Perceiver IO: A General Architecture for Structured Inputs & Outputs

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu,

David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier Hénaff,

Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, João Carreira

DeepMind

Abstract

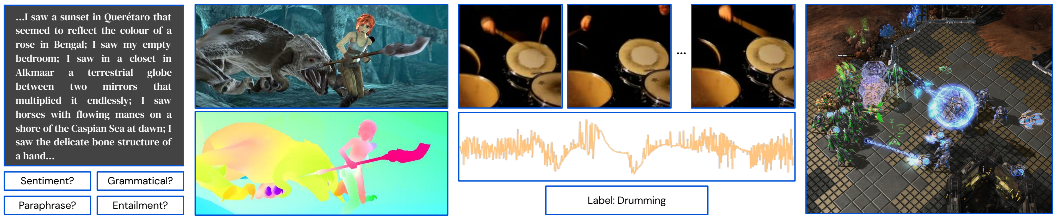
The recently-proposed Perceiver model obtains good results on several domains (images, audio, multimodal, point clouds) while scaling linearly in compute and memory with the input size. While the Perceiver supports many kinds of inputs, it can only produce very simple outputs such as class scores. Perceiver IO overcomes this limitation without sacrificing the original's appealing properties by learning to flexibly query the model's latent space to produce outputs of arbitrary size and semantics. Perceiver IO still decouples model depth from data size and still scales linearly with data size, but now with respect to both input and output sizes. The full Perceiver IO model achieves strong results on tasks with highly structured output spaces, such as natural language and visual understanding, StarCraft II, and multi-task and multi-modal domains. As highlights, Perceiver IO matches a Transformer-based BERT baseline on the GLUE language benchmark without the need for input tokenization and achieves state-of-the-art performance on Sintel optical flow estimation. Code: https://dpmd.ai/perceiver-code

1 Introduction

Humans and other animals have a remarkable ability to take in data from many sources, integrate it seamlessly, and deploy it flexibly in the service of a range of goals. Yet most machine learning research focuses on building bespoke systems to handle the stereotyped set of inputs and outputs associated with a single task. This is the case even for models that handle inputs or outputs of different modalities: a typical approach is to process each input independently with a deep, modality specific architecture (for example using a 2D ResNet [30] for vision and a Transformer [88] for language), integrate them afterwards using a third fusion network, and read the result out in a task-specific manner. The complexity of a system like this can grow dramatically as the inputs or outputs grow more diverse (e.g. [1, 89, 65]), and the shape and structure of a task's inputs and outputs may place strong constraints on how such a system processes data, making it difficult to adapt to new settings.

Is the development of problem-specific models for each new set of inputs and outputs unavoidable? Life would be drastically simpler if a single neural network architecture could handle a wide variety of both input modalities and output tasks. In this work, we propose such an architecture, with the ultimate goal of building a network that can easily integrate and transform arbitrary information for arbitrary tasks. Our starting point is the Perceiver [35], an architecture which has demonstrated a remarkable ability to handle data from many modalities with minimal changes to the network architecture. The Perceiver uses attention (building on Transformers [88]) to map inputs of a wide range of modalities

Preprint. Under review.



...I saw a sunset in Querétaro that seemed to reflect the colour of a rose in Bengal; I saw my empty bedroom; I saw in a closet in Alkmaar a terrestrial globe between two mirrors that multiplied it endlessly; I saw horses with flowing manes on a shore of the Caspian Sea at dawn; I saw the delicate bone structure of a hand...

Figure 1: The Perceiver IO architecture can be used on domains with a wide variety of input and output spaces, including multi-task language understanding, dense visual tasks like optical flow, hybrid dense/sparse multimodal tasks such as video+audio+class autoencoding, and tasks with symbolic outputs like StarCraft II.

to a fixed-size latent space, that is further processed by a deep, fully attentional network. This process decouples the bulk of the network's processing from the size and modality-specific details of the input, allowing it to scale to large and multimodal data.

However, the Perceiver can only handle simple output spaces like classification. Much of the complexity of real-world tasks comes from the variety, size, and structure of their *outputs*, and in this regard the original Perceiver can't be considered general purpose. In this work, we develop a mechanism for decoding structured outputs – language, optical flow fields, audiovisual sequences, symbolic unordered sets, etc. – directly from the Perceiver latent space, which allows the model to handle a host of new domains without sacrificing the benefits of deep, domain-agnostic processing. Our key insight is to produce each output by attending to the latent array using a specific *output query* associated with that particular output. For example if we wanted the model to predict optical flow on one particular pixel we could compose a query from the pixel's xy coordinates plus an optical flow task embedding: the model would then attend using the query and produce a single flow vector. As a result, our architecture can produce many outputs, each with arbitrary shape and structure, and yet the latent features in our architecture remain agnostic to the shape and structure of the outputs.

Perceiver IO does this using a fully attentional read-process-write architecture: inputs are encoded (read) to a latent space, the latent representation is refined (process) via many layers of processing, and the latent space is decoded (write) to produce outputs. This approach inherits the best features of both Transformers [88] – which leverage domain agnostic primitives for nonlocal processing of inputs – and the encoder-decoder architectures (e.g. [67, 57]) that are in widespread use in high-bandwidth domains – computer vision, multimodal processing, and elsewhere in machine learning. This approach allows us to decouple the size of elements used for the bulk of the computation (the latent) from the size of the input and output spaces, while making minimal assumptions about the spatial or locality structure of the input and output.

Our main technical contribution is Perceiver IO's decoding procedure. We use a cross-attention mechanism to map from latents to arbitrarily sized and structured outputs using a querying system that can flexibly specify the semantics needed for outputs on a wide range of domains, including dense and multitask settings. The design improvement that comes by incorporating this decoder with the original Perceiver allows Perceiver IO to serve as a drop-in replacement for a wide range of specialist networks currently in use on a set of challenging domains, while improving the architecture's performance on tasks like classification that could be handled by the original architecture.

We demonstrate the generality of the proposed model by tackling several challenging tasks, showing that Perceiver IO can be used to replace the Transformers used in both BERT [21] and AlphaStar [89]. At the same time, Perceiver IO produces state-of-the-art results on the Sintel optical flow benchmark [10], compelling results on multimodal (video, audio, label) auto-encoding in Kinetics [71], and results comparable to state-of-the-art models on ImageNet image classification [19]. In some cases, Perceiver IO allows us to simplify pipelines and remove domain-specific assumptions. For example, we process language without using tokenizers and fine-tune on multiple classification tasks simultaneously and without requiring `[CLS]` tokens (see Sec. 4.1), estimate optical flow without relying on explicit architectural features for multiscale correspondence (see Sec. 4.2), and perform image classification with no information about the 2D structure of images (see Sec. 4.5). Perceiver IO can be used as a standalone architecture, in conjunction with {pre, post}-processing steps, and as a part of sophisticated pipelines, in each case matching or beating strong baselines.

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.

2 Related Work

Complex output spaces have long been a focus of neural network research. Autoencoding [32], especially for images, was among the first attempts to provide a representation which can encode and reproduce its inputs. As hardware acceleration grew more powerful and neural nets proved themselves in image understanding [41, 101, 78], interest intensified: autoregressive models were developed that could take samples of handwriting and complete them [28], and new convolutional network designs led to good results in structured output spaces like semantic segmentation [25, 46, 67], pose estimation [85], detection [70], captioning [97], and optical flow [26]. At the same time, natural language applications research has made extensive progressive in capturing the structured nature of language, typically via autoregressive models [9, 16, 62, 76, 88] or context prediction [21, 59, 54].

Similar to our work, several models have been proposed to solve tasks in several domains (e.g. [2, 3, 38]), but typically across a fixed and predefined set of modalities by means of domain-specific networks. In vision, multi-task learning has become popular [22, 40, 56, 100]. Although single-task specialist networks remain dominant in the field, some models achieve generality in a restricted domain: e.g. Mask-RCNN [29] handles object detection, segmentation, and pose estimation. In language, training or evaluation on multiple tasks has also become common [15, 21, 50, 44, 63]. Several groups have demonstrated that Transformers (originally designed for language) can be used or adapted to non-language tasks (e.g. [13, 48]), but the limited scalability of Transformers restricts their usefulness as general-purpose architectures.

Several groups have proposed to use Transformer-like attention to manipulate the size of input and output arrays or to introduce bottlenecks in processing. Set Transformers [43] use a learned encoder query (which they call "inducing points") to map input sets to a latent space and learned decoder queries (which they call "seed vectors") to map latents to outputs (a process they call "pooling by multiheaded attention"). Set Transformers and related models (e.g. [27]) use this procedure to efficiently map set inputs to a simple, homogeneous output spaces on small-scale problems.[1] Our work uses attention over inputs and outputs of different sizes in part to produce an architecture that's more efficient than Transformers, and several other approaches have been proposed to solve this problem (e.g. [94, 93, 80, 5], and see [81]). Much of this work has been restricted to fairly small-scale problems and restricted to language; the focus of our work is not only developing efficient attention architectures but also developing an architecture that performs well in many settings with a wide range of inputs and outputs. Several other works use attention to process or refine latent spaces that interface with the input/output data using task- or domain-specific architectures [12, 45, 92]. Cross-attention itself is in widespread use as a mechanism for allowing information from a source to influence processing of a target domain of a different size or structure [18, 20, 53, 88, 64, 68, 34, 51]. Perceiver IO builds on this body of work – and on the original Perceiver [35] – to produce a simple, general purpose architecture that can be easily applied to many domains.

One application of Perceiver IO is byte-level language processing, which has concurrently been addressed by several other groups. [14] trains models on Unicode code points and shows results competitive with subword-based models on a multilingual question answering dataset. [83] trains on UTF-8 bytes directly by introducing a hand-designed module that is trained end-to-end to perform subword tokenization and produces results on-par with and sometimes better than subword-based models. [95] trains encoder-decoder T5 models on UTF-8 bytes directly and shows that making the encoder 3x deeper than the decoder leads to comparable performance with subword baselines.

[1] For a discussion of the relationship between Set Transformers and Perceivers see Sections 2 and A of [35].

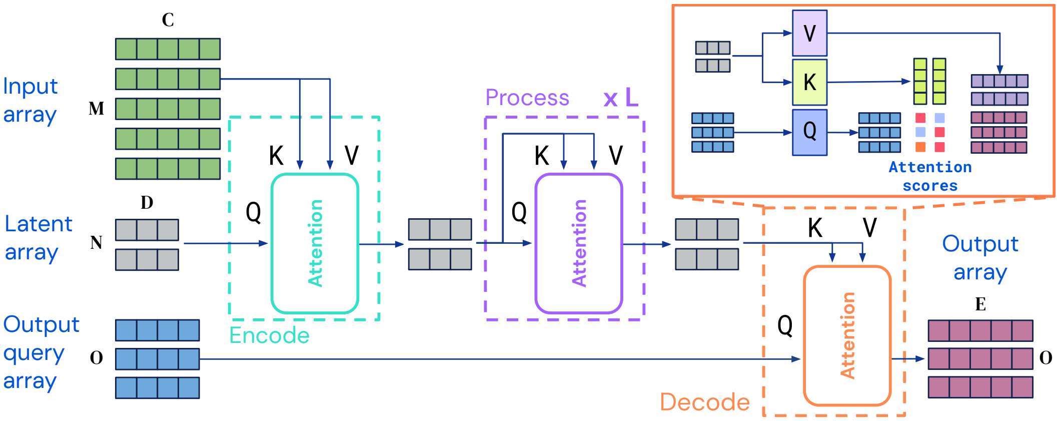


Figure 2: The Perceiver IO architecture. Perceiver IO maps arbitrary input arrays to arbitrary output arrays in a domain agnostic process. The bulk of the computation happens in a latent space whose size is typically smaller than the inputs and outputs, which makes the process computationally tractable even for very large inputs & outputs.

3 The Perceiver IO architecture

The Perceiver IO model builds on the Perceiver architecture [35], which achieved its cross-domain generality by assuming that its input is a simple 2D byte array: a set of elements (which might be pixels or patches in vision, characters or words in language, or some form of embedding, learned or otherwise), each described by a feature vector. The model then encodes information about the input array using a smaller number of latent feature vectors, using Transformer-style attention, followed by iterative processing and a final aggregation down to a category label.

Rather than output a single category, Perceiver IO aims to have the same level of generality with respect to its *outputs* as the Perceiver has with respect to its *inputs*: that is, it should produce arbitrary output arrays. The key insight is that we can predict each element of the output array using another attention module, by simply *querying* the latent array using a query feature vector unique to the desired output element. In other words, we define a query array with the same number of elements as the desired output. The queries may be hand-designed, learned embeddings, or a simple function of the input. They attend to the latents to yield an output array of the desired shape.

3.1 Encoding, processing, and decoding

Fig. 2 illustrates the Perceiver IO. In detail, we first **encode** by applying an attention module that maps input arrays $x \in \mathbb{R}^{M \times C}$ to arrays in a latent space $z \in \mathbb{R}^{N \times D}$. We next **process** the latents z by applying a series of modules that take in and return arrays in this latent space. Finally, we **decode** by applying an attention module that maps latent arrays to output arrays $y \in \mathbb{R}^{O \times E}$. M, C, O, and E are properties of the task data and can be very large (see Table 1), while N and D are hyperparameters and can be chosen to make model computation tractable. Following the design of the Perceiver, we implement each of the architecture's components using Transformer-style attention modules.

Each of these modules applies a global query-key-value (QKV) attention operation followed by a multi-layer perceptron (MLP). As usual in Transformer-style architectures, we apply the MLP independently to each element of the index dimension. Both encoder and decoder take in two input arrays, the first used as input to the module's key and value networks, and the second used as input to the module's query network. The module's output has the same index dimension (i.e. the same number of elements) as the query input: this is what allows the encoder and decoder module to produce outputs of different sizes.

The Perceiver IO architecture relies on the same primitives as Transformers: so why aren't Transformers all you need? The answer is that Transformers scale very poorly in both compute and memory [82]. A Transformer deploys attention modules homogeneously throughout its architecture, using its full input to generate queries and keys at every layer. As discussed in [35], this means each layer scales quadratically in compute and memory, which currently makes it impossible to apply Transformers

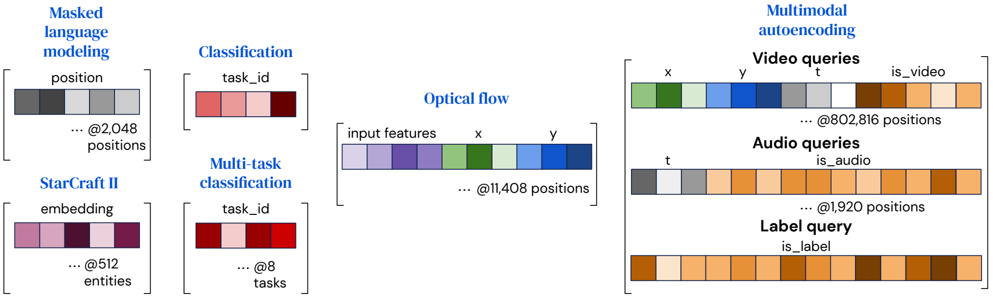


Figure 3: Perceiver IO uses queries constructed with output-specific features to produce outputs with different semantics. For settings with homogeneous outputs, like language, where each output point depends only on its position, a position embedding (learned or constructed) can be used. Input features for the target output can also be used to query, either alone (as for StarCraft II) or alongside position features (as for flow). For multi-{task, modality} settings we use a different embedding for each {task, modality} instead of each position. A single learned embedding suffices for simple classification tasks, like ImageNet. For tasks with heterogeneous outputs, like multimodal autoencoding, features that are specific to some queries (like position within a modality's data) can be combined with modality embeddings, which also serve to pad each embedding to a target length.

on high-dimensional data like images without some form of preprocessing. Even on domains like language where Transformers shine, preprocessing (e.g. tokenization) is often needed to scale beyond short input sequences. On the other hand, Perceiver IO uses attention non-homogeneously, first using it to map inputs to a latent space, then using it to process in that latent space, and finally using it to map to an output space. The resulting architecture has no quadratic dependence on the input or output size: encoder and decoder attention modules depend linearly on the input and output size (respectively), while latent attention is independent of both input and output sizes. Because of this structure, and the corresponding reduction in compute and memory requirements, Perceivers scale to much larger inputs and outputs. While Transformers are typically used in settings with inputs and outputs of at most a few thousand dimensions [9, 63], we show good results on domains with hundreds of thousands of input and output dimensions.

The encode and processing stages re-use the basic structure of the original Perceiver.[2] The encoder consists of a single attention module, where the queries are learned and have shape independent of the input data. Because the queries do not depend on the input, this module has complexity linear in the input size [35]. The processor is composed of a stack of several attention modules: it is essentially a latent-space Transformer. The decoder works similarly to the encoder, but instead of mapping inputs to latents, it maps latents to outputs. It does this by using latents as inputs to its keys and value networks and passing an output query array to its query network. The resulting output has the same number of elements as the decoder query array.

Because of this structure, this architecture can be applied to inputs of any shape or spatial layout and even to inputs or outputs which don't share the same spatial structure (e.g. sound and video). However, in contrast to the latent spaces used elsewhere in vision (e.g. [67]) the latent does not explicitly share the structure (spatial or otherwise) of the inputs. To decode this information, we query for it using cross-attention.

3.2 Decoding the latent representation with a query array

Our goal is to produce a final output array of size $O \times E$, given a latent representation of size $N \times D$. As noted above, we can produce an output of this size by feeding an input with index dimension O as a query to the decoder. To capture the structure of the output space, we must make sure this query contains the appropriate information. This means the information query of the byte should reflect the downstream task, and ideally captures any structure needed in the outputs. This may include the spatial position in an image or the position of an output word in a sequence.

[2] With the exception of the repeated encoder cross-attends. We found it simpler to omit these modules, as they increase compute without dramatically increasing performance, as reported in Appendix Table 6 in [35].

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.

We construct queries by combining (concatenating or adding) a set of vectors into a query vector containing all of the information relevant for one of the O desired outputs. This process is analogous to the way that positional information is used to query implicit functions like NeRF [55]. We illustrate the query structure for the tasks we consider here in Fig. 3. For tasks with simple outputs, such as classification, these queries can be reused for every example and can be learned from scratch. For outputs with a spatial or sequence structure, we include a position encoding (e.g. a learned positional encoding or a Fourier feature) representing the position to be decoded in the output. For outputs with a multi-task or multimodal structure, we learn a single query for each task or for each modality: this information allows the network to distinguish one task or modality query from the others, much as positional encodings allow attention to distnguish one position from another. For other tasks, the output should reflect the content of the input at the query location: for instance, for flow we find it helpful to include the input feature at the point being queried, and for StarCraft II we use the unit information to associate the model's output with the corresponding unit. In general, we find that even very simple query features can produce good results, suggesting that the latent attention process is able to learn to organize the relevant information in a way that's easy to query.

4 Experiments

To evaluate the generality of Perceiver IO, we evaluate it on tasks in several domains, including language understanding (masked language modeling and downstream fine-tuning) and visual understanding (optical flow and image classification), symbolic representations for games (StarCraft II), and multi-modal and multi-task settings. The inputs and outputs in each case are listed in Table 1. We give additional details of the experiments on each domain in the Appendix. All experiments were conducted using JAX [7] and the DeepMind JAX ecosystem [4].

4.1 Language

First, we explore the use of Perceiver IO for language understanding, in particular to see how it performs compared to standard Transformers in a setting whether they the latter are known to shine. We evaluate the quality of the Perceiver IO learned representation on the GLUE benchmark [91] and report our results in Table 2. Unlike recent language understanding models such as BERT [21] or XLNet [96], Perceiver IO scales effectively with the input length (the Perceiver's latent size does not depend on the input length). For a given FLOPs budget, this allows us to train a tokenizer-free language model that matches the performance of a baseline model trained with a SentencePiece tokenizer, hence removing the need for hand-crafted and potentially harmful tokenization schemes [6, 14].

Pretraining setting. We pretrain on the Masked Language Modeling (MLM) task proposed in BERT [21] using a large text corpus obtained by combining English Wikipedia and C4 [63]. For both the SentencePiece and the byte-level models, we mask 15% of the words (here a word is defined to be a space-delimited sequence of characters). We use input sequence lengths of 512 SentencePiece tokens or 2048 UTF-8 bytes (as a token contains many bytes on average, we need to increase the sequence length to input a similar amount of text). For the SentencePiece models we use a vocabulary size of $32,000$ following [21]. For the byte-level models, the vocabulary size is much smaller: 256 bytes and 4 special tokens (`[PAD]`, `[MASK]`, `[CLS]`, `[SEP]`). Perceiver IO produces one output vector per masked input by using learnable position-dependent vectors to query the output of the final latent processing layer. We then apply a position-wise linear layer on top of these output vectors and train the model using a softmax cross-entropy loss to predict the original non-masked input as target. The full details of the architecture are given in Appendix Section C.1. See Appendix Fig. 6 for analysis and visualization of the learnt features.

Finetuning setting. We finetune Perceiver IO on the GLUE Benchmark [91], reporting the best performance on the dev set for a fixed size sweep of finetuning hyper-parameters. Individual task results and hyper-parameters are given in Appendix Section C.3.

Perceiver IO on SentencePiece tokens. We first observe that the Perceiver IO architecture applied on SentencePiece tokenized [69, 42] input sequences is slightly outperforming a strong BERT baseline applied on the same inputs (81.2 vs 81.1). As a result of the reduced latent size of 256 we can train a much deeper network with 26 processing layers compared to BERT Base (12 layers) while maintaining a similar FLOPs budget.

Perceiver IO on UTF-8 bytes. Next, we show that we can leverage Perceiver IO to run on much longer sequences than a regular Transformer can afford. This allows us to avoid using a fixed, handcrafted vocabulary. Our model works directly with the raw byte inputs: we simply feed in and predict the UTF-8 bytes of the input string. Perceiver IO significantly outperforms a byte-level BERT baseline at the same FLOPs budget, demonstrating the real advantage of Perceiver IO architecture for language. Remarkably, the bytes Perceiver IO is actually on par with BERT running on SentencePiece tokens, showing that Perceiver IO is also competitive against strong baselines relying on handcrafted tokenizers. Finally, the performance of Perceiver IO on bytes scales well with more FLOPs where we obtain 81.8 on the GLUE benchmark. Note that the byte-level Perceiver IO shares some similarities with the concurrent CANINE work [14]. While [14] still rely on a relatively sophisticated pipeline that maps Unicode codepoints to hash embeddings [77], we embed raw UTF-8 bytes directly. [14] also uses a bottleneck architecture to scale to longer text inputs, but their upsampling strategy differs from ours: they concatenate raw inputs with their aligned downsampled latent representation, apply a 1D convolution and then run a shallow transformer stack on the resulting upsampled sequence. Hence, their approach scales quadratically with respect to the original input length while the Perceiver IO's decoder scales *linearly* with respect to the target output size. Finally, our work scales to byte-level inputs without making any assumptions about the structure of the input, which allows it to be used beyond language as shown in the following sections.

Multitask Perceiver IO on UTF-8 bytes. We apply the multitask architecture detailed in Section 3.2 to the 8 GLUE benchmark tasks and report the results in Table 3. We compare to our previous result obtained in the single task regime where the model is trained independently on each task as well as a more standard approach that consists of inputting a special token in the input, and reusing the corresponding learned output query vector (aligned with its position in the sequence). This is reminiscent of the `[CLS]` token employed in BERT [21]. We consider two cases: a single token shared among tasks (*Shared input token*), or task-specific tokens (*Task specific input token*). In both cases a task specific head, a 2-layer MLP, is applied to generate the output logits corresponding to each task. We observe that our multitask approach outperforms the single task regime, and that our proposed multitask architecture is on par with the task-specific input token conditioning. However, our proposed approach is more generic as it decouples the output array from the input array by not relying on the `[CLS]` token. This is especially appealing when the number of tasks becomes high, as we show in section 4.3.

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).

4.2 Optical flow

Optical flow is a decades-old open problem in computer vision [49, 33]. Given two images of the same scene (e.g. two consecutive frames of a video), the task is to estimate the 2D displacement for each pixel in the first image. This has many broader applications, such as navigation and visual odometry in robots [11], estimation of 3D geometry [66], and even to aid transfer of more complex, learned inference such as 3D human pose estimation from synthetic to real images [23]. Optical flow is challenging for neural networks for two reasons. First, optical flow relies on finding correspondence: a single frame provides no information about flow, and images with extremely different appearance can produce the same ground truth flow. Second, flow is extremely difficult to annotate, meaning that the few existing datasets with realistic images and high-quality ground truth are small and biased. While it is straightforward to generate large synthetic datasets as training data, e.g. AutoFlow [74], there is still a large domain gap.

Algorithms for optical flow must therefore learn to accomplish several steps in a way that transfers from synthetic to real data. First, the algorithm must find correspondence between points. Then it must compute the relative offset of these points. Finally it must propagate the estimated flow across large regions of space, including to parts of the image which have no texture that can be used for correspondence. To generalize to real data, the learned procedure needs to also work for objects and textures that weren't seen in the synthetic training data.

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].

These difficulties have led flow researchers to develop some of the most involved architectures in the computer vision literature. State of the art algorithms, such as PWCNet [75], RAFT [84] or GMA [36], use explicit machinery to make sure that each of these steps is performed correctly, even on out-of-domain data. Expensive global correlation volumes are used to explicitly compare features with a spatiotemporal neighborhood across images in order to find correspondences. Flows are computed iteratively in 2D space, often hierarchically, with explicit lookup operators to verify correctness; these are often slow on hardware like TPUs [37].

Perceiver IO on Flow In contrast, we apply Perceiver IO to flow in a straightforward manner. We concatenate the frames along the channel dimension and extract a 3×3 patch around each pixel (leading to $3 \times 3 \times 3 \times 2 = 54$ values for each pixel). We concatenate a fixed position encoding to these features and then apply the Perceiver. To decode, we query the latent representation using the same encoding used for the input. See Appendix Section E for training details and results with various forms of pre- and post-processing, which typically perform similarly. We also test a version with convolutional downsampling and RAFT-style upsampling, which performs only slightly worse while improving computation time.

It may seem counter-intuitive to append the images along the channel dimension, as large motions might result in pixels on entirely different objects being concatenated. However, this kind of operation isn't unprecedented: one of the earliest optical flow algorithms, Lucas-Kanade [49], makes explicit use of the *temporal* image gradient, which is approximated by the difference in intensities at a given pixel across two frames. Specifically, the algorithm uses the fact that, ignoring lighting effects, the temporal gradient of the image is equal to the spatial gradient times the spatial velocity. This is true (and in fact, a better approximation) for image regions with very little texture. Such regions are a challenge for algorithms which attempt to find explicit correspondence in feature space, especially if feature encoding involves any normalization operations which may destroy intensity information.

Results Table 4 shows our results, following the standard protocol for training on AutoFlow [74]. We compare to PWCNet and RAFT baselines trained by the AutoFlow authors. On Sintel [10], our results are slightly better than RAFT on Sintel and outperform PWCNet on KITTI [52]. As far as we are aware, this result is state of the art on Sintel.final (GMA [36] produces slightly better numbers us on the somewhat easier Sintel.clean evaluation set using different training data). This is surprising considering how different our architecture is from PWCNet and RAFT, and how little additional tuning was required for our Perceiver implementation. We use no cost volumes or explicit warping, our model is not explicitly hierarchical, and the latent representation doesn't even maintain the 2D layout of the inputs. Also note that we reuse RAFT's AutoFlow augmentation parameters, which were tuned specifically for RAFT using population-based training [74]. As shown in Appendix Fig. 7, qualitatively Perceiver IO is good at following object boundaries, and can easily propagate motion across image regions with little texture.

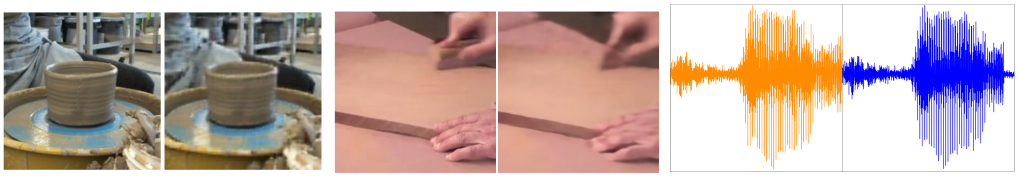


Figure 4: Audio-visual autoencoding with 88x compression. Side-by-side: inputs on left, reconstructions right (class label not shown).

4.3 Multimodal autoencoding

We explore using Perceiver IO for audio-video-label multimodal autoencoding on the Kinetics-700-2020 dataset [71]. The goal of multimodal autoencoding is to learn a model that can accurately reconstruct multimodal inputs in the the presence of a bottleneck induced by an architecture. This problem has been previously studied using techniques such as Restricted Boltzmann Machines [58], but on much more stereotyped and smaller scale data.

Kinetics-700-2020 has video, audio and class labels. We consider all three as separate modalities and train a model to reconstruct them. With traditional autoencoding models like convolutional encoder-decoders, it is not obvious how to combine these 3 modalities, because each consists of data of different dimensions – 3D (video), 1D (raw audio) and 0D (class labels) – and with vastly different numbers of elements. With a Perceiver IO model, we pad the inputs with modality-specific embeddings [35], serialize all of them into a 2D input array and query the model using queries containing Fourier-based position embeddings (for video and audio) and modality embeddings.

We train on 16 frames at 224×224 resolution, preprocessed into 50k 4x4 patches as well as 30k raw audio samples, patched into a total of 1920 16d vectors and one 700d one-hot representation of the class label. We decode directly into pixels, raw audio and the one-hot label without any post-processing. To prevent the model from encoding the label directly into one of the latent variables, we mask the class label (similarly to BERT) in training 50% of the time.

Compression Ratio	Audio PSNR	Video PSNR	Top-1 Accuracy
88x	26.97	24.37	10.2%
176x	25.33	24.27	8.6%
352x	14.15	23.21	11.5%

Table 5: Multimodal autoencoding results. Higher is better for accuracy and PSNR.

Due to the scale of inputs and outputs in this task we tried subsampling decoding in training, while fully decoding in testing: we sampled 512 audio samples and 512 pixels and the class label for every training example. This allows us to directly decode to a video-sized array, which would otherwise not be feasible given memory constraints. We used 3 different latent array bottleneck sizes with 512 channels and 784, 392 and 196 latents, resulting in compression ratios of 88x, 176x and 352x respectively.

Numerical results are in Table 5 and Fig. 4 shows reconstructions. By masking the classification label during evaluation, our auto-encoding model becomes a Kinetics 700 classifier. Because the latent variables are shared across modalities and not explicitly allocated between them, the quality of reconstructions for each modality is sensitive to the weight of its loss term and other training hyperparameters. Table 5 shows one tradeoff, where we emphasized video and audio PSNR at the expense of classification accuracy. By putting stronger emphasis on classification accuracy, we can reach 45% top-1 accuracy while maintaining 20.7 PSNR for video. This shows that the model learns a joint distribution across modalities.

4.4 StarCraft II

To further demonstrate Perceiver IO's capabilities on discrete modalities and to serve as a drop-in replacement for Transformers, we use Perceiver IO to replace the Transformer in AlphaStar, the state-of-the-art system for the complex game of StarCraft II. At its core, AlphaStar [89] represents the units in the game as a discrete, unordered set of symbols (the "units"). These units are represented by a vector of properties such as unit type, position, health, etc. At each timestep, the architecture encodes up to 512 units "tokens" with a vanilla Transformer. This representation is used both as a summary of the state (after pooling) and as a rich representation of the 512 units. This representation is used by a pointer network [90], to assign a probability to each possible unit selection, effectively parameterizing the agent's unit selection policy (see [89] and Appendix Section G for more details). We replaced the Transformer that inputs and outputs 512 units with Perceiver IO with a latent size of 32. Without tuning any additional parameters, we observed that the resulting agent reached the same level of performance as the original AlphaStar agent, reaching an 87% win-rate versus the Elite bot after behavioral cloning [61] on human data.

4.5 Image classification

The original Perceiver did as well on ImageNet [19] classification as models that used 2D structure in the design of the architecture, but used simple average pooling and a linear layer to generate class scores. Here, we evaluate the effect of the more general decoder.

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

Table 6: Results on ImageNet image classification (top-1 accuracy, higher is better). The first two blocks show representative baseline results. "Perceiver" uses an average+project decoder while "Perceiver IO" uses a cross-attend decoder.

Results Table 6 shows our results alongside representative numbers from the literature. Perceiver and Perceiver IO differ only in their decoder, and neither model uses convolutional preprocessing by default. Perceiver IO consistently improves over the original architecture. After large-scale pretraining on JFT [73], Perceiver IO produces results in the ballpark of strong models designed primarily for image classification. Perceiver IO is competitive with members of the Vision Transformer (ViT) [24] family without relying on 2D convolutions. But Perceiver IO is also compatible with convolutional preprocessing, and adding a simple 2D conv+maxpool preprocessing network leads to a moderate improvement in performance. We have incorporated several minor improvements to the training protocol reported in [35] and we use it for all experiments. See Appendix Section H for details.

While neither the Perceiver and Perceiver IO incorporate any 2D spatial structure architecturally, they use positional features that inject 2D spatial information (see Sec. 3.2 and Appendix sec. D of [35]). By replacing these 2D position features with a fully learned position encoding (the same strategy used on language), we can learn an image classification model that is given no privileged information about the structure of images. The results of this model on ImageNet are shown in the table: to our knowledge, this is the best result by any model on ImageNet without 2D architectural or feature information. As discussed in [35] (sec. 4.1), this model is also permutation invariant.

5 Conclusion

In this work, we introduce Perceiver IO, an architecture capable of handling general purpose inputs and outputs while scaling linearly in both input and output sizes. As we show, this architecture achieves good results in a wide variety of settings, making it a promising candidate for a *general purpose* neural network architecture. Perceiver IO leverages the expressive power of latent attention and uses learned queries to expose a simple and unified interface that can handle multimodal and multitask settings. Our work has limitations: for example, we don't currently address generative modeling, and we haven't explored mechanisms to automatically tune the size of the latent space. From an ethical perspective, our model may be susceptible to the drawbacks common to deep networks trained from big data: namely, they may replicate harmful biases present in the training data, and they may not be robust to domain shift or adversarial attacks, meaning that care must be taken in safety-critical applications. Overall, Perceiver IO offers a promising way to simplify the construction of sophisticated neural pipelines and facilitate progress on multimodal and multiask problems.

Acknowledgments

We are grateful to Ankush Gupta and Adrià Recasens Continente for reviewing drafts of this paper and to Deqing Sun for sharing code and helpful advice on the optical flow experiments.

References

[1] J. Abramson, A. Ahuja, A. Brussee, F. Carnevale, M. Cassin, S. Clark, A. Dudzik, P. Georgiev, A. Guy, T. Harley, F. Hill, A. Hung, Z. Kenton, J. Landon, T. Lillicrap, K. Mathewson, A. Muldal, A. Santoro, N. Savinov, V. Varma, G. Wayne, N. Wong, C. Yan, and R. Zhu. Imitating interactive intelligence. *arXiv preprint arXiv:2012.05672*, 2020. 1

[2] H. Akbari, L. Yuan, R. Qian, W.-H. Chuang, S.-F. Chang, Y. Cui, and B. Gong. Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text. *arXiv preprint arXiv:2104.11178*, 2021. 3

[3] J.-B. Alayrac, A. Recasens, R. Schneider, R. Arandjelović, J. Ramapuram, J. De Fauw, L. Smaira, S. Dieleman, and A. Zisserman. Self-supervised multimodal versatile networks. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2020. 3

[4] I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, C. Fantacci, J. Godwin, C. Jones, T. Hennigan, M. Hessel, S. Kapturowski, T. Keck, I. Kemaev, M. King, L. Martens, V. Mikulik, T. Norman, J. Quan, G. Papamakarios, R. Ring, F. Ruiz, A. Sanchez, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, W. Stokowiec, and F. Viola. The DeepMind JAX Ecosystem, 2020. 6

[5] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document Transformer. *arXiv preprint arXiv:2004.05150*, 2020. 3

[6] K. Bostrom and G. Durrett. Byte pair encoding is suboptimal for language model pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624, Online, Nov. 2020. Association for Computational Linguistics. 6

[7] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 6

[8] A. Brock, S. De, S. L. Smith, and K. Simonyan. High-performance large-scale image recognition without normalization. In *ICML*, 2021. 10, 23

[9] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 3, 5

[10] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2012. 2, 8

[11] J. Campbell, R. Sukthankar, and I. Nourbakhsh. Techniques for evaluating optical flow for visual odometry in extreme terrain. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004. 7

[12] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with Transformers. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2020. 3

[13] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever. Generative pretraining from pixels. In *Proceedings of International Conference on Machine Learning (ICML)*, 2020. 3

[14] J. H. Clark, D. Garrette, I. Turc, and J. Wieting. CANINE: pre-training an efficient tokenization-free encoder for language representation. *arXiv preprint arXiv:2103.06874*, 2021. 3, 6, 7

[15] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of International Conference on Machine Learning (ICML)*, 2008. 3

[16] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 2011. 3

[17] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. RandAugment: Practical automated data augmentation with a reduced search space. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020. 23

[18] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the Annual Meetings of the Association for Computational Linguistics (ACL)*, 2019. 3

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 2, 10

[20] K. Desai and J. Johnson. VirTex: Learning Visual Representations from Textual Annotations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 3

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional Transformers for language understanding. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2019. 2, 3, 6, 7, 18, 20

[22] C. Doersch and A. Zisserman. Multi-task self-supervised visual learning. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2017. 3

[23] C. Doersch and A. Zisserman. Sim2real transfer learning for 3d human pose estimation: motion to the rescue. *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2019. 7

[24] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021. 10

[25] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012. 3

[26] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2015. 3

[27] A. Goyal, A. Didolkar, A. Lamb, K. Badola, N. R. Ke, N. Rahaman, J. Binas, C. Blundell, M. Mozer, and Y. Bengio. Coordination among neural modules through a shared global workspace. *arXiv preprint arXiv:2103.01197*, 2021. 3

[28] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. 3

[29] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017. 3

[30] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 10

[31] D. Hendrycks and K. Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016. 17

[32] G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length, and Helmholtz free energy. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 1994. 3

[33] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 1981. 7

[34] D. A. Hudson and C. L. Zitnick. Generative adversarial transformers. In *Proceedings of International Conference on Machine Learning (ICML)*, 2021. 3

[35] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira. Perceiver: General perception with iterative attention. In *Proceedings of International Conference on Machine Learning (ICML)*, 2021. 1, 3, 4, 5, 9, 10, 16, 17, 19, 20, 23

[36] S. Jiang, D. Campbell, Y. Lu, H. Li, and R. Hartley. Learning to estimate hidden motions with global motion aggregation. *arXiv preprint arXiv:2104.02409*, 2021. 8

[37] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017. 8

[38] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017. 3

[39] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 16

[40] I. Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2012. 3

[42] T. Kudo and J. Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the Annual Meetings of the Association for Computational Linguistics (ACL)*, 2018. 7

[43] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set Transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2019. 3

[44] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. 3

[45] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, pages 11525–11538, 2020. 3

[46] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 3

[47] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017. 23

[48] K. Lu, A. Grover, P. Abbeel, and I. Mordatch. Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*, 2021. 3

[49] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1981. 7, 8

[50] M.-T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser. Multi-task sequence to sequence learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2016. 3

[51] X. Ma, X. Kong, S. Wang, C. Zhou, J. May, H. Ma, and L. Zettlemoyer. LUNA: Linear unified nested attention. *arXiv preprint arXiv:2106.01540*, 2021. 3

[52] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 8, 21

[53] A. Miech, J.-B. Alayrac, I. Laptev, J. Sivic, and A. Zisserman. Thinking fast and slow: Efficient text-to-visual retrieval with Transformers. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 3

[54] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2013. 3

[55] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorth, and R. Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2020. 6, 19

[56] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[57] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2016. 2

[58] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *Proceedings of International Conference on Machine Learning (ICML)*, 2011. 9

[59] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global Vectors for word representation. In *Proceedings of the Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 3

[60] H. Pham, Z. Dai, Q. Xie, M.-T. Luong, and Q. V. Le. Meta pseudo labels. *arXiv preprint arXiv:2003.10580*, 2021. 10

[61] D. A. Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 1989. 10

[62] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. 3, 16

[63] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. 3, 5, 6, 18

[64] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of Machine Learning Research*, 2020. 3

[65] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021. 1

[66] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 7

[67] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2015. 2, 3, 5

[68] A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, and T. Lillicrap. Relational recurrent neural networks. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2018. 3

[69] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the Annual Meetings of the Association for Computational Linguistics (ACL)*, 2016. 7

[70] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2014. 3

[71] L. Smaira, J. Carreira, E. Noland, E. Clancy, A. Wu, and A. Zisserman. A short note on the Kinetics-700-2020 human action dataset. *arXiv preprint arXiv:2010.10864*, 2020. 2, 9, 22

[72] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(20):131 – 162, 2007. 19

[73] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2017. 10, 23

[74] D. Sun, D. Vlasic, C. Herrmann, V. Jampani, