



# Vision Transformers

DL4DS – Spring 2025

*A survey in three papers.*

# Reminders

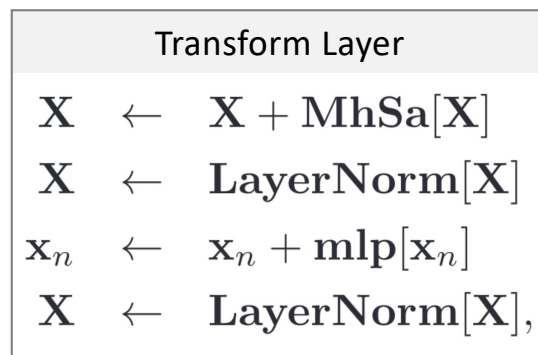
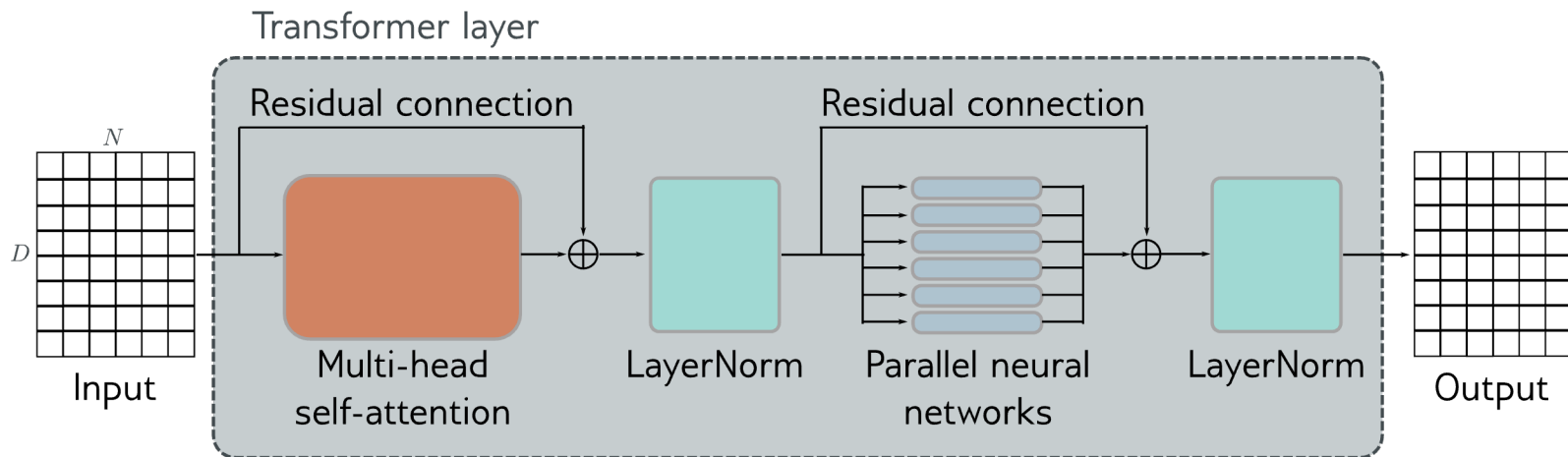
- Mid-project check-in

# Topics

- Transformers Recap
- ImageGPT
- Vision Transformer (ViT)
- CLIP – Contrastive Learning w/ Image Pre-Training

# Transformers Recap

# Transformer Layer -- Complete

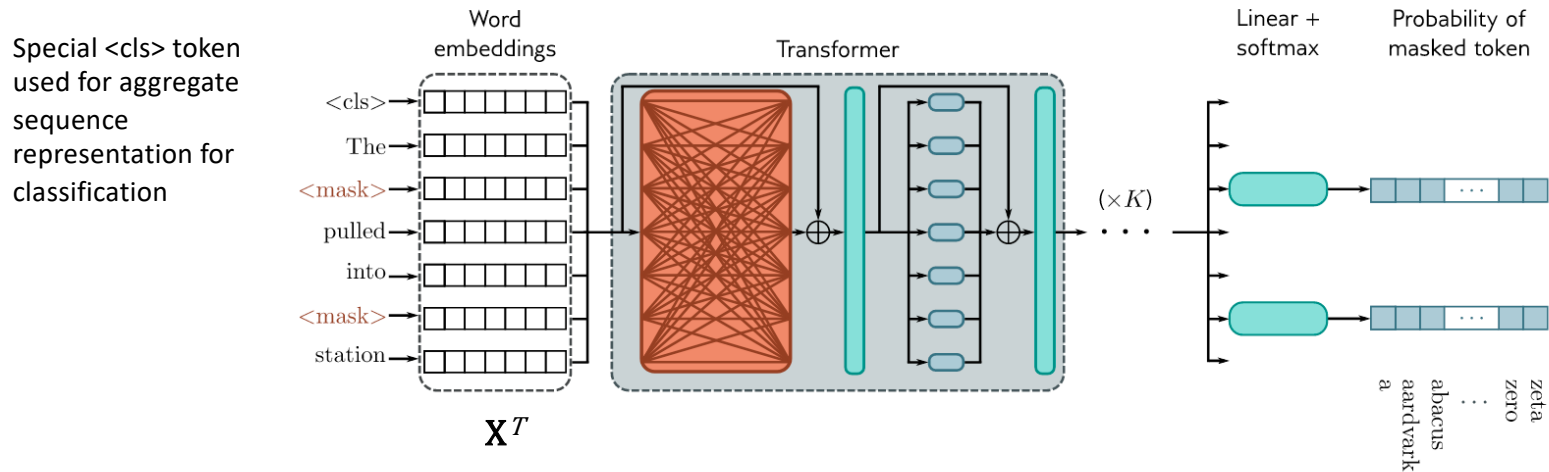


**LayerNorm**

$$y = \frac{x - \mathbf{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

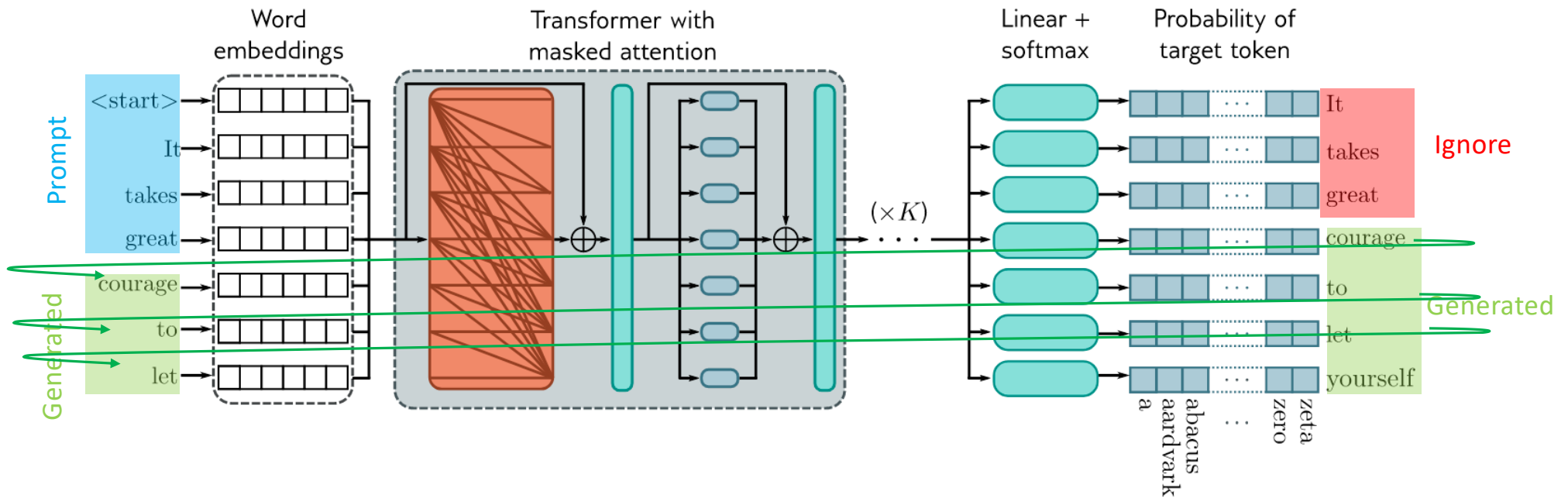
Calculated column-wise

# Encoder Pre-Training



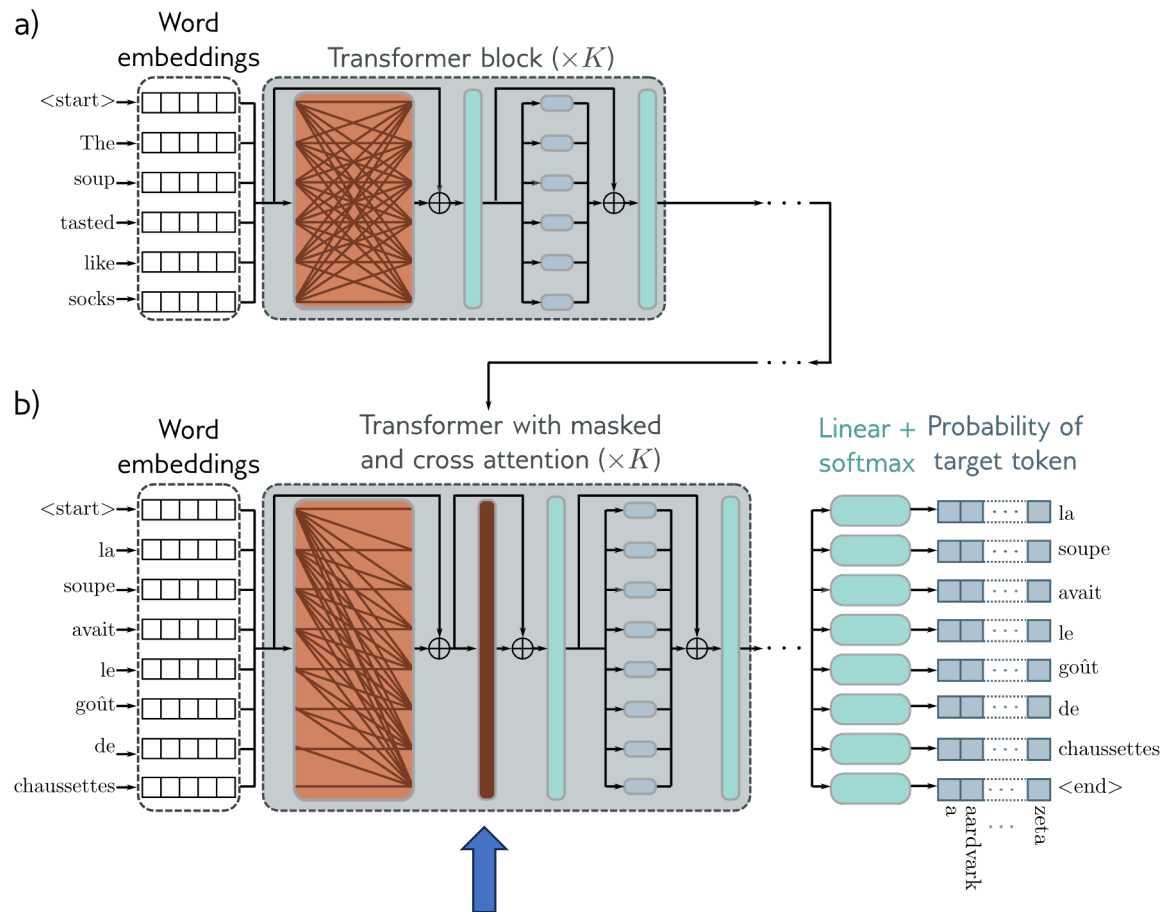
- A small percentage of input embedding replaced with a generic <mask> token
- Predict missing token from output embeddings
- Added linear layer and softmax to generate probabilities over vocabulary
- Trained on BooksCorpus (800M words) and English Wikipedia (2.5B words)

# Decoder: Text Generation (Generative AI)



- Feed the output back into input

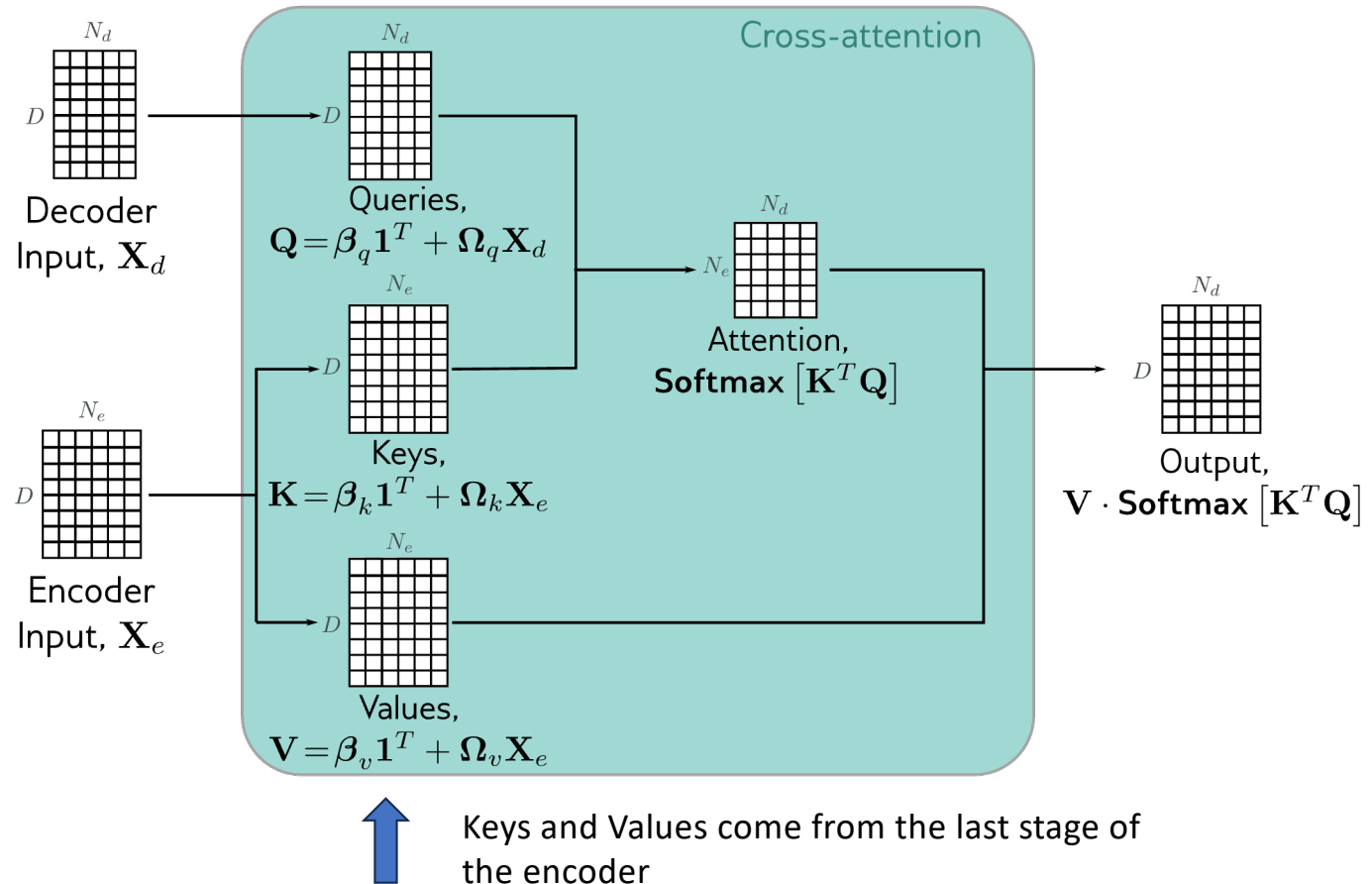
# Encoder Decoder Model



- The transformer layer in the decoder of the encoder-decoder model has an extra stage
- Attends to the input of the encoder with *cross attention* using Keys and Values from the output of the encoder



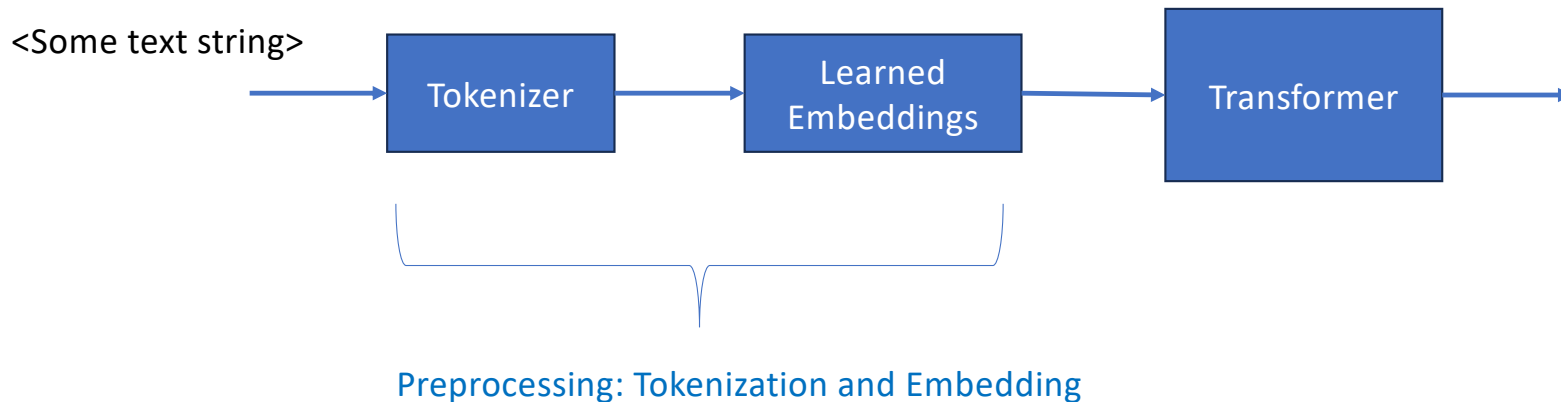
# Cross-Attention



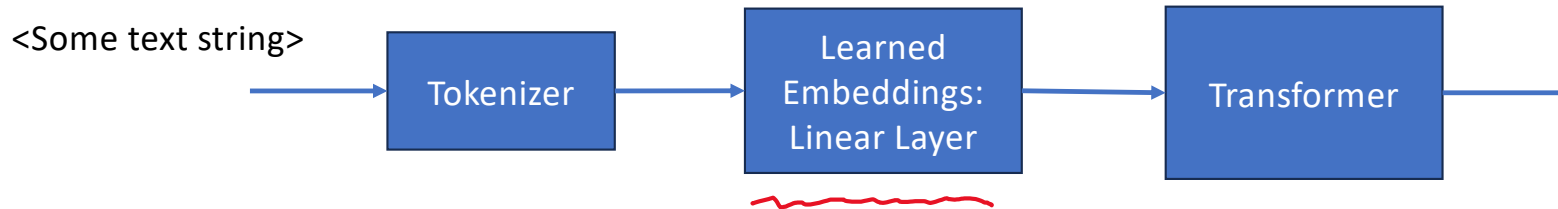
# NLP Preprocessing Pipeline

Transformers don't work on character string directly, but rather on vectors.

The character strings must be converted to vectors



# Learned Embeddings

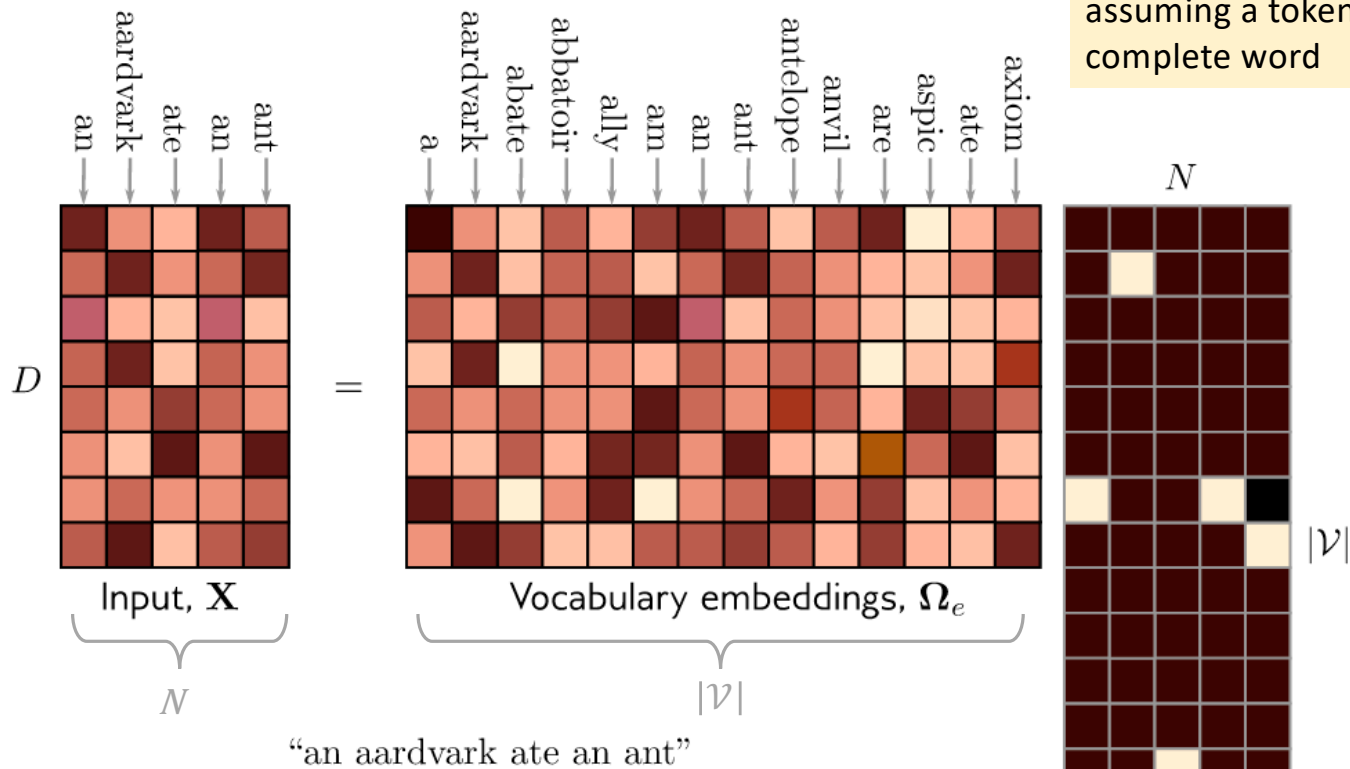


- After the tokenizer, you have an updated "vocabulary" indexed by token ID
- Next step is to translate the token into an embedding vector
- Translation is done via a linear layer which is typically learned with the rest of the transformer model

```
self.embedding = nn.Embedding(vocab_size, embedding_dim)
```

- Special layer definition, likely to exploit sparsity of input

# Embeddings Output

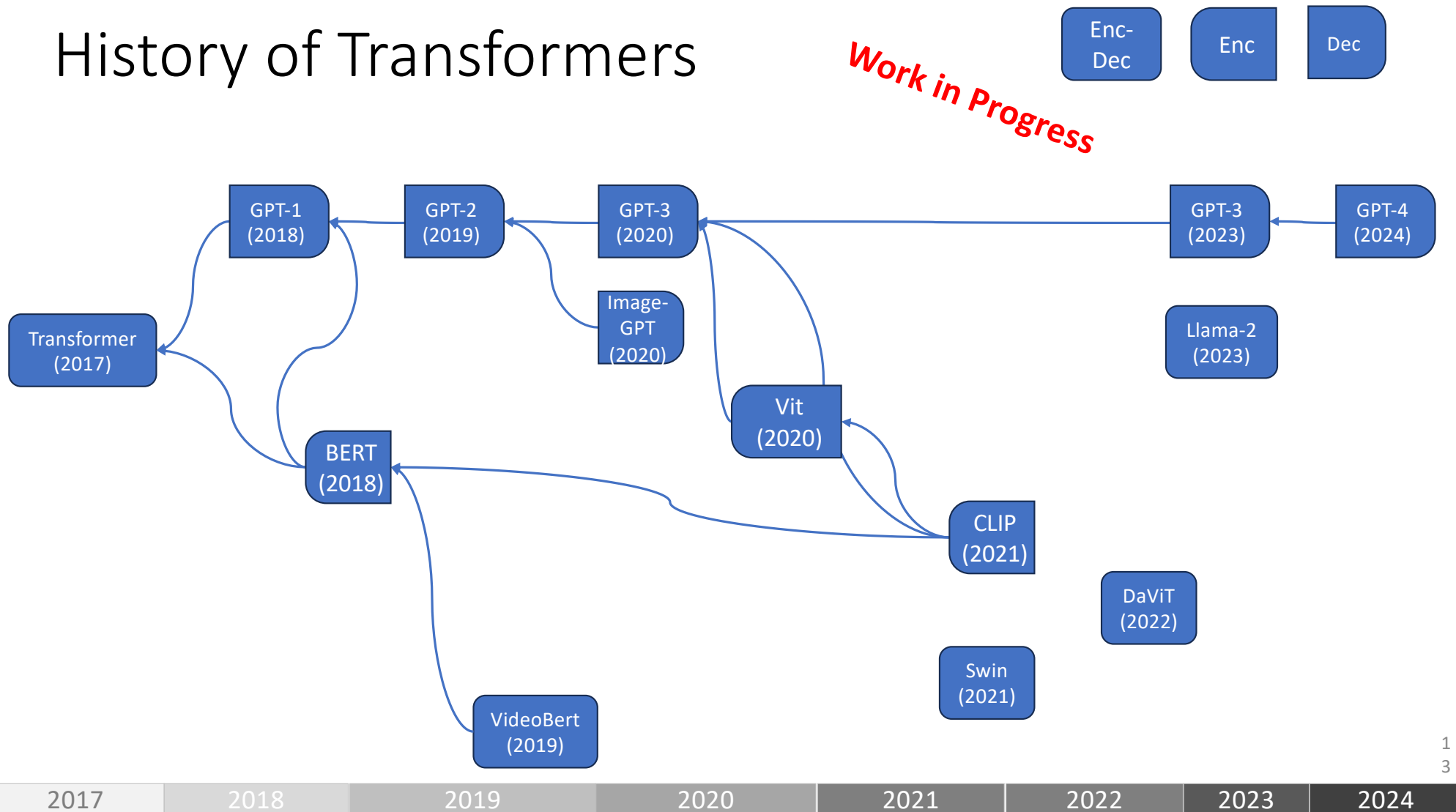


In this example, we are assuming a token is simply a complete word

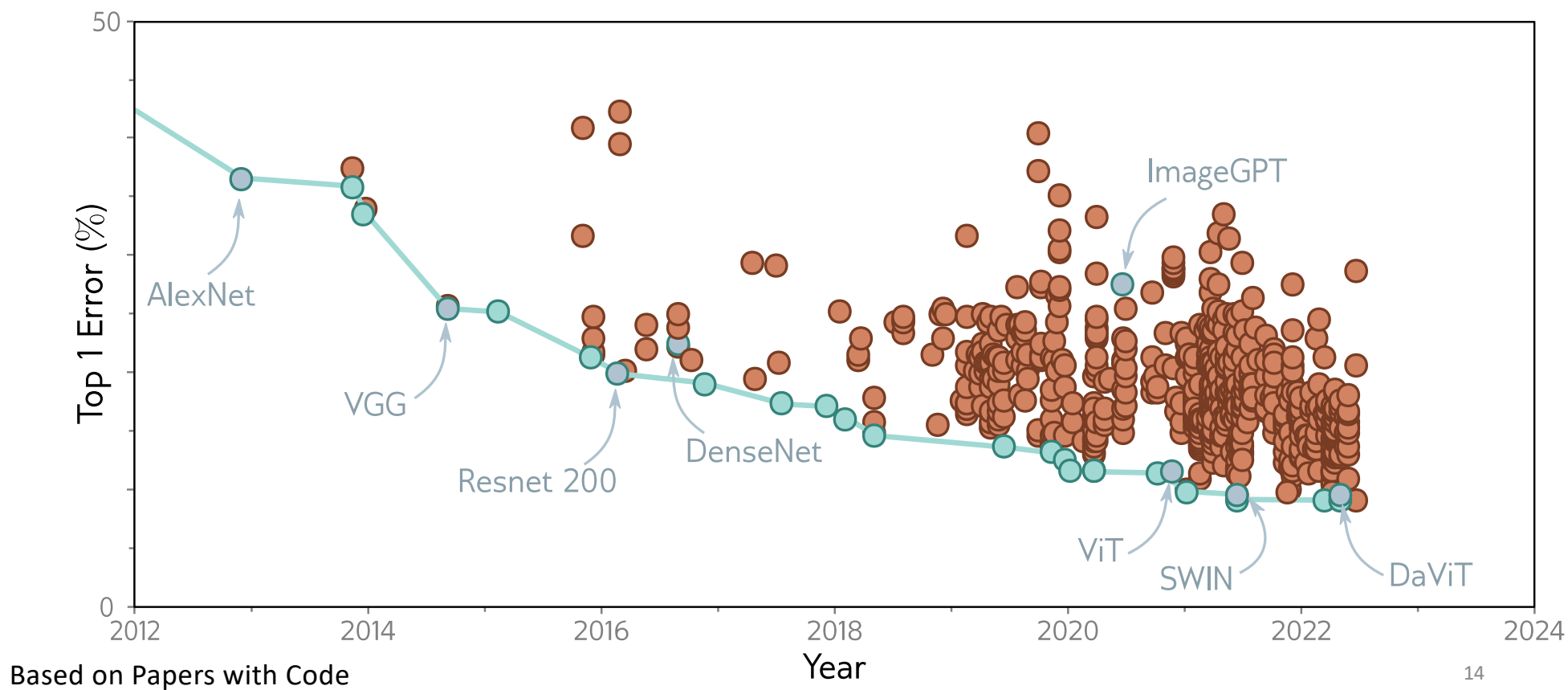
"One hot encoding"

- Typical embedding size,  $D$ , is 1024
- Typical vocabulary size,  $|\mathcal{V}|$ , is 30,000
- So 30M parameters just for this matrix!

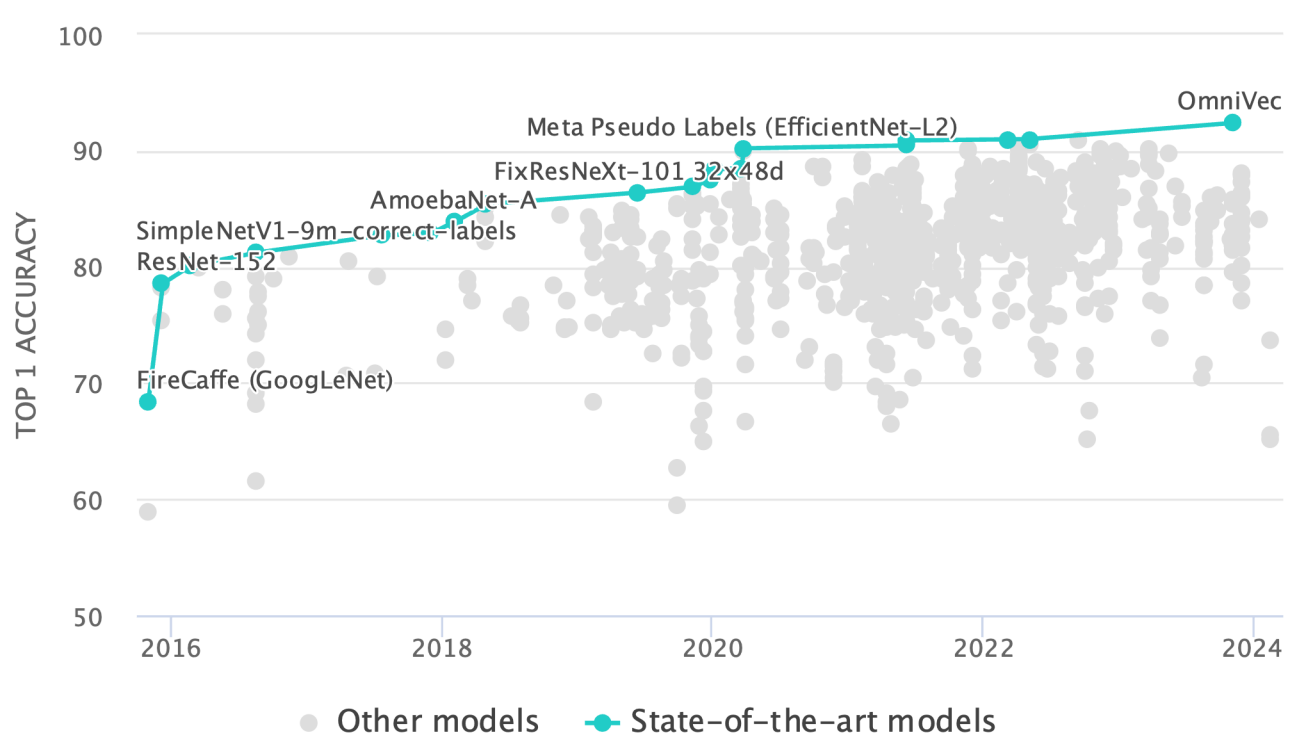
# History of Transformers



# ImageNet History – Top-1 Error

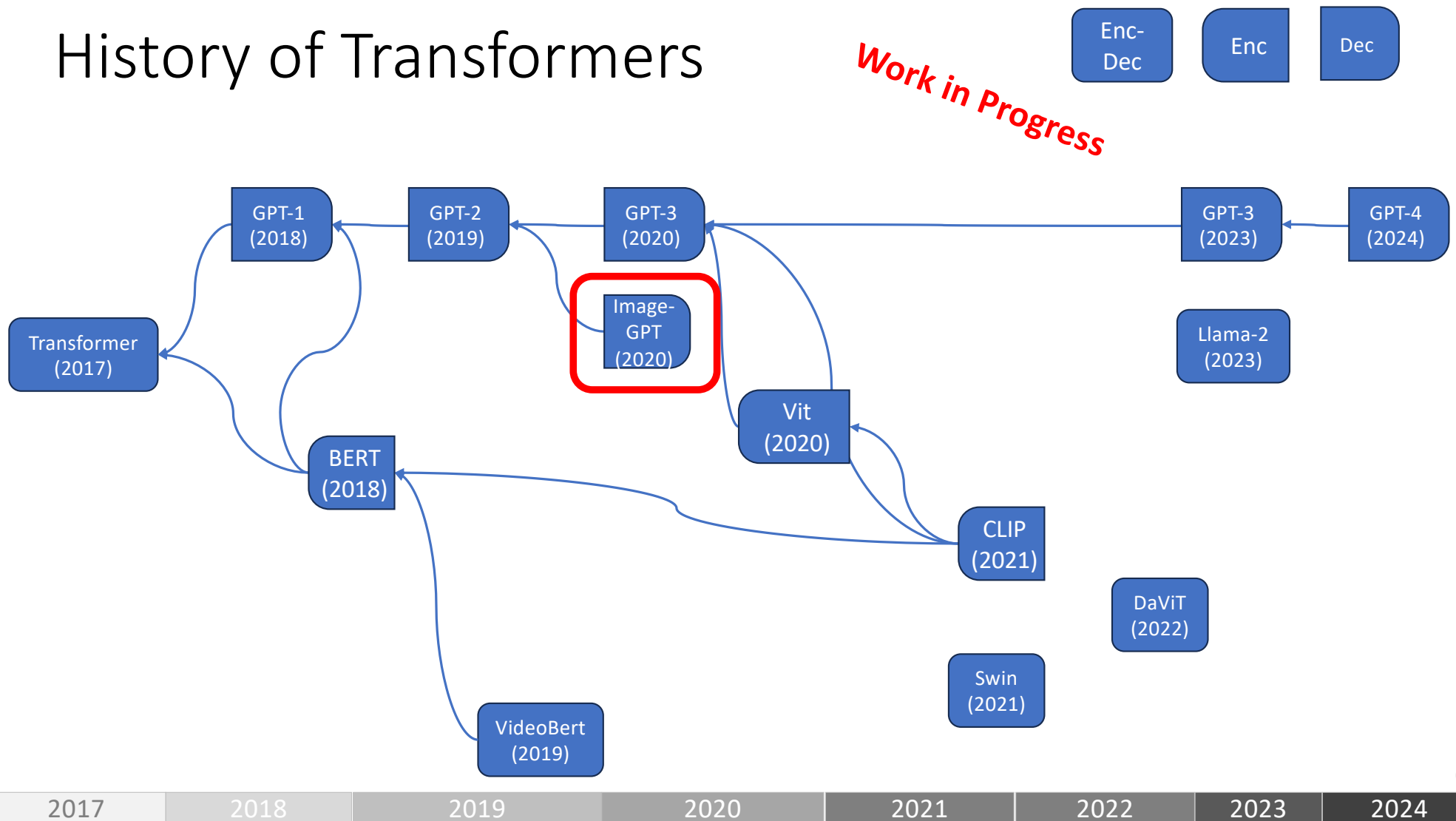


# ImageNet Top-1 Accuracy



<https://paperswithcode.com/sota/image-classification-on-imagenet>

# History of Transformers





# Image GPT – June 2020

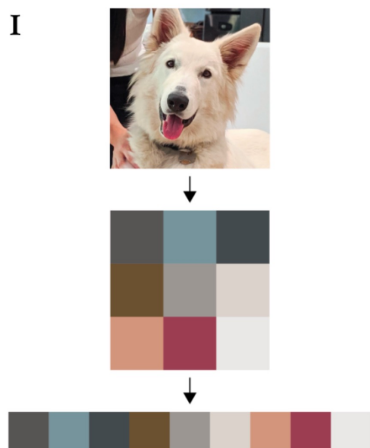


- Train GPT-2 scale sequence Transformer to auto-regressively predict pixels, w/o 2D input structure
- Use GPT-2 with only minor changes
- ImageNet Top-1 72% accuracy (not great), trained on ImageNet and web images
- Primary objective is to explore the representation accuracy of internal features

<https://openai.com/research/image-gpt>  
<https://github.com/openai/image-gpt> (deprecated)  
[https://huggingface.co/docs/transformers/model\\_doc/imagegpt](https://huggingface.co/docs/transformers/model_doc/imagegpt)

M. Chen *et al.*, “Generative Pretraining from Pixels,” OpenAI, Technical Report, Jun. 2020.

# Image GPT – Inputs



- Reduced resolution to reduce context size:  
 $32 \times 32$ ,  $48 \times 48$  or  $64 \times 64$
- Also reduced color palette from  $3 \times 8 = 24$  bit (16M colors) to a 9-bit (512 colors) color palette by clustering (R, G, B) pixels with  $k = 512$  colors.
- The “vocabulary” is then the 512 palletized colors
- Which can be input as one-hot encoded

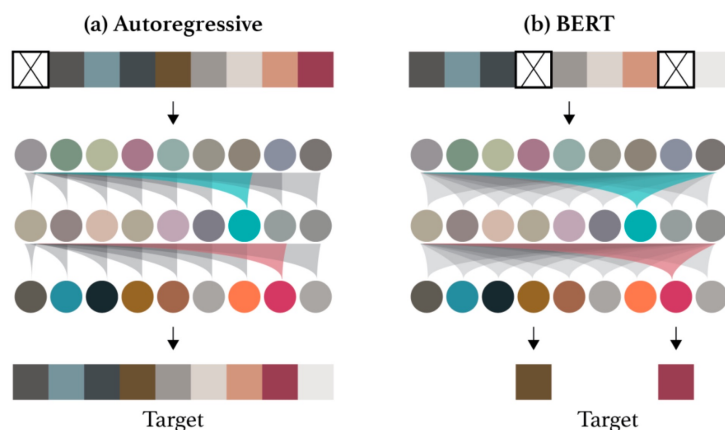
<https://openai.com/research/image-gpt>  
<https://github.com/openai/image-gpt> (deprecated)  
[https://huggingface.co/docs/transformers/model\\_doc/imagegpt](https://huggingface.co/docs/transformers/model_doc/imagegpt)

