AIM 2: Artificial Intelligence in Medicine II

Generative AI

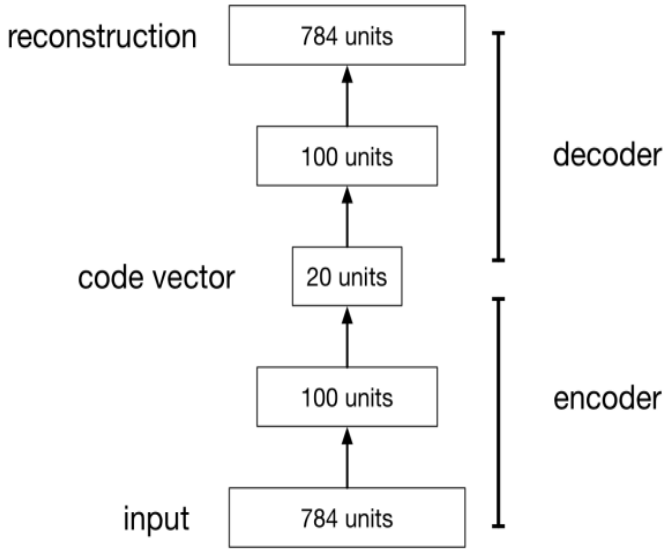
HARVARD
MEDICAL SCHOOL

Outline for Today's class

- Variational Autoencoders
- Generative adversarial Networks
- Generative AI
- LLMs and Multimodal LLMs
- Grounding and RAG
- Healthcare Applications of Generative AI
- Synthetic data Generation
- Data Privacy

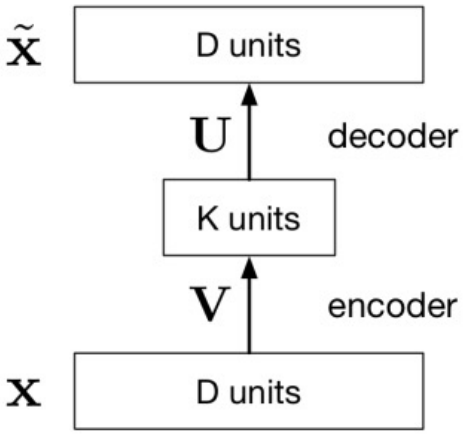
Autoencoders

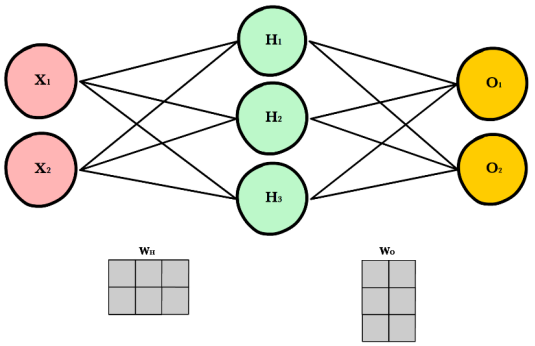
- An autoencoder is a feed-forward neural net whose job it is to take an input **x** and reconstruct **x**.
- To make this non-trivial, we need to add a bottleneck layer whose dimension is much smaller than the input.
- Basically, what is happening here, we are reducing 784 dimension input to only 20 dimension.
  - We are compressing features, only trying to keep important ones
  - Next we are trying to reproduce that using only those important ones



How Autoencoders Works

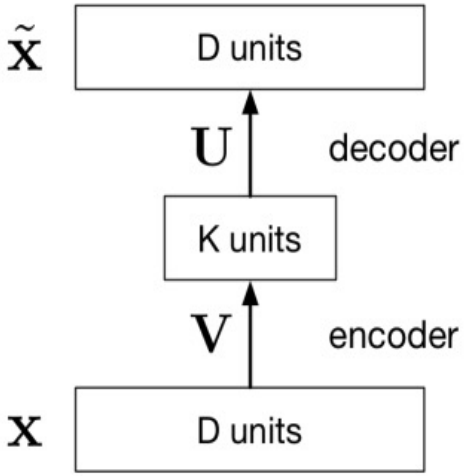
- **x**: Input vector or array
- $\hat{x}$: Output vector or array
- **V**: A matrix of weights. For example if D has 3 neurons, K has 2 neuron, for every connection there will be a weight and it is a 3 * 2 weight matrix
- **U**: Another matrix of weights. For example if D has 3 neurons, K has 2 neuron, for every connection there will be a weight and it is a 2 * 3 weight matrix





How it works

- **Vx**: A matrix multiplication project D dimensional x to k dimensional plane. Shown for 3 to 2. **Dimensionality Reduction**
- **Ux** : Project k dimensional hidden unit to D dimensional $\hat{\boldsymbol{x}}$
- Overall Output is: $\hat{\boldsymbol{x}} = UV\mathrm{x}$ [A linear function]
- How it learn:
  - Learn the U, V matrix or all weights so that we get minimum loss
  - L(x, $\hat{\boldsymbol{x}}$) $= ||x - \hat{\boldsymbol{x}}||^2$
  - We minimize L to learn U,V



Why Autoencoders?

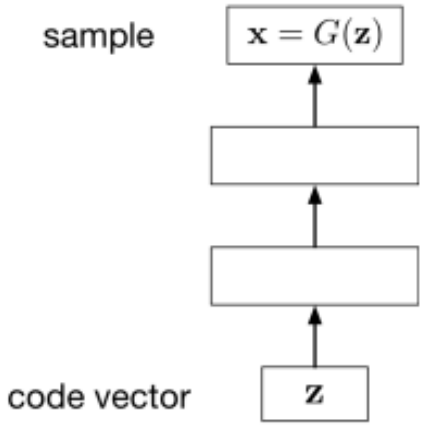
- Map high-dimensional data to two dimensions for visualization
- Compression
- Learn abstract features in an unsupervised way so you can apply them
  to a supervised task
  - Unlabeled data can be much more plentiful than labeled data
- Learn a semantically meaningful representation where you can, e.g.,
  interpolate between different images.

Some limitations of autoencoders

- They’re not generative models, so they don’t define a distribution
- How to choose the latent dimension?

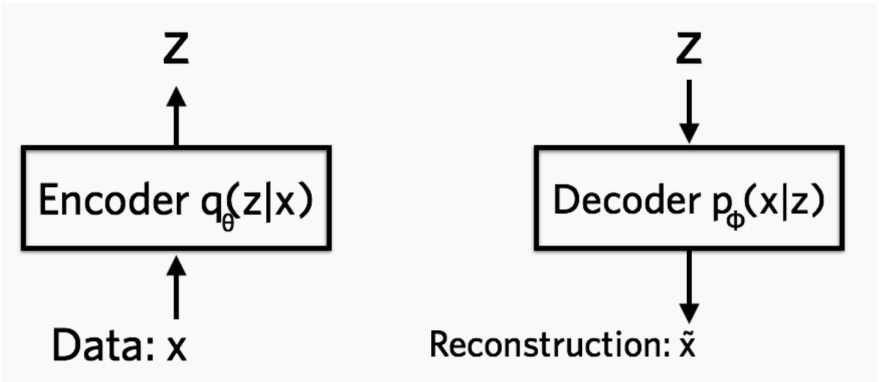
Generative Models

- One of the goals of unsupervised learning is to learn representations of images, sentences, etc.
- A generative model is a function of representation z.
  - X= G(z) [G is the generator function]
- Example: VAE, GAN



Variational Autoencoder (VAE)

- Key idea: make both the encoder and the decoder probabilistic.
- I.e., the latent variables, z, are drawn from a probability distribution depending on the input, X, and the reconstruction is chosen probabilistically from z.



https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

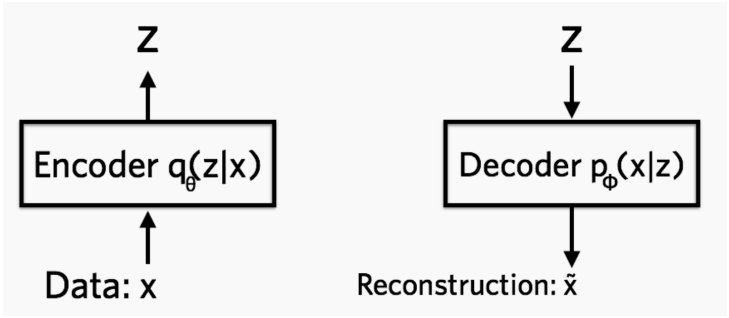
VAE Encoder

- The encoder takes input and returns parameters for a probability density (e.g., Gaussian): I.e., $q_\theta(z \mid x)$gives the mean and co-variance matrix.
- We can sample from this distribution to get random values of the lower-dimensional representation *z*.
- Implemented via a neural network: each input *x* gives a vector mean and diagonal covariance matrix that determine the Gaussian density
- Parameters θ for the NN need to be learned – need to set up a loss function.

https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

VAE Decoder

- The decoder takes latent variable *z* and returns parameters for a distribution. E.g., $p_\phi(x|z)$ gives the mean and variance for each pixel in the output.
- Reconstruction $\tilde{x}$ is produced by sampling.
- Implemented via neural network, the NN parameters ϕ are learned.



https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

VAE loss function

- Loss function for autoencoder: L_2 distance between output and input (or clean input for denoising case)
- For VAE, we need to learn parameters of two probability distributions. For a single input, x_i, we maximize the expected value of returning x_i or minimize the expected negative log likelihood.

$$-\mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)]$$

- This takes expected value w.r.t. z over the current distribution $q_\theta(z|x_i)$ of the loss $-\log p_\phi(x_i|z)$

https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

VAE loss function

- **Problem:** the weights may adjust to memorize input images via *z*. I.e., input that we regard as similar may end up very different in *z* space.
- We prefer continuous latent representations to give meaningful parameterizations. E.g., smooth changes from one digit to another.
- **Solution:** Try to force $q_\theta(z|x_i)$ to be close to a standard normal (or some other simple density).

https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

VAE loss function

- For a single data point x_i we get the loss function

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

- The first term promotes recovery of the input.
- The second term keeps the encoding continuous – the encoding is compared to a fixed $p(z)$ regardless of the input, which inhibits memorization.
- With this loss function the VAE can (almost) be trained using gradient descent on minibatches.

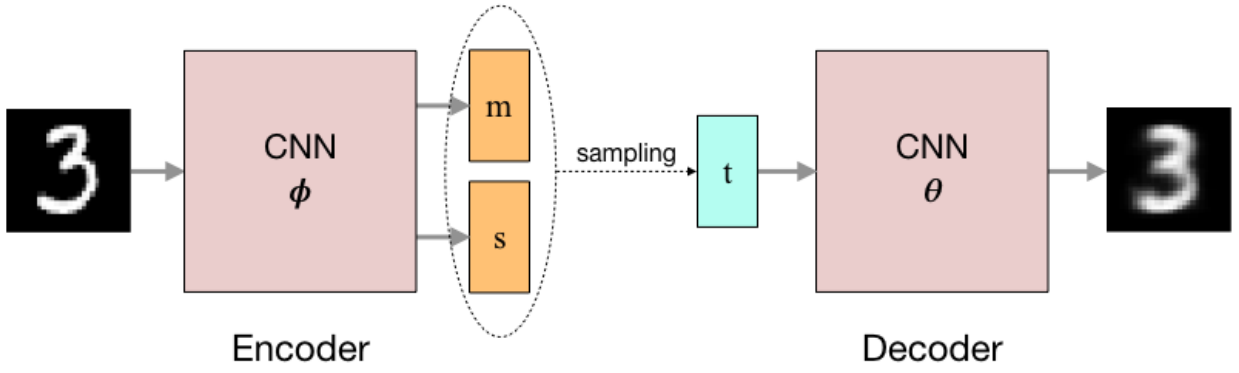
https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

VAE loss function

- For a single data point x_i we get the loss function

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \,||\, p(z))$$

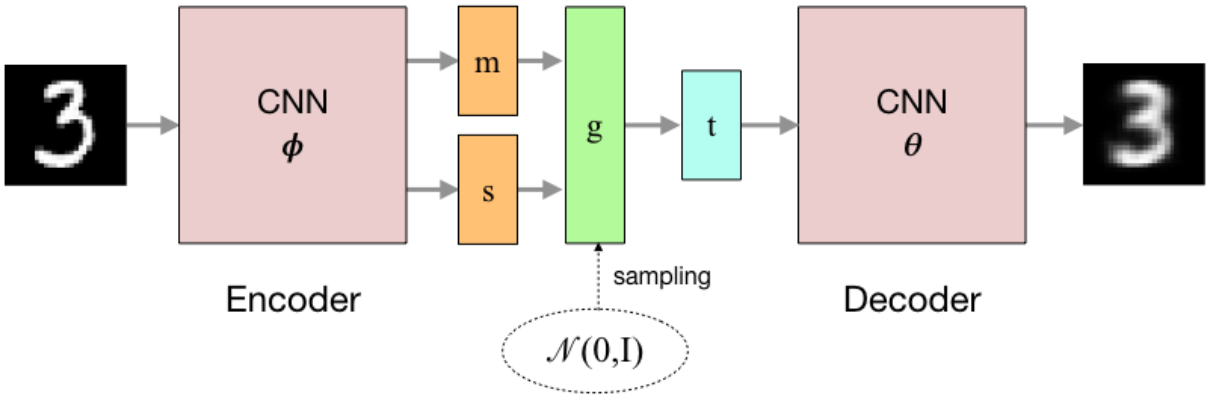
- **Problem:** The expectation would usually be approximated by choosing samples and averaging. This is not differentiable w.r.t. θ and ϕ.



https://nbviewer.jupyter.org/github/krasserm/bayesian-machine-learning/blob/master/variational_autoencoder.ipynb

VAE loss function

- **Reparameterization:** If z is $N(\mu(x_i), \Sigma(x_i))$, then we can sample z using $z = \mu(x_i) + \sqrt{(\Sigma(x_i))}\,\epsilon$, where ϵ is $N(0,1)$. So we can draw samples from $N(0,1)$, which doesn't depend on the parameters.



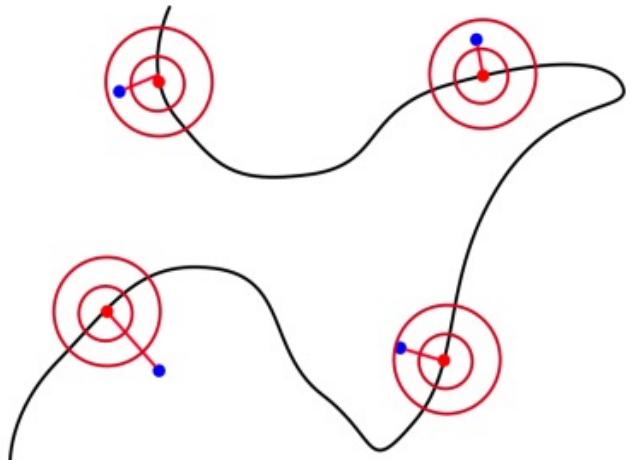
https://nbviewer.jupyter.org/github/krasserm/bayesian-machine-learning/blob/master/variational_autoencoder.ipynb

VAE generative model

- After training, $q_\theta(z|x_i)$ is close to a standard normal, N(0,1) – easy to sample.
- Using a sample of *z* from $q_\theta(z|x_i)$ as input to sample from $p_\phi(x|z)$ gives an approximate reconstruction of x_i, at least in expectation.
- If we sample any *z* from N(0,1) and use it as input to to sample from $p_\phi(x|z)$ then we can approximate the entire data distribution *p*(*x*). I.e., we can generate new samples that look like the input but aren't in the input.

How VAE Learn Distribution

- VAE based on variational inference
- Decoder does is calculate p(z|x)= p(x,z)/p(x)
- What generator part does is:
  - $p(x) = \int p(z)p(x|z)dz$
- A noisy observation model is learnt
  - $p(x|z) = N(x; G_\theta(z), \eta I)$
  - θ is μ and α learnt by the decoder



How VAE Learn Distribution

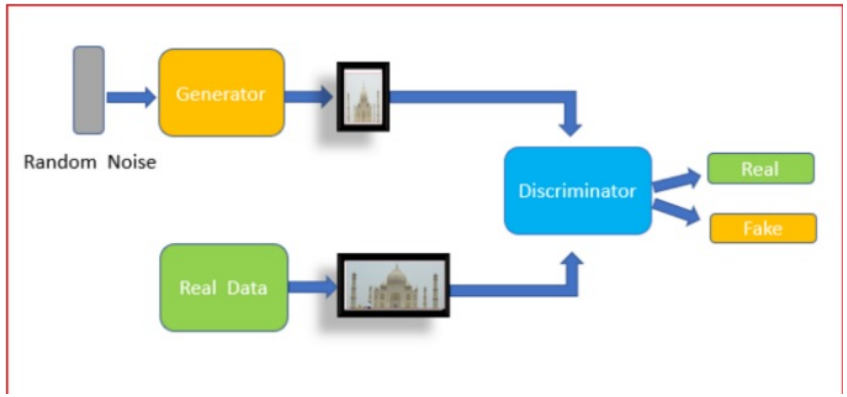
- Direct computation of p(x) is costly
- It introduce a further function to approximate the posterior distribution as
  - $q_\phi(z|x) \approx p_\theta(z|x)$
  - Approximate q(z) which is known to p(z|x)
- the idea is to jointly optimize the generative model parameters θ to reduce the reconstruction error between the input and the output and ϕ to make $q_\phi(z|x)$ as close as $p_\theta(z|x)$
  - Min $D_{KL}(q_\phi(.|x)||p_\theta(.|x))$
- It uses KL divergence to make the parameters similar and define new loss function using ELBO method that does both task at the same time
  - $L_{\theta,\phi}(x) = \log p_\theta(x) - D_{KL}(q_\phi(.|x)||p_\theta(.|x))$

Generative Adversarial Networks (GANs)

- VAEs approximate P(X) using latent variables z, with the mapping between X and z pushed through a NN function approximation that ensures that the transformed data can be well represented by a mixture of Gaussians
- But approximating P(X) directly is complicated, and approximating it well in the space of an arbitrarily defined reconstruction error does not generalize well in practice
- GANs go about approximating P(X) using an indirect approach

Adversarial training

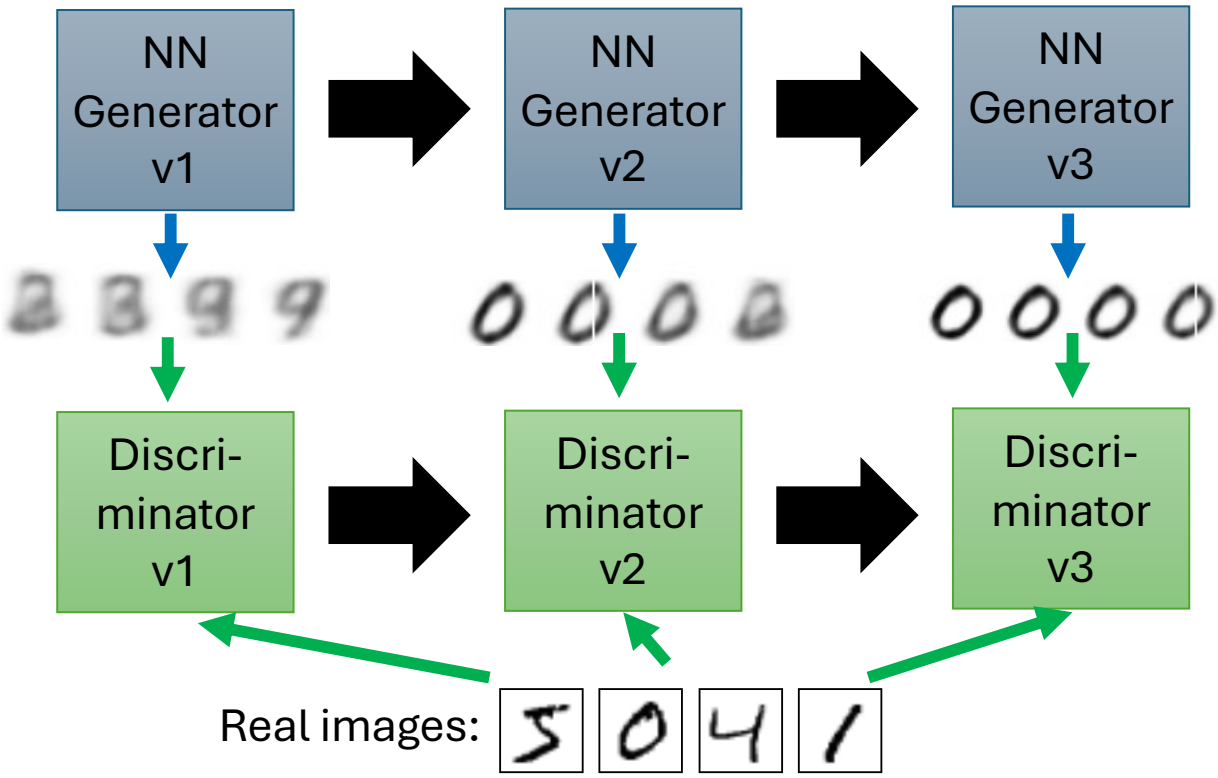
- Two models are trained – a generator and a discriminator
- The goal of the discriminator is to correctly judge whether the data it is seeing is real, or synthetic
  - Objective function is to maximize classification error
- The goal of the generator is to fool the discriminator
  - It does this by creating samples as close to real data as possible
  - Objectively tries to minimize classification error
- No longer reliant on reconstruction error for quality assessment



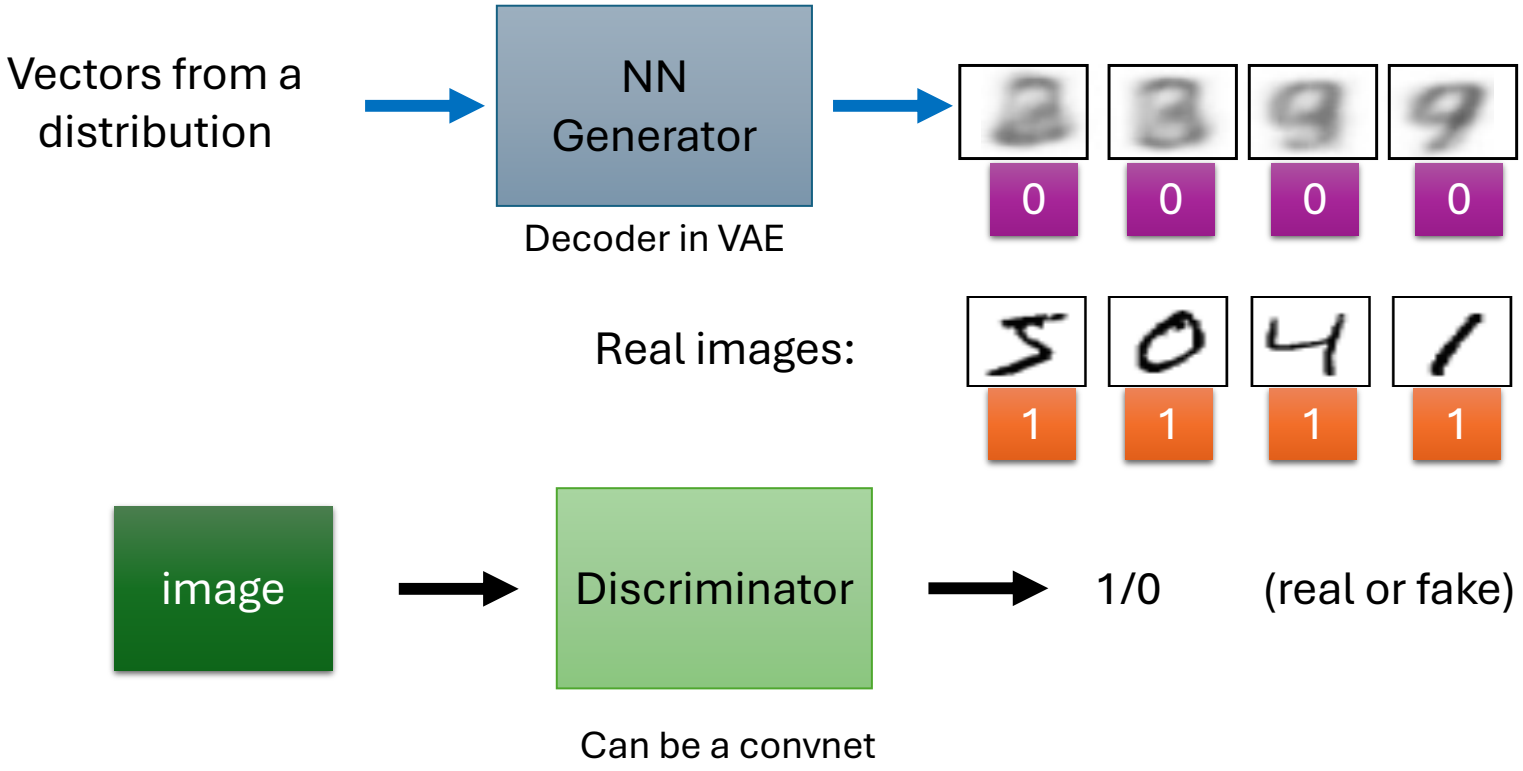
Realistic

Fake

GANs


NN Generator v1
NN Generator v2
NN Generator v3
Discri-minator v1
Discri-minator v2
Discri-minator v3
Real images:

GAN - Discriminator



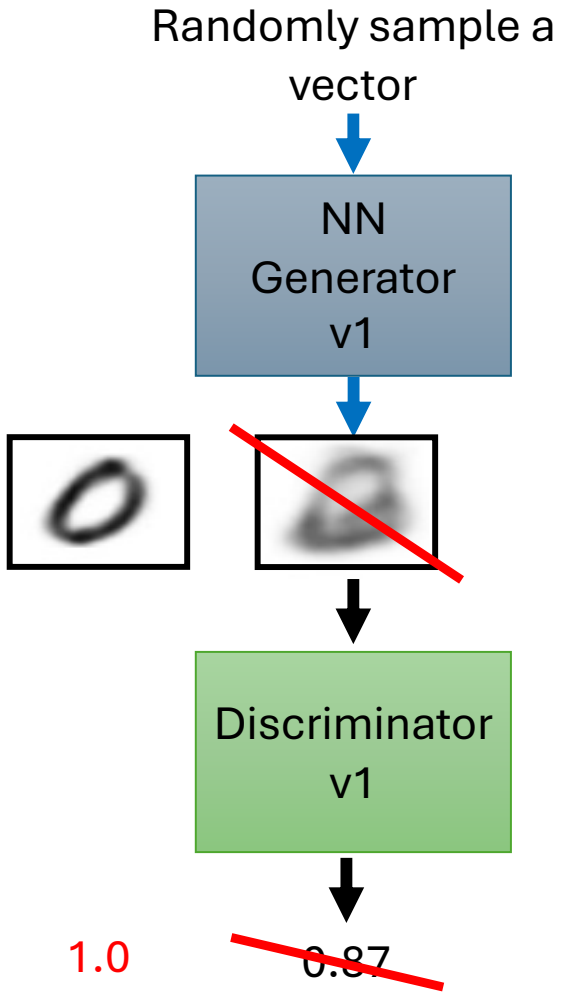
GAN - Generator

"Tuning" the parameters of generator

➡ The output be classified as "real" (as close to 1 as possible)

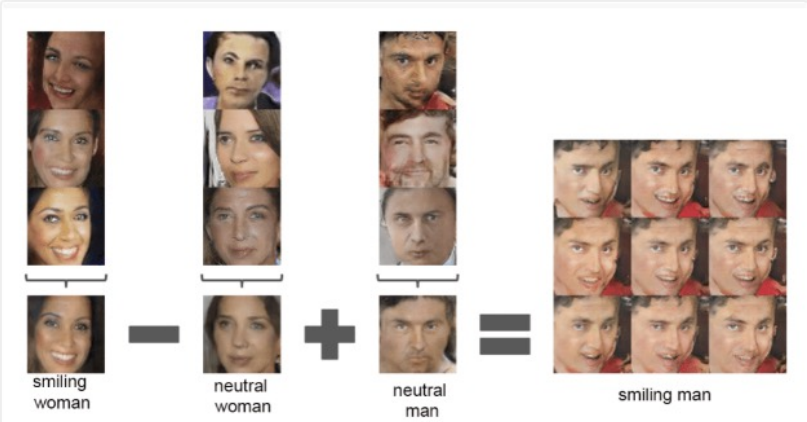
Generator + Discriminator = a network

Use gradient descent to find the parameters of generator

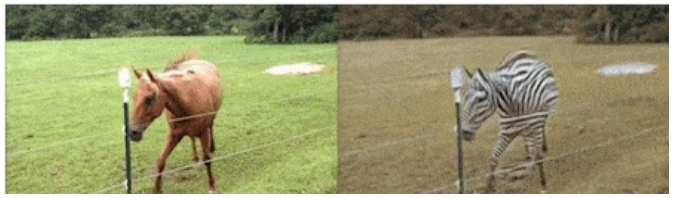


GAN outputs

- The latent space learned in GANs is very interesting
- People have showed that vector additions and subtractions are meaningful in this space
- Can control novel item compositions almost at will
- A big ‘deepfakes’ industry is growing up around this



Example of Vector Arithmetic on Points in the Latent Space for Generating Faces With a GAN.
Taken from Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.



https://machinelearningmastery.com/how-to-interpolate-and-perform-vector-arithmetic-with-faces-using-a-generative-adversarial-network/

Deep Learning

- Specialized DL (Before BERT/Elmo in 2018)
  - Design specialized model architectures.
  - Leveraging task-specific features.
  - Train the specialized models with limited data.
- Transfer DL (Between 2018 - 2021)
  - Train a model with large amount of training data.
  - Use the features of the trained model to initialize part of the architecture
  - Design specialized modules on top of the trained features.
  - Train the partially specialized model with limited data.
- Foundation Model (After 2021)
  - Train a single huge model on astronomical amount of data
  - Prompt the single model for everything

Pros and Cons of Specialized DL

- Pros
  - The model considers the inductive bias for architecture design
  - The model can be effectively trained with limited amount of data
  - The model is normally small in size, easy to deploy for applications

- Cons
  - Each task requires lots of expertise for architecture design
  - Each task requires annotating specialized dataset
  - The model cannot benefit from other annotated data, it needs to start from scratch literally to gain its skill
  - Hosting many specialized models incur high costs

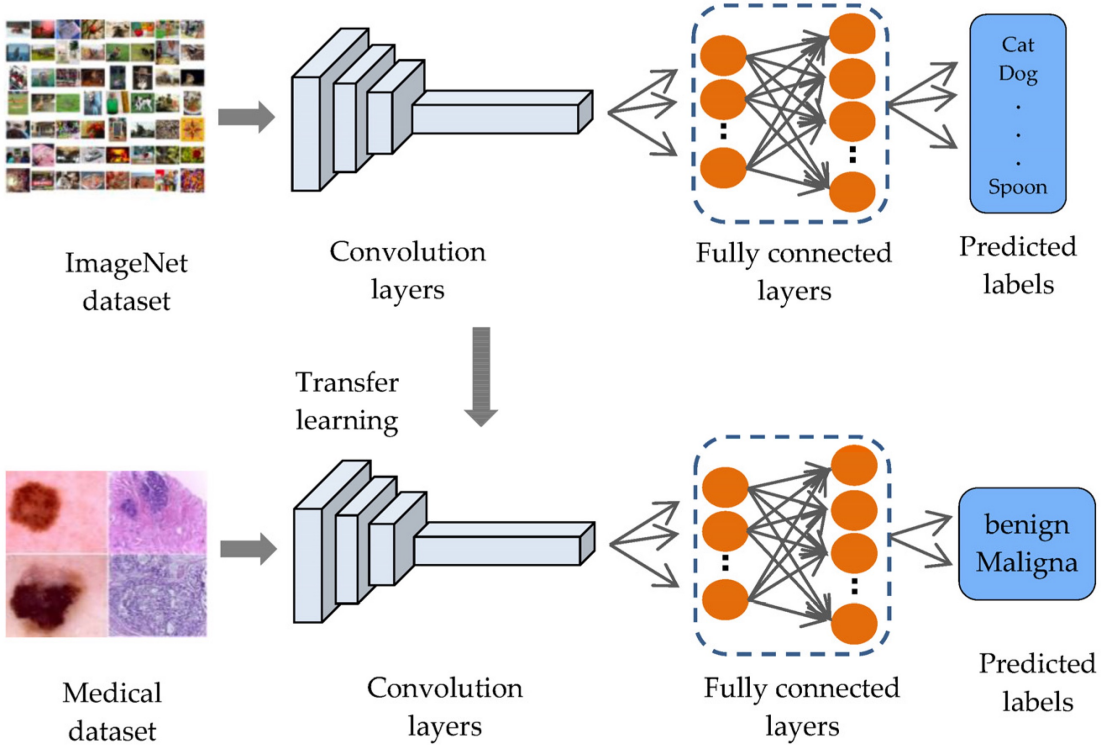
Transfer Learning

- We can train our model on massive amount of data to learn neural representation or initialize certain part of the weights.

- Then transfer to new tasks by adding layers on on top of the learned neural representation.

- This can leverage some cross-task similarity to enhance model performance across different tasks.

