Perceiver IO: A General Architecture for Structured Inputs & Outputs

by DeepMind

GLAMI AI
Radek Bartyzal
7. 9. 2021

Motivation

Current state:

- NN architectures usually differ based on the input modality = audio, image, video, …
- e.g. CNNs and Visual Transformers rely on image specific locality assumption
    - => convolutions, splitting the image into a grid
- Perceiver can only predicts logits

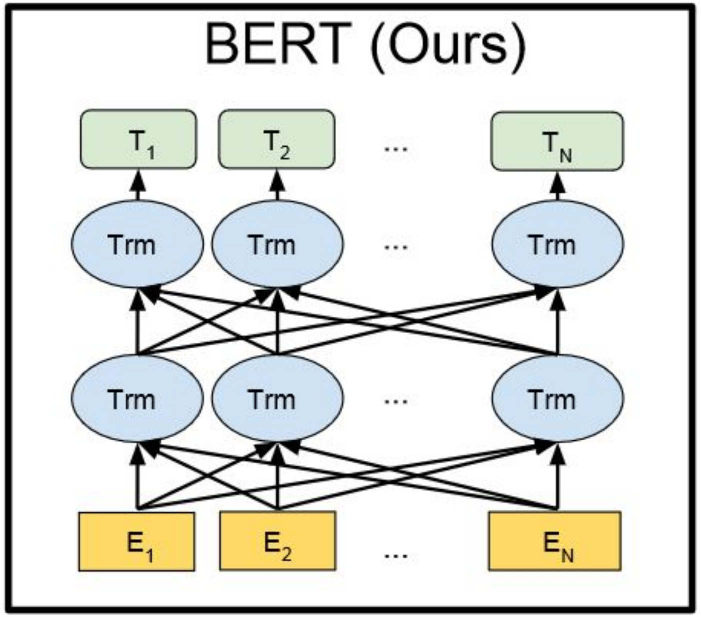
Author’s goal:

- create competitive architecture without relying on these assumptions
    - => attend to the individual pixels in an image
    - => use same architecture for audio, video, 3D point cloud
- enable larger inputs to Transformers
- enable any output

Regular Transformers

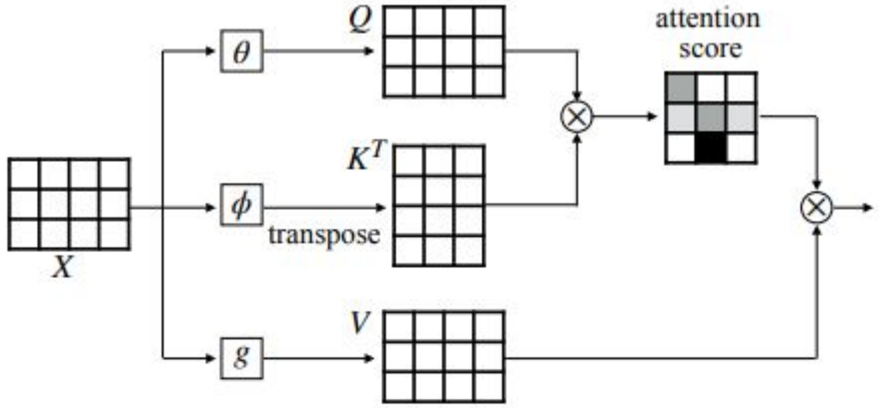
- transform a set of tokens into another set of tokens of same length = 1 transformer layer

- core of the transformer layer = self-attention



Self Attention

- M input tokens = e.g. word embeddings
- from each input token, create Query, Key, Value vectors
- dot product Query * Key vectors => attention matrix of MxM
- dot product Att matrix * Value vectors => new set of token vectors
- => O(M^2) space, O(M^2 * d) time

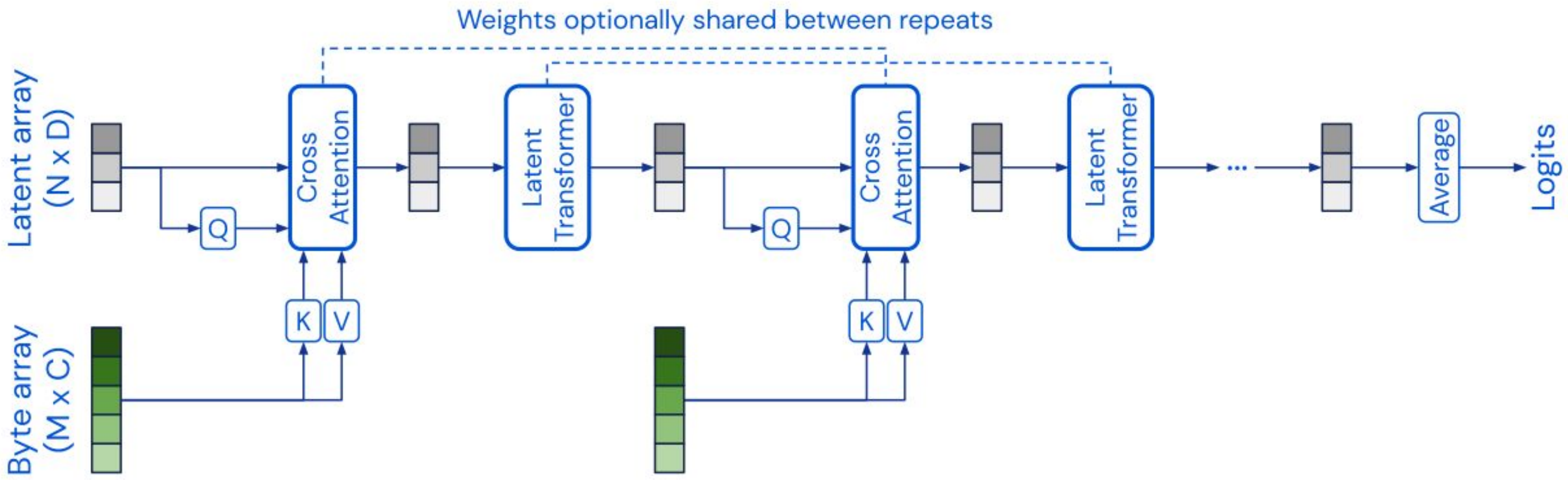


Perceiver architecture

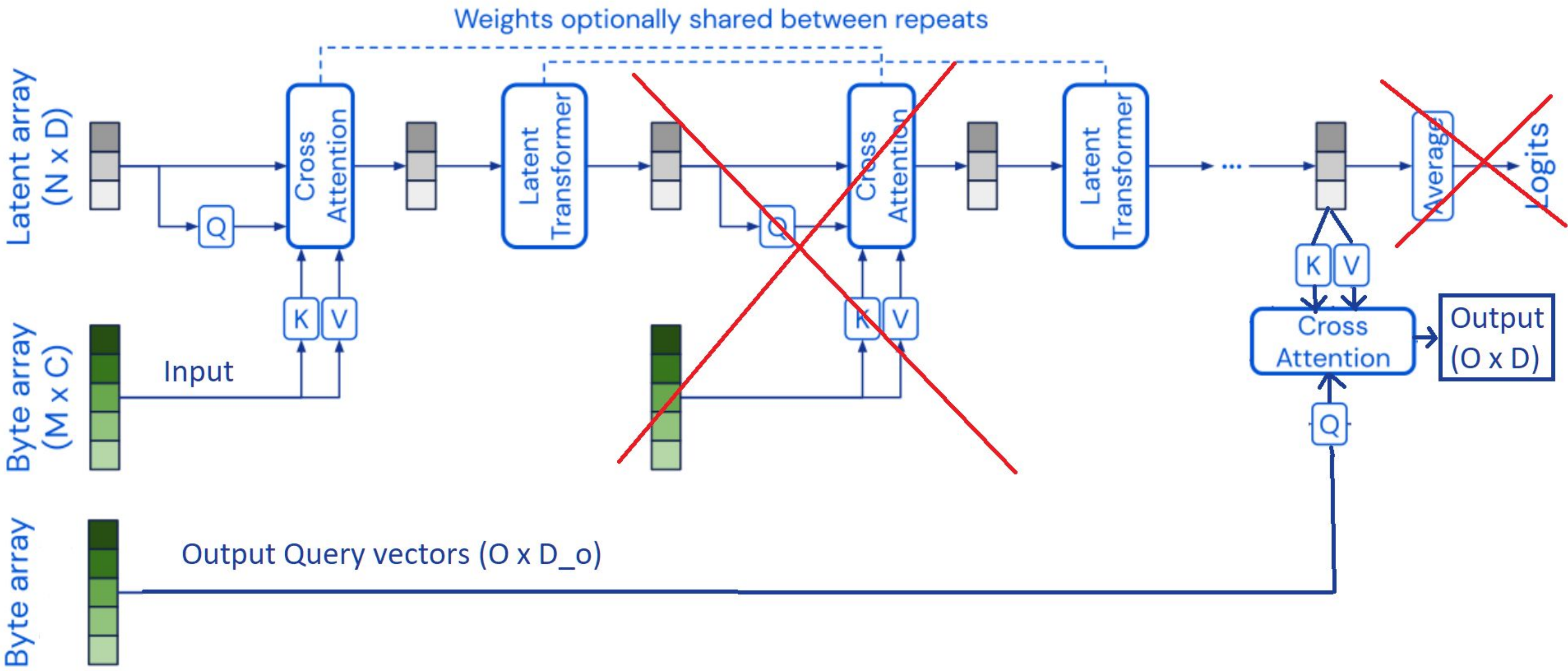
- 224x224 image = 50 000 pixels = M
- => impossible to calculate self-attention with O(50000*50000)
- => use **cross-attention** to inject the input information into the latent representation
- => attention from N latent vectors to M input tokens = **O(N*M)**, N << M