M. Chen *et al.*, “Generative Pretraining from Pixels,” OpenAI, Technical Report, Jun. 2020.

# Image GPT – Training Objectives



2

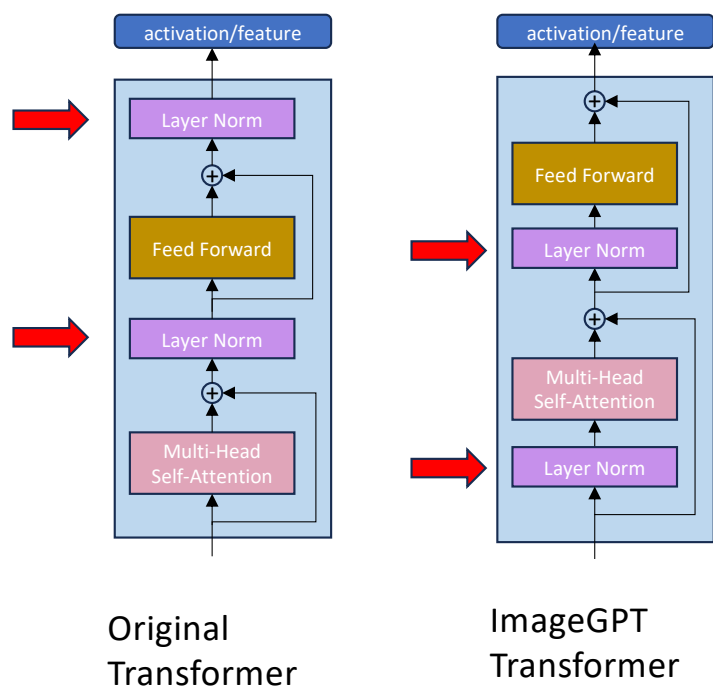


- Trained on large collection of unlabeled images.
- Tried training with either *Autoregressive* or *BERT* style training objective
- Predict the masked pixels

<https://openai.com/research/image-gpt>  
<https://github.com/openai/image-gpt> (deprecated)  
[https://huggingface.co/docs/transformers/model\\_doc/imagegpt](https://huggingface.co/docs/transformers/model_doc/imagegpt)

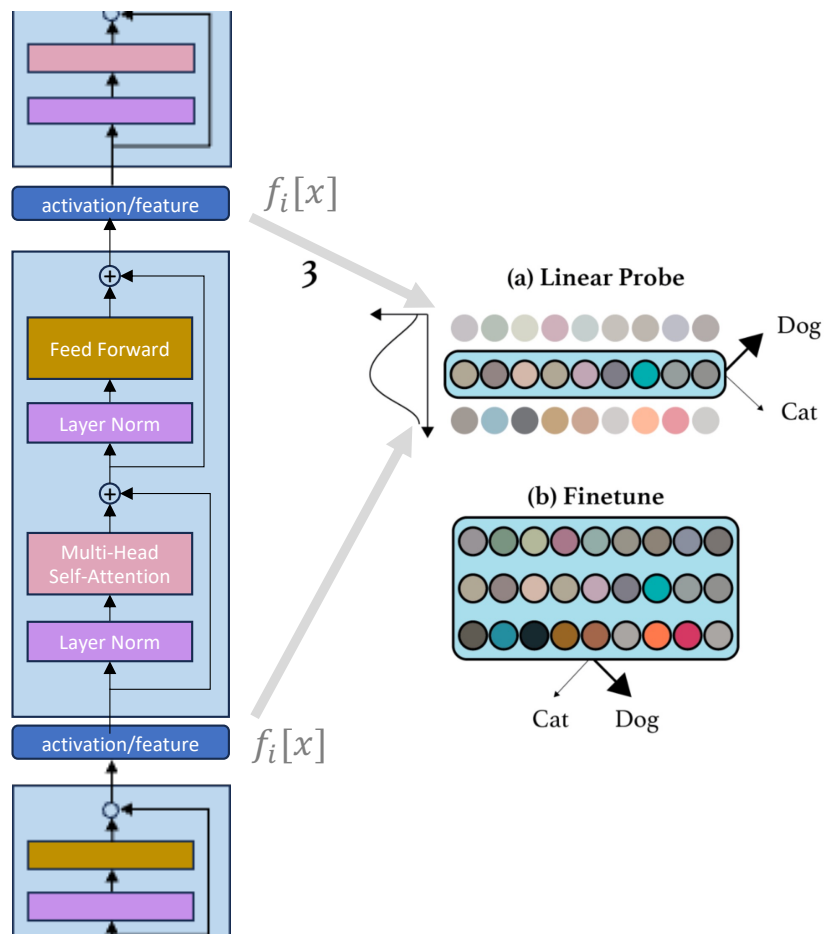
M. Chen *et al.*, "Generative Pretraining from Pixels," OpenAI, Technical Report, Jun. 2020.

# Image GPT – Transformer Layer



- LayerNorm moved to precede Self-Attention and Feed Forward block
- In the residual path

# Image GPT – Linear Probes

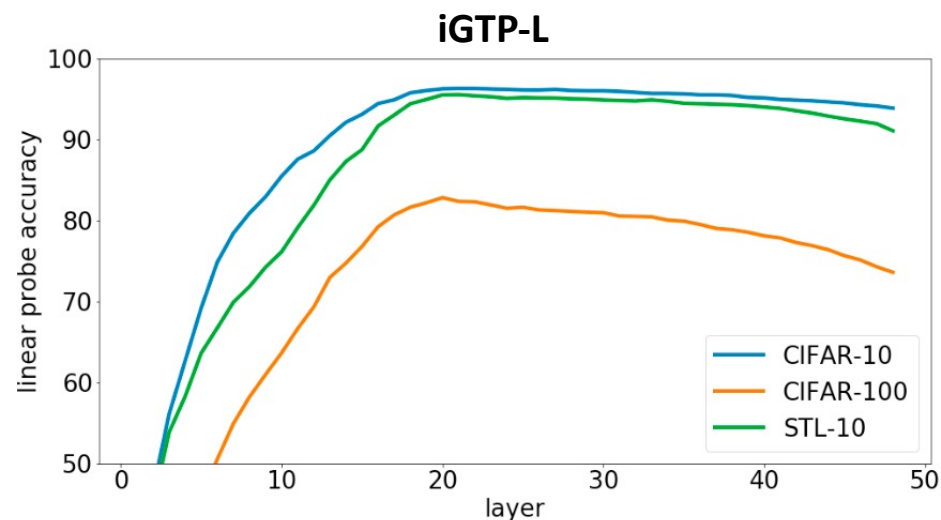


- Use pre-trained model as a “feature extractor”
- Activations after each layer → Features
  - call  $i^{th}$  feature:  $f_i[x]$
- Good features should linearly separate the classes of transfer tasks
- → linear classifier trained on  $(f_i[x], Y)$
- Do this with each feature layer and see which performs best

# Image GPT – Representation Quality



Size	Layers	d	# parms
iGPT-S	24	512	76M
iGPT-M	36	1024	455M
iGPT-L	48	1536	1.4B
iGPT-XL	60	3072	6.8B



- Feature layer linear classifier trained for each dataset
- Classification representation quality by feature layer
- Best representation seems to lie in the middle
- As opposed to supervised-training where the best representations lie at the end of the network

slido



**Why is best representation in the middle as opposed to the end of the network like in supervised training?**

① Start presenting to display the poll results on this slide.

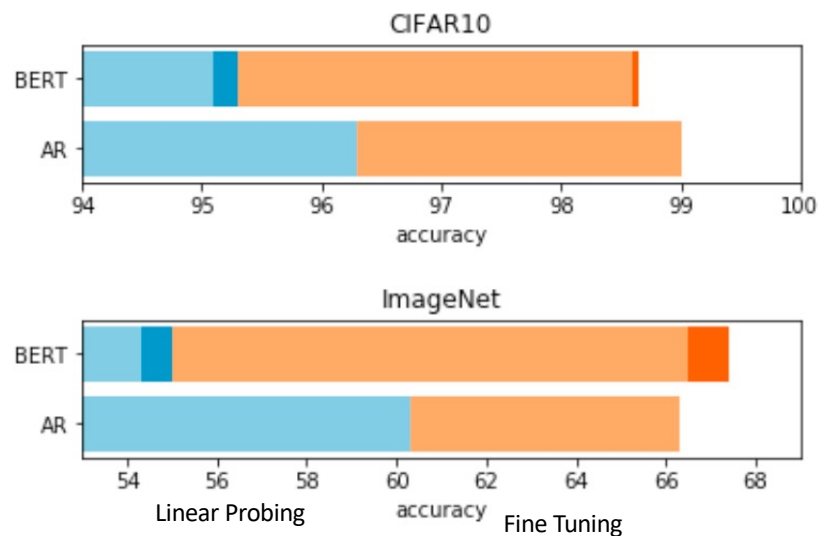
# Image-GPT –

Perhaps generative model operates in two phases:

1. The first phase gathers information from surrounding context in order to build a more global representation.
2. In 2<sup>nd</sup> phase, contextualized input is used to solve conditional next pixel prediction task



# Image GPT – Finetuning for Classification



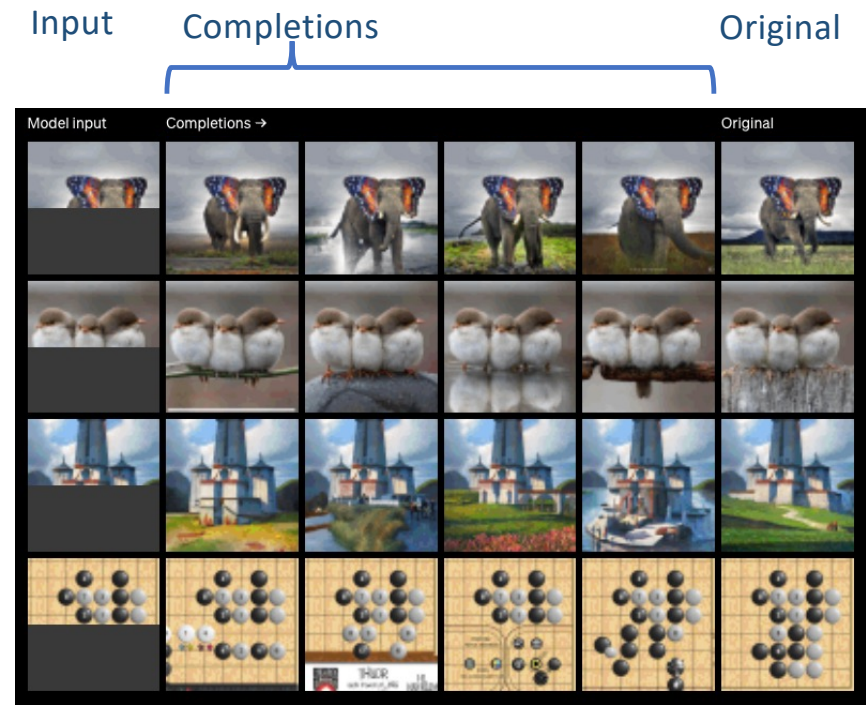
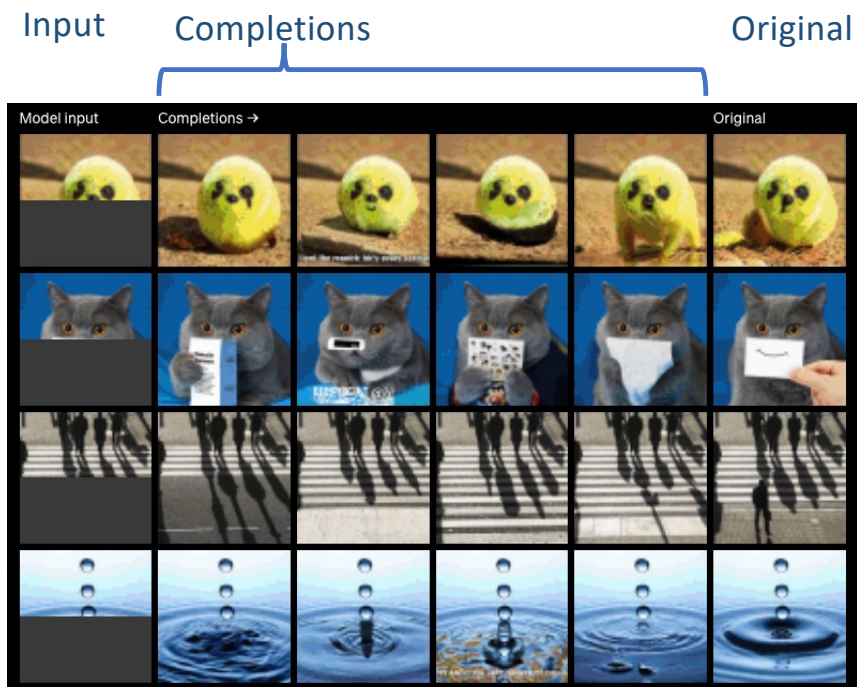
Comparison of auto-regressive pre-training with BERT pre-training using iGPT-L at an input resolution of  $322 \times 3$ .

Blue bars display linear probe accuracy and orange bars display finetune accuracy.

**Bold** colors show the performance boost from ensembling BERT masks – use multiple BERT-style masking of each image then combine.

- Finetuning on the target dataset further improves accuracy
- Finetuning the entire model outperformed finetuning the best linear probe feature
- auto-regressive models produce much better features than BERT models after pre-training,
- but BERT models catch up after fine-tuning.

# Image GPT – AR Pixel Prediction Results



# Image GPT – Sampling the Distribution



# Image GPT – Pros and Cons

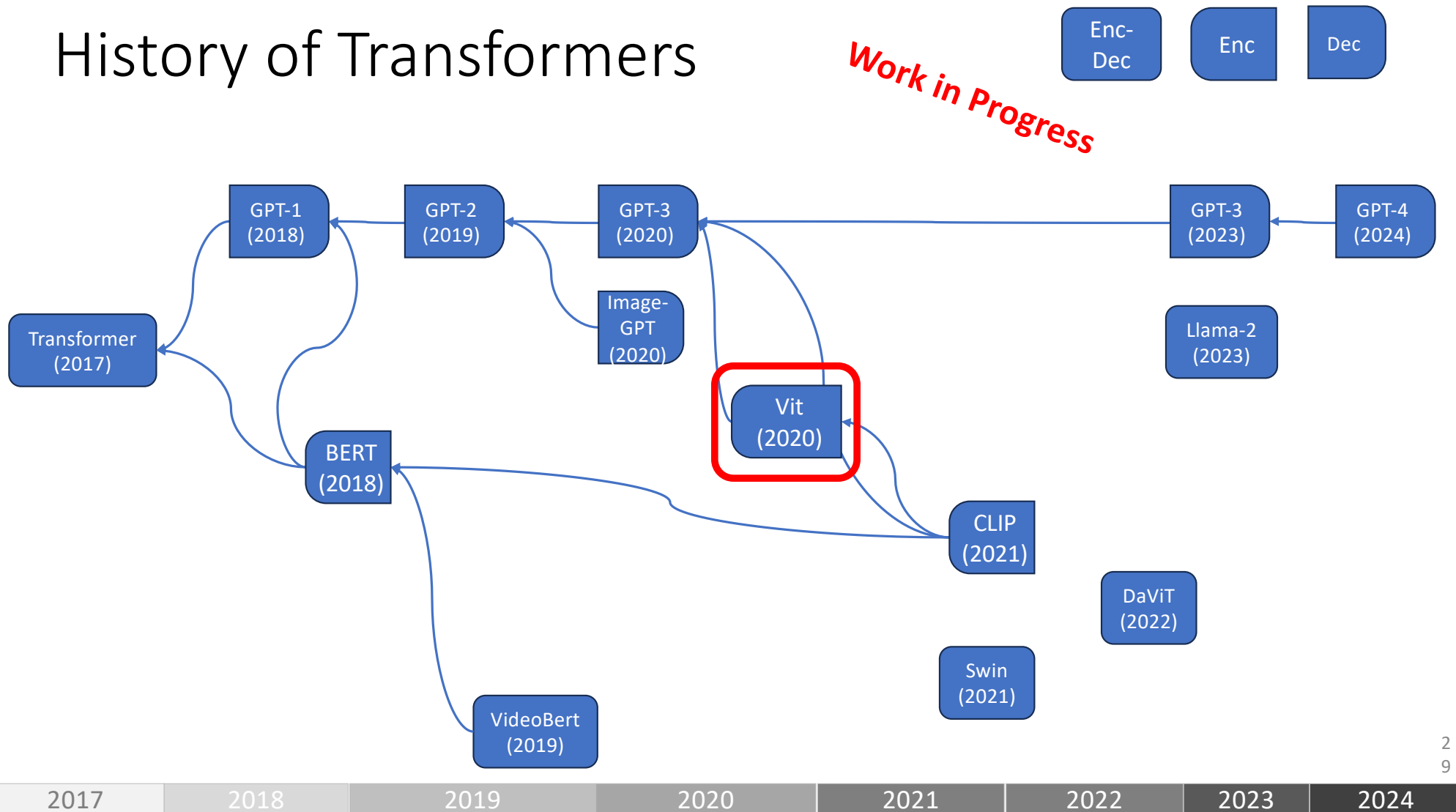
## Pro:

- Gave insights into the representational power of Transformers with unsupervised training

## Con:

- Worked on downscaled images of size 32x32 to 64x64

# History of Transformers



# Vision Transformer (ViT) – June 2021

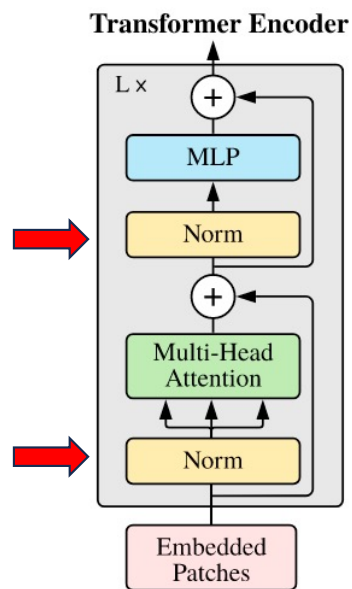
- Overcomes resolution limitation of ImageGPT by using patches
- Follows scalable NLP Transformer architectures to benefit from efficient implementations
- ImageNet Top-1 accuracy: [88.55%](#)
- Performs poorly if just trained on ImageNet
  - → can be expected since Transformers lack the inductive bias of CNNs
- Competitive when pre-trained on very large datasets (e.g. 14M – 300M) images – all supervised at this point

Large scale training trumps inductive bias.

<https://arxiv.org/abs/2010.11929v2>

[https://github.com/google-research/vision\\_transformer](https://github.com/google-research/vision_transformer)

# Vision Transformer (ViT) – June 2021



- Uses same Transformer layer as ImageGPT and scalable NLP Transformers

<https://arxiv.org/abs/2010.11929v2>

[https://github.com/google-research/vision\\_transformer](https://github.com/google-research/vision_transformer)

# ViT: Putting it all together

1. Divide image into  $P \times P$  patches





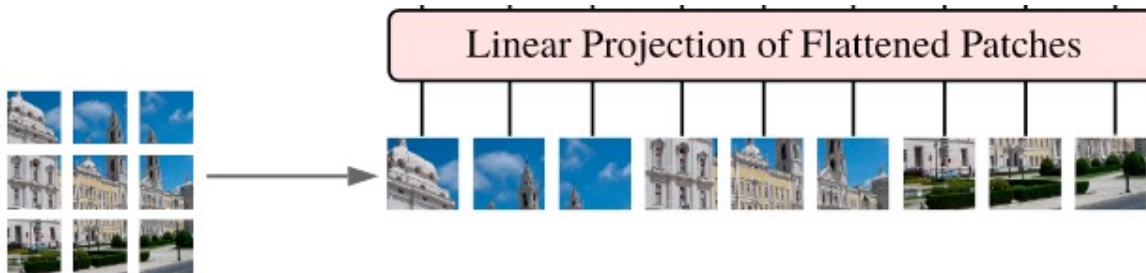
# ViT: Putting it all together

1. Divide image into  $P \times P$  patches
2. Create sequence of length  $N = HW/P^2$



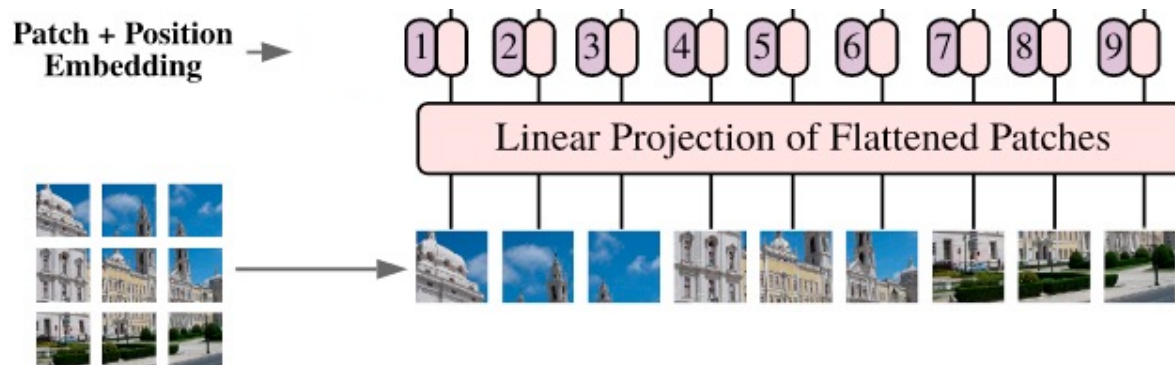
# ViT: Putting it all together

1. Divide image into  $P \times P$  patches
2. Create sequence of length  $N = HW/P^2$
3. Flatten the patches and map to  $D$  dimensions with a trainable linear projection



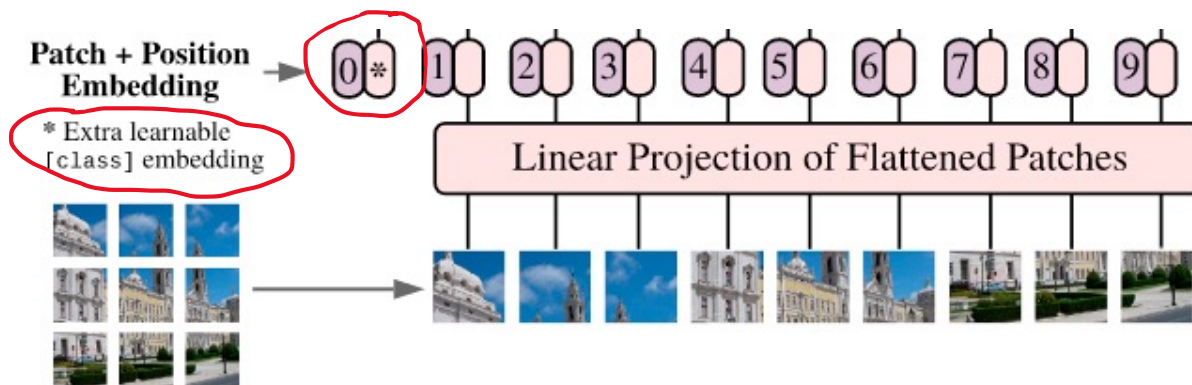
# ViT: Putting it all together

1. Divide image into  $P \times P$  patches
2. Create sequence of length  $N = HW/P^2$
3. Flatten the patches and map to  $D$  dimensions with a trainable linear projection
4. Add a learned 1-D position embedding

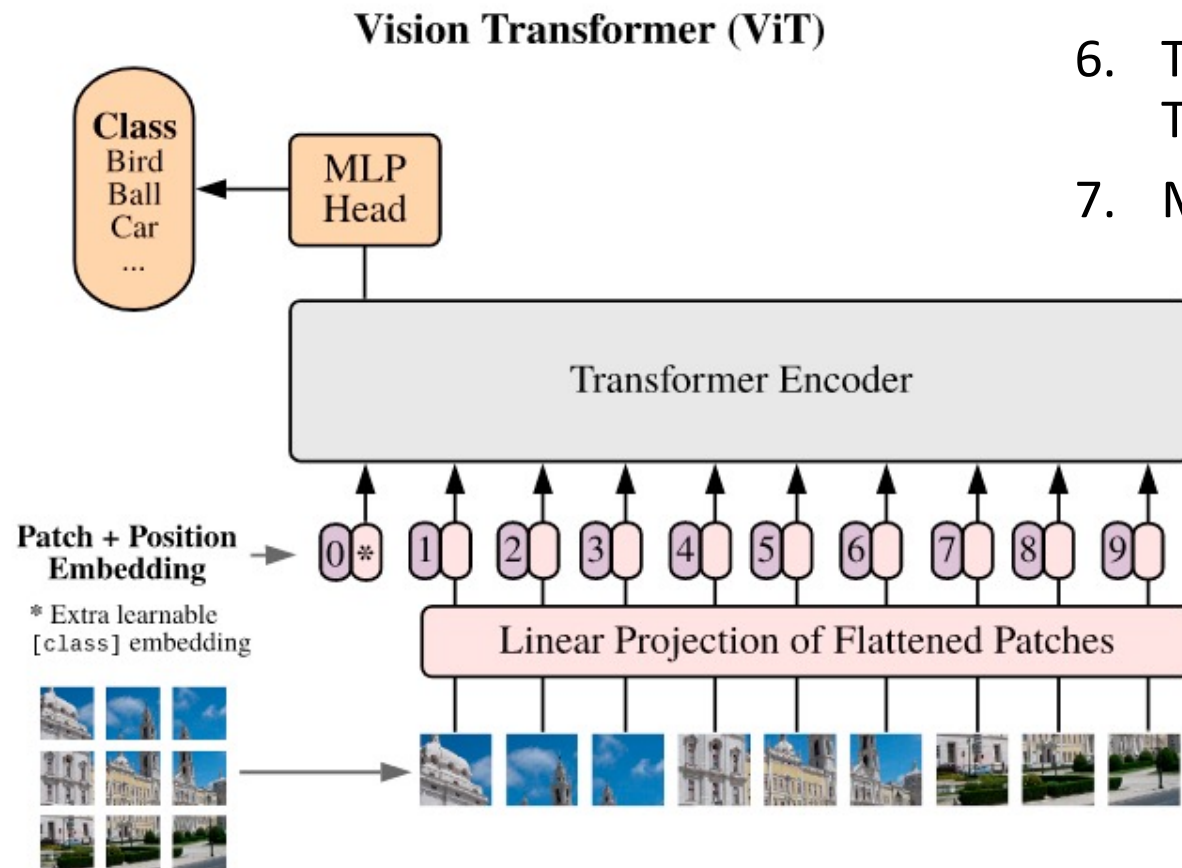


# ViT: Putting it all together

1. Divide image into  $P \times P$  patches
2. Create sequence of length  $N = HW/P^2$
3. Flatten the patches and map to  $D$  dimensions with a trainable linear projection
4. Add a learned 1-D position embedding
5. Include a learnable [class] embedding



# ViT: Putting it all together



6. Then through a multi-layered Transformer encoder to a
7. MLP classification head.

# ViT Training Datasets & Model Variants

Dataset	# Classes	# Images
ILSVRC-2012 (ImageNet-1K)	1K	1.3M
ImageNet-21K	21K	14M
JFT	18K	303M

Model	Layers	Hidden size $D$	MLP size	Heads	Params	
ViT-Base	12	768	3072	12	86M	Same as BERT
ViT-Large	24	1024	4096	16	307M	Same as BERT
ViT-Huge	32	1280	5120	16	632M	New for ViT

Notation: ViT-L/16 -- "Large" variant with  $16 \times 16$  input size.