Transfer Learning in Vision

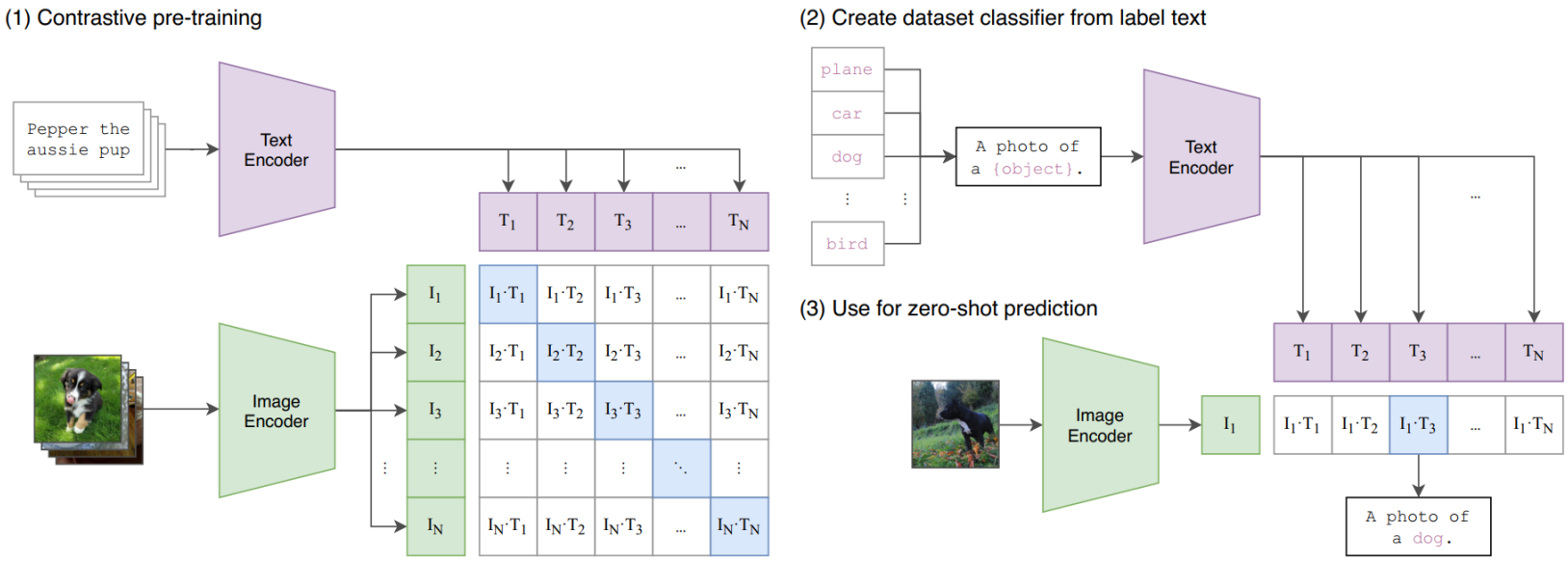


Transfer Learning in BERT

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

Transfer Learning in Vision

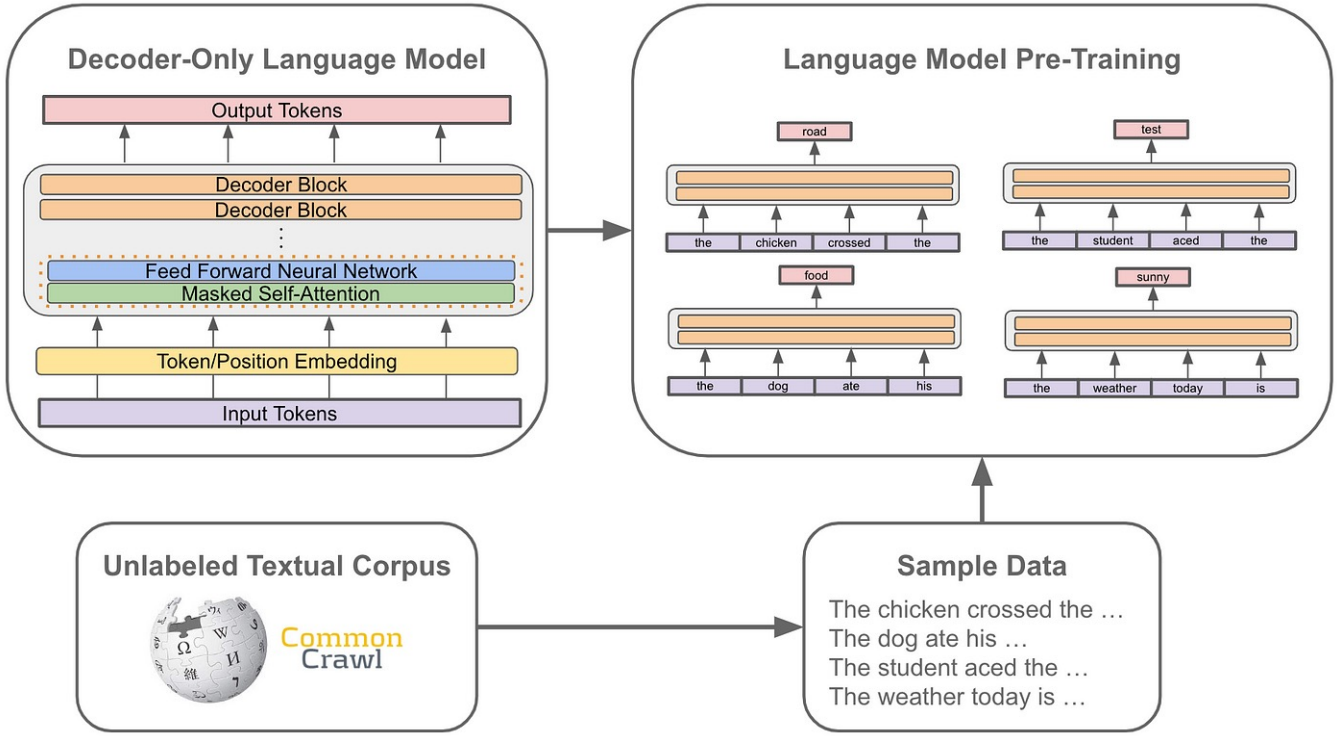


Pros and Cons of Transfer DL

- Pros:
  - The model shows much stronger capability than Specialized DL
  - The model can generalize to unseen cases
  - The model requires very few fine-tuning

- Cons:
  - The model's performance is still not perfect.
  - There is still fine-tuning needed for the downstream tasks

GPT-2 (Radford et al. 2019)



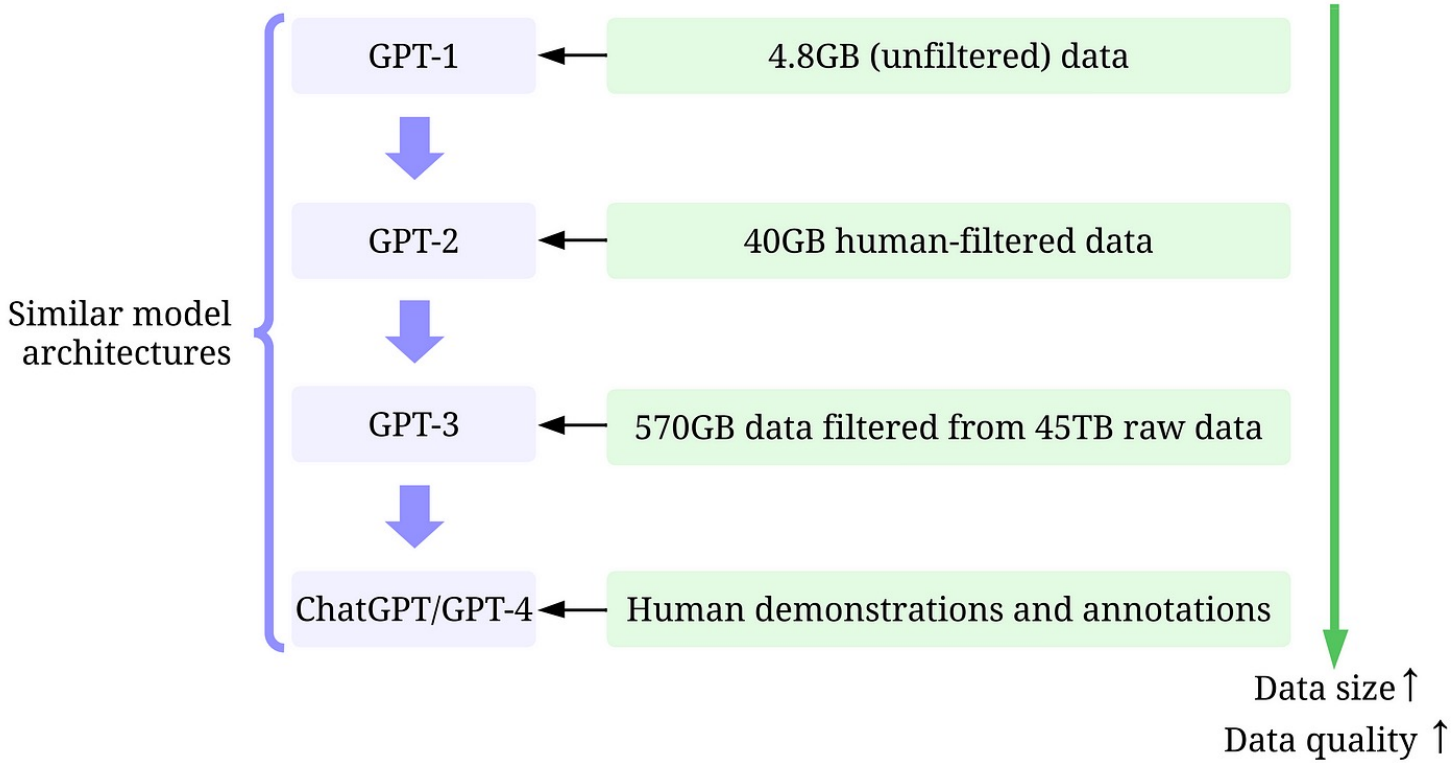
GPT-2 (Radford et al. 2019)

- Training Objective: next work prediction
- Nothing surprising here, these are the traditional LM loss:

$$P(w_{0:n}) = \prod_{i=0}^{n} P(w_i | w_{0:i-1})$$

- We want to estimate the probability distribution of sequence of words (or tokens in general).

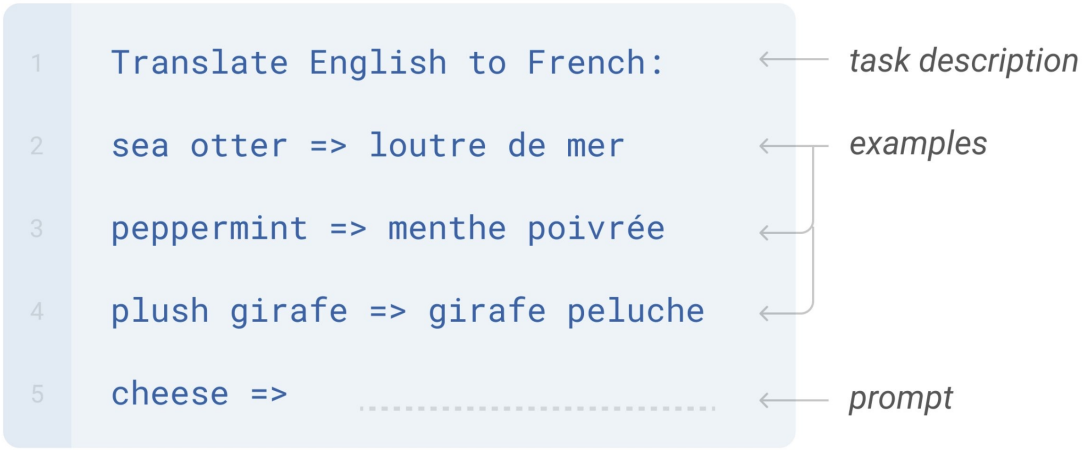
GPT-3 (Brown et al. 2020)



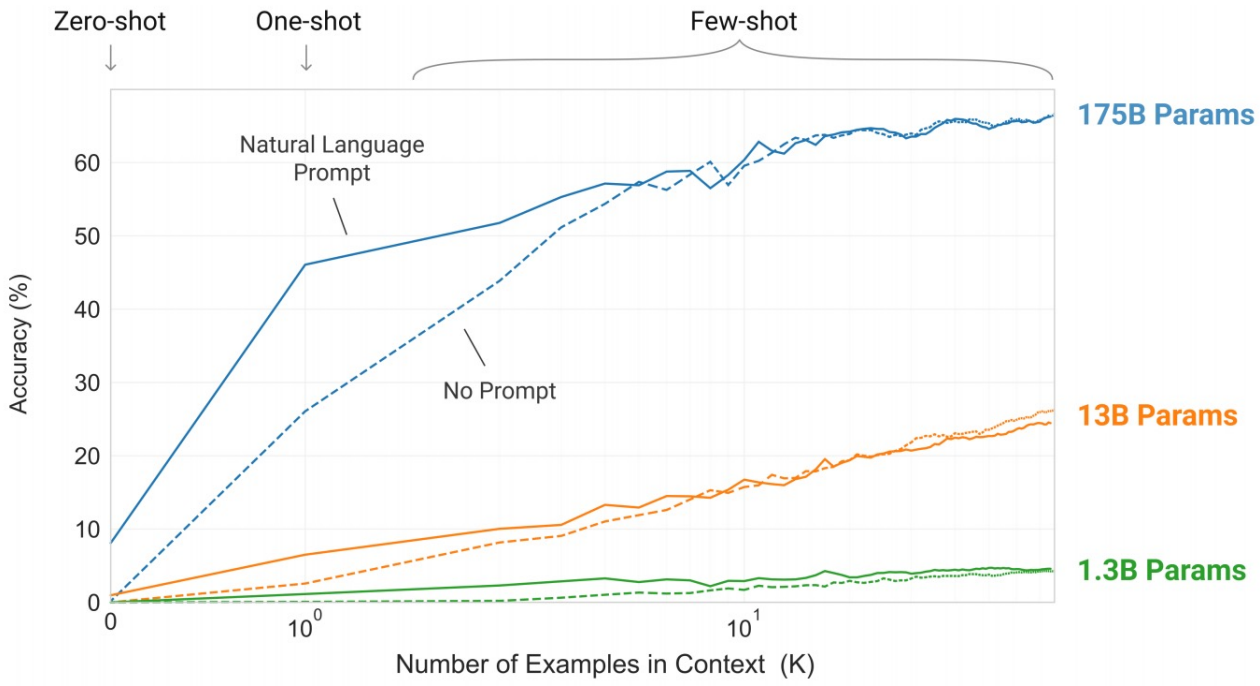
In-context Learning

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

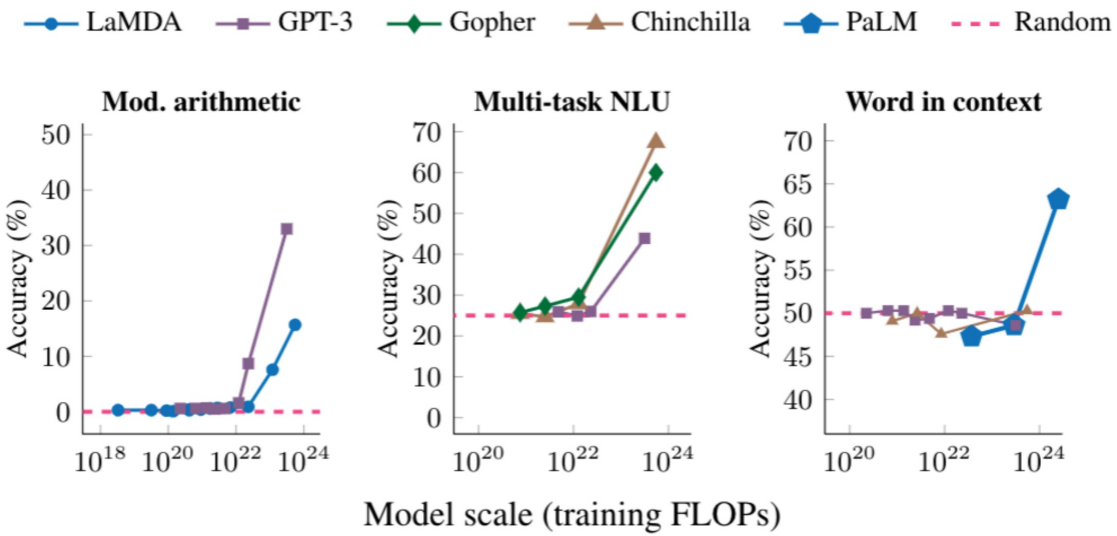


Few-shot Learning

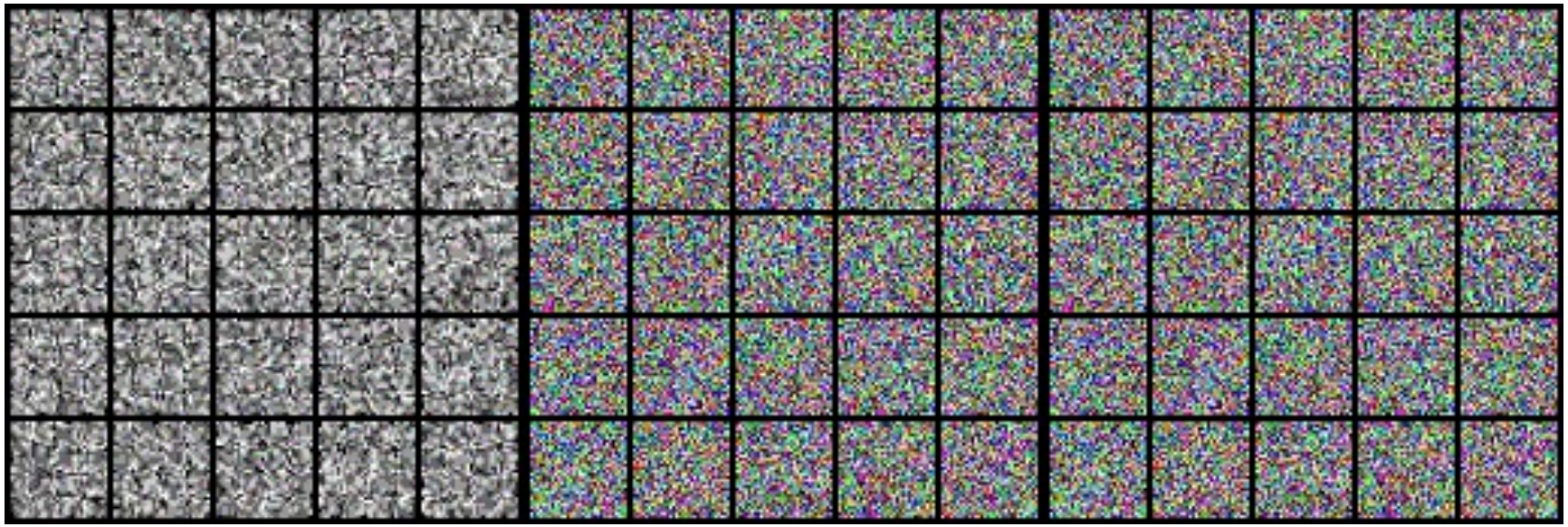


Emergent Ability (Wei et al. 2022)

- When the model size grows from 0.1B -> 1.5B -> 175B, the model starts be really good in zero-shot and few-shot tasks
- This is called “emergent abilities”.

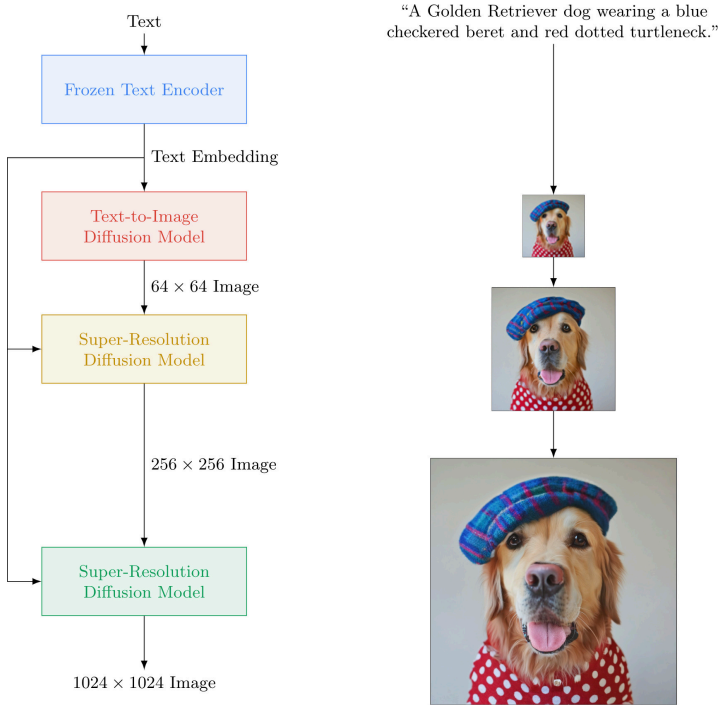


Sampling from above



Song, Yang, and Stefano Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution." NeurIPS 2019.

Diffusion Models



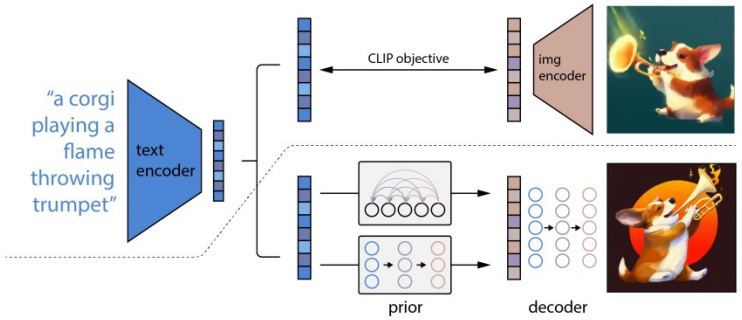
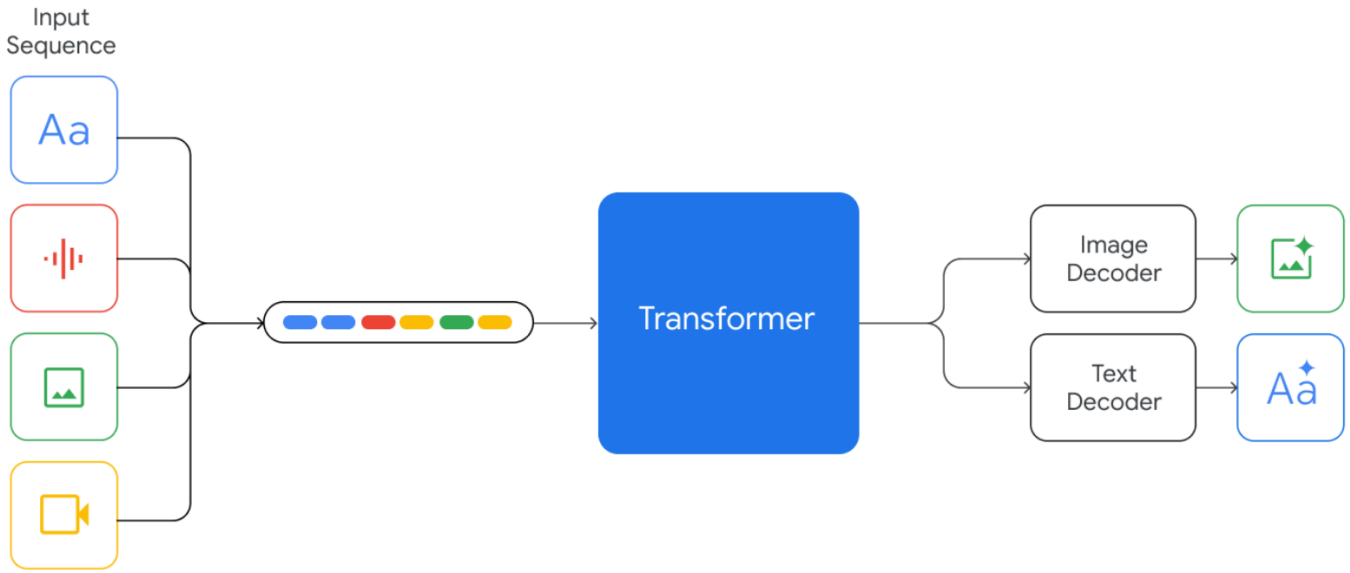


Figure 2: A high-level overview of unCLIP. Above the dotted line, we depict the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, we depict our text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.

Diffusion Models (Videos)

Gemini (Google et al. 2023)

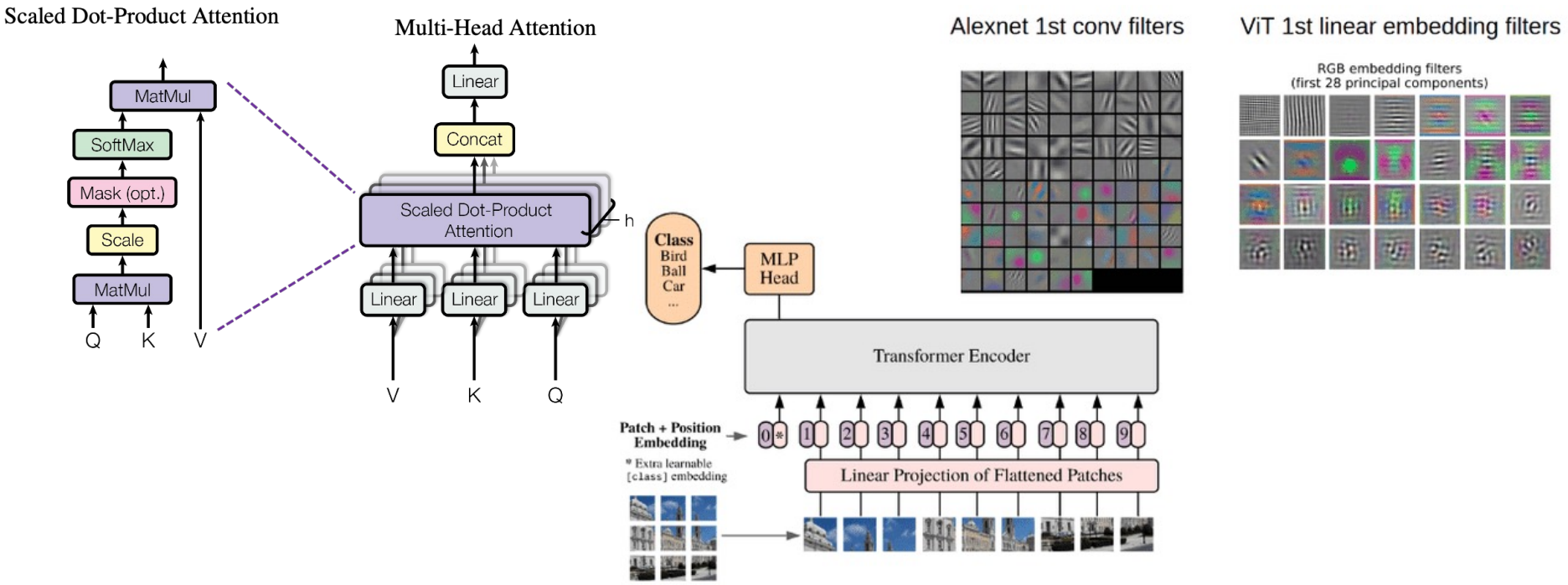


What can Gemini do?

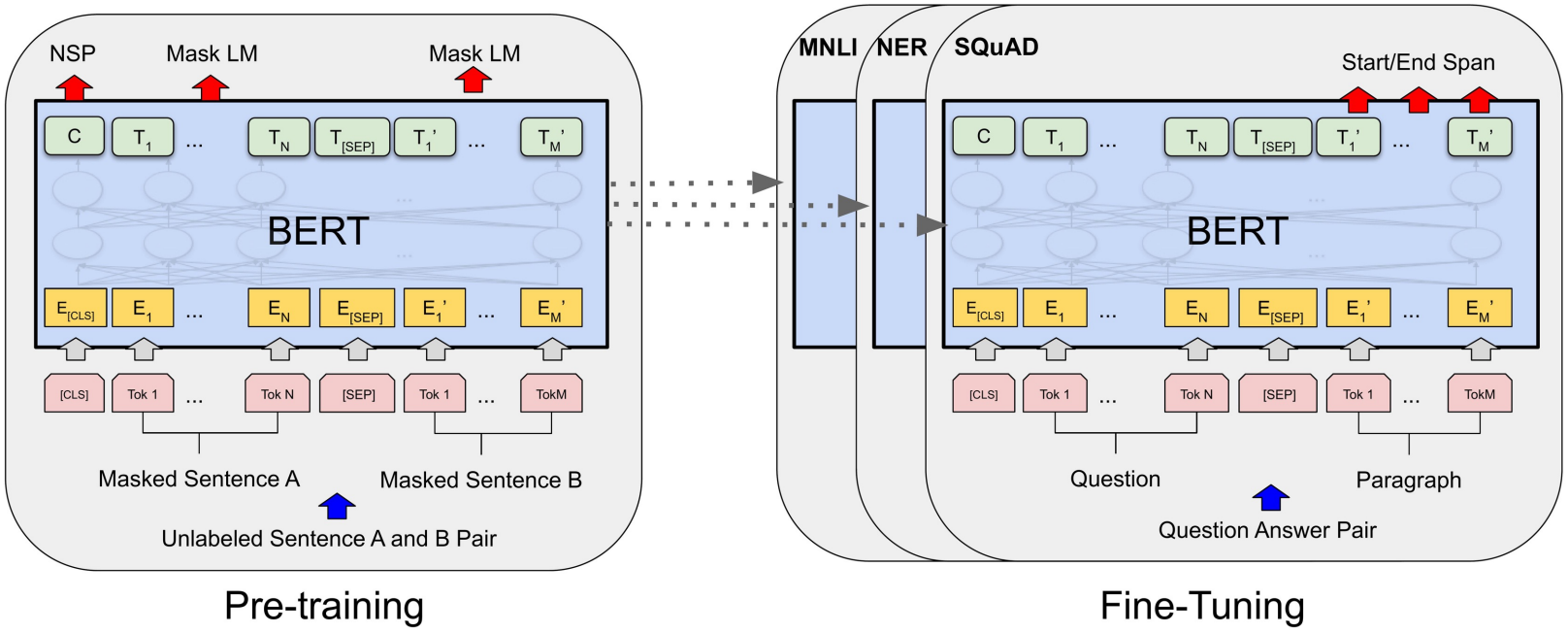
Parameters of Foundation DL

Parameters	Specialized DL	Transfer DL	Foundation DL
Model Size	< 100M parameters	100M -> 1B parameters	7B -> 1T parameters
Data Size	10K -> 1M tokens	100M -> 10B tokens	100B -> 30T tokens
Architecture	Specialized	General	General
Generalization	None	Reasonable	Strong

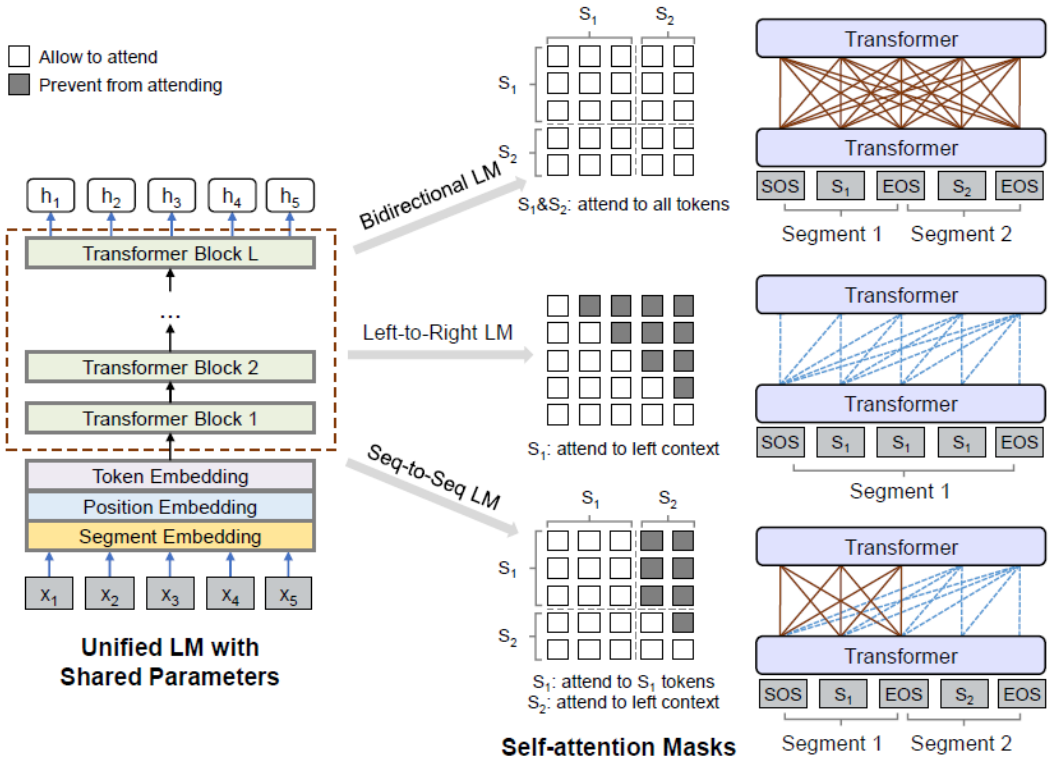
Self-Attention and Transformer



Language Model Pre-training



Language Model Pre-training

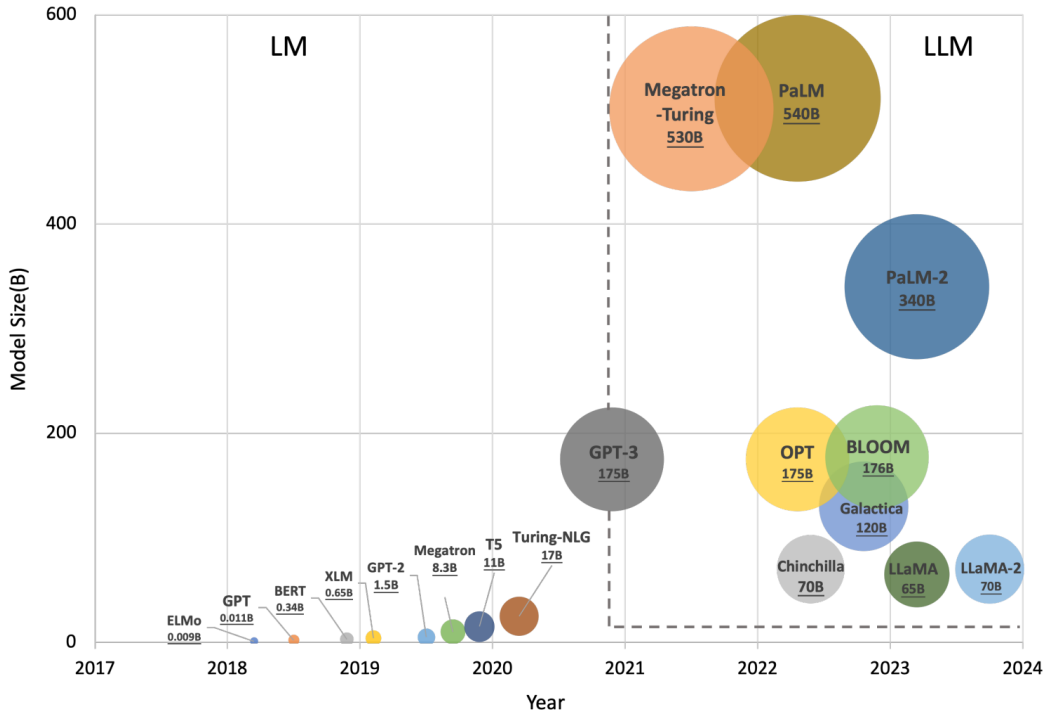


What is Large Language Model (LLM)?[1]

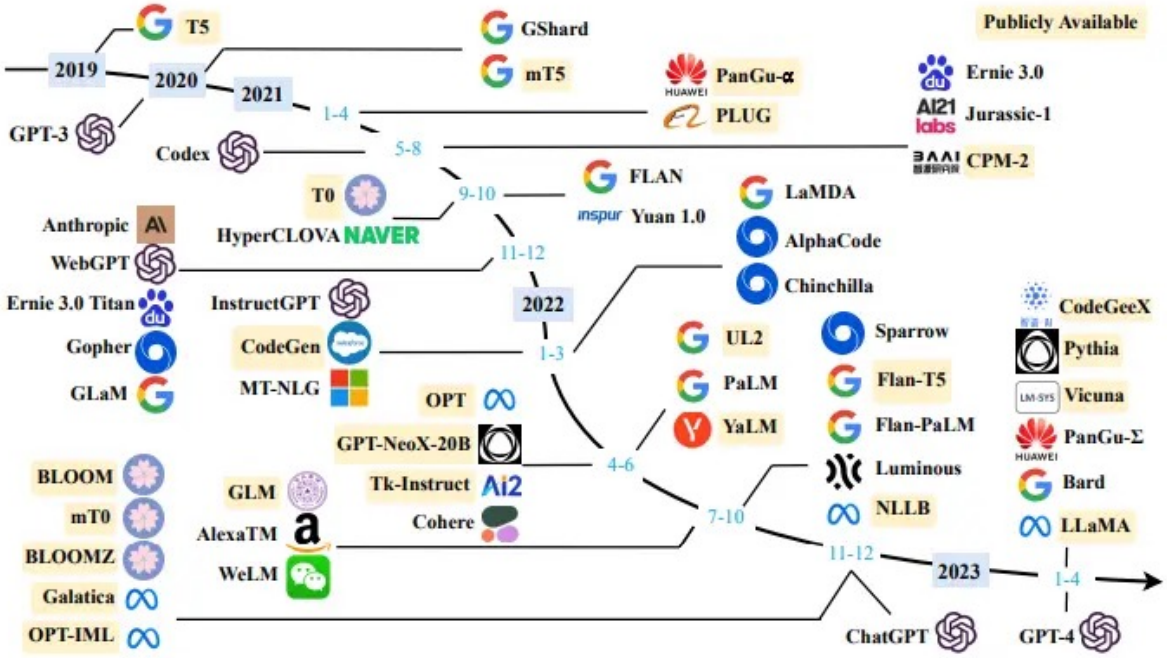
- Language models are computational models that have the capability to understand and generate human language.
- Deep learning algorithm that can perform various NLP tasks
- Are trained on massive datasets that allow them to recognize, translate, predict, or generate text or other content
- Unsupervised multi-task learners

1. Chang, Y., Wang, X., Wang, J., Wu, Y., Zhu, K., Chen, H., Yang, L., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P.S., Yang, Q., & Xie, X. (2023). **A Survey on Evaluation of Large Language Models.** *ArXiv, abs/2307.03109*.

Large Language Models



Large Language Models

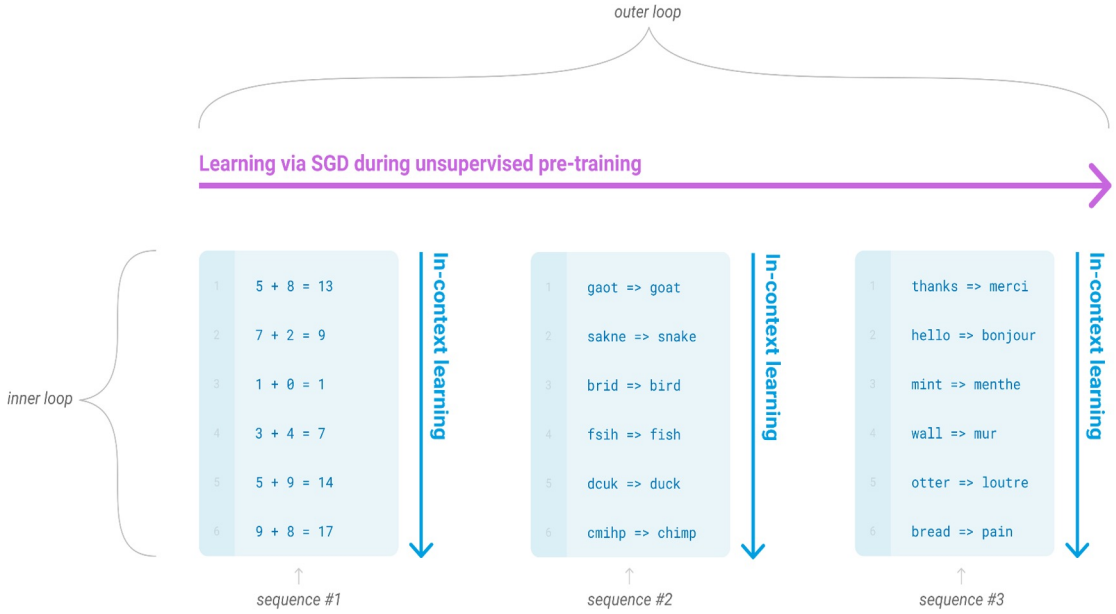


In-Context Learning

- Need for a large dataset for every task limits applicability of language models - its not practical
- Potential to exploit spurious correlations in training data grows with the expressiveness of the model and narrowness of the training distribution Language models are few-shot learners
- Humans do not require large supervised datasets to learn most language tasks - a brief directive is sufficient
- Meta-learning or zero-shot transfer allows the model to develop a broad set of skills and pattern recognition abilities at training time
- Not as performant as reinforcement learning from human feedback (RLHF)

Language model meta-learning[4]

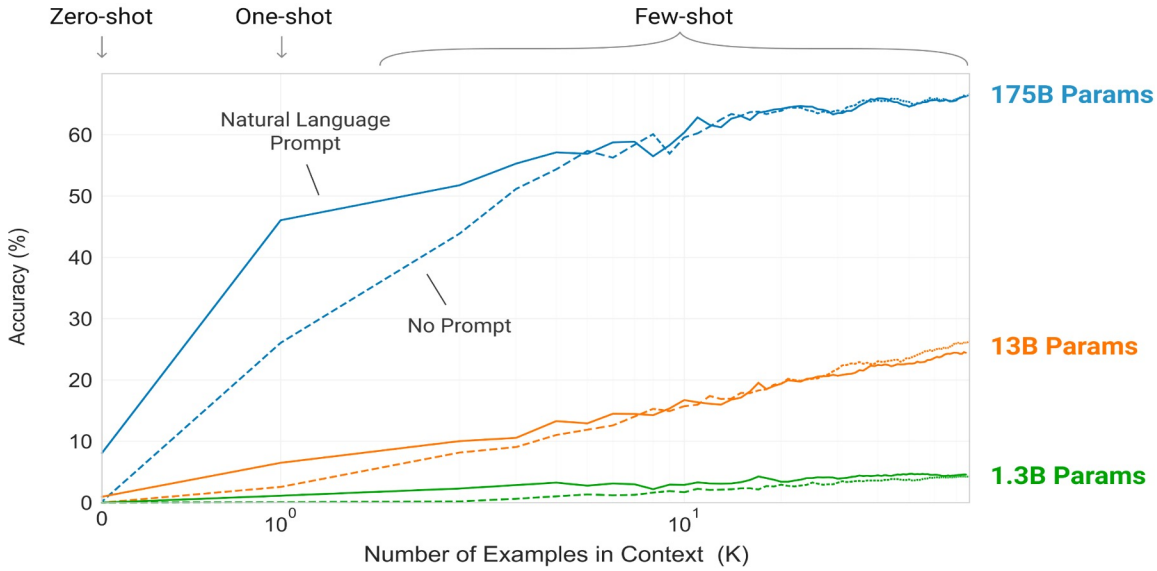
- Language model develops a broad set of skills and pattern recognition abilities during training



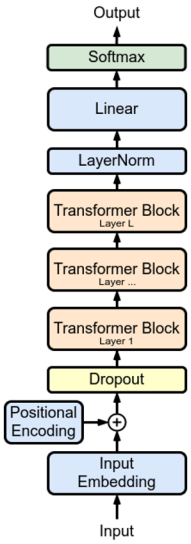
4. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877-1901.

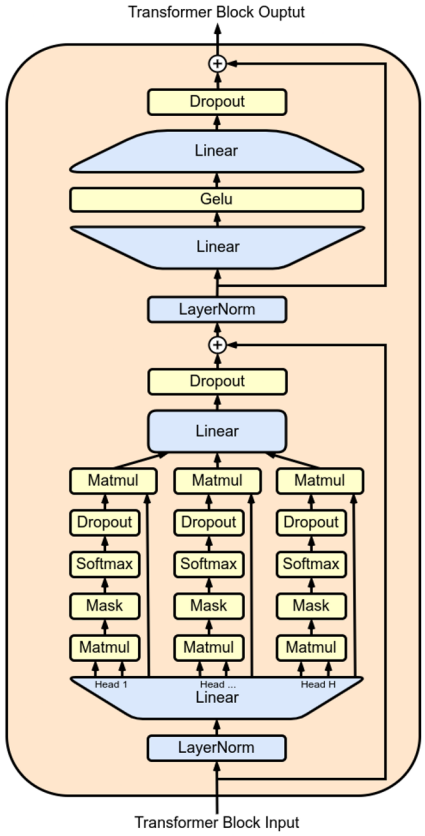
Language model meta-learning

- Task: Remove random symbols from a word
- Larger models make increasingly efficient use of in-context info
- Params: weights + biases
- Eval: GPT-3



GPT-3 Architecture & Training Approaches





The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
cheese =>                           ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese =>                           ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                           ← prompt
```

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
sea otter => loutre de mer          ← example #1
```

gradient update

```
peppermint => menthe poivrée        ← example #2
```

gradient update

...

```
plush giraffe => girafe peluche     ← example #N
```

gradient update

```
cheese =>                           ← prompt
```

GPT-3 - Training dataset

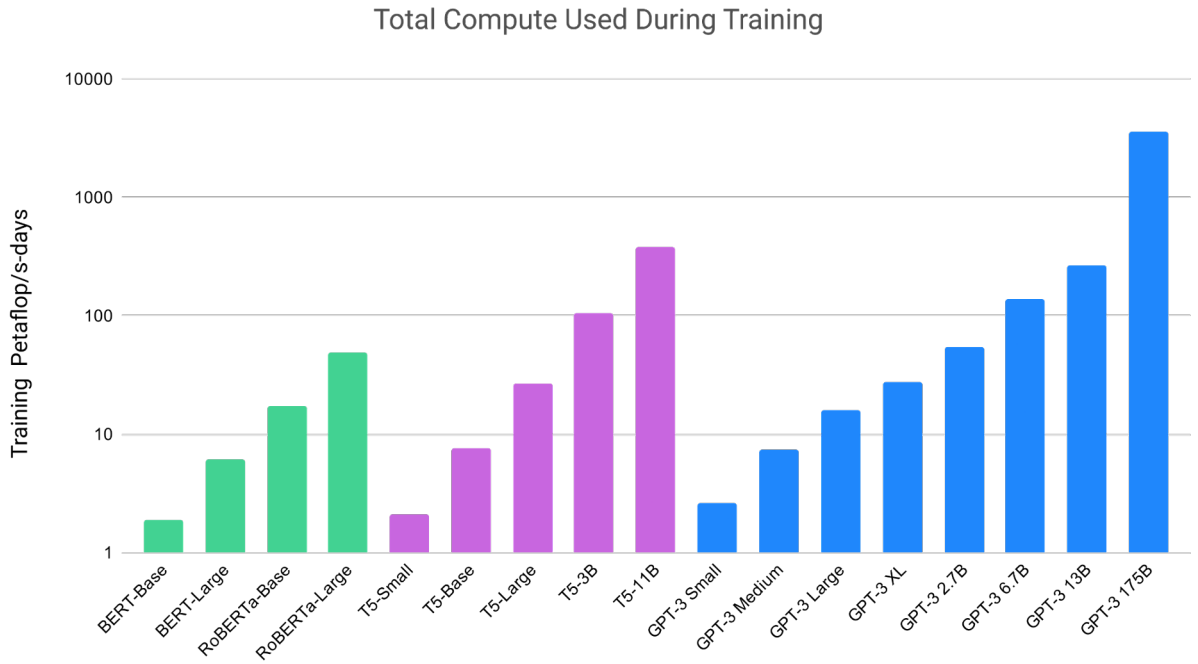
- Based on raw Common Crawl dataset of up to 1T words
- Cleaned up original datasets by:
  - Filtered Common Crawl based on similarity to a range of high-quality reference corpora
  - Performed fuzzy deduplication at the document level
  - Added known high-quality reference corpora to the training mix
- Used final cleaned up data in training

GPT-3 - Training dataset

- Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models trained
- All models were trained for a total of 300 billion tokens

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	BatchSize	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	196	0.5M	2.5×10^{-4}
GPT-3 XL	1.3GB	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7GB	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7GB	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5144	40	128	2M	1.0×10^{-4}
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

GPT-3 Compute Consumption



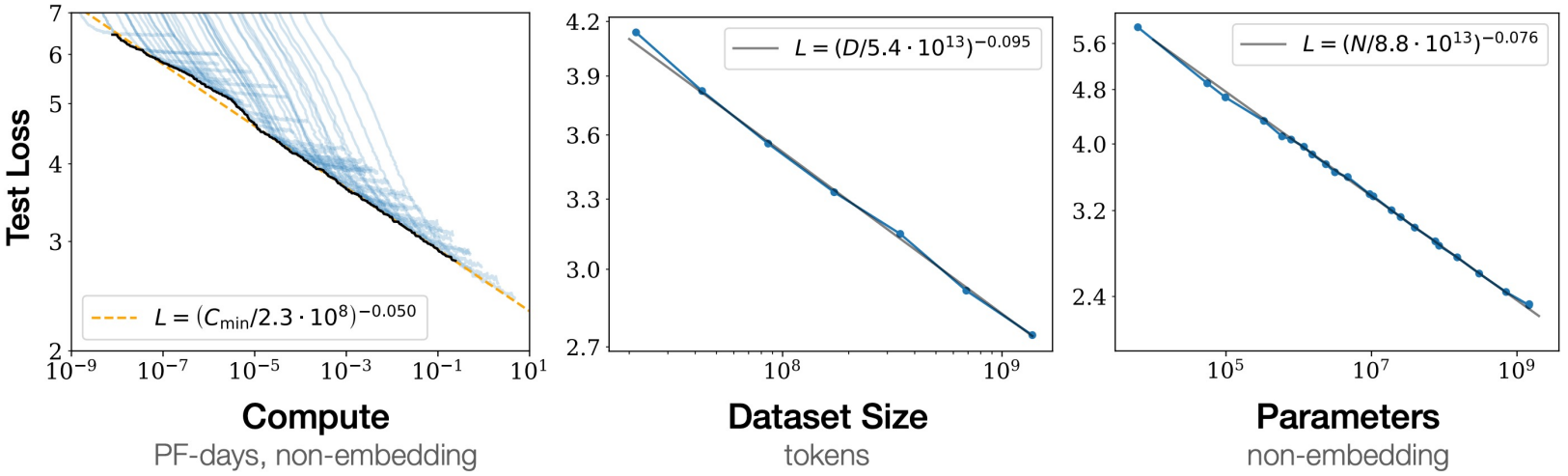
GPT-3 Limitations

- Limitations in text synthesis
- Structural and algorithmic limitations
- Poor sample efficiency during pre-training
- Lack of interpretability
- Can perpetuate and amplify existing biases and unfairness in society
- Multilingualism - majority language model researches are done in English

LLMs Comparison

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Learning Rate	Context length	Architecture
T5	125M	12	768	12	64	1.0×10^{-3}		Transformer – Encoder-Decoder
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	0.6×10^{-4}		Transformer – Decoder
CodeX 12B	12B	N/A				0.6×10^{-4}	4k	Transformer - Decoder
Llama-2 70B	70B	N/A		1		1.5×10^{-4}	4k	Transformer with Pre-Normalization, SwiGLU activation, RoPE, GQA
Mixtral 8x7B	13B active, 47B sparse	32	14436	32	128		32k	Transformer with Pre-Normalization, SwiGLU activation, RoPE, SWA, and Mixture of Experts instead of feed-forward layer
PaLM	540B	118	18432	48		5×10^{-5}		Transformer – Encoder-Decoder

Scaling Law

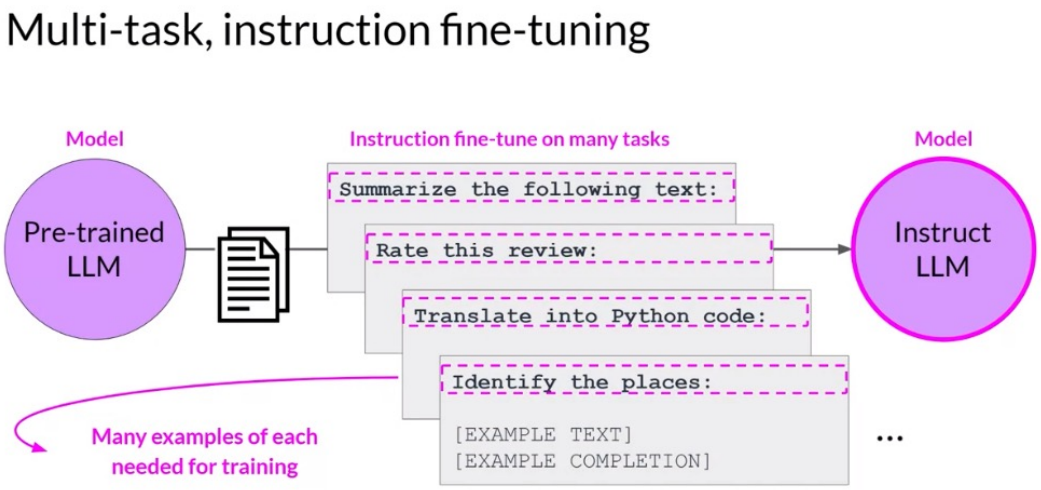


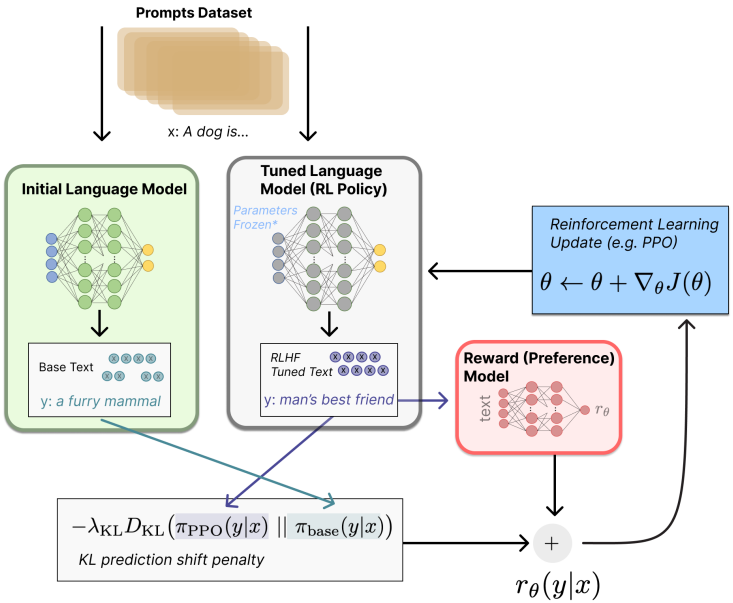
For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two

Scaling Laws for Neural Language Models, Kaplan et al. 2020

Instruction Tuning & RLHF

Unify all the NLP tasks as instruction following format to enhance its generalization





Proximal Policy Optimization (PPO)

- Introduced by OpenAI in 2017 and is considered as a state-of-the-art policy-based RL algorithm
- Enhances training stability in policy updates by limiting the magnitude to prevent large, disruptive changes
- Balances between exploration and exploitation by maintaining proximity to proven policies

Source: RL - Proximal Policy Optimization (PPO) Explained by Jonathan Hui

PPO

- **Idea**: constrain our policy update with a new objective function called the *clipped surrogate objective function* that will constrain the policy change in a small range using a clip

- Designed to avoid destructive large weight updates

Clipped Surrogate Objective Function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

$$\boldsymbol{r_t(\theta)} = \frac{\boldsymbol{\pi_\theta(a_t|s_t)}}{\boldsymbol{\pi_{\theta_{old}}(a_t|s_t)}}$$

InstructGPT

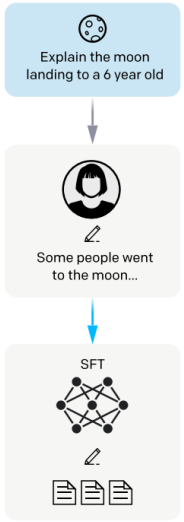
Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



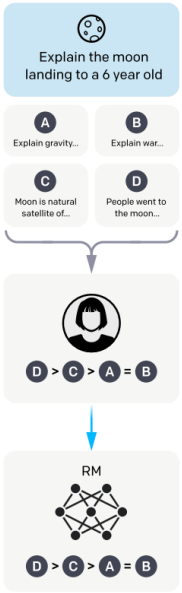
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

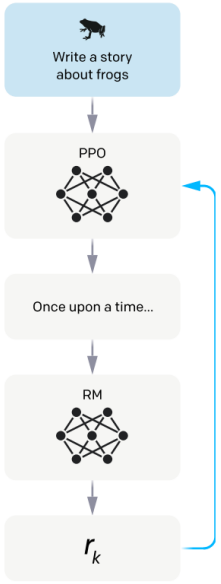
Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

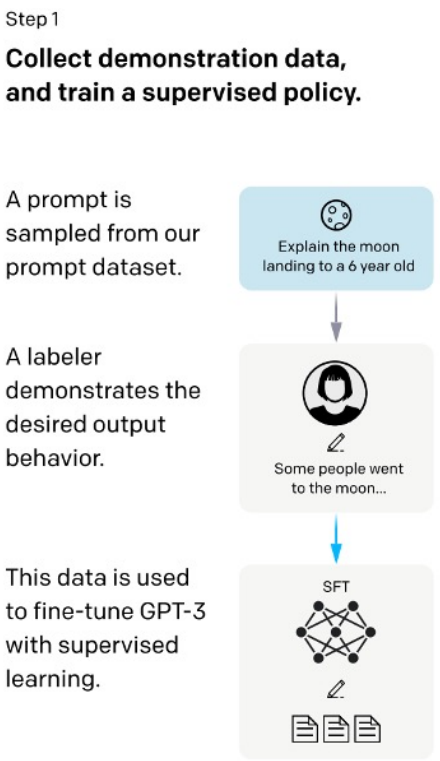
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



Supervised Fine-Tuning Model (SFT)

- Initiated by recruiting 40 contractors through a selective screening process
- Compiled a dataset of *human-written demonstrations of the desired output behaviour* on prompts (mostly English) submitted to the OpenAI API and some labeler-written prompts
- Used this dataset as source of supervised learning training

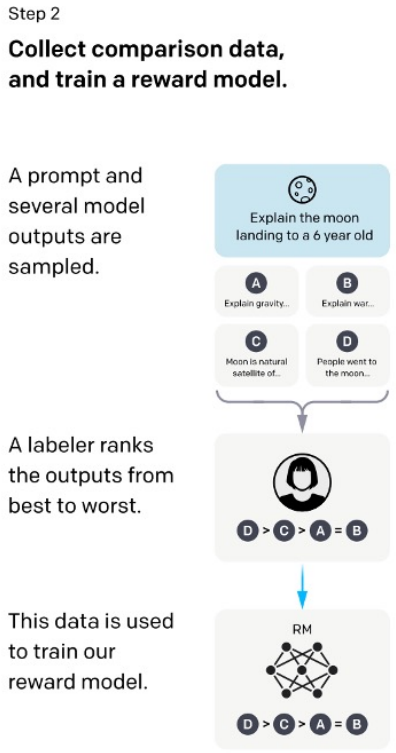


Reward Model (RM)

- Collected a dataset from human-labeled comparisons between GPT3 model outputs on a larger collection of OpenAI API prompts
- Using this dataset a reward model was trained to predict which model output a human would prefer
- Wish to minimize this loss function:

$$loss(\theta) = -\frac{1}{\binom{K}{2}} E_{(x,y_w,y_l)\sim D}[\log(\sigma(r_\theta(x,y_w) - r_\theta(x,y_l)))]$$

- x is the prompt, y_w is preferred completion and y_l is the unpreferred completion, all from the dataset D
- K is the number of outputs to rank, making $\binom{K}{2}$ possible comparisons
- $r_\theta(x,y)$ produces the scalar output of the reward model

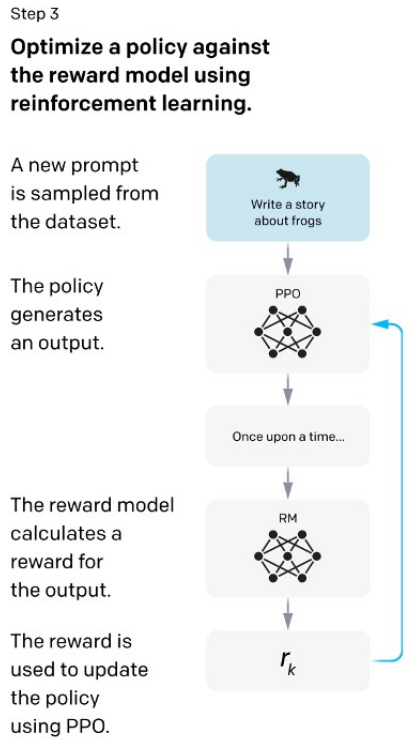


RLHF

- Using the RM as a reward function the SFT model is fine-tuned to maximize this reward using the PPO algorithm
- $r_\theta(x, y)$ is our RM's scalar reward for the generated output given the prompt

$$MAX\ objective(\phi)$$
$$= E_{(x,y)\sim D_{\pi_\phi^{RL}}}[r_\theta(x, y) - \beta \log(\pi_\phi^{RL}(y\,|x)/\pi^{SFT}(y\,|x))]$$
$$+ \gamma\, E_{(x)\sim D_{pretrain}}[\log(\pi_\phi^{RL}(x))]$$

- KL Penalty to keep policy gradients from deviating far
  - π_ϕ^{RL} is the learned RL policy, π^{SFT} is the supervised trained model's policy
  - β is a hyperparameter for the strength of this term

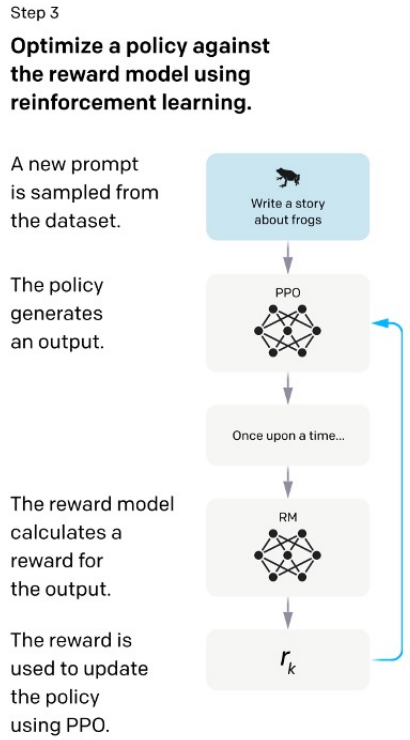


RLHF

- Using the RM as a reward function the SFT model is fine-tuned to maximize this reward using the PPO algorithm
- Wish to maximize this objective function:

$$objective(\phi) = E_{(x,y)\sim D_{\pi_\phi^{RL}}}[r_\theta(x,y) - \beta \log(\pi_\phi^{RL}(y\,|x)/\pi^{SFT}(y\,|x))] + \gamma\, E_{(x)\sim D_{pretrain}}[\log(\pi_\phi^{RL}(x))]$$

- Mixing pretraining gradients into PPO gradients to fix the performance regressions on public NLP datasets controlled by γ



Multimodal Pre-training

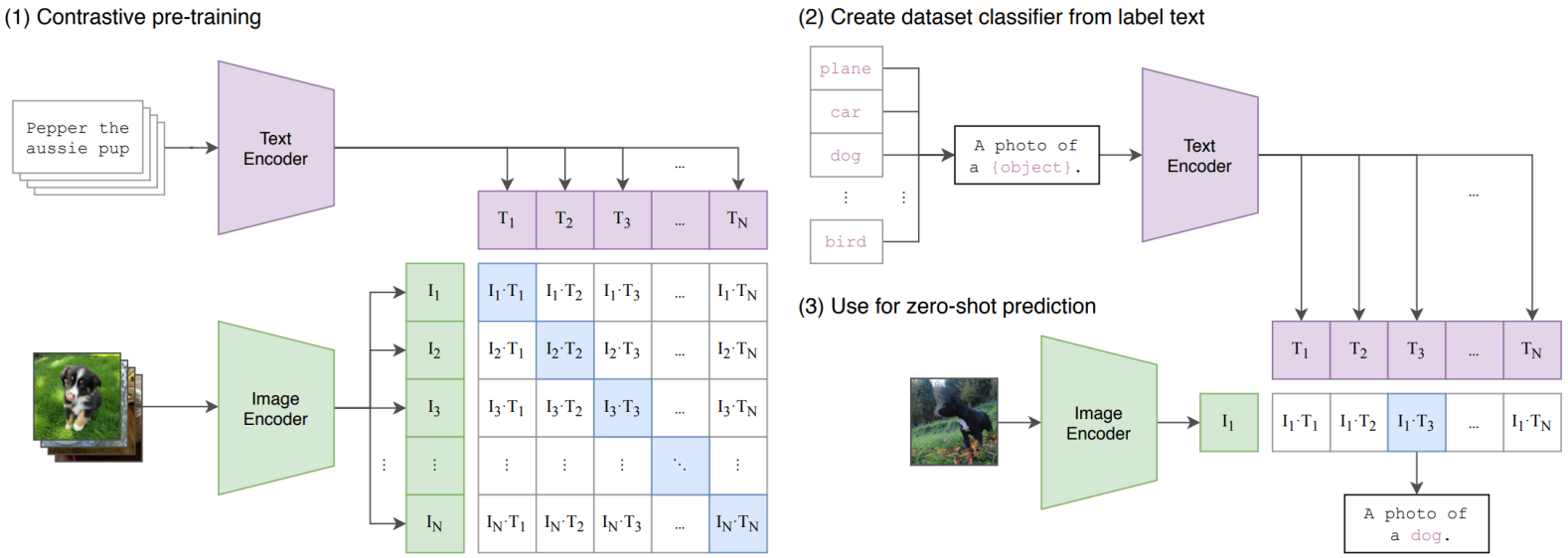
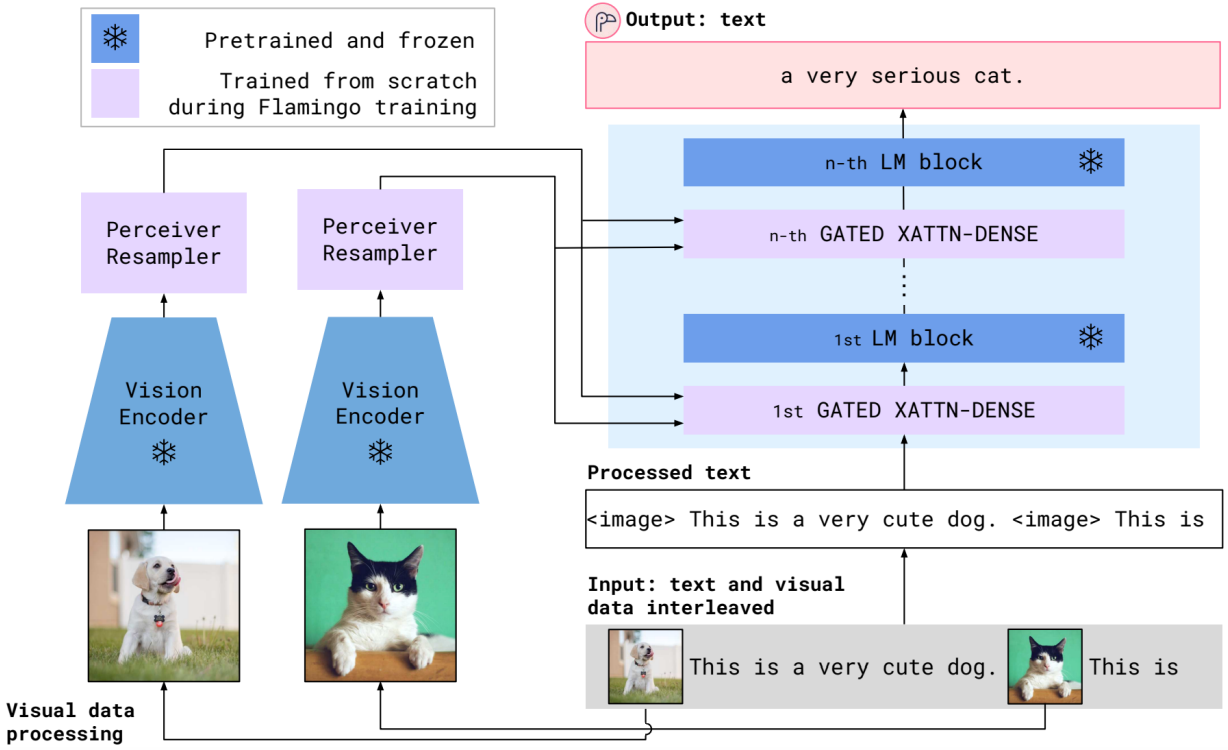


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

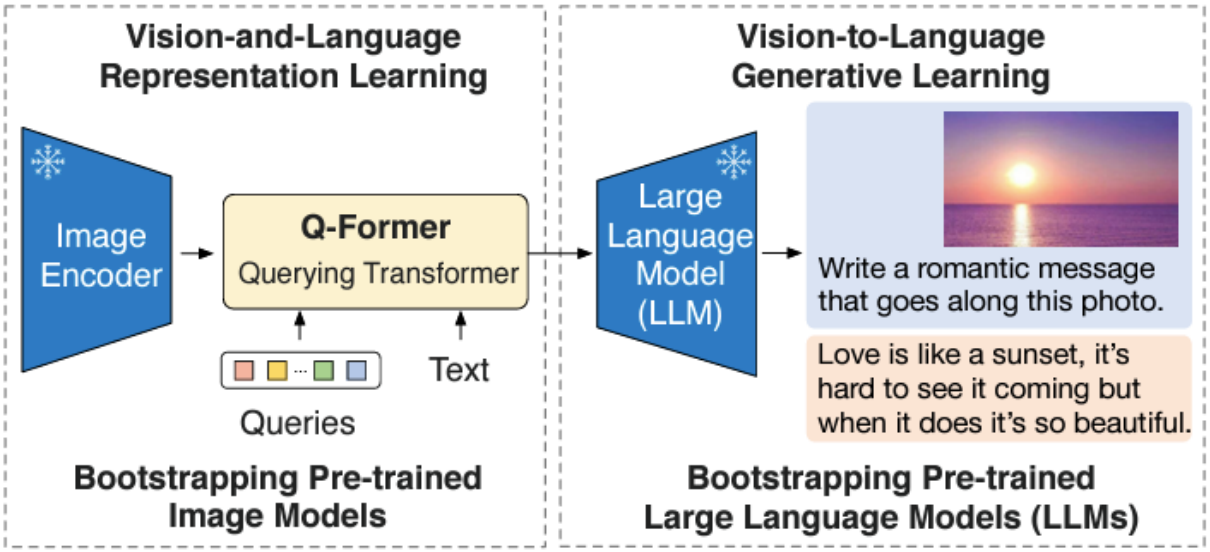
Large Multimodal Models



Q-Former: Lightweight Querying Transformer to bridge the modality gap

• **Goal:**

The queries can learn to extract visual representation that is most informative of the text



Q-Former functions as an **information bottleneck** that feeds the most useful information to the LLM while removing irrelevant visual information

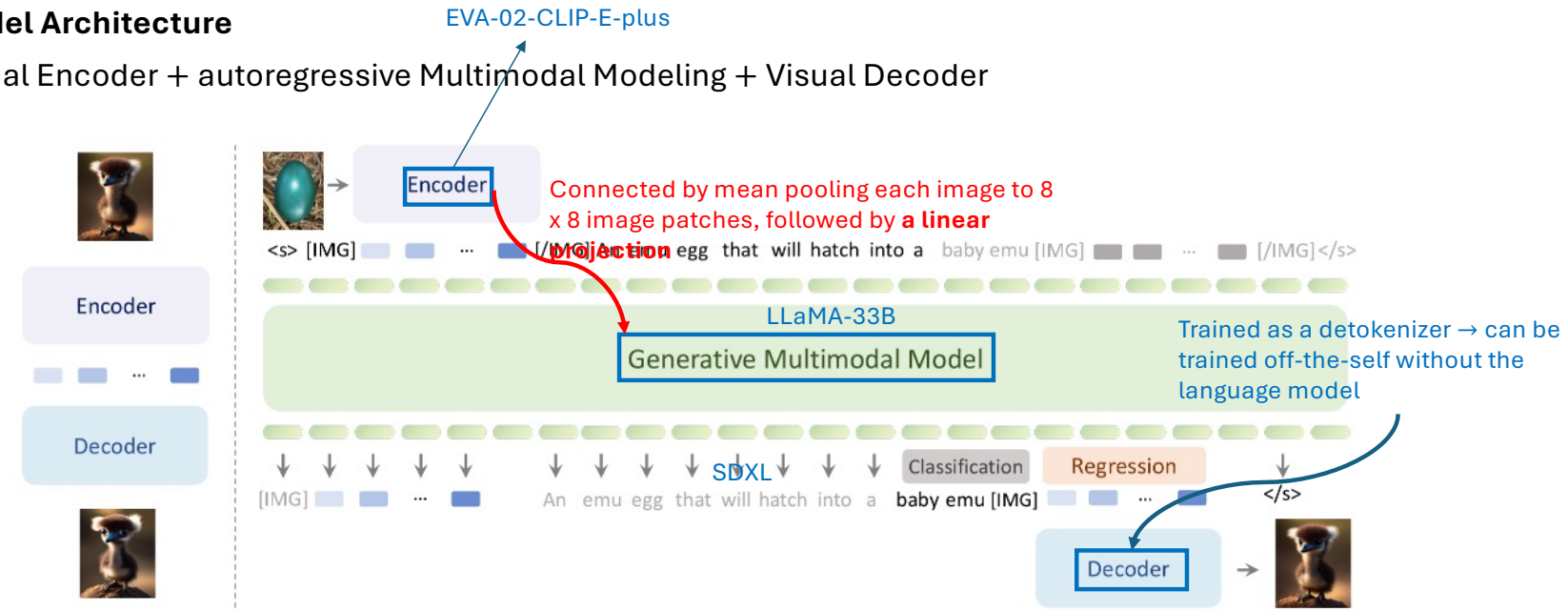
Emu2: Objective & Architecture

- **Unified autoregressive objective:**

  Predict-the-next-multimodal-element (either visual embeddings or textual tokens)

- **Model Architecture**

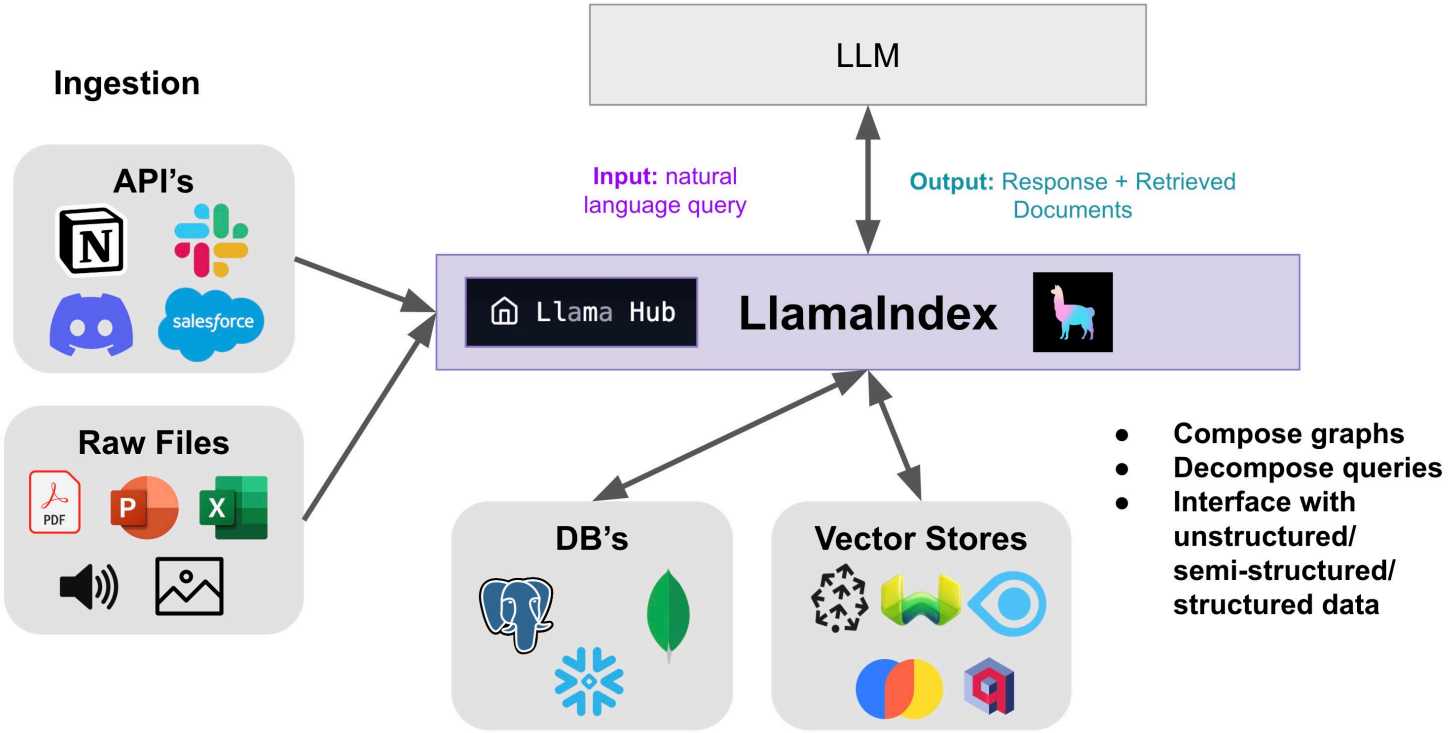
  Visual Encoder + autoregressive Multimodal Modeling + Visual Decoder



Overall Comparison

Model	Contrib.	Zero/few shot or fine-tuned	size	VQAv2	NoCap	GQA	TextVQA
Flamingo	The first LMM	Few shot	80B	56.3	-	-	36.0 few shot 57.1 fine-tuned
LLaVA	Visual instruction tuning	Few shot	1.5-13B	80.0	-	63.3* few-shot	61.3* few-shot
InstructBLIP	Visual instruction tuning & instruction aware visual feature extraction		13B	-	121.9 few-shot	49.5 f-show	50.7 few-shot
PALI	Scaling up the Vision, multilingual	Fine-tuned	55B	86.1	126.3 zero-shot	54.2 fine-tuned	73.06 zero-shot
PALI-3	Contrastively trained visual encoder	Fine-tuned	5B	85.2	-	-	79.51 fine
EMU2	Strong in-context learning ability	Few shot	37B	84.9	-	65.1* few-shot	66.6* few-shot
InternVL	Bridge the gap between vision encoder and LLM	Zero shot	24B	81.2	126.2 few-shot	66.6* few-shot	-
Gemini	Natively multimodal	Zero shot	-	77.8	-	-	-

LLM & Tool Augmentation



Two Types of RAG Models

- $p_{RAG-Sequence}(y|x) \approx \sum_{z \in top-k(p(\cdot|x))} p_\eta(z|x) \prod_i^N p_\theta(y_i|x, z, y_{1:i-1})$
  - Marginalized over each top-k document…
  - Generate probability for entire output sequence

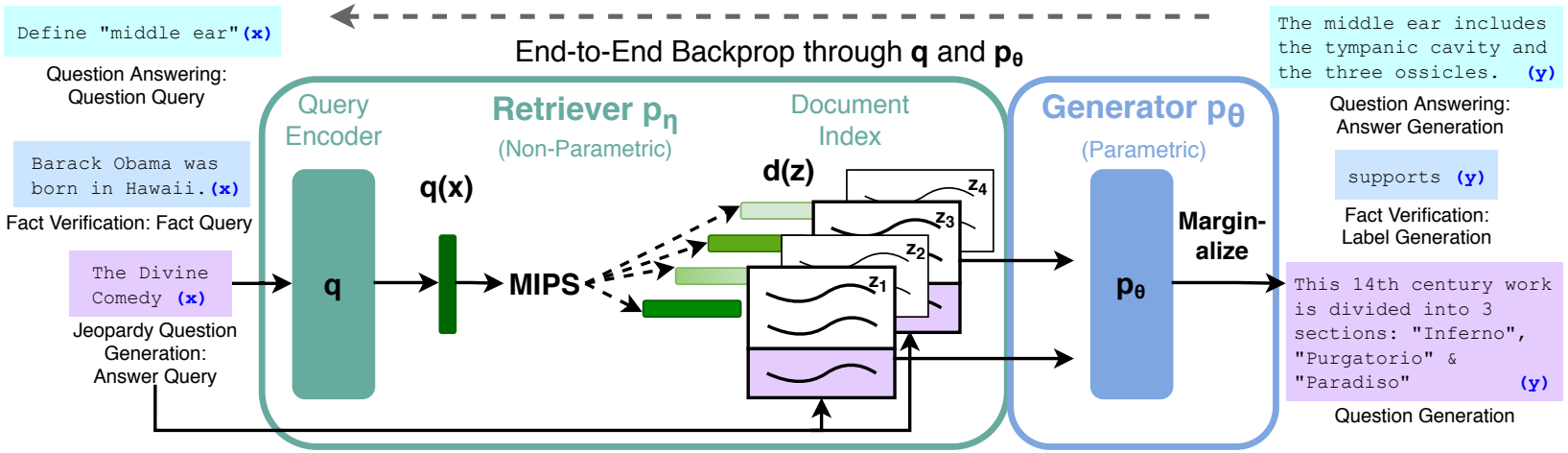
- $p_{RAG-Token}(y|x) \approx \prod_i^N \sum_{z \in top-k(p(\cdot|x))} p_\eta(z|x) p_\theta(y_i|x, z, y_{1:i-1})$
  - For each token…
  - Marginalized over each each top-k document…
  - Generate distribution for next output token

Implementation: Fine-Tuning

- **Joint Fine-Tuning Update:**
  - **Update:** Retriever query encoder and generator
  - **Don't Update:** Retriever document encoder
    1. Expensive since document index must be periodically updated
    2. Authors find it isn't necessary for strong performance
    3. Aside: REALM does this during pre-training

- **Fine-Tuning Data:** A collection of pairs (x_j, y_j)
- **Objective:** Minimize $\sum_j -\log p(y_j|x_j)$ using SGD (via Adam)

Retrieval Augmented Generation



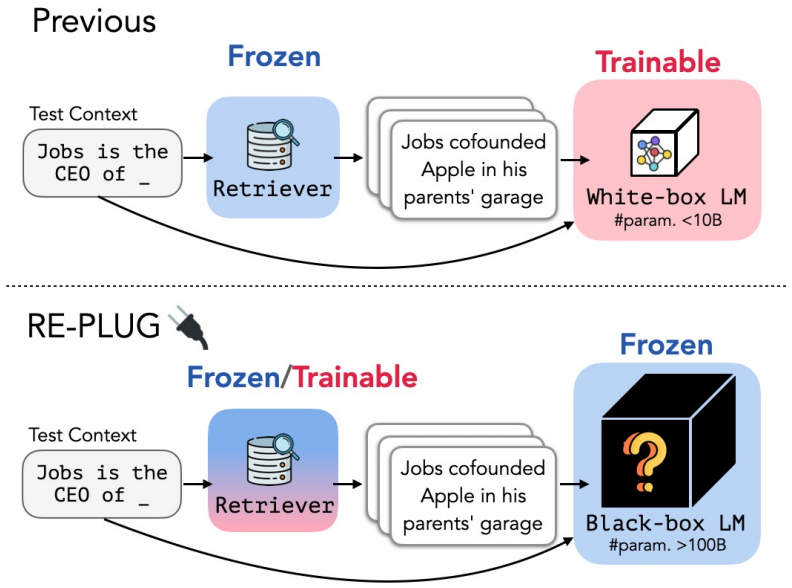
Motivation: RAG for Black-Box LLMs

- **Issue:** Fine-tuning LLMs with retrieval requires white-box access (i.e., access to the LLM parameters)
  - To train the model
  - To index the datastore
- **In Practice:** Many LLMs are only accessible via API
  - Can't access model parameters!
  - Can't fine-tune!




REPLUG

- **Focus:** Retrieval-augmentation in the black-box setting (no access to model parameters)
  - LM is a black box
  - Retriever is optionally tunable
- **How does it work?**
  1. Get documents from retriever
  2. Prepend documents to LM input
  3. Feed input to LM



REPLUG LSR (LLM-Supervised Retrieval)

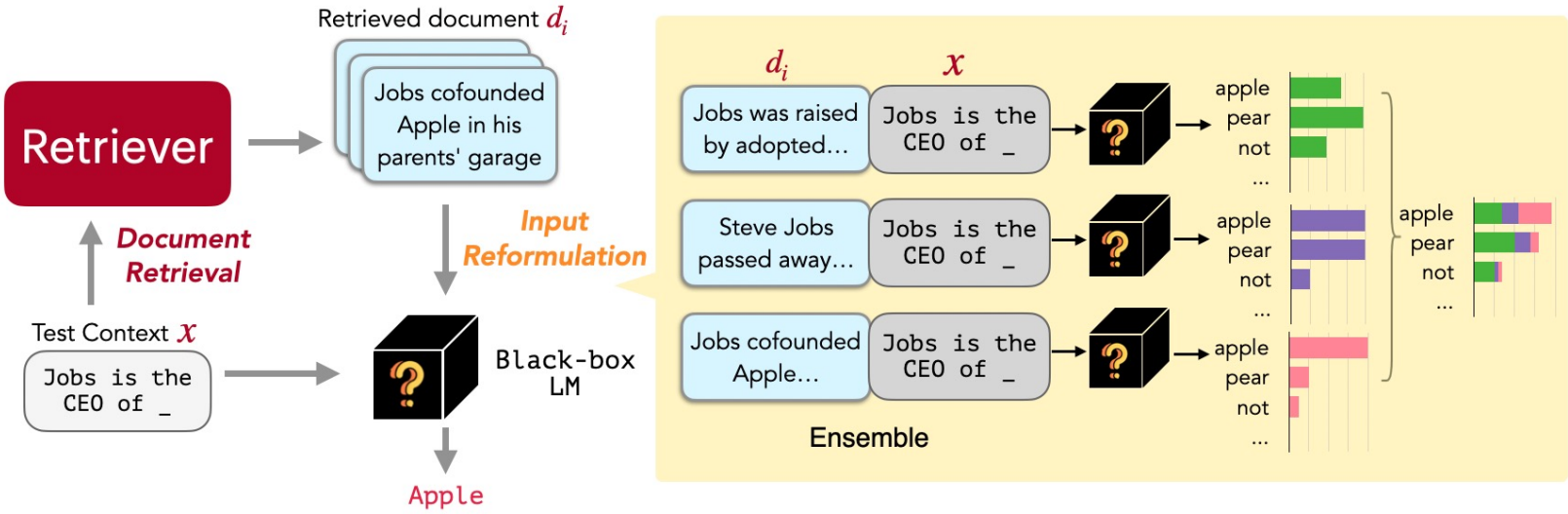
Idea: Train retriever to find documents that minimize LM perplexity

1. Retrieve k relevant documents $D' \subset D$
2. Compute retrieval likelihood of each document
   - $P_R(d|x) = \frac{e^{s(d,x)/\gamma}}{\sum_{d' \in D'} e^{s(d',x)/\gamma}}$
3. Compute LM likelihood of each document (y = ground truth)
   - $Q_{LM}(d|x,y) = \frac{e^{P_{LM}(y|d,x)/\beta}}{\sum_{d' \in D'} e^{P_{LM}(y|d',x)/\beta}}$
4. Minimize KL divergence between retriever and LM ($\mathcal{B}$ = contexts)
   - $\mathcal{L} = \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} KL\big(P_R(d|x) \parallel Q_{LM}(d|x,y)\big)$

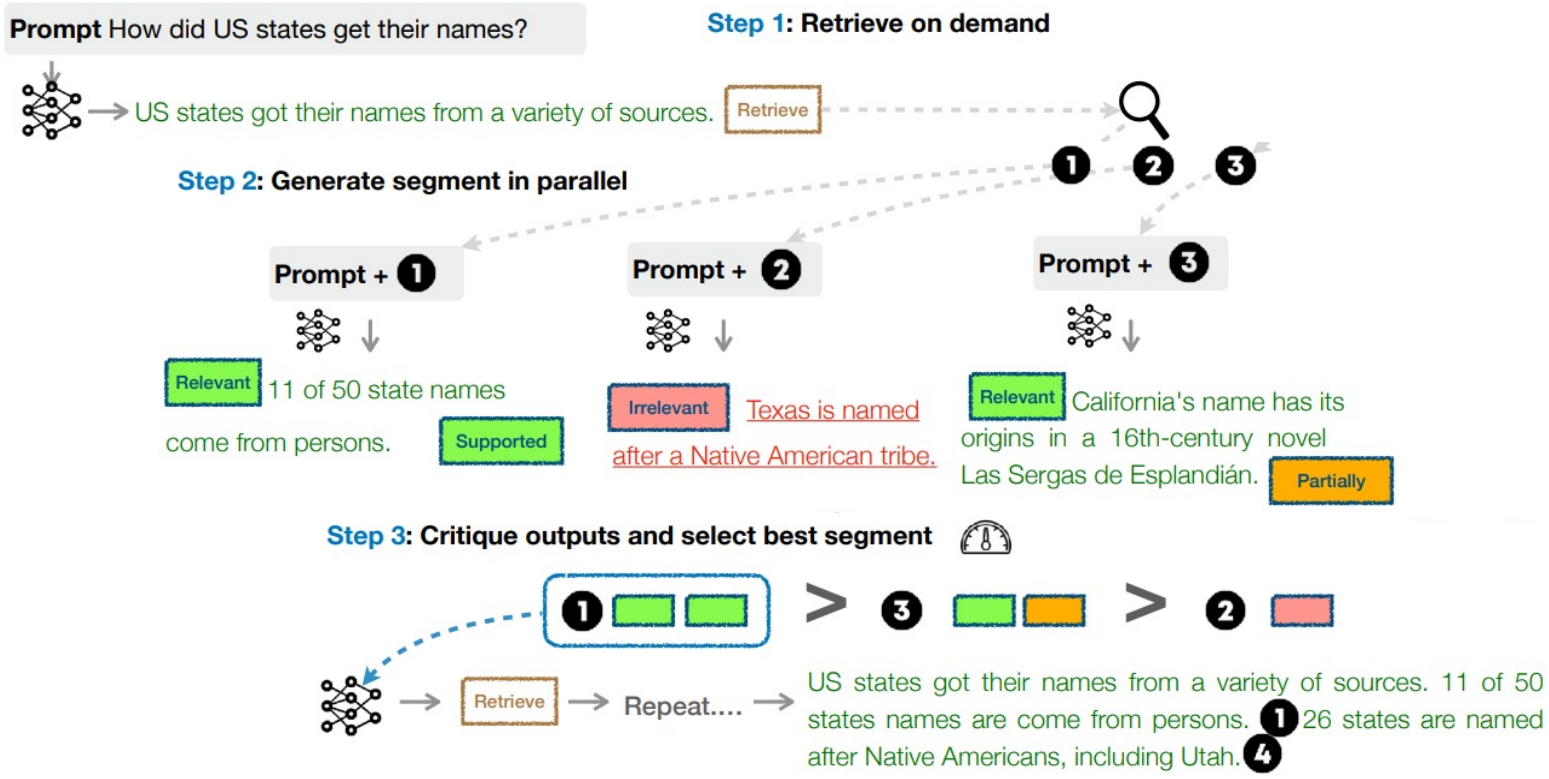
Implementation: Training

- **REPLUG:** No training, use off-the-shelf retriever (Contriever)
- **REPLUG LSR:** Use retriever (Contriever) + LM (GPT-3 Curie)
  - Sample training queries from Pile dataset (256 tokens x 800K sequences)
  - Query split into:
    - Input context x (128 tokens)
    - Ground truth continuation y (128 tokens)
  - Sample external documents from Pile (128 tokens x 36M documents)
  - Training queries and documents are verified for non-overlapping

REPLUG


Retrieved document d_i
Retriever
Jobs cofounded Apple in his parents' garage
Document Retrieval
Input Reformulation
Test Context x
Jobs is the CEO of _
Black-box LM
Apple
d_i
x
Jobs was raised by adopted…
Jobs is the CEO of _
Steve Jobs passed away…
Jobs is the CEO of _
Jobs cofounded Apple…
Jobs is the CEO of _
apple
pear
not
…
Ensemble

Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection



Problems of RAG Models

- Robustness to Irrelevant Contexts

Ram, Ori, et al. "In-context retrieval-augmented language models." *arXiv preprint arXiv:2302.00083* (2023).

Discussion

- What types of tasks are not suitable for RAG?
  - Lack of expert knowledge (i.e., retrieval coverage)
  - Doesn't require factual supports (i.e., writing a fiction)
- What's the down-side of Self-RAG and RePLUG?
  - Latency: Processing multiple passages in parallel still bring up computational cost (depending on the GPU usage).
- What are the pros and cons of different types of RAG?
  - Input text concatenation, cross-attention, output interpolation
- Do we need RAG with long-context large language models?
  - Lost in the Middle: How Language Models Use Long Contexts

Care Coordination

- Data fragmentation across the healthcare continuum

Poor connection in the healthcare system can spur redundant costs.

A study in the Journal of the American Medical Association found that "failures in care coordination" cost between $27.2 and $78.2 billion per year.

Generative AI could transform a considerable percentage of tasks.

Generative AI has high potential to automate or augment 39% of all working hours in the health industry, per an Accenture study.

Leading electronic health vendors are adopting generative AI to advance their systems.

EHR vendor Epic has integrated with Microsoft's Azure OpenAI Service to allow healthcare providers to automatically generate replies to messages and find ways to cut costs.

GenAI in Healthcare

Patients are fending for themselves over navigating healthcare systems.

Nearly 40% of Americans feel unsupported in understanding their healthcare, while 70% say that the system is hard to navigate, per a study from Maestro Health. As a result, many patients look up their symptoms online, which may lead to misinformation.

Greater access to health information enables patients to better participate in their own care.

Patients who can review their physician's notes are more likely to adhere to their treatment plan than those who cannot.

Healthcare LLMs put patients in the center.

Hippocratic AI is building the first safety-focused large language model (LLM) for healthcare, allowing for more solutions to be designed specifically for patients.

COMPANION BOTS

SENSELY

notable

soul machines

SYMPTOM CHECKERS

buoy

clearstep

ada

mediktor

PATIENT EDUCATION

SayHeart

vital

ANDOR HEALTH

ASSISTIVE TECH

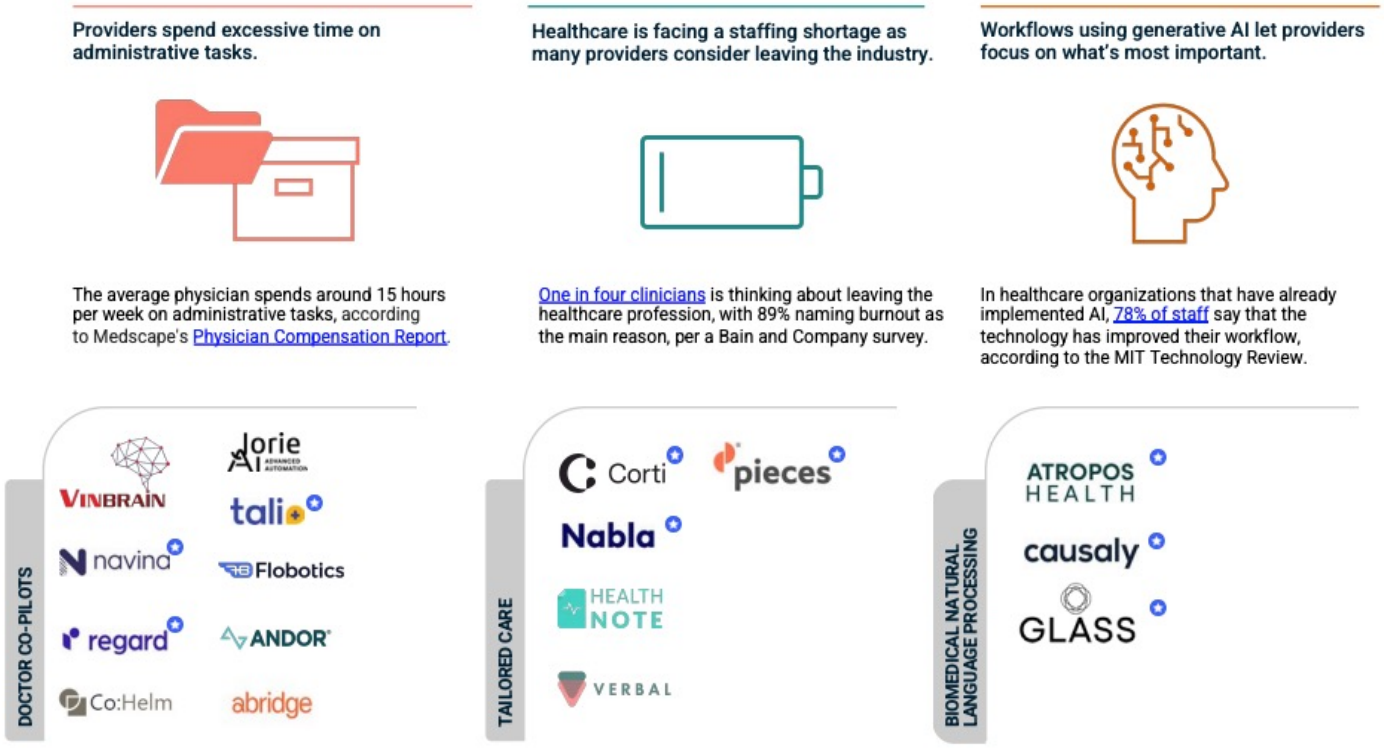
be my eyes

OurAbility

USERWAY

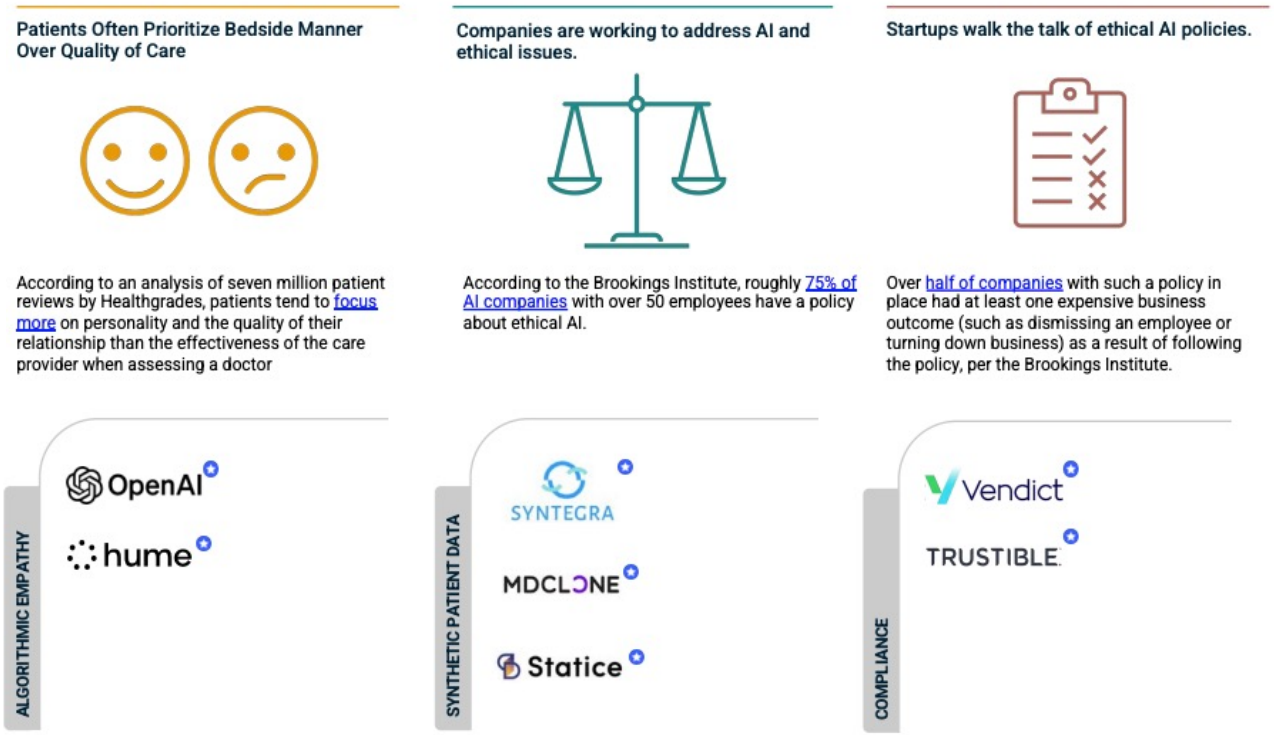
Providers

- Paperwork and repetitive administrative tasks contribute to healthcare worker burnout



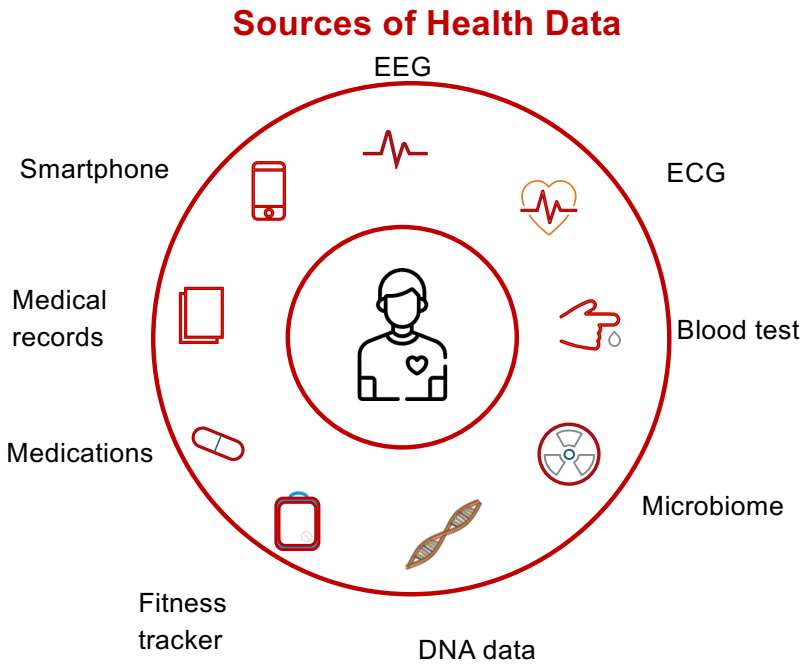
Ethical Behavior

- AI solutions that support algorithmic empathy, synthetic patient data and greater compliance



Health data

- How much health data the average person generates?
- How much data a consumer's smart phone, smart watch, or fitness tracker generates daily?
- The difference between health information and Protected Health Information (PHI)
- Health Insurance Portability & Accountability Act (HIPAA) provides Covered Entities (plans, providers & clearing houses) with requirements to protect the privacy and security of health information and must provide individuals with certain rights with respect to their health information



Risks associated with AI in healthcare

Privacy issues for data used for AI model training

- Pregnancy tracking app Ovia Health
- Helpful monitoring tool for millions of users
- Monitoring tool for employers
- Employers could pay the app developer for access to aggregated data of the users

Privacy experts worry that issues like this could impact the career path of employees without their knowledge

Pregnancy app-maker Ovia Health is selling data to employers

Ovia Health has a suite of apps to help women keep tabs on their own fertility cycles. Employers are also getting a peek

Amy Martyn
Reporter

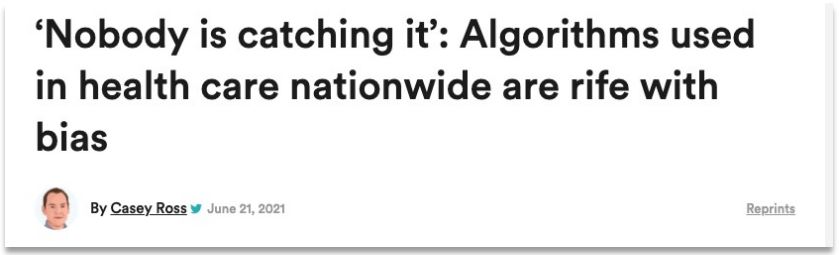
Risks associated with AI in healthcare

- Big Tech and startups are investing heavily in AI health care. They will need vast amounts of data to stay relevant
- For AI to function accurately it needs to leverage data, which often is coming from patients themselves
- Some algorithms are biased and scaling that bias [Ross 2021]
- Achieving generalization in AI requires exposing algorithms to diverse data while training (e.g., Evaluation of the clinical value of the Epic Sepsis Model [Wong, et al. 2021])

A hospital algorithm designed to predict a deadly condition misses most cases

A new study found that it also had many false alarms

By Nicole Wetsman | Jun 22, 2021, 11:31am EDT



'Nobody is catching it': Algorithms used in health care nationwide are rife with bias

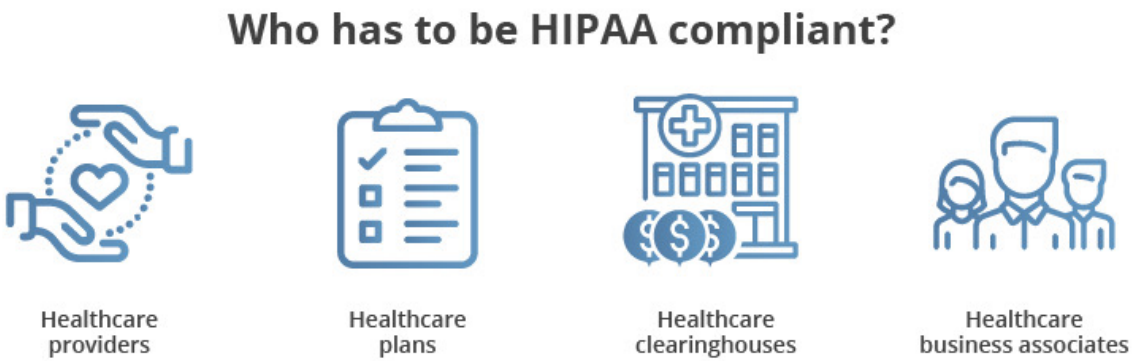
By Casey Ross June 21, 2021 Reprints

https://www.statnews.com/2021/06/21/algorithm-bias-playbook-hospitals/
https://www.theverge.com/2021/6/22/22545044/algorithm-hospital-sepsis-epic-prediction
https://jamanetwork.com/journals/jamainternalmedicine/article-abstract/2781307

HIPAA

HIPAA Privacy and Security Rules require healthcare organizations
to adopt processes and procedures

HIPAA compliance



HIPAA defines three categories of covered entities:

- **Health care providers:** Hospitals, clinics, medical laboratories, pharmacies, nursing homes, doctors, psychologists, dentists, chiropractors, et cetera
- **Health care plans:** Health insurance and health maintenance companies, government programs such as Medicare and Medicaid, military healthcare programs
- **Health care clearinghouses:** Organizations that create, receive, maintain, edit, or transmit any protected health information (PHI)

Thank you!