- follow the cross-attention by regular self-attention on the latent space = O(N*N)
- repeat blocks of [cross-attention->regular-latent-attention]

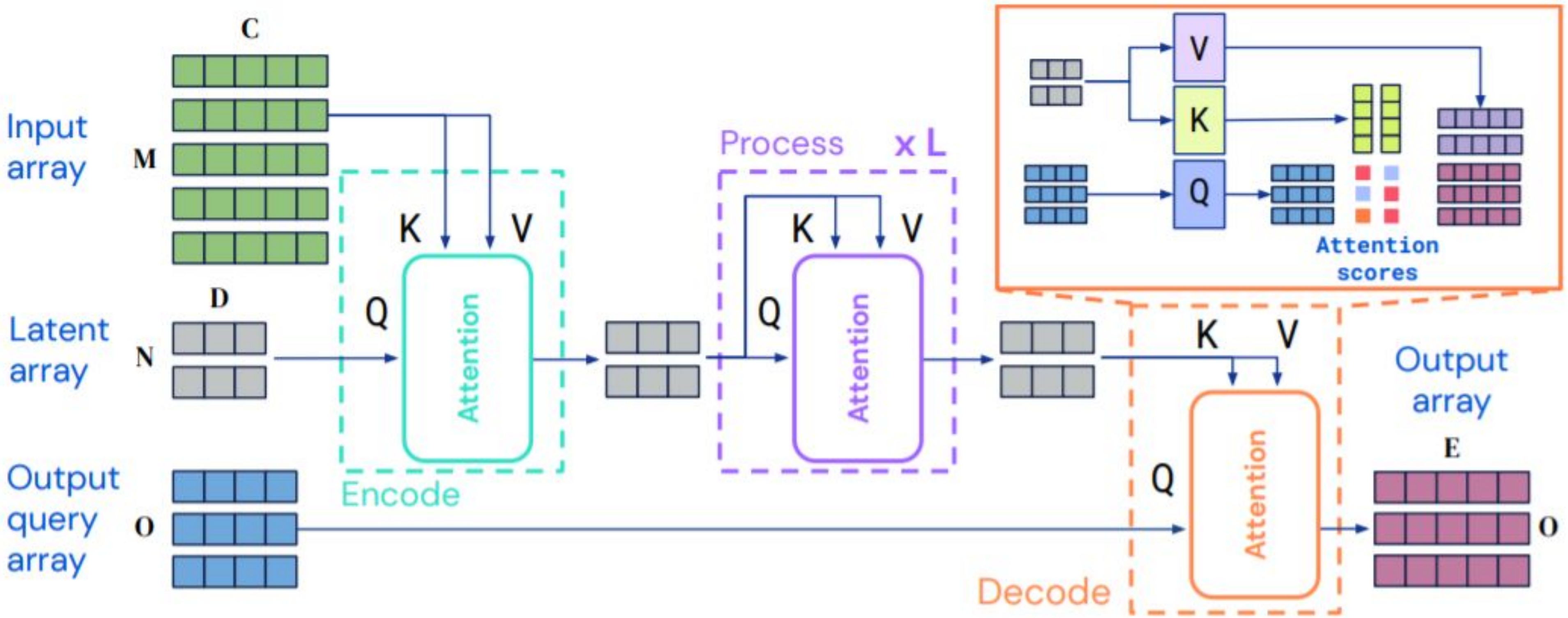
Perceiver architecture



Perceiver IO



Perceiver IO



Perceiver IO

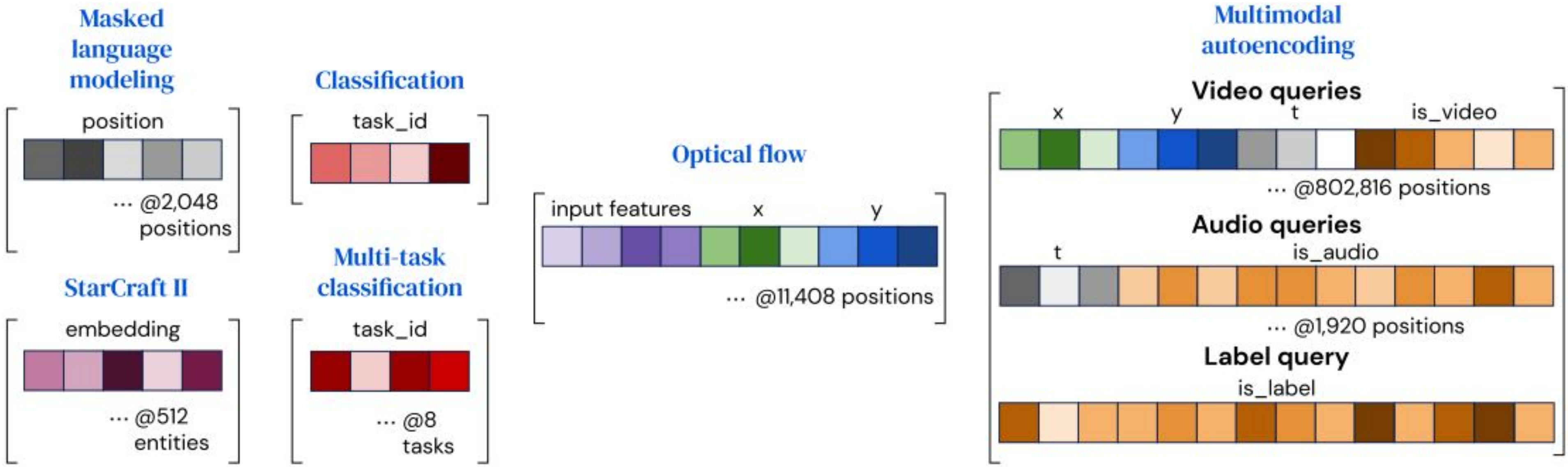
- autoencoder like structure
- encode = cross-attention of: input -> latent
- decode = cross-attention of: latent -> output
- don’t repeatedly pass in input = does not help that much
- => not a RNN anymore :(

- positional encoding = domain specific = hand crafted
  - 1D = sequence = words
  - 2D = image
  - 3D = 3D point cloud
  - 4D = video

Output Query vectors = decoder vectors

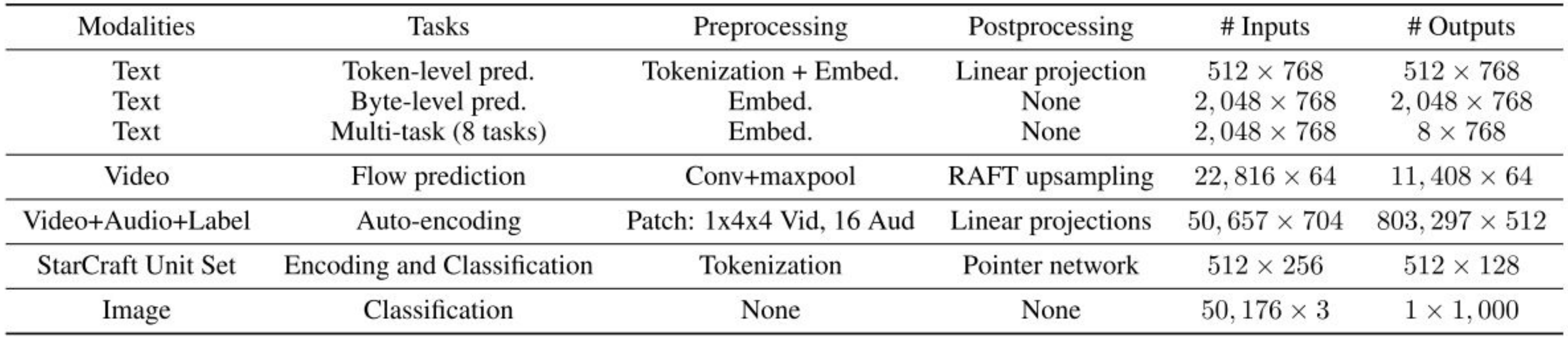
- Can be:
  - learned
  - hand crafted
  - simple function of the input
- number of decoder vectors = number of outputs = dynamic
- output dimension is fixed = task specific = e.g. 1000 for imagenet
- decode query vectors = task specific = hand crafted **structure**
  - simple classification => single learned embedding = token = query vector
  - word prediction => position in sentence + <predict word task token>
  - optical flow of pixel => x,y coords of pixel + <optical flow task token>
  - Starcraft unit information => embedding of unit description
  - task specification token is needed only for multi-task models

Decoder vectors



- task_id = single learned embedding is enough for class. task = imagenet
- position = pos. embedding as query which will return predicted word distribution
- is_label = <learned token of give me label logits = 1000 dim for imagenet>

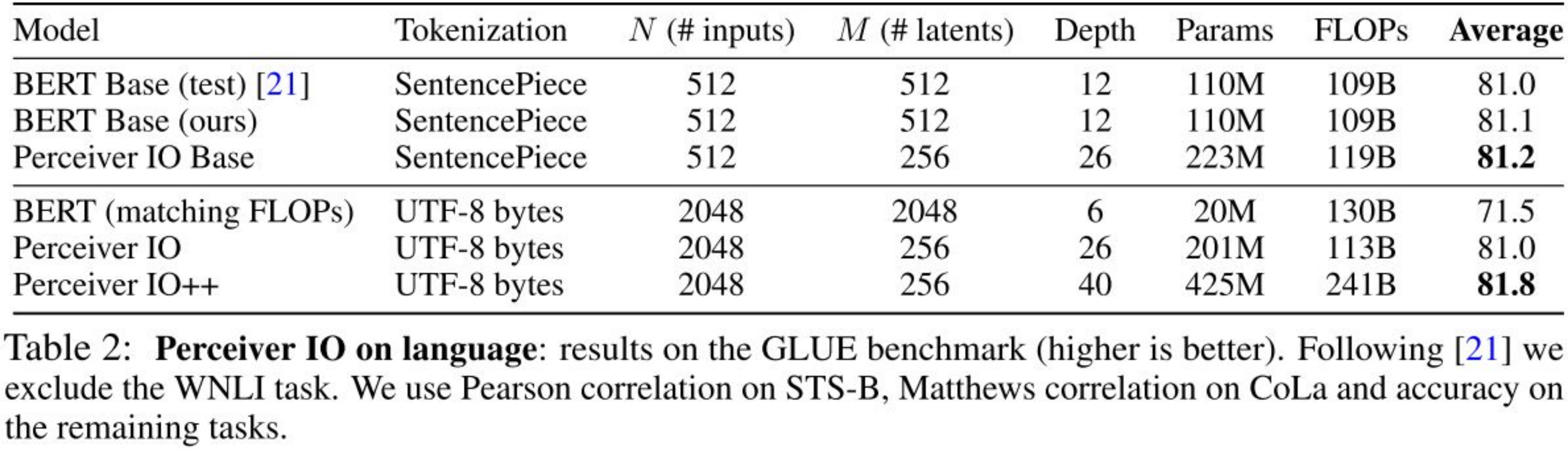
Tasks



Modalities	Tasks	Preprocessing	Postprocessing	# Inputs	# Outputs
Text	Token-level pred.	Tokenization + Embed.	Linear projection	512×768	512×768
Text	Byte-level pred.	Embed.	None	$2,048 \times 768$	$2,048 \times 768$
Text	Multi-task (8 tasks)	Embed.	None	$2,048 \times 768$	8×768
Video	Flow prediction	Conv+maxpool	RAFT upsampling	$22,816 \times 64$	$11,408 \times 64$
Video+Audio+Label	Auto-encoding	Patch: 1x4x4 Vid, 16 Aud	Linear projections	$50,657 \times 704$	$803,297 \times 512$
StarCraft Unit Set	Encoding and Classification	Tokenization	Pointer network	512×256	512×128
Image	Classification	None	None	$50,176 \times 3$	$1 \times 1,000$

Table 1: We give details of each of the tasks we use to evaluate Perceiver IO here. The positional and task embeddings appended to inputs for each case are listed in Appendix Table 7.

GLUE

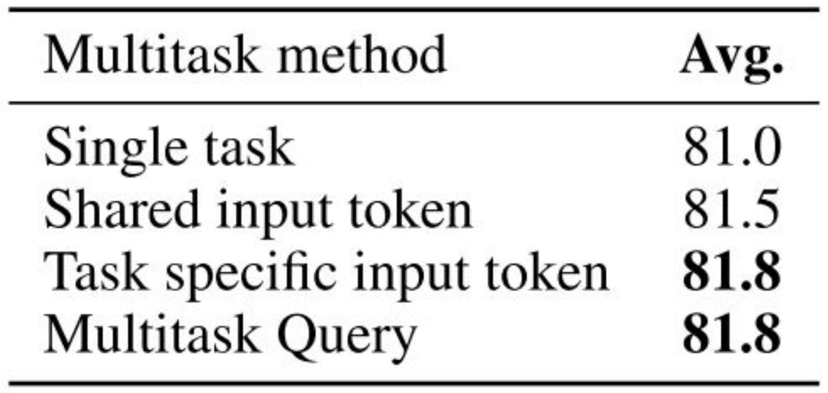


Model	Tokenization	N (# inputs)	M (# latents)	Depth	Params	FLOPs	**Average**
BERT Base (test) [21]	SentencePiece	512	512	12	110M	109B	81.0
BERT Base (ours)	SentencePiece	512	512	12	110M	109B	81.1
Perceiver IO Base	SentencePiece	512	256	26	223M	119B	**81.2**
BERT (matching FLOPs)	UTF-8 bytes	2048	2048	6	20M	130B	71.5
Perceiver IO	UTF-8 bytes	2048	256	26	201M	113B	81.0
Perceiver IO++	UTF-8 bytes	2048	256	40	425M	241B	**81.8**

Table 2: **Perceiver IO on language**: results on the GLUE benchmark (higher is better). Following [21] we exclude the WNLI task. We use Pearson correlation on STS-B, Matthews correlation on CoLa and accuracy on the remaining tasks.

GLUE

- multitask = 1 model for all tasks
- input concat with special token:
  - 1 for all tasks = shared input token
  - 1 for each = task specific input token
  - = like CLS token = decoder vector
  - a task specific head, a 2-layer MLP, is applied to generate the output logits corresponding to each task
- multitask query = use decoder vectors specific to each task
  - no input token
  - no output 2-layer heads?

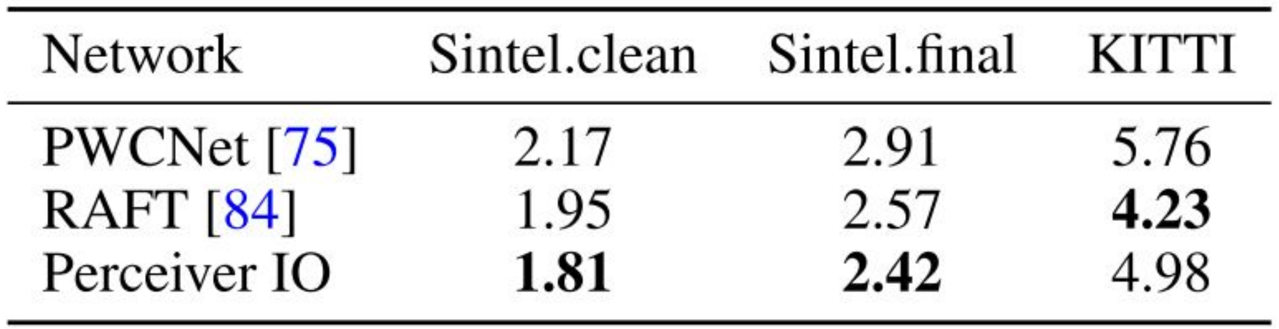


Multitask method	Avg.
Single task	81.0
Shared input token	81.5
Task specific input token	**81.8**
Multitask Query	**81.8**

Table 3: Multitask Perceiver IO. Results use the same metric as Table 2 (higher is better).

SOTA on Optical flow

- input: 2 images = consecutive frames from video
- output: displacement of each pixel between the 2 images
- PWCNet, RAFT = slow, hand-crafted architectures, very task specific



Network	Sintel.clean	Sintel.final	KITTI
PWCNet [75]	2.17	2.91	5.76
RAFT [84]	1.95	2.57	**4.23**
Perceiver IO	**1.81**	**2.42**	4.98

Table 4: Optical Flow evaluated on Sintel [10] and KITTI with average end-point error (EPE) (lower is better). Baselines are reported from [74].

Optical flow

Input:

- concat the frames along channels
- extract 3x3 area around each pixel
- => **3x3 x 3**=channels **x 2**=frames = 54 values for each pixel
- then concatenate a fixed position encoding to the 54 values

Decoder vector = same embedding as used for input

Audio Video Class reconstruction

- inputs:
  - padded with modality-specific embeddings
  - raw video pixels, audio, one-hot label
- outputs = raw reconstruction of video, audio, one-hot label

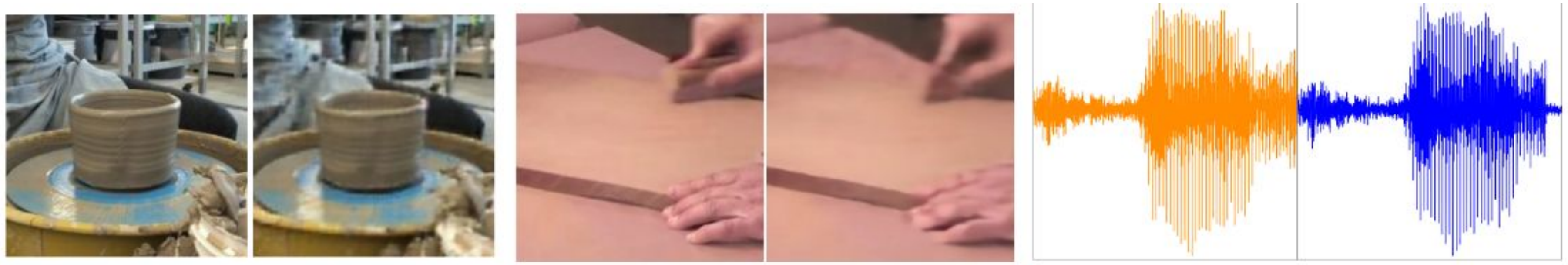
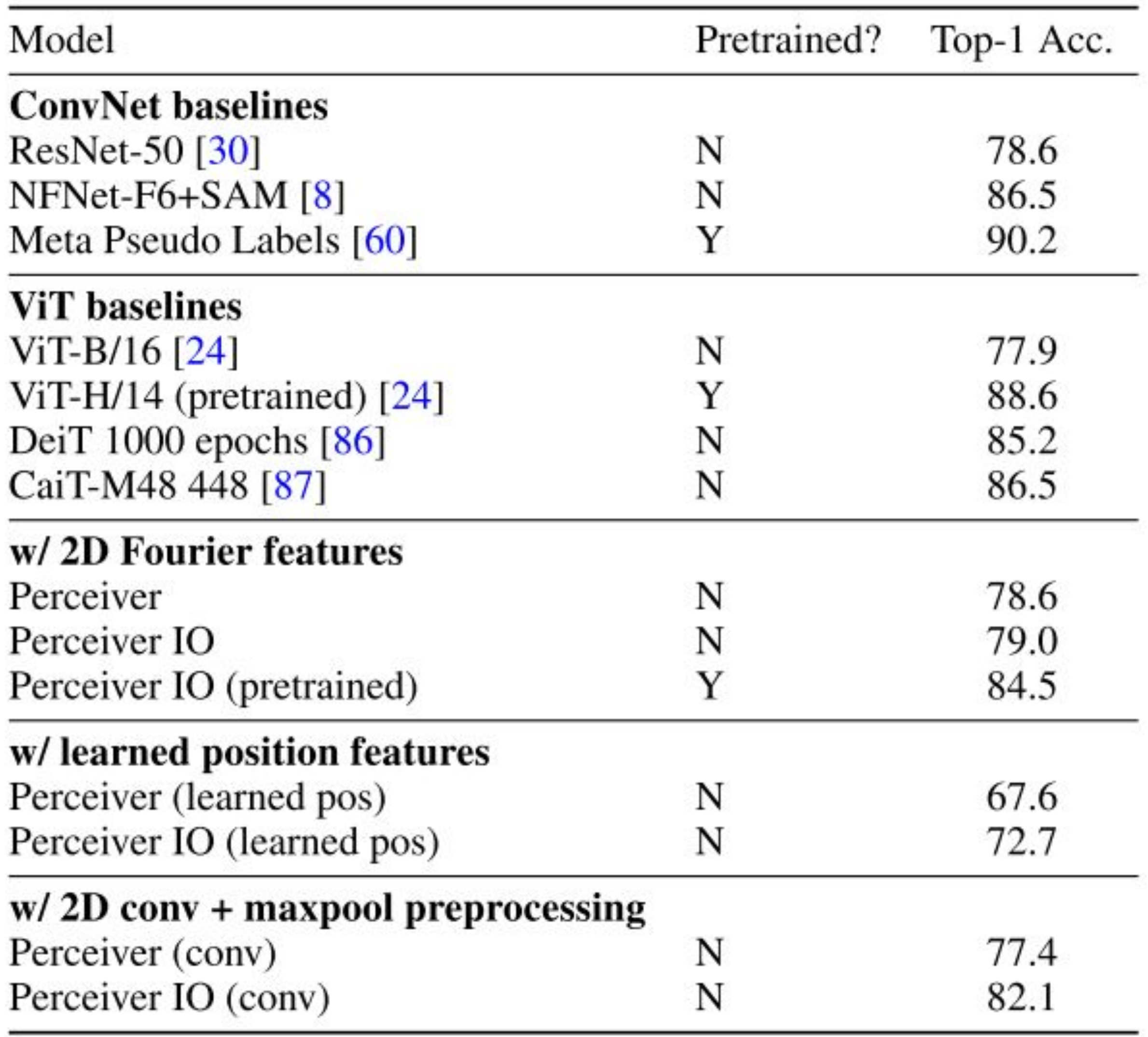


Figure 4: Audio-visual autoencoding with 88x compression. Side-by-side: inputs on left, reconstructions right (class label not shown).

Image classification

- learned pos embeddings = no hand-crafted information about the structure
- => pretty cool
- also permutation invariant



Model	Pretrained?	Top-1 Acc.
ConvNet baselines		
ResNet-50 [30]	N	78.6
NFNet-F6+SAM [8]	N	86.5
Meta Pseudo Labels [60]	Y	90.2
ViT baselines		
ViT-B/16 [24]	N	77.9
ViT-H/14 (pretrained) [24]	Y	88.6
DeiT 1000 epochs [86]	N	85.2
CaiT-M48 448 [87]	N	86.5
w/ 2D Fourier features		
Perceiver	N	78.6
Perceiver IO	N	79.0
Perceiver IO (pretrained)	Y	84.5
w/ learned position features		
Perceiver (learned pos)	N	67.6
Perceiver IO (learned pos)	N	72.7
w/ 2D conv + maxpool preprocessing		
Perceiver (conv)	N	77.4
Perceiver IO (conv)	N	82.1

Conclusion

Pros:

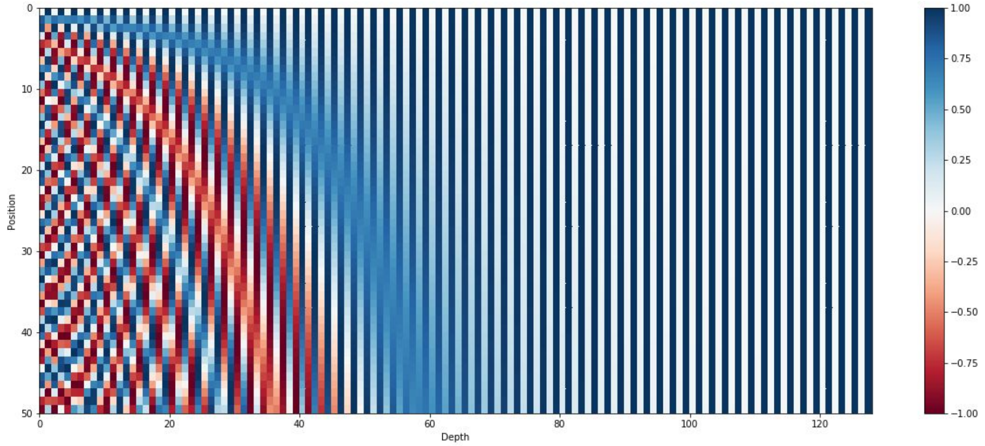
- latent space transformer
- linear scaling with input size
- multiple domains, tasks supported

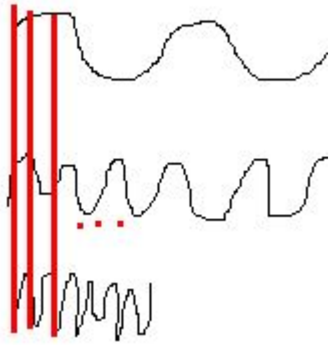
Cons:

- does not support generative modeling

Positional encodings = these are domain dependent

- spatial information is important, Transformer are invariant to it
- => use positional encodings = **Scalable Fourier features**
    - get k frequency bands, k-th band has frequency 2^k
    - sample positional encodings along those bands
    - concat these encodings to the input token vectors
    - 1 row on the figure below = 1 positional encoding





Sources

- Paper: https://arxiv.org/abs/2107.14795
- Pytorch: https://github.com/lucidrains/perceiver-pytorch
- JAX: https://github.com/deepmind/deepmind-research/tree/master/perceiver