Note:  $16 \times 16 \times 3 = 768$

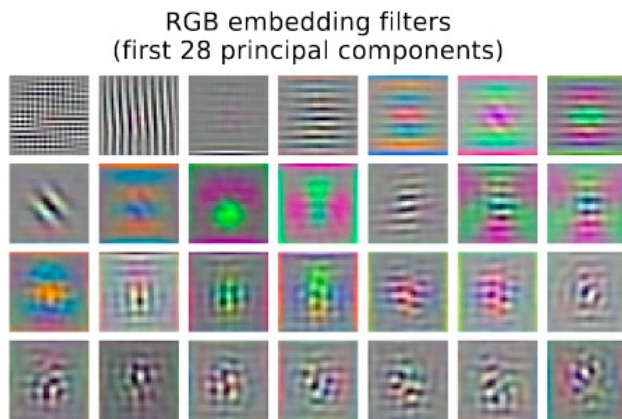
# ViT: Image Classification Results

Pre-Trained On	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	<b>88.55</b> $\pm 0.04$	87.76 $\pm 0.03$	85.30 $\pm 0.02$	87.54 $\pm 0.02$	88.4/88.5*
ImageNet ReaL	<b>90.72</b> $\pm 0.05$	90.54 $\pm 0.03$	88.62 $\pm 0.05$	90.54	90.55
CIFAR-10	<b>99.50</b> $\pm 0.06$	99.42 $\pm 0.03$	99.15 $\pm 0.03$	99.37 $\pm 0.06$	—
CIFAR-100	<b>94.55</b> $\pm 0.04$	93.90 $\pm 0.05$	93.25 $\pm 0.05$	93.51 $\pm 0.08$	—
Oxford-IIIT Pets	<b>97.56</b> $\pm 0.03$	97.32 $\pm 0.11$	94.67 $\pm 0.15$	96.62 $\pm 0.23$	—
Oxford Flowers-102	99.68 $\pm 0.02$	<b>99.74</b> $\pm 0.00$	99.61 $\pm 0.02$	99.63 $\pm 0.03$	—
VTAB (19 tasks)	<b>77.63</b> $\pm 0.23$	76.28 $\pm 0.46$	72.72 $\pm 0.21$	76.29 $\pm 1.70$	—
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

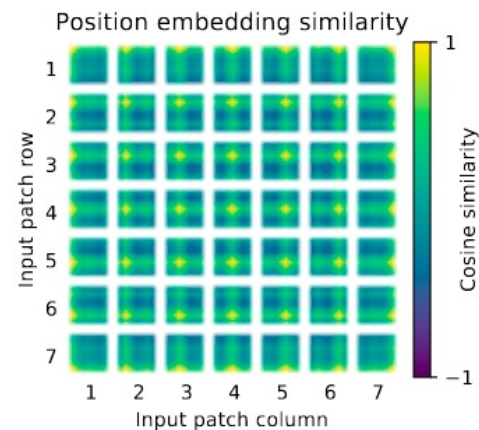
Model	Layers	Hidden size $D$	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Notation: ViT-L/16 -- "Large" variant with 16 $\times$ 16 input size.

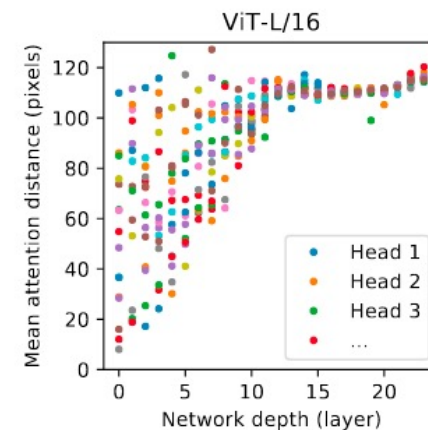
# ViT: Visualizing Internals



Filters of the initial linear embedding of RGB values of ViT-L/32.



Similarity of position embeddings of ViT-L/32. Tiles show the cosine similarity between the position embedding of the patch with the indicated row and column and the position embeddings of all other patches.



Size of attended area by head and network depth. Each dot shows the mean attention distance across images for one of 16 heads at one layer.



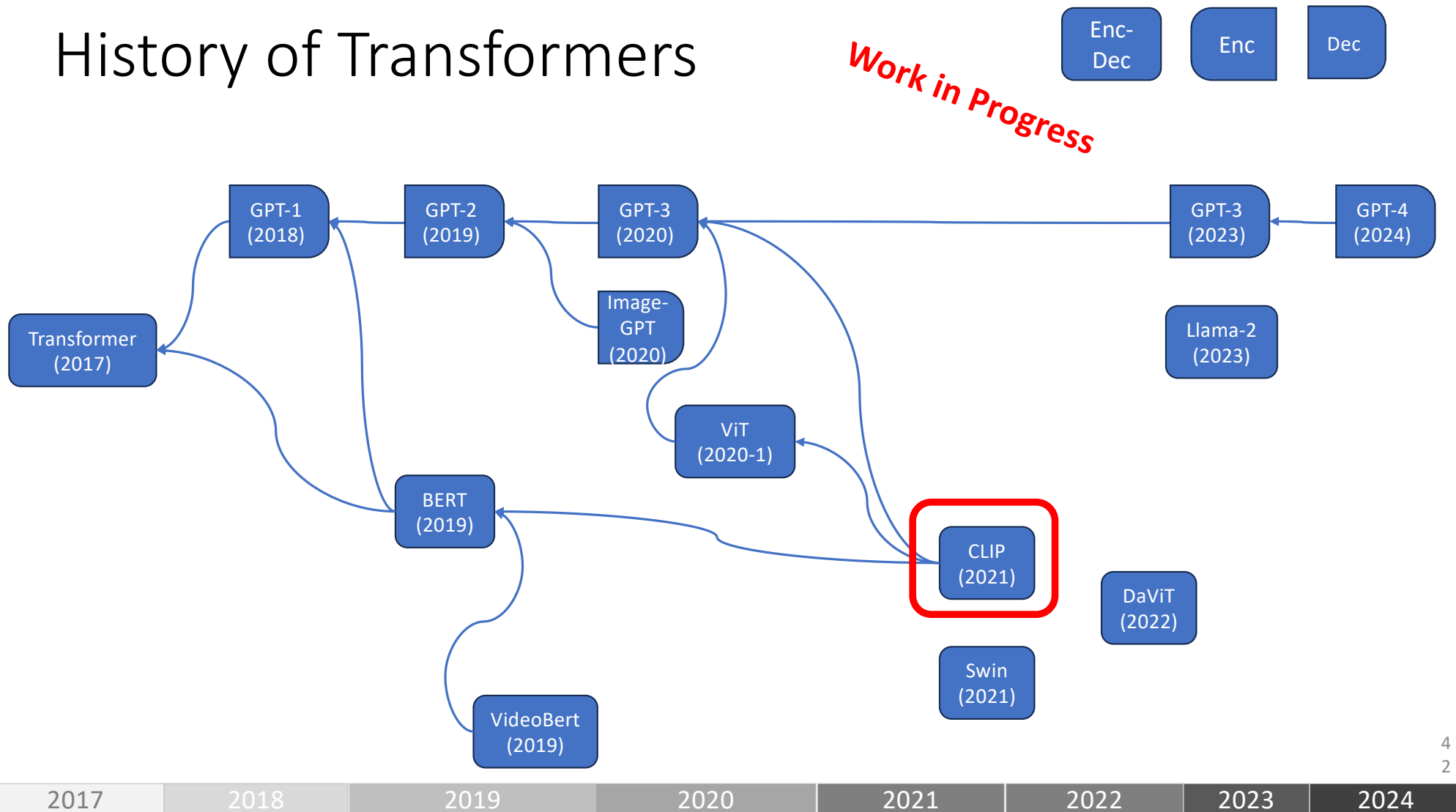
# Scaling Vision Transformers (2022)

- Explore scaling up and down
- Achieves new state-of-the-art on ImageNet top-1: 90.45% *with 2B parameter model*

- **Significantly scaled up model size and training data**, leading to new state-of-the-art results. While the original ViT paper demonstrated the potential of Transformers for vision by training models up to the "Huge" variant (632M parameters), "Scaling Vision Transformers" successfully trained a **two-billion parameter model (ViT-G/14)**, leveraging a much larger dataset (JFT-3B) and achieving **substantially improved top-1 accuracy on ImageNet (90.45% vs. 88.55%)**.
- **Enhanced few-shot transfer learning capabilities** through improved training techniques. "Scaling Vision Transformers" discovered that applying **very strong L2 regularization specifically to the final linear prediction layer** significantly boosts few-shot performance. This resulted in a remarkable **84.86% top-1 accuracy on ImageNet with only 10 examples per class**, a substantial improvement over the few-shot results presented in the original ViT paper.
- **Improved memory efficiency and training methodologies** to enable the scaling of larger models. "Scaling Vision Transformers" addressed memory limitations by **exploring alternatives to the [class] token like Multihead Attention Pooling (MAP)** and by utilizing **memory-efficient optimizers like a modified AdaFactor**. These modifications, along with other training recipe improvements, facilitated the training of much larger ViT models that were previously infeasible.

X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling Vision Transformers," presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 12104–12113. Accessed: Mar. 18, 2024.

# History of Transformers



# CLIP (2021) – Contrastive Language Image Pretraining

- Learn directly from raw text about images
- Created a new 400m (image, text) pair dataset called WebImageText (WIT) scraped from the internet
- “Simple” pre-training task:
  - Predict which caption goes with which image from scratch on a dataset of 400 million (image, text) pairs
  - Efficient and scalable
  - Learn state-of-the-art image representations from scratch
- Zero-shot transfer to many image classification datasets
- Shows promise for zero-shot transfer for other tasks: e.g. OCR, facial expression recognition, ...

A. Radford *et al.*, “Learning Transferable Visual Models From Natural Language Supervision,” in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, Jul. 2021, pp. 8748–8763.  
<https://proceedings.mlr.press/v139/radford21a.html>

# CLIP (2021) – Contrastive Language Image Pretraining

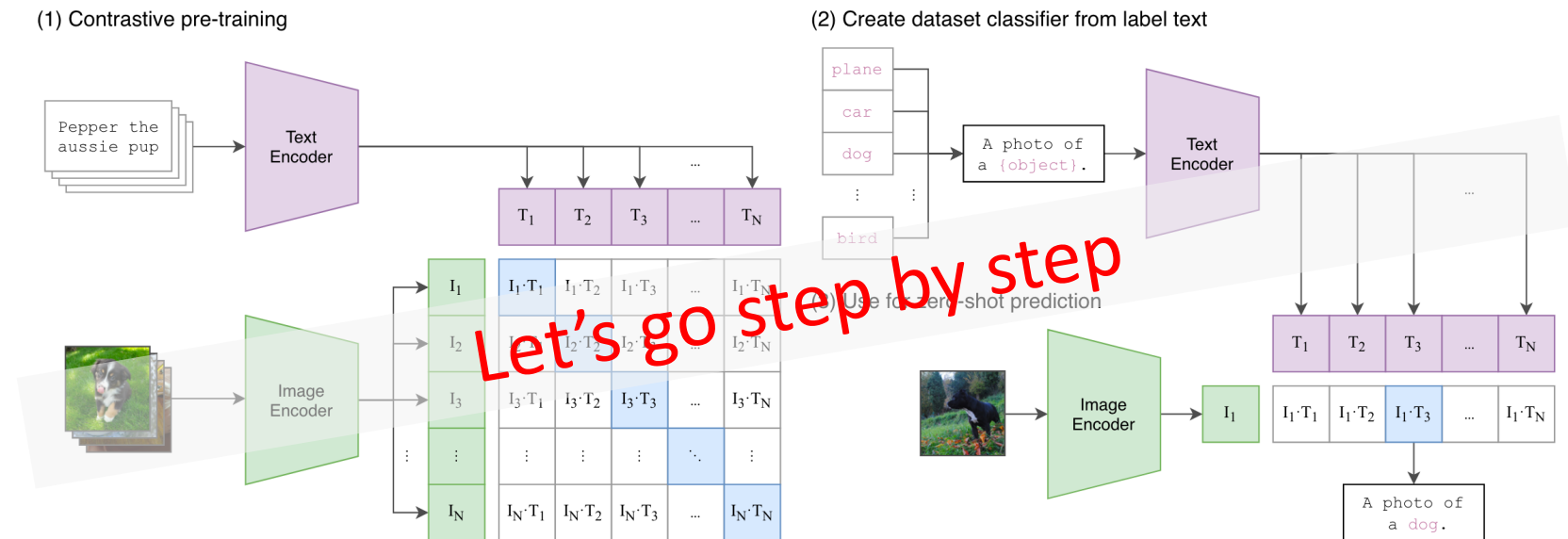
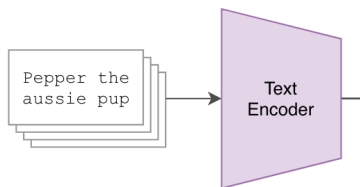


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the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

## CLIP (2021) – Text Encoder



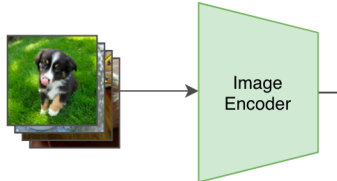
### Embedding

- lower-cased byte pair encoding (BPE)
- bracketed with `[SOS]` and `[EOS]` tokens

### Transformer

- 12-layer
- 512-wide
- 8 attention heads

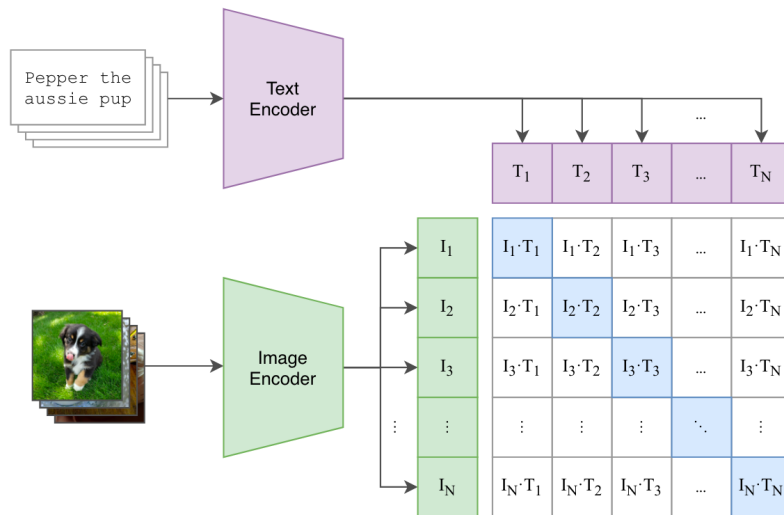
## CLIP (2021) – Image Encoder



- Trained and compared 5 ResNets and 3 vision transformers
  - ResNet50, ResNet101, RN50x4, x16, x64
  - ViT-B/32, ViT-B/16 and ViT-L/14
- Best model: ViT-L/14@336px
  - e.g. ViT-Large with 336×336 pixel resolution and 14×14 patch resolution
- Found vision transformers ~3x more compute efficient than CLIP ResNets
  - RN50x64 took 18 days to on 592 V100 GPUs
  - ViT took 12 days on 256 V100 GPUs

# CLIP (2021) – Contrastive Language Image Pretraining

(1) Contrastive pre-training

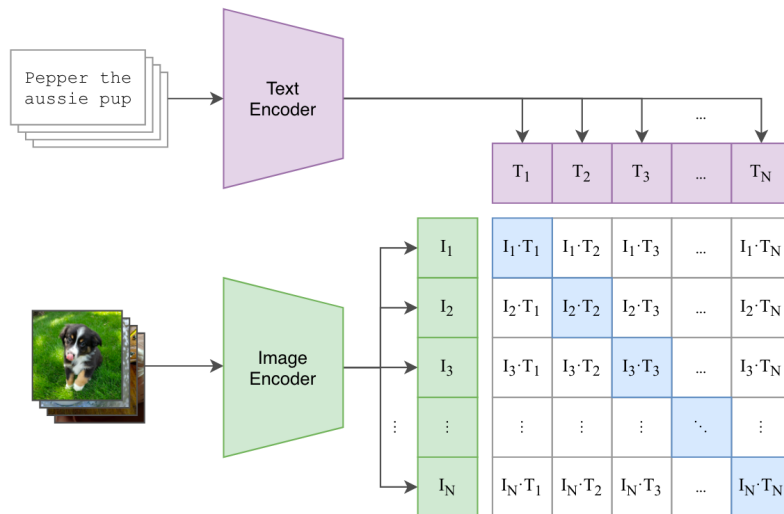


```
# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l] - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t - learned temperature parameter
```

- **Input Shapes**
  - $I[n, h, w, c]$  – A minibatch of images. Here,  $n$  is the batch size,  $h$  is the image height,  $w$  is the image width, and  $c$  is the number of color channels.
  - $T[n, l]$  – A minibatch of  $n$  texts.  $l$  is the sequence length for each text.
- **Projection Matrices**
  - $W_i[d_i, d_e]$  – A learned projection matrix that maps the image features (of dimension  $d_i$ ) into a joint embedding space of dimension  $d_e$ .
  - $W_t[d_t, d_e]$  – A learned projection matrix that maps the text features (of dimension  $d_t$ ) into the same joint embedding space.
- **Temperature Parameter**
  - $t$  – A learned scalar that controls the sharpness of the softmax (through scaling the logits). Its exponential,  $\exp(t)$ , is used to scale the cosine similarity scores.

# CLIP (2021) – Feature Extraction

(1) Contrastive pre-training



```
# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l] - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t - learned temperature parameter
```

```
# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T) #[n, d_t]
```

- `I_f = image_encoder(I)`

The image encoder processes the input batch of images  $I$  and outputs feature representations  $I_f$  for each image. The output shape is  $[n, d_i]$ , where each image is represented by a feature vector of dimension  $d_i$ .

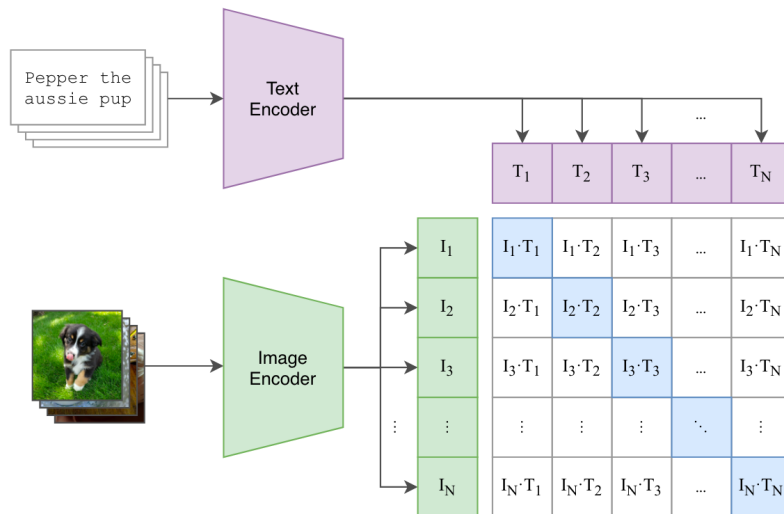
- `T_f = text_encoder(T)`

The text encoder processes the input batch of texts  $T$  and outputs feature representations  $T_f$  for each text. The output shape is  $[n, d_t]$ , where each text is represented by a feature vector of dimension  $d_t$ .



# CLIP (2021) – Multimodal Embeddings

## (1) Contrastive pre-training



```
# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l] - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t - learned temperature parameter
```

```
# extract feature representations of each modality
```

```
I_f = image_encoder(I) #[n, d_i]
```

```
T_f = text_encoder(T) #[n, d_t]
```

```
# joint multimodal embedding [n, d_e]
```

```
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
```

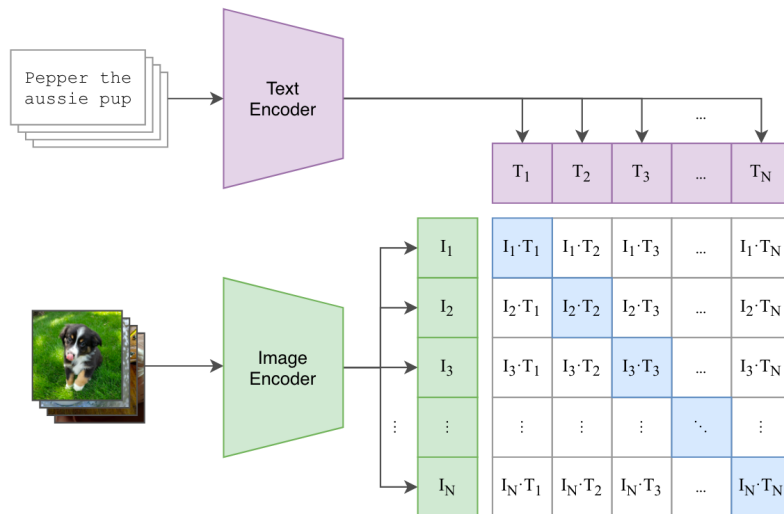
```
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)
```

- `I_e = l2_normalize(np.dot(I_f, W_i), axis=1)`
  - The image features `I_f` are projected into the common embedding space using the matrix `W_i`. This is done via a dot product, converting each feature from dimension `d_i` to `d_e`.
  - The resulting vectors are then L2-normalized (i.e., scaled to have unit length) along each row (`axis=1`). This normalization is key for cosine similarity computation.
- `T_e = l2_normalize(np.dot(T_f, W_t), axis=1)`
  - Similarly, the text features `T_f` are projected into the joint embedding space using the matrix `W_t`.
  - The projected text vectors are also L2-normalized along each row.

After this step, both `I_e` and `T_e` have the shape `[n, d_e]` and lie in the same embedding space, making them directly comparable.

# CLIP (2021) – Pairwise Cosine Similarity

(1) Contrastive pre-training



```
# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l] - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T) #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)
```

Cosine Similarity:

$$\cos(\theta) = \frac{(A \cdot B)}{||A|| \times ||B||} = \begin{cases} 1, & \text{for } \theta = 0^\circ \\ 0, & \text{for } \theta = 90^\circ \end{cases}$$

**`np.dot(I_e, T_e.T)`**

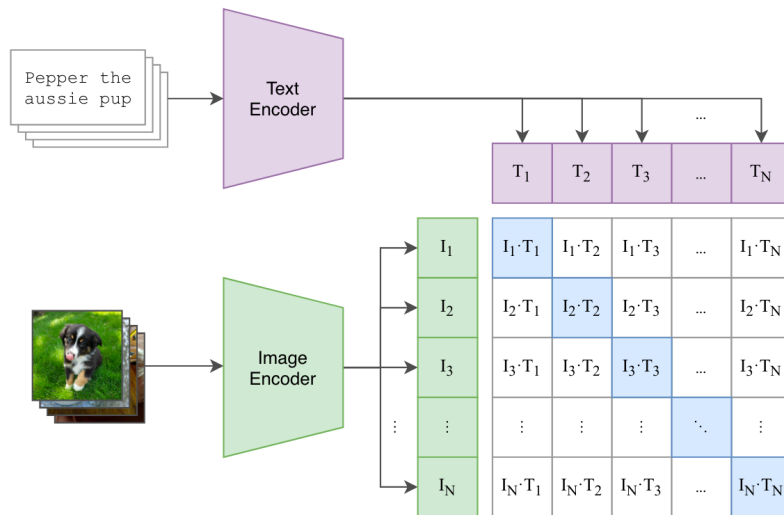
- Since  $I_e$  and  $T_e$  are both L2-normalized, their dot product yields the cosine similarity between each image and each text.
- The resulting matrix has shape  $[n, n]$  where each element  $(i, j)$  represents the similarity between image  $i$  and text  $j$ .

**`*np.exp(t)`**

- The cosine similarities are then scaled by the factor  $\exp(t)$ . The temperature parameter  $t$  controls the distribution of these similarity scores. A higher temperature (after exponentiation) makes the softmax distribution sharper, emphasizing the highest similarities.

# CLIP (2021) – Contrastive Language Image Pretraining

(1) Contrastive pre-training



```
labels = np.arange(n)
```

- This creates an array of integers `[0, 1, 2, ..., n-1]` which serves as the ground-truth labels. It assumes that the  $i$ -th image corresponds to the  $i$ -th text (i.e., they are aligned).

```
loss_i = cross_entropy_loss(logits, labels, axis=0)
```

- This computes the cross-entropy loss using the logits, **treating each image as a query against all texts**. Here, the correct text for each image is given by labels. The loss is computed along the columns ('axis=0').

```
loss_t = cross_entropy_loss(logits, labels, axis=1)
```

- Similarly, this computes the cross-entropy loss **treating each text as a query against all images**, with loss computed along the rows ('axis=1').

```
loss = (loss_i + loss_t) / 2
```

- The final loss is the average of the two losses, making it symmetric. This symmetric loss ensures that both image-to-text and text-to-image retrieval tasks are optimized simultaneously.

```
# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l] - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t - learned temperature parameter
```

```
# extract feature representations of each modality
```

```
I_f = image_encoder(I) #[n, d_i]
```

```
T_f = text_encoder(T) #[n, d_t]
```

```
# joint multimodal embedding [n, d_e]
```

```
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
```

```
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)
```

```
# scaled pairwise cosine similarities [n, n]
```

```
logits = np.dot(I_e, T_e.T) * np.exp(t)
```

```
# symmetric loss function
```

```
labels = np.arange(n)
```

```
loss_i = cross_entropy_loss(logits, labels, axis=0)
```

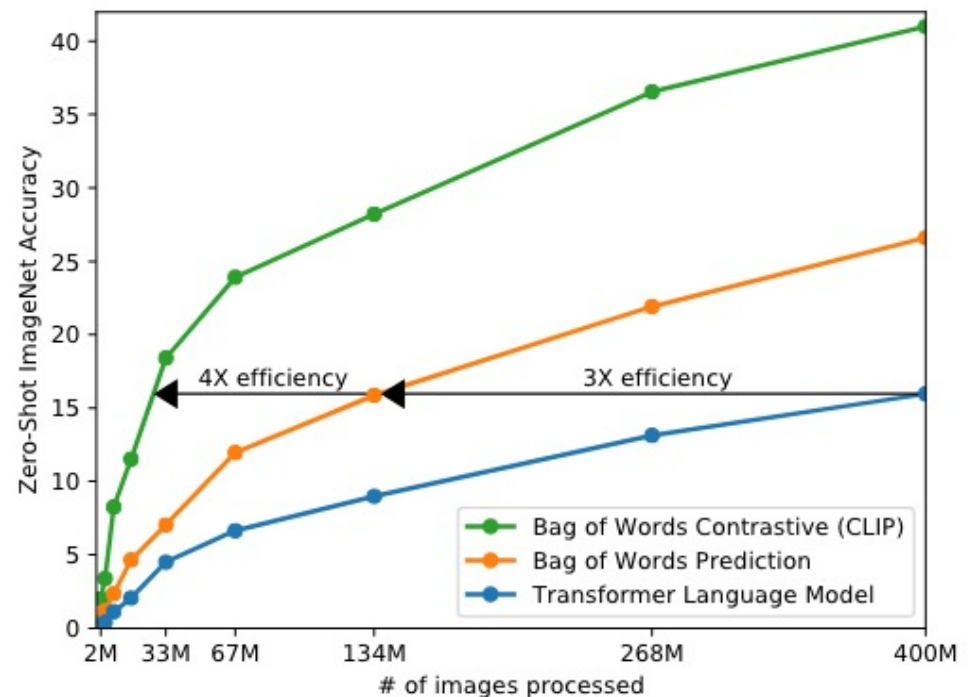
```
loss_t = cross_entropy_loss(logits, labels, axis=1)
```

```
loss = (loss_i + loss_t) / 2
```

## CLIP (2021) – Contrastive Loss

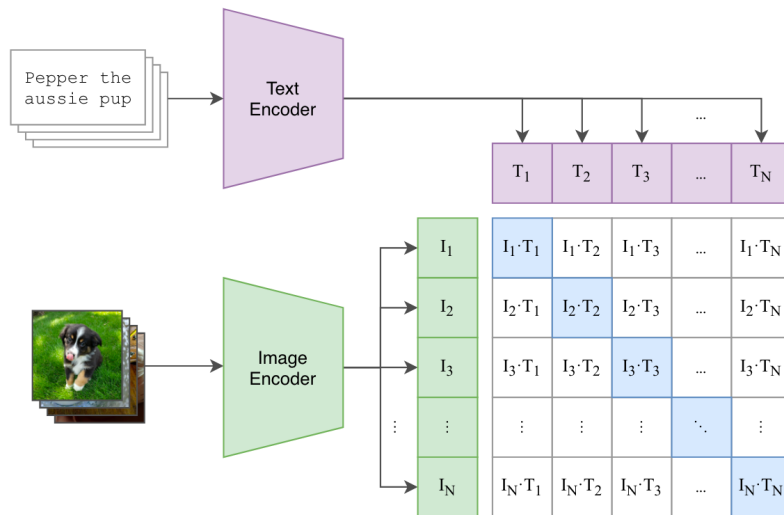
- Initially tried to train to predict caption of image (blue curve)
- bag-of-words encoding of same text is 3X more efficient (orange) curve
- Contrastive Objective improved another 4X (green curve)

Contrastive Loss: Maximize cosine similarity measure between matching (image, text) pairs and simultaneously minimize similarity between non-matching pairs

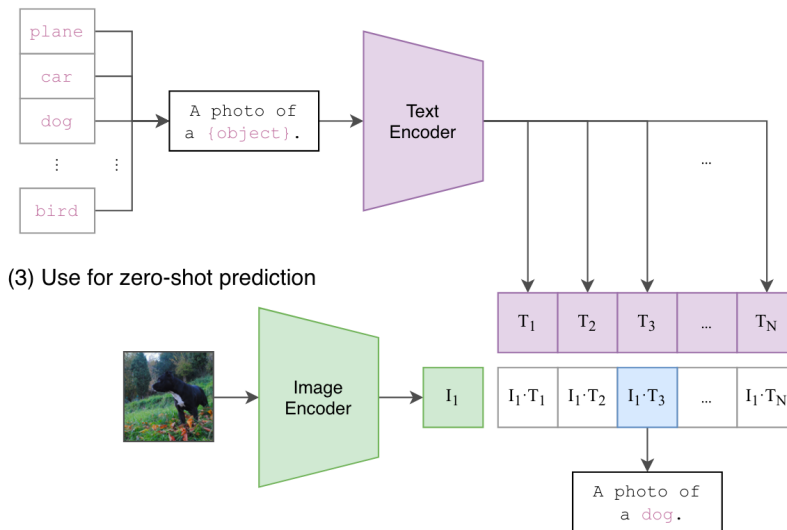


# CLIP (2021) – Zero-Shot Image Classification

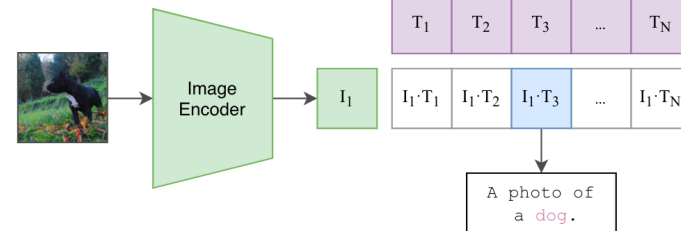
(1) Contrastive pre-training



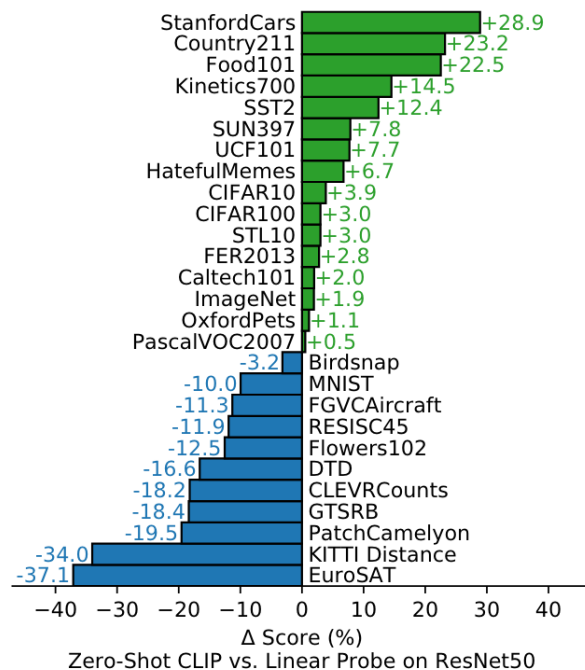
(2) Create dataset classifier from label text



(3) Use for zero-shot prediction



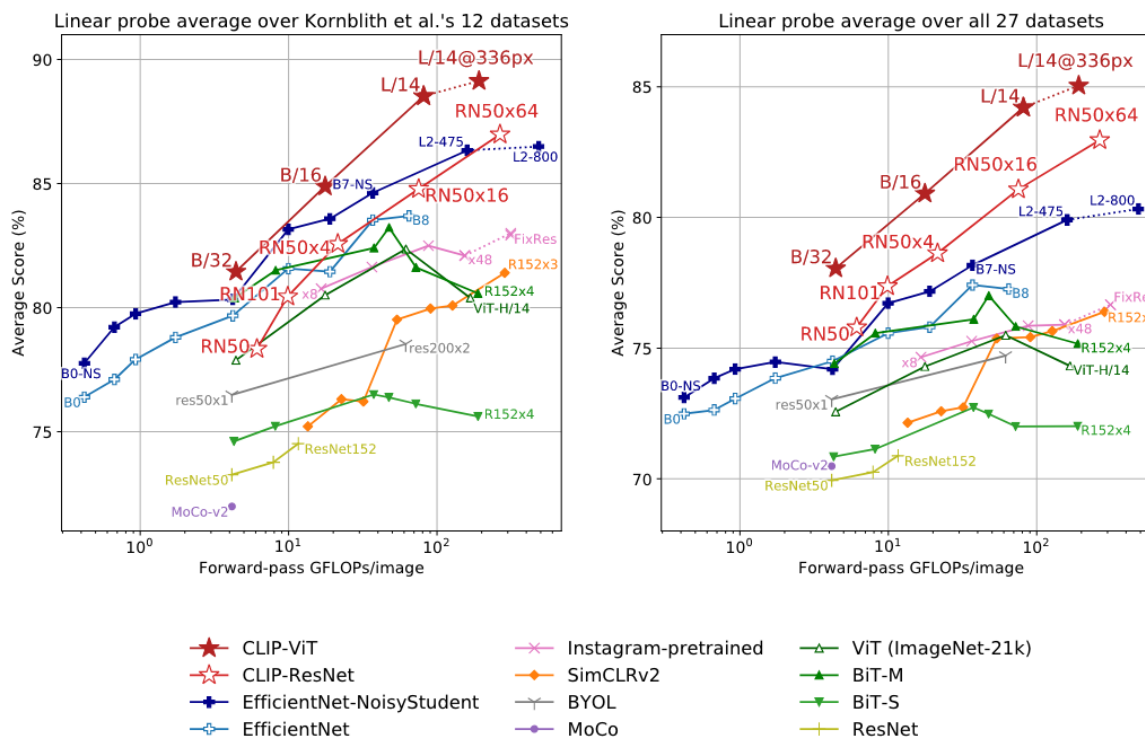
## CLIP (2021) – Zero-Shot Image Classification



- Evaluated across 27 datasets
- Compared to ResNet50 trained in supervised manner
- Beat ResNet50 on 16 of the 27
- Produced new SoTA on STL10 (99.3%)

*Figure 4. Zero-shot CLIP is competitive with a fully supervised baseline.* Across a 27 dataset eval suite, a zero-shot CLIP classifier outperforms a fully supervised linear classifier fitted on ResNet50 features on 16 datasets, including ImageNet.

# CLIP (2021) – Compute Efficiency



Scores are averaged over 12 datasets studied by Kornblith et al. (2019).

Scores are averaged over 27 datasets that contain a wider variety of distributions. Dotted lines indicate models fine-tuned or evaluated on images at a higher-resolution than pre-training.

# CLIP(2021) – Zero-Shot Classification Examples




**Food101**  
**guacamole (90.1%)** Ranked 1 out of 101 labels



- ✓ a photo of **guacamole**, a type of food.
- ✗ a photo of ceviche, a type of food.
- ✗ a photo of edamame, a type of food.
- ✗ a photo of tuna tartare, a type of food.
- ✗ a photo of hummus, a type of food.


**SUN397**  
**television studio (90.2%)** Ranked 1 out of 397 labels



- ✓ a photo of a **television studio**.
- ✗ a photo of a podium indoor.
- ✗ a photo of a conference room.
- ✗ a photo of a lecture room.
- ✗ a photo of a control room.

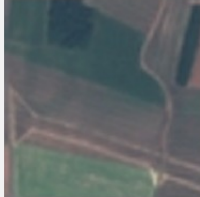


**Youtube-BB**  
**airplane, person (89.0%)** Ranked 1 out of 23 labels



- ✓ a photo of a **airplane**.
- ✗ a photo of a bird.
- ✗ a photo of a bear.
- ✗ a photo of a giraffe.
- ✗ a photo of a car.

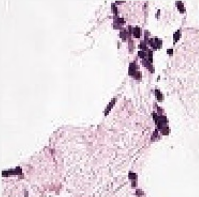
**EuroSAT**  
**annual crop land (46.5%)** Ranked 4 out of 10 labels



- ✗ a centered satellite photo of permanent crop land.
- ✗ a centered satellite photo of pasture land.
- ✗ a centered satellite photo of highway or road.
- ✓ a centered satellite photo of **annual crop land**.
- ✗ a centered satellite photo of brushland or shrubland.



**PatchCamelyon (PCam)**  
**healthy lymph node tissue (77.2%)** Ranked 2 out of 2 labels



- ✗ this is a photo of lymph node tumor tissue
- ✓ this is a photo of **healthy lymph node tissue**

**ImageNet-A (Adversarial)**  
**lynx (47.9%)** Ranked 5 out of 200 labels

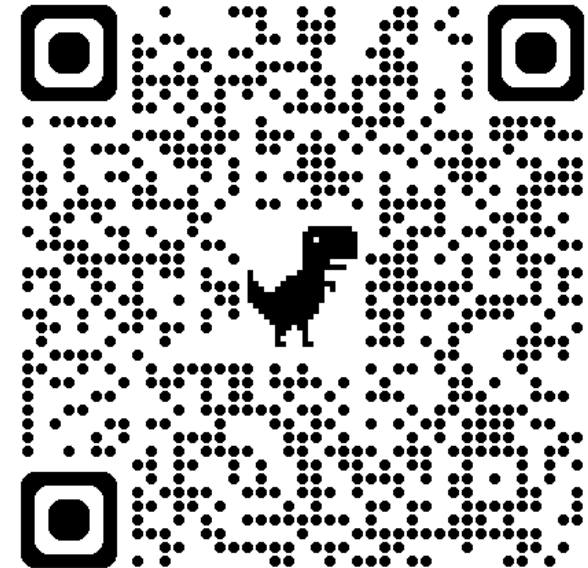


- ✗ a photo of a fox squirrel.
- ✗ a photo of a mongoose.
- ✗ a photo of a skunk.
- ✗ a photo of a red fox.
- ✓ a photo of a **lynx**.





Thank You



[Link](#)