cdot[1\ 1] = [1\ 1]$
4. Compute Key Vectors: Multiply each word by W_K: $k = W_K \cdot e$
The → [1,0] → [2,1]; Early → [1,1] → [3,1]; Bird → [1,2] → [5,2]; Catches → [0,2] → [4,2]; Worm → [1,1] → [3,1]
5. Dot Product Similarity: $q = [1,1]$ Score: $q \cdot k$
The [1,1] [2,1] 1×1+1×2 = 3
Early [3,1] 2+1 = 3
Bird [5,2] 5+2 = 7
Catches [4,2] 4+2 = 6
Worm [3,1] 3+1 = 4
6. Softmax apply to scores: $softmax(s_i) = \frac{e^{s_i}}{\sum_j e^{s_j}}$
The e^3 = 20.09 → 20.09/1577.47 = 0.0127
Early e^3 = 20.09 → 0.0127
Bird e^7 = 54.6 → 0.0346
Catches e^6 = 403 → 0.2556
Worm e^4 = 1096 → 0.6949
1577.47

1. Why is attention better than using only the last hidden state of the encoder?
Using only the last hidden state in an encoder compresses all input information into a single fixed-length vector, which loses important details — especially for long sequences.
Attention, on the other hand:
Lets the decoder access all encoder hidden states.
Learn which parts of the input are relevant at each decoding step.
Provides flexibility and better performance for long or complex sequences.

2. What problem does self-attention solve compared to RNNs?
Allows parallel processing (RNNs are sequential).
Captures long-range dependencies more efficiently.
Learns relationships between all words in a sequence, not just adjacent ones.
This makes self-attention the backbone of models like Transformers (e.g., BERT, GPT).

3. How does softmax contribute to attention calculation?
It Converts raw attention scores (e.g., dot products between query and keys) into probabilities.
Ensures all attention weights sum to 1, and all weights are powitive
Gives higher weight to more relevant input tokens and suppresses less relevant ones. $a_{ti} = e^{s_{ti}} / \sum_j e^{s_{tj}}$

4. What are query, key, and value in attention?
Query (Q):What we're looking to match (from decoder or input token)
Key (K): What we're comparing the query against (all tokens)
Value (V): The actual data we use to generate output (usually same as key)
Attention computes how much each value should contribute to the output based on query–key similarity.

5. Why do we call attention a "weighted average"?
Because the output of attention is:
$\sum(\text{weight}_i \times \text{value}_i)$
The weights come from softmax scores between query and keys.
The values are what we're averaging.
So it's literally a weighted average of the value vectors — focused more on the relevant inputs.

Dot Product based Attention

B) $Q: [1,0], K_1: [1,0], K_2: [0,1]$
$v_1 = 5, v_2 = 10$

1. Dot Product (Similarity Score)
$S: q \cdot k$
$S_1 = (1,1) \cdot (0,0) = 1$
$S_2 = (1,0) \cdot (0,1) = 0$

2. Softmax over scores
$softmax(s_i): e^{s_i} / \sum_j e^{s_j}$
$a_1 = \frac{e^1}{e^1 + e^0} = 0.731$
$a_2 = \frac{e^0}{e^1 + e^0} = 0.269$

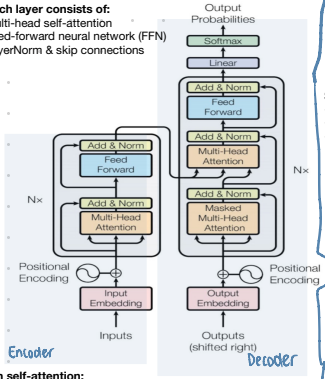
3. Weighted sum
$= 0.731(5) + 0.269(10)$
$= 6.345$

10. Transformer

What Is a Transformer?
A Transformer is a deep learning architecture
Unlike RNNs, it:
Processes all words in parallel
Uses self-attention to understand relationships between words
Is the foundation for models like BERT, GPT, T5, etc.

Encoder	Decoder
Processes input	Generates output
Uses self-attention	Uses masked self-attention + cross-attention
Has 6 identical layers	Also has 6 identical layers

Each layer consists of:
Multi-head self-attention
Feed-forward neural network (FFN)
LayerNorm & skip connections



In self-attention:
Every word compares itself to every other word.
It answers: "How important is word X to word Y?"

Final Output Layer
Linear Projection:
•Primary Role: Adjusting dimensionality.
•The linear layer serves to change the dimensionality of the feedforward network's output to match the size of the vocabulary.
•This ensures that the output has a dimension corresponding to every word in the dictionary.
Softmax Activation:
•This function transforms the linear layer's output into probabilities,
•Its applied to predict the next word.

ENCODER

A Transformer encoder has two main parts:
Self-Attention
Feedforward Neural Network (FFN)
The image focuses on the Self-Attention part

▸ **Self-Attention:**
▸ Input: Matrix X
▸ Linear Transformations to generate Query (Q), Key (K), and Value (V) matrices:

$$Q = W_Q^T X, \quad K = W_I^T X, \quad V = W_V^T X$$

▸ Compute attention output Z using the formula:

$$Z = V\text{softmax}\left(\frac{1}{\sqrt{p}} Q^T K\right)$$

▸ Residual Connection:

$$X + Z$$

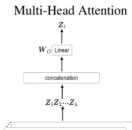
▸ Normalization:

$$(X + Z)$$

Structure of the Feed Forward Network
• Linear Layer 1
• ReLU Activation
• Linear Layer 2

FEED-FORWARD NETWORK
After attention, each token goes through a feedforward neural net, individually.
• The Feed Forward Network (FF) is applied independently to each position in the input sequence.
• Despite individual processing, all positions share the same set of weights and biases in the FFN.
• Key Points:
• Shared parameters ensure consistency in processing across all positions.
• Enables the model to generalize learnings from one position to all positions.
• Facilitates parallel processing of the sequence, enhancing computational efficiency.

MULTI-HEAD ATTENTION
Instead of doing self-attention once, we do it multiple times in parallel using different learned projections:
Why?
Each head focuses on different parts of the sentence (e.g., grammar, position, syntax).



Multi-Headed Attention

$$Q_1 = W_Q^{1\,T} X$$
$$K_1 = W_K^{1\,T} X$$
$$V_1 = W_V^{1\,T} X$$
$$Z_1 = V\text{softmax}\left(\frac{1}{\sqrt{p}} Q_1^T K_1\right)$$
$$\vdots$$
$$Q_h = W_Q^{h\,T} X$$
$$K_h = W_K^{h\,T} X$$
$$V_h = W_V^{h\,T} X$$
$$Z_h = V\text{softmax}\left(\frac{1}{\sqrt{p}} Q_h^T K_h\right)$$

$$\begin{pmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_h \end{pmatrix} = \text{Concat}(head_1, \ldots, head_h)$$

$$Z = \text{MultiHead}(Q, K, V) = W_0^T \begin{pmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_h \end{pmatrix}$$

$$\text{MultiHead}(Q, K, V) = W_O{}^T \text{Concat}(head_1, \ldots, hea$$

Global vs Local
Attention Mechanism:
• Global Understanding: Captures relationships among different
• Context Aggregation: Spreads relevant information across the sequence, enabling each position to see a broader context.
positions in the sequence.
Feed-Forward Networks (FFN):
• Local Processing: While attention looks across the entire sequence, FFN zooms back in to process each position independently.
• Individual Refinement: Enhances the representation of each position based on its own value, refining the information gathered so far.

Synergy of Attention and FFN:
• Holistic Understanding: The combination of global interaction observation (attention) and individual assessment (FFN) provides a holistic understanding of both group dynamics and individual performances.
• Balanced Processing: A balanced approach to processing global relationships and local, position-specific information, leading to richer representations and enhanced learning.

DECODER

$$Z := \text{maskedAttention}(Q, K, V) = V\text{softmax}\left(\frac{1}{\sqrt{p}}(Q^T K + M)\right),$$

where the mask matrix $M \in \mathbb{R}^{n \times n}$ is:

$$M(i,j) := \begin{cases} 0 & \text{if } j \le i, \\ -\infty & \text{if } j > i. \end{cases}$$

• Masked self attention.
• Cross Attention : allows each position in one sequence to attend over all positions in another sequence.
•Query (Q): Originates from a position in the first sequence, i.e. the output of a previous layer in the decoder.
•Memory Keys (K) and Values (V): Both come from all positions in the second sequence, i.e. the output of the encoder.

In the decoder:
MASKED SELF-ATTENTION
While generating output word-by-word, we mask future tokens.
The model can't "cheat" by looking at words it hasn't generated yet.
CROSS-ATTENTION
After masked self-attention:
Each decoder word uses cross-attention to attend to encoder outputs.
So the decoder knows:
"What parts of the input should I focus on to generate this word?"

Positional Encoding
• Problem: Transformers don't have recurrence, so they don't know order. To fix this, we add positional information to token embeddings.
• Solution: It creates unique values for each position, letting the model differentiate:
"This word came first, this one came third..."

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \qquad PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

1. Why do Transformers need Positional Encoding?
Transformers don't use RNNs or CNNs, so they have no built-in sense of order.
Positional Encoding adds information about the position of each word/token in the sequence to the input embeddings so the model can still understand sequence structure.

2. What is the difference between Self-Attention and Cross-Attention?

Self-Attention	Cross-Attention
Input = Output = Same seq.	Input ≠ Output sequences
Used in Encoder/Decoder	Used only in Decoder
Each token attends to all others in the same sequence	Each token in the decoder attends to the encoder output

3. What is Multi-Head Attention and why is it useful?
Multi-head attention:
Splits queries, keys, and values into multiple smaller projections ("heads").
Each head learns different aspects of relationships (e.g., syntax, coreference).
The results of all heads are concatenated and passed through a final linear layer.
It increases the model's ability to attend to multiple things at once.

4. What's the purpose of the Feed-Forward Network (FFN) after Attention?
Attention captures global dependencies across tokens.
FFN refines each token individually using learned transformations.
Together, they model both interactions between tokens and position-wise updates.

5. What does the softmax in scaled dot-product attention do?
Converts raw attention scores into a probability distribution.
Ensures weights sum to 1 and highlight more relevant positions.
The "scaling" (division by √d) avoids extreme softmax outputs when dot products are large.

MUITI HEAD Attention Scaled Dot Product

Q) x=[1,2,3,4], use 2 attention heads
Each projection matrix is size (4 × 2) → reducing dimension from 4 to 2.

Head 1:

$$W_Q^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad W_K^{(1)} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \quad W_V^{(1)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}$$

Head 2:

$$W_Q^{(2)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \quad W_K^{(2)} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \quad W_V^{(2)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$$

For Head 1

Step 1: Compute Q, K, V

$$Q^{(1)} = x \cdot W_Q^{(1)} = [1,2,3,4] \cdot W_Q^{(1)} = [1\times1+2\times0+3\times1+4\times0, 1\times0+2\times1+3\times0+4\times1] = [4,6]$$
$$K^{(1)} = x \cdot W_K^{(1)} = [1,2,3,4] \cdot W_K^{(1)} = [1\times1+2\times1+3\times0+4\times1, 1\times1+2\times1+3\times1+4\times0] = [7,6]$$
$$V^{(1)} = x \cdot W_V^{(1)} = [1,2,3,4] \cdot W_V^{(1)} = [1\times0+2\times1+3\times1+4\times0, 1\times1+2\times0+3\times1+4\times0] = [5,4]$$

Step 2: Scaled Dot-Product Attention

$$\text{Dot}(Q,K) = Q \cdot K = 4\times7 + 6\times6 = 28 + 36 = 64$$
$$\text{Scale: } \frac{64}{\sqrt{2}} \approx \frac{64}{1.414} = 45.24$$
$$\text{Softmax}(45.24) \approx 1.0 \quad \text{(Only one value, so weight = 1)}$$
$$\text{So output} = 1.0 \cdot V = [5,4]$$

For Head 2

Step 1: Compute Q, K, V

$$Q^{(2)} = x \cdot W_Q^{(2)} = [1,2,3,4] \cdot W_Q^{(2)} = [1\times0+2\times1+3\times0+4\times0, 1\times1+2\times0+3\times1+4\times1] = [5,8]$$
$$K^{(2)} = x \cdot W_K^{(2)} = [1,2,3,4] \cdot W_K^{(2)} = [1\times1+2\times1+3\times0+4\times1, 1\times0+2\times1+3\times1+4\times1] = [7,9]$$
$$V^{(2)} = x \cdot W_V^{(2)} = [1,2,3,4] \cdot W_V^{(2)} = [1\times1+2\times0+3\times1+4\times0, 1\times1+2\times1+3\times0+4\times0] = [4,3]$$

Step 2: Scaled Dot-Product Attention

$$\text{Dot}(Q,K) = 5\times7 + 8\times9 = 35 + 72 = 107$$
$$\text{Scale: } \frac{107}{\sqrt{2}} = 75.64$$
$$\text{Again, softmax}(75.64) = 1$$
$$\rightarrow \text{Output} = 1.0 \cdot V = [4,3]$$

Final Answer

Final Multi-Head Attention Output:

We concatenate the outputs of both heads:

$$\text{Head 1 output} = [5,4] \text{Head 2 output} = [4,3]$$
$$\text{Final Output} = [5,4,4,3]$$

Scaled Dot Product Attention

Q) $q_1: [2,1]$ $k_1: [1,0]$ $k_2: [0,1]$ $k_3: [1,1]$
$v_1: 4$ $v_2: 6$ $v_3: 10$

1. Dot Product
$s_1: 2\cdot1+1\cdot0 = 2$
$s_2: 2\cdot0+1\cdot1 = 1$
$s_3: 2\cdot1+1\cdot1 = 3$

2. Scale the scores
$s_1: 2/\sqrt{2} \cdot 1.414$
$s_2: 1/\sqrt{2} \cdot 0.707$
$s_3: 3/\sqrt{2} \cdot 2.121$

3. Softmax
$a_1: \frac{e^2}{e^2+e^1+e^3} \cdot 0.284$
$a_2: \frac{e^1}{e^2+e^1+e^3} \cdot 0.14$
$a_3: \frac{e^3}{e^2+e^1+e^3} \cdot 0.576$

4. Sum of weights
$: 4(0.284) + 6(0.14) + 10(0.576)$
$= 7.796$

Positional Encoding

Q) POS: 1, d: 4 → so i:0 and i:1 → offset every 2 dim : increases

1. Dim 0 → sin → even
$PE(1,0) = \sin\left(1/10000^{0/4}\right) = 0.8415$
2. Dim 1 → cos → odd
$PE(1,1) = \cos\left(1/10000^{0/4}\right) = 0.5403$
3. Dim 2 → sin
$PE(1,2) = \sin\left(1/10000^{2/4}\right) = 0.0099$
4. Dim 3 → cos
$PE(1,3) = \cos\left(1/10000^{2/4}\right) = 0.99995$

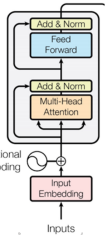
So final Positional Encoding Vector for Position 1 and d:4
(0.8415, 0.5403, 0.0099, 0.9995)

Q) POS: 2, d: 4

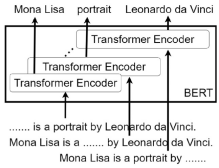
i:0
1. Dim 0 → Sin
$PE(2,0) = \sin\left(2/10000^{0/4}\right) = 0.9093$
2. Dim 1 → cos
$PE(2,1) = \cos\left(2/10000^{0/4}\right) = -0.4161$
i:1
3. Dim 2 → sin
$PE(2,2) = \sin\left(2/10000^{2/4}\right) = 0.0199$
4. Dim 3 → cos
$PE(2,3) = \cos\left(2/10000^{2/4}\right) = 0.9998$

11. Self Supervised Learning → BERT, GPT

Bidirectional Encoder Representations from Transformers (BERT)
•Pre-training of Deep Bidirectional Transformers for Language Understanding
• BERT is built using **transformer encoder blocks.**



Masked language model
• Masks words in the input and asks the model to predict the missing word.



BERT: Bidirectional language model
• Masks words in the input and asks the model to predict the missing word.
• Additional task: Given two sentences (A and B), is B likely to be the sentence that follows A, or not?

• BERT is designed to pretrain bidirectional representations from unlabeled text.
• Jointly conditioning on both left and right context.
• The pre-trained BERT model can be finetuned with just one
• It creates state-of-the-art models for a wide range of tasks, such as question answering and language inference.
additional output layer.

[CLS] Token in BERT
• The [CLS] token is prepended to the input text and travels through the Transformer layers alongside other tokens.
• All tokens, including [CLS], gather contextual information from the entire sequence due to the self-attention mechanism.
• For sentence-level tasks, the final hidden state of the [CLS] token is used as the sentence representation.
• During fine-tuning on a specific task, the model learns to imbue the [CLS] token with a meaningful representation of the entire sentence, optimized for that task.
• Example Usage: In classification tasks, the [CLS] token representation is fed into a classifier to determine the sentence's class.

BERT is basically a trained Transformer Encoder stack
1.Transformer:
1. Encoder Layers: 6
2. FFNN Hidden Layer Units: 512 3. Attention Heads: 8
2.BERT Base:
1. Encoder Layers: 12
2. FFNN Hidden Layer Units: 768 3. Attention Heads: 12
4. Total Parameters: 110 million
3.BERT Large:
1. Encoder Layers: 24
2. FFNN Hidden Layer Units: 1024 3. Attention Heads: 16
4. Total Parameters: 340 million

1.Multilingual BERT:
1. Trained on 104 different languages, capable of "zero-shot" adaptation to a new language domain.
2.Domain Specific BERT Variants:
1. BioBERT: Retrained on a biomedical corpus.
2. SciBERT: Trained on over one million published articles.
3. BERTweet: A RoBERTa model trained on 850 million tweets. 4. FinBERT: Adapted to the financial domain.

• RoBERTa:
•OptimizesBERT'strainingprocessbyusingmoredata,largerbatchsizes,and longer training times, resulting in improved performance on NLP tasks
• TinyBERT:
AsmallerandfasterversionofBERTdesignedforresource-constrained environments, retaining competitive performance with significantly fewer parameters.

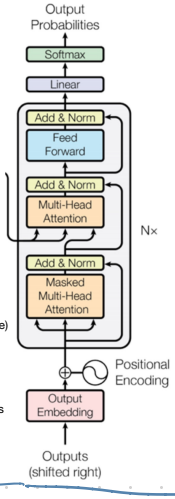
Pretraining Objectives
• Supervised training conducted on downstream tasks provided by GLUE and SuperGLUE benchmarks, reformulated into text-to-text tasks.
• Self-supervised training employs corrupted tokens: 15% of tokens are randomly removed and replaced with sentinel tokens.

Generative Pre trained Transformer (GPT)
• Improving Language Understanding by Generative Pre-Training
• The GPT is built using **transformer decoder blocks.**

Predict the next word, given all of the previous words
• **Architecture:**
• StackofTransformerdecoderblocks.
• No encoder, hence no cross-attention module.
• Components: Positional encoding, masked multihead self-attention, and feedforward network.
• **Directionality:**
• Only considers previous (left) words in attention, not bidirectional like BERT.
• Utilizesmaskedmultiheadself-attention for this purpose.

GPT 1
• Released: 2018
• Parameters: 117 Million
• Layers: 12
• Training Data: Books1 Corpus
• Focus: Unsupervised pre-training, Transformer architecture, large-scale language modeling
GPT 2
• Released: 2019
• Parameters: ~1.5 Billion
• Layers: 48
• Training Data: 40GB (English)
• Focus: Transformer architecture, self-attention mechanism

GPT 3
• Released: 2020
• Parameters: 175 Billion
• Layers: 175
• Training Data: 570GB (Multilingual)
• Focus: Few-shot learning, prompt engineering, Python support
GPT 4
• Release: Not Yet
• Parameters: ~100 Trillion (speculative)
• Layers: Unknown
• Training Data: Larger, more diverse (speculative)
• Focus: GPT-4 is known to be a multimodal model, capable of processing both text and image inputs to generate text outputs. Advanced few-shot learning, improved NLU and NLG, reasoning and inference



T5 - Text-to-Text Transfer Transformer.
• Introduced as a unified framework for addressing NLP tasks by converting them into a text-to-text format.
• Key Focus: Transfer learning - Pre-training on data-rich tasks followed by fine-tuning on downstream tasks.

• Operates on an **encoder-decoder** model.
• Pretrained on a multi-task mixture of both unsupervised and supervised tasks, each converted into a text-to-text format.
• Utilizes relative scalar embeddings; encoder input padding can be performed on both left and right.

Task Adaptation
• T5 adapts to various tasks by prepending task-specific prefixes to the input, e.g., for translation: "translate English to German: ...", for summarization: "summarize: ...".
• Achieves very good results on many benchmarks including summarization, question answering, and text classification.

The Challenge:
GPT just predicts what comes next — but it doesn't inherently follow instructions.
The Fix: RLHF (Reinforcement Learning with Human Feedback)
3-Step Process:
Supervised Fine-Tuning (SFT): Train with human-curated answers
Reward Model (RM): Learn what answers humans prefer
Reinforcement Learning (RLHF): Optimize GPT to produce better aligned outputs

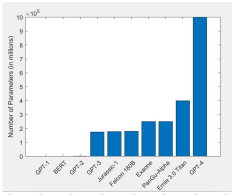
Core Challenge with GPT Models:
• GPT models predict the next token based on historical data, lacking an innate ability to follow instructions.
• GPT (Generative Pre-trained Transformer) models, at their core, predict the next word or token in a sequence based on the probabilities derived from pre-training on extensive text corpora.
• They don't inherently "understand" instructions or follow commands but generate what's statistically likely to come next, given their training.

The Goal of Alignment:
• The aim is to align GPT's responses with specific user instructions and ethical standards, beyond just generating probable text.
• The primary objective is to bridge the gap between these statistical predictions and meaningful adherence to instructions provided by users.
• This involves ensuring that the AI's responses are not just contextually appropriate or conversationally relevant but also aligned with the specific intentions, ethical expectations, and task-oriented goals of the user.

Why It Matters:
•This alignment is crucial for enhancing GPT's reliability, ensuring it respects user intent and ethical norms.
• Without this alignment, while a model like GPT might produce grammatically correct and contextually relevant content, it might diverge from user instructions or produce content that's inappropriate or misaligned with the user's ethical, cultural, or personal expectations.
• The goal is to enhance the model's reliability in following directives accurately, respecting ethical boundaries, and fulfilling the user's actual needs, thereby making the technology more trustworthy and effective in real-world applications.

Key Areas of Focus in AI Alignment
1.Learning from Human Feedback:
• Tailoring AI through human interaction.
2. Training to Follow Instructions:
• Teaching AI specific task adherence.
3. Evaluating AI Harms:
• Identifying risks in AI outputs.
4. Modifying Behavior to Mitigate Harms:
• Adjusting AI to prevent negative impacts.

Number of Parameters in Various Language Models



GPT 3 Applications
• Create fiction so close to human level
• Write Poem
• Chat similar to humans
• Write computer codes
• Design applications
• Summarize texts
• Answer questions
• Design application (Figma): Enter the description of an application. Then the Software can design an app for that description.
•Write code (): Enter the description of an application. Then the Software will generate react code for described app
• Fill a table: Make an empty table with some column names. Then the model fills the table by predicting what the entries of the table should be
• Semantic search: Go to a web page like Wikipedia and ask a question. Then the model can show you the paragraph where the answer of your question can be found there.

1. Main difference between self-supervised and supervised learning?
Self-supervised learning uses unlabeled data and creates learning objectives from the data itself (e.g., predicting a missing word).
Supervised learning requires labeled datasets with human-annotated answers.

2. Why does BERT use bidirectional attention but GPT does not?
BERT is trained to understand context from both sides of a word to capture deeper meaning (e.g., in masked word prediction).
GPT is trained for generating text, so it must only look at previous tokens, not future ones — hence unidirectional.

3. How does Masked Language Modeling work in BERT?
Random words in a sentence are replaced with a [MASK] token.
The model learns to predict the masked word based on its left and right context.
Example: "The cat [MASK] on the mat." → predict "sat"

4. What is the purpose of the [CLS] token in BERT?
The [CLS] token is prepended to every input. After going through the Transformer, it represents the whole sentence.
Used for sentence-level tasks like classification.

5. Function of masked multi-head self-attention in GPT?
It ensures that each word only attends to previous words, not future ones — important for generating text sequentially.

6. Why can GPT generate text but BERT cannot?
GPT is trained to predict the next word, making it naturally suited for text generation.
BERT is bidirectional and masked, so it lacks a natural left-to-right generation structure.

7. What does NSP (Next Sentence Prediction) teach BERT?
NSP trains BERT to understand sentence relationships — whether one sentence follows another in natural text.

8. 3 steps in GPT alignment with RLHF?
Supervised Fine-Tuning (SFT)
Training a Reward Model (RM)
Reinforcement Learning from Human Feedback (RLHF)

12. CLIP

📌 1. Introduction
Motivated by NLP's success with task-agnostic web-scale pretraining (e.g., GPT-3).
Goal: Can we do the same for vision, by training on natural language supervision instead of human-labeled datasets?
CLIP achieves zero-shot performance (without training on the target task).
Core Idea: Align images and texts in a joint embedding space using contrastive learning.

📌 2. Approach
2.1 Natural Language Supervision
Uses image-text pairs from the internet.
Treats language as a rich source of supervision, more scalable than labels.
Leverages contrastive learning to associate correct (image, text) pairs and separate mismatched ones.

2.2. Dataset Construction
400 million image-text pairs scraped from the internet.
Different from prior datasets like ImageNet – unfiltered and noisy but massive in size.
No manual labeling.

2.3. Model Architecture
Dual-encoder architecture:
Image Encoder: ResNet or Vision Transformer (ViT)
Text Encoder: Transformer (e.g., 12-layer, 8-head, 63M params)
Each encoder maps its input to a 512-dimensional embedding.
Uses dot product similarity in the shared embedding space.

📌 Figure 1 (Three-Step Summary):
Contrastive Pre-training: Learn to associate matching image-text pairs.
Dataset Classifier from Label Text: Text prompts like "A photo of a {label}" are encoded.
Zero-Shot Prediction: Predict the class whose text embedding best matches the image embedding.

2.4 Training
Large-scale training: up to 592 GPUs, 18 days, 32 epochs.
Optimizer: Adam with weight decay + cosine LR schedule.
Large batch sizes: 32,768 samples.
Uses InfoNCE loss (a symmetric contrastive loss).
Learns a temperature parameter τ to scale logits.

📌 3. Experiments
3.1 Zero-Shot Transfer
Instead of evaluating representation quality (like in self-supervised learning), they measure task generalization.

Zero-shot: CLIP predicts by choosing the class text whose embedding is closest to the image.

3.2 Robustness
CLIP models are more robust to distribution shifts than supervised baselines.
Shows better generalization to datasets outside training distribution.
📌 Figure 2: Shows CLIP's 3x to 4x greater efficiency compared to Bag-of-Words and language-modeling baselines.

📌 4. Results
Evaluated on 30+ datasets: OCR, action recognition, geo-localization, emotion recognition.
CLIP outperforms supervised models on many.
Example: Matches ResNet-50 on ImageNet without any ImageNet training.
Few-shot learning is possible via fitting linear classifiers on frozen CLIP features.
📌 Figure 15: Shows few-shot performance drops compared to zero-shot due to lack of sample efficiency.
📌 Table 2: Human vs CLIP accuracy on Oxford Pets. CLIP zero-shot beats human zero-shot.

📌 5. Data Overlap Analysis
Checks if evaluation datasets leaked into the training set.
Built a duplicate detector using ResNet + custom augmentation.

📌 6. Bias and Fairness (FairFace Analysis)
Evaluated CLIP on bias across race, gender, and age.
CLIP shows significant biases (especially in zero-shot mode).
Example: Underperforms on non-white FairFace categories compared to white.
📌 Table 3 & 4: CLIP's accuracy on race/gender/age for white and non-white categories.

📌 7. Broader Impacts
CLIP can classify arbitrary categories (e.g., shoplifters) which raises ethical concerns.
Promotes flexibility but also risk of misuse.
Large models like CLIP require societal accountability.

📌 8. Related Work
Builds on contrastive learning, webly supervised learning, image-text retrieval (e.g., VirTex, ConVIRT, etc.)
Distinct from works that rely on complex attention-based architectures like ViLBERT or VisualBERT.

📌 9. Conclusion
CLIP demonstrates the feasibility of language-supervised pretraining for vision.
Offers strong zero-shot generalization and task-agnostic transfer.
Promising direction, but has biases, poor few-shot performance, and data limitations.

🧪 Result Interpretation
Q10: What is one area where CLIP underperforms?
A: Image retrieval in MS-COCO when compared to fine-tuned models.

Q11: What datasets are used to evaluate OCR performance?
A: MNIST, SVHN, IIIT5K, Hateful Memes, SST-2.

Q12: What ethical concerns does CLIP raise?
A: Potential for misuse in surveillance, bias in classification, and unintended applications.

Q7: On how many datasets was CLIP evaluated?
A: Over 30 datasets across OCR, geo-localization, fine-grained classification, facial emotion recognition, and more.
Two primary evaluation suites:
Kornblith's 12-dataset benchmark
A broader 27-dataset suite

Q8: Which tasks does CLIP outperform ResNet-50 significantly?
A: Video action recognition (e.g., Kinetics700 and UCF101), OCR datasets, and general object classification.
Example: +14.5% on Kinetics700

Q9: On which tasks does CLIP underperform?
A: More specialized or abstract tasks:
Satellite imagery (EuroSAT)
Tumor detection (PatchCamelyon)
Synthetic counting (CLEVRCounts)
Traffic signs (GTSRB)

🧪 ZERO-SHOT vs FEW-SHOT vs SUPERVISED
Q10: Why does CLIP's zero-shot outperform few-shot in some cases?
A: CLIP is not optimized for few-shot learning. The zero-shot classifier is globally optimized via contrastive learning, while few-shot uses a weak linear probe over fixed features, often lacking enough data to generalize

Q11: What happens when CLIP is fine-tuned on ImageNet?
A: Accuracy increases by 9.2% on ImageNet, but robustness decreases on other datasets, suggesting overfitting to ImageNet distribution.

Q12: What's CLIP's advantage in terms of robustness?
A: Compared to ImageNet-trained models, CLIP shows up to 75% improvement in robustness across distribution shift datasets (e.g., ImageNet-A, ImageNet-R, ObjectNet)

⚖️ BIAS, FAIRNESS & ETHICAL IMPLICATIONS
Q13: What benchmark was used to analyze bias in CLIP?
A: FairFace – diverse face dataset categorized by race, gender, and age
.
Q14: What are CLIP's bias-related issues?
Zero-shot CLIP underperforms on non-white races compared to white.
Class design bias: If the user defines biased class labels (e.g., "criminal"), CLIP may reinforce harmful stereotypes.
Representation harms: Model classifies people of color more often into "animal" or "crime" categories in label sets containing such terms

Q15: How was intersectional bias studied?
A: Accuracy measured across race × gender categories. CLIP performs above 95% across most gender categories but shows disparity in age and race

Q16: What other methods were used to detect duplicates or data contamination?
A: A custom near-duplicate detector using synthetic augmentations (zoom, rotation, crop, JPEG compression, etc.) trained using a modified ResNet-50 and InfoNCE loss

📚 BROADER IMPACTS
Q17: What are potential misuses of CLIP?
A: Tasks like:
Surveillance (e.g., classifying "shoplifters")
Biased classification based on race/gender
Unethical applications in facial recognition or social profiling

Q18: Why is CLIP's interface considered "flexible but dangerous"?
A: Users can easily define arbitrary categories using natural language. This makes CLIP powerful but vulnerable to misuse if the prompts encode bias or offensive semantics
.

✅ PRACTICE QUESTIONS & ANSWERS
Q1: What is CLIP and what problem does it solve?
A: CLIP is a vision-language model that learns to associate images and text using contrastive learning on a web-scale dataset. It enables zero-shot learning for image classification tasks without needing task-specific training.

Q2: How does CLIP perform zero-shot classification?
A: By embedding a set of text prompts (like "A photo of a cat") and selecting the one whose embedding is closest (dot product similarity) to the image embedding.

Q3: What are the two types of encoders used in CLIP?
A: Image encoder (ResNet or ViT) and Text encoder (Transformer).

Q4: What loss function does CLIP use?
A: Symmetric InfoNCE loss (cross-entropy over pairwise cosine similarities).

📊 Technical / Architecture Questions
Q5: What is the dimensionality of the embedding space?
A: 512-dimensional shared embedding space.

Q6: How is the temperature parameter τ used?
A: Scales the logits before softmax to control sharpness in the similarity distribution.

Q7: Why does few-shot performance drop compared to zero-shot in CLIP?
A: Because CLIP isn't optimized for few-shot settings, unlike humans who benefit significantly from even a single example.

Q4: What are the image encoder variants used in CLIP?
A:ResNet-50 family (e.g., RN50x4, RN50x16, RN50x64).
Vision Transformer (ViT-B/32, ViT-B/16, ViT-L/14).
ViTs are found to be 3× more compute efficient for CLIP's objectives

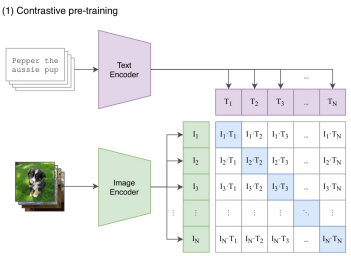
Q5: How is attention used in CLIP's ResNet variant?
A: Replaces global average pooling with a single-layer multi-head attention pooling. This improves performance by letting the network learn which regions to emphasize for the image representation.

Q6: Why is the temperature τ learned during training?
A: It controls the scale of logits before softmax in the contrastive loss. Learning it avoids manual tuning and helps balance similarity scores across batches

🔬 Dataset and Training
Q8: How many image-text pairs are used in CLIP's training?
A: 400 million unfiltered internet (image, text) pairs.

Q9: What was the largest model trained?
A: ViT-L/14@336px and RN50x64, taking up to 592 GPUs and 18 days.

Q1: What is the loss function used in CLIP, and how does it work?
A: CLIP uses the InfoNCE loss, a symmetric contrastive loss. For a batch of image-text pairs:
It maximizes cosine similarity for matching (image, text) pairs.
Minimizes it for all non-matching pairs in the batch (N^2–N negative pairs).
Implements it as a cross-entropy loss over similarity logits, scaled by a learnable temperature parameter τ
.
Q2: What augmentations are applied to images during training?
A: Only a random square crop from resized images. The training is kept minimal and straightforward to avoid data-specific tuning
.
Q3: How does CLIP differ from prior contrastive image-text models like Visual N-Grams?
A: CLIP trains on a dataset 10× larger, uses 100× more compute, leverages transformers, and shows a 95% top-5 ImageNet accuracy, outperforming Visual N-Grams on multiple benchmarks



ishma hafeez notes

13. TVLT

Abstract & Introduction
TVLT is a transformer-based model that learns visual-linguistic representations from raw visual and audio inputs—without relying on text, tokenization, or ASR (Automatic Speech Recognition). Instead of converting speech to text, TVLT directly aligns video frames with audio spectrograms using two core training strategies: masked autoencoding and vision-audio matching. The model is efficient (1/3 the parameters, 28x faster) and achieves comparable or better results than text-based counterparts on tasks like VQA, sentiment analysis, and retrieval

Related Work
The paper contrasts TVLT with:
Text-based models like BERT, which use written language.
Audio-based models that use spectrograms and modality-specific architectures.
VL (vision-language) models that rely on ASR for aligning text and vision.
TVLT fills the gap as the first modality-agnostic, homogeneous transformer handling raw video + audio without text

Architecture: TVLT Overview
Input Embeddings:
Video: 224×224 images → 16×16 patches → Linear projection → 768-dim embeddings.
Audio: Log-mel spectrograms (128 freq bins) → patches → 768-dim embeddings.
Added: modality embeddings + temporal/spatial (for video) or temporal/frequency (for audio) embeddings
Transformer Encoder: 12-layer, modality-agnostic.
Transformer Decoder: 8-layer, used only for masked autoencoding reconstruction.

Training Objectives
Two losses:
Vision-Audio Matching (VAM): Predict if paired audio & video belong together. Uses [CLS] token with a sigmoid classifier. Binary cross-entropy loss
Masked Autoencoding (MAE): Randomly mask 75% of patches in both video and audio inputs. Decoder reconstructs original data. Uses MSE loss on only masked patches

Efficiency and Design
ASR modules in traditional pipelines dominate inference time (e.g., 2890ms just for ASR).
TVLT runs in 103ms for same input due to bypassing ASR.
1/3 the parameters (88M vs. 283M)

Results on Benchmark Tasks
TVLT outperforms text-based counterparts in audio-to-video retrieval and sentiment analysis (e.g., on CMU-MOSEI).
Slightly underperforms on audio-to-image retrieval and VQA but still competitive
Handles emotion categories better than text-based models except for 'disgust'.

Ablation Studies
Joint encoder > separate encoders (better fusion).
Separate decoders > joint decoders (better reconstruction).
VAM + MAE > each alone

Diagram Explanations
Figure 1 – Efficiency Diagram
Shows traditional VL models (with ASR) vs. TVLT:
ASR alone takes ~2.8s.
TVLT pipeline takes ~100ms.
TVLT is 27–28x faster for inference.

Figure 2 – TVLT Architecture
(a) Vision-Audio Matching: Both inputs go through the encoder. A [CLS] token predicts if they are matched (label 0 or 1):
(b) Masked Autoencoding: Masked input → encoder → decoder → reconstructed output. Applies to both spectrogram and video.

Figure 3 & 4 – Reconstruction Visualization
Left: masked input. Middle: reconstruction. Right: original.
Shows TVLT can successfully reconstruct masked video and audio using its learned embeddings

Exam Practice Questions & Answers
Q1: What is the main novelty of the TVLT model compared to traditional VL models?
A: TVLT removes reliance on text and ASR by using a modality-agnostic transformer to learn directly from video frames and audio spectrograms.

Q2: What are the two main training objectives used in TVLT?
A: Vision-Audio Matching (VAM) and Masked Autoencoding (MAE).

Q3: How does Vision-Audio Matching work?
A: It creates positive (matched) and negative (random) video-audio pairs and uses binary cross-entropy loss to train a [CLS] token classifier.

Q4: How is Masked Autoencoding applied in TVLT?
A: Random patches from both audio and video are masked; the decoder reconstructs the original input using MSE loss only on masked patches.

Q5: What does the efficiency comparison show about TVLT?
A: TVLT is over 27x faster in inference than text-based VL models due to removing ASR, and it uses 1/3 the parameters.

Q6: Why is TVLT considered more "green" than other models?
A: It reduces computation by eliminating ASR modules and uses a simpler, unified architecture, although large-scale pretraining is still required.

Q7: What happens when you use joint vs. separate encoders/decoders?
A: Joint encoders perform better than separate ones, while separate decoders perform better than joint ones in masked autoencoding.

Q8: How does TVLT perform on CMU-MOSEI emotion analysis?
A: TVLT outperforms text-based models in recognizing emotions like happy, sad, angry, and fear, thanks to its ability to learn tone and loudness from raw audio.

Q9: What kind of input preprocessing is used for audio in TVLT?
A: Raw audio is converted to 128-bin log-mel spectrograms; patches are created and embedded with temporal/frequency encodings.

Q10: Why does masking speech spans help in audio MAE?
A: Because speech spans contain semantic information; masking them improves the model's ability to learn useful acoustic features.

ARCHITECTURE & ENCODING
Q1: What are the input types accepted by TVLT?
A: TVLT accepts raw video frames and audio spectrograms, not text. These are processed into patch embeddings with added modality, spatial/temporal, and frequency encodings.

Q2: What is the size of each visual patch, and how are visual embeddings created?
A: Each 224×224 frame is divided into 16×16 patches. Each patch is linearly projected into a 768-dimensional embedding.

Q3: How are audio signals transformed for use in TVLT?
A: 1D waveforms are converted into 128-dimensional log mel-spectrograms (T×128), divided into patches (16×16 or 2×128), and projected to 768-dimensional embeddings.

Q4: What is the purpose of modality embeddings in TVLT?
A: They are trainable vectors that inform the model whether the input patch is from vision or audio.

Q5: What is the architecture of the encoder and decoder in TVLT?
A: The encoder is a 12-layer transformer (hidden size 768). The decoder is an 8-layer transformer (hidden size 512), used only during pretraining for masked autoencoding.

PRETRAINING OBJECTIVES
Q6: Why does TVLT use a dual-objective setup for pretraining?
A: To learn strong unimodal and cross-modal representations: Vision-Audio Matching (VAM) captures cross-modal alignment, while Masked Autoencoding (MAE) strengthens unimodal reconstructions.

Q7: How are positive and negative examples generated for the VAM task?
A: Positive: matched video and audio from the same clip. Negative: video from one clip paired with audio from a different one.

Q8: What is the exact loss used for VAM?
A: Binary cross-entropy loss applied to the [CLS] token output of the encoder after passing through a sigmoid classifier.

Q9: How much of the input is masked during MAE?
A: 75% of both visual and audio patches are masked independently.

Q10: What is the final pretraining loss formula?
A: loss = λVAM × lossVAM + λMAE × lossMAE
(where λVAM = 1.0 and λMAE = 0.3)

ABLATION STUDIES
Q11: What did the decoder architecture ablation show?
A: Using separate decoders (one for audio, one for vision) outperformed using a joint decoder.

Q12: What did the encoder ablation study reveal?
A: A joint encoder (shared for vision and audio) outperformed separate encoders followed by a fusion layer.

Q13: How did speech-span masking affect performance?
A: Masking only the speech-active regions in the spectrogram improved performance on both MSR-VTT and VQAv2.

Q14: Between 16×16 and 2×128 patch sizes for audio, which is better?
A: Mixed results. 2×128 was better on MSR-VTT; 16×16 was better on VQAv2. Default chosen is 16×16 for modality alignment.

PERFORMANCE & RESULTS
Q15: Which benchmark tasks were used for evaluation?
A: Audio-to-video retrieval (MSR-VTT, YouCook2, CrossTask), multimodal sentiment (CMU-MOSEI), audio-to-image (Places-400k), and visual QA (VQAv1/VQAv2).

Q16: How does TVLT compare to its text-based version on video tasks?
A: TVLT consistently outperforms the text-based version on video tasks like retrieval and sentiment/emotion analysis.

Q17: Why is TVLT more robust in emotion classification?
A: It directly captures acoustic features like tone and loudness from raw audio, unlike text-based models limited to transcript content.

Q18: In which areas does the text-based model slightly outperform TVLT?
A: On tasks with clean, written text like VQA and audio-to-image retrieval, text-based models have a slight edge.

Q19: What efficiency gains does TVLT offer over traditional ASR pipelines?
A: TVLT is up to 28× faster in inference and uses only 88M parameters vs. 283M+ in ASR-based pipelines.

Q20: What task head is used during finetuning on retrieval tasks?
A: A two-layer MLP maps the encoder's [CLS] token to a score ∈ [0,1] used for match classification.

DATASETS & TRAINING
Q21: Which datasets were used for pretraining?
A: HowTo100M and a 20% subset of YTTemporal180M (called YTT-S).

Q22: How long does pretraining TVLT take?
A: 200k steps over 2 weeks using 4 NVIDIA RTX A6000 GPUs (49GB each).

Q23: How is ASR used in the text-based version?
A: ASR (like SpeechBrain) transcribes speech audio to text, which is then tokenized and embedded.

Q24: How is TTS used in the paper?
A: Text questions from VQAv1/VQAv2 are converted to synthetic speech via WaveNet and used as audio queries.

Q25: What are the hyperparameters used during pretraining?
A: LR = 1e-5, batch size = 4096, weight decay = 0.001, cosine learning rate schedule.

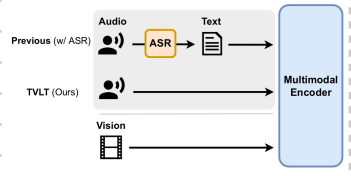
BROADER INSIGHTS
Q26: What key insight about vision-language learning does TVLT challenge?
A: That high-quality vision-language representations require written language. TVLT shows this can be achieved with raw audio instead.

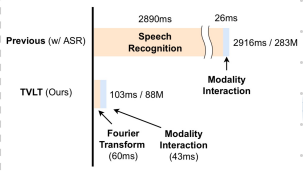
Q27: Why is TVLT considered more compact?
A: Because it avoids heavy modules like ASR and text tokenizers, reducing overall architecture complexity and compute.

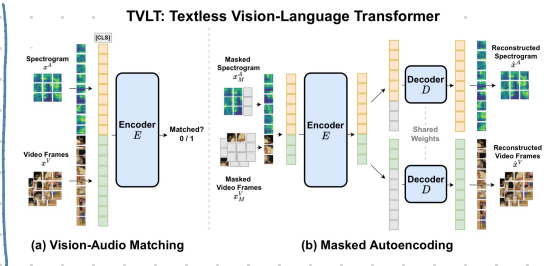
Q28: What advantage does TVLT have in real-time systems like smart assistants?
A: Faster inference and direct audio input makes it ideal for deployment in systems with limited latency budgets.

Q29: How does TVLT handle image-only tasks like VQA?
A: By converting the text questions into synthetic speech via TTS, enabling consistent input modality (audio + vision).

Q30: What is a practical use case where TVLT would outperform text-based models?
A: Emotion recognition from speech and video in real-time, such as in therapeutic AI or expressive virtual agents.







Q2: Attention Masking
You are training GPT on this sequence:
"The dog barked loudly"
Which tokens can the word "barked" attend to during training?
We are predicting the word "barked".
✅ It can only attend to:
"The"
"dog"
✅ It cannot attend to:
"loudly" (it comes after)

Q3: Token Flow
Draw or describe the token flow in a BERT-based classification task using this input:
[CLS] The movie was great [SEP]
What is the role of the [CLS] token?
What happens to the token embeddings after self-attention?
Tokens: [CLS], The, movie, was, great, [SEP]
All tokens go into self-attention.
Each token attends to all other tokens (because it's bidirectional).
The final representation of the [CLS] token becomes the sentence embedding.
For classification tasks (like sentiment analysis), the [CLS] token is passed to a classifier head.

Feature	BERT	GPT	T5
Architecture	Encoder-only	Decoder-only	Encoder–Decoder
Direction	Bidirectional	Left-to-right	Bidirectional (enc), causal (dec)
Pretraining Task	MLM + NSP	Next token prediction	Span corruption
Input Format	Tokenized text + \[CLS]	Plain text	Text-to-text
Token Masking?	Yes	Yes (causal)	Yes (with sentinel tokens)
Can generate text?	No (not directly)	Yes	Yes

what is projection?
Suppose you have an input word embedding:
$x \in \mathbb{R}^{512}$ (i.e., 512-dimensional vector)
You apply a projection matrix $W_Q \in \mathbb{R}^{64\times512}$
$q = W_Q x$ Then $q \in \mathbb{R}^{64}$ is your **projected query vector** projection from 512-D into a **64-D**

All model comparisions

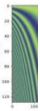
Model	Architecture	Training Objective	Strengths	Limitations	Typical Use Cases
Diffusion Models	Markov chain noise addition + iterative denoising	Denoising score matching	Stable training, diverse & high-fidelity samples	Slow sampling, computationally heavy	High-quality image generation, inpainting
GANs	Generator + Discriminator (adversarial)	Adversarial loss	Fast generation, sharp images	Training instability, mode collapse	Real-time image/video synthesis, deepfakes
VAEs	Encoder-decoder with probabilistic latent space	Variational lower bound	Stable, diverse outputs, latent space	Blurry outputs, posterior collapse	Representation learning, anomaly detection
AEs	Encoder-decoder (deterministic)	Reconstruction loss	Simple, fast, unsupervised learning	No explicit generative modeling	Dimensionality reduction, denoising
BERT	Transformer encoder (bidirectional)	Masked Language Modeling + NSP	Deep understanding, strong NLP performance	Not generative, heavy compute	NLP understanding, classification
GPT	Transformer decoder (unidirectional)	Next-token prediction	Strong text generation	Limited context, large compute	Text generation, chatbots, summarization

Positional Encoding

$\text{position } z_i = x_i + \text{Positional encoding}_i$

Step 1: Get token embeddings
Imagine the embedding layer gives us fixed vectors for tokens:

Token	Embedding vector (dim 6)
I	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
love	[0.5, 0.4, 0.3, 0.2, 0.1, 0.0]
AI	[0.9, 0.8, 0.7, 0.6, 0.5, 0.4]



Step 2: Calculate positional encoding vectors
formula:
$\text{PE}(pos,2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \text{PE}(pos,2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$
For $d = 6$, dimensions $i = 0, 1, 2$:
Calculate denominator $10000^{\frac{2i}{d}}$ for each i:

i	$2i/d$	Denominator = $10000^{2i/6}$
0	0	$10000^0 = 1$
1	2/6=0.333	$10000^{0.333} \approx 21.54$
2	4/6=0.666	$10000^{0.666} \approx 464.16$

Calculate each position:
For **pos = 0**:
$\sin(0/1) = 0, \quad \cos(0/1) = 1$
$\sin(0/21.54) = 0, \quad \cos(0/21.54) = 1$
$\sin(0/464.16) = 0, \quad \cos(0/464.16) = 1$
$\text{PE}_0 = [0, 1, 0, 1, 0, 1]$
For **pos = 1**: $\sin(1/1) = 0.8415, \cos(1) \approx 0.5403$
$\sin(1/21.54) \approx 0.0464, \cos(1/21.54) \approx 0.9989$
$\sin(1/464.16) \approx 0.00215, \cos(1/464.16) \approx 0.9999977$
$\text{PE}_1 = [0.8415, 0.5403, 0.0464, 0.9989, 0.00215, 0.9999977]$
For **pos = 2**:
$\sin(2/1) = \sin(2) \approx 0.9093, \quad \cos(2) \approx -0.4161$
$\sin(2/21.54) \approx 0.0927, \cos(2/21.54) \approx 0.9957$
$\sin(2/464.16) \approx 0.00431, \cos(2/464.16) \approx 0.9999907$
$\text{PE}_2 = [0.9093, -0.4161, 0.0927, 0.9957, 0.00431, 0.9999907]$

Step 3: Add positional encoding to token embeddings

Token	Result (input to Transformer)
I	[0.1+0, 0.2+1, 0.3+0, 0.4+1, 0.5+0, 0.6+1] = [0.1, 1.2, 0.3, 1.4, 0.5, 1.6]
love	[1.3415, 0.9403, 0.3464, 1.1989, 0.10215, 0.9999977]
AI	[1.8093, 0.3839, 0.7927, 1.5957, 0.50431, 1.3999907]

Formula Sheet

Convolutional Layers (CNN)

- **Conv2D Output Size**:
$$\text{Output size} = \left\lfloor \frac{\text{Input size} + 2\times\text{padding} - \text{kernel size}}{\text{stride}} \right\rfloor + 1$$
- **Pooling Output**:
$$\text{Output size} = \left\lfloor \frac{\text{Input size} - \text{kernel size}}{\text{stride}} \right\rfloor + 1$$
- **Number of Weights (Conv Layer)**:
$\text{Weights} = \text{Out channels} \times \text{In channels} \times \text{Kernel height} \times \text{Kernel width}$
- **Number of Biases (Conv Layer)**:
$\text{Biases} = \text{Out channels}$

Positional Encoding (Transformers)

$$\text{PE}(\text{pos}, i) = \begin{cases} \sin\left(\frac{\text{pos}}{10000^{i/d}}\right) & \text{if } i \text{ is even} \\ \cos\left(\frac{\text{pos}}{10000^{(i-1)/d}}\right) & \text{if } i \text{ is odd} \end{cases}$$

Scaled Dot Product Attention

- **Attention Score**:
$\text{score} = \frac{q \cdot k^T}{\sqrt{d_k}}$
- **Softmax**:
$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$
- **Weighted Sum** (Attention output):
$\text{Output} = \sum \text{softmax}(score_i) \cdot v_i$

Diffusion/VAEs

KL Divergence (VAE Loss)

$$D_{KL}(q(z|x)\|p(z)) = \frac{1}{2}\sum(\mu^2 + \sigma^2 - \log(\sigma^2) - 1)$$
$$D_{KL}(P\|Q) = \sum_i P(i)\log\left(\frac{P(i)}{Q(i)}\right)$$

Note: Use ln (natural log)

Harris Corner Detection

Used for detecting corners in an image.

Step 1: Compute image gradients
$I_x = \frac{\partial I}{\partial x}, \quad I_y = \frac{\partial I}{\partial y}$

Step 2: Compute structure matrix MMM
$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$ (smoothed over a wind

Step 3: Harris Response
$R = \det(M) - k \cdot (\text{trace}(M))^2$
- $\det(M) = I_x^2 I_y^2 - (I_x I_y)^2$
- $\text{trace}(M) = I_x^2 + I_y^2$
- $k \in [0.04, 0.06]$

If R
- Large **positive** → **corner**
- Large **negative** → **edge**
- Close to **zero** → **flat region**

Additive Attention/Bahdanau Attention:

You are doing **machine translation** using seq2seq + attention.
We are at **decoder timestep** $i = 1$
We have **3 encoder hidden states**: h_1, h_2, h_3
We have **1 decoder hidden state from previous timestep**: s_0
$h_1 = [1, 0] \quad h_2 = [0, 1] \quad h_3 = [1, 1] \quad s_0 = [1, 2]$
We'll use a score function of the form:
$e_{ij} = v^T \tanh(W_s s_{i-1} + W_h h_j)$ attention vector
$W_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad W_h = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad v = [1, 1]^T$
For $h_1 = [1, 0]$
$W_s s_0 = [1, 2]$
$W_h h_1 = [1 \times 1 + 1 \times 0,\ 1 \times 1 + (-1) \times 0] = [1, 1]$
$W_s s_0 + W_h h_1 = [1 + 1,\ 2 + 1] = [2, 3]$
$\tanh([2, 3]) \approx [0.964, 0.995]$
$e_{i1} = v^T \cdot [0.964, 0.995] = 0.964 + 0.995 = * * 1.959 * *$
For $h_2 = [0, 1]$
$W_h h_2 = [0 \times 1 + 1 \times 0,\ 0 \times 1 + (-1) \times 1] = [1, -1]$
$W_s s_0 + W_h h_2 = [1 + 1,\ 2 + (-1)] = [2, 1]$
$\tanh([2, 1]) \approx [0.964, 0.761]$
$e_{i2} = 0.964 + 0.761 = * * 1.725 * *$
For $h_3 = [1, 1]$
$W_h h_3 = [1 \times 1 + 1 \times 1,\ 1 \times 1 + (-1) \times 1] = [2, 0]$
$W_s s_0 + W_h h_3 = [1 + 2,\ 2 + 0] = [3, 2]$
$\tanh([3, 2]) \approx [0.995, 0.964]$
$e_{i3} = 0.995 + 0.964 = * * 1.959 * *$
Apply softmax to get attention weights a_{ij}
$\text{softmax}(e_{i1}, e_{i2}, e_{i3}) = \frac{e^{e_{ij}}}{\sum e^{e_{ij}}}$
$e^{1.959} \approx 7.09, \quad e^{1.725} \approx 5.61$
$\text{Denominator} = 7.09 + 5.61 + 7.09 = 19.79$
$a_{i1} = \frac{7.09}{19.79} \approx 0.358 \quad a_{i2} = \frac{5.61}{19.79} \approx 0.283 \quad a_{i3} = \frac{7.09}{19.79} \approx 0.358$
Compute context vector c_i
$c_i = a_{i1} h_1 + a_{i2} h_2 + a_{i3} h_3$
$h_1 = [1, 0], \quad h_2 = [0, 1], \quad h_3 = [1, 1]$
$c_i = 0.358[1, 0] + 0.283[0, 1] + 0.358[1, 1] =$
$= [0.358, 0] + [0, 0.283] + [0.358, 0.358] =$
$= [0.716, 0.641]$
Attention weights $[0.358,\ 0.283,\ 0.358]$
Context vector c_i $[0.716,\ 0.641]$

dim of output: $\frac{\text{input} + 2p - k}{s} + 1$ (input size, padding, kernel size, stride)

num. parameters: weights + biases
$i \times f \times f \times O + O$ (1 filter is applied to every output map)
(i: no. of input images/channels; f: filter size only height; O: no. of output maps)

grey scale image - 1 channel
RGB image - 3 channels

Transformers

8) $X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $W_Q \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $W_K \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $W_V = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$

1. Compute Q, k, V

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

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

$V = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$

2. Compute Attention Scores

$Q \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} K^T \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

$QK^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

Scaled by $\sqrt{D}$: $\sqrt{2}$ | 1 | 1

$\frac{QK^T}{\sqrt{2}} \begin{bmatrix} 0.71 & 0.71 \\ 0.71 & 0.71 \end{bmatrix}$

3. Apply Softmax

Softmax (0.71, 0.71) (0.5, 0.5)

$\text{Softmax}\left(\frac{QK^T}{\sqrt{2}}\right) = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$

4. Multiply V

2. Softmax. V

$\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}$

5. Residual Connection

$= X + Z$

$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}$

$= \begin{bmatrix} 1.25 & 0.25 \\ 0.25 & 1.25 \end{bmatrix}$

Hello - [1,2] world [3,4]

8) $X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $W_Q \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $W_K \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $W_V = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$

1. Compute Q, k, V

$Q = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

$K = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$

$V = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.5 & 1 \\ 1.5 & 2 \end{bmatrix}$

2. Compute Attention Scores

$Q \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} K^T \begin{bmatrix} 3 & 7 \\ 3 & 7 \end{bmatrix}$

$QK^T = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 3 & 7 \\ 3 & 7 \end{bmatrix} = \begin{bmatrix} 9 & 21 \\ 21 & 49 \end{bmatrix}$

Scaled by $\sqrt{d}$: $\sqrt{2}$ ∴ d: 2

$\frac{QK^T}{\sqrt{2}} \begin{bmatrix} 6.36 & 14.85 \\ 14.85 & 34.64 \end{bmatrix}$ | $\frac{21}{\sqrt{2}}$

3. Apply Softmax

$\text{Softmax}\left(\frac{QK^T}{\sqrt{2}}\right) = \begin{bmatrix} 0.00021 & 0.99979 \\ 0 & 1 \end{bmatrix}$

4. Multiply V

2. Softmax. V ∴ $V = \begin{bmatrix} 0.5 & 1 \\ 1.5 & 2 \end{bmatrix}$

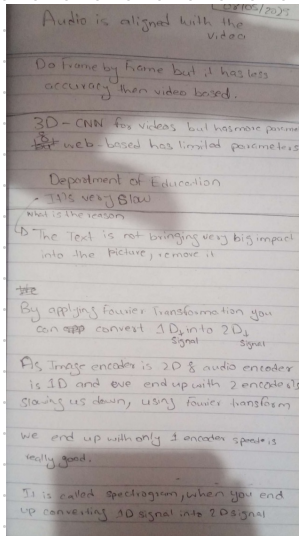
2. $\begin{bmatrix} 0.00021 & 0.99979 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 1 \\ 1.5 & 2 \end{bmatrix} = \begin{bmatrix} 1.49979 & 1.99979 \\ 1.5 & 2 \end{bmatrix}$

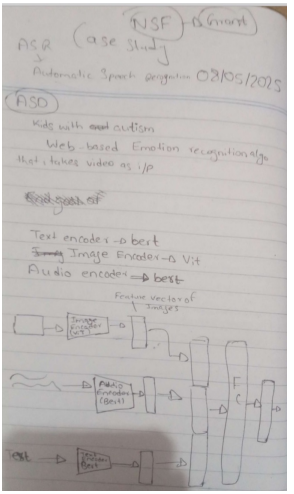
5. Residual Connection

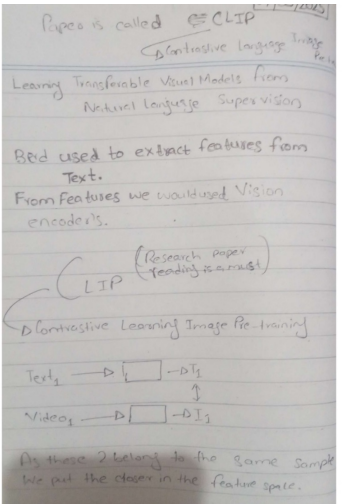
$= X + Z$

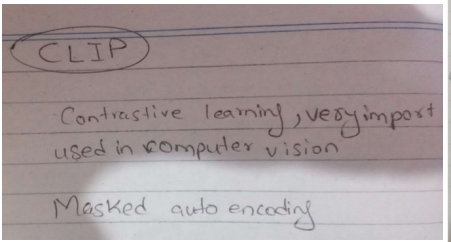
$= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1.49979 & 1.99979 \\ 1.5 & 2 \end{bmatrix}$

$= \begin{bmatrix} 2.5 & 4 \\ 4.5 & 6 \end{bmatrix}$









ishma hafeez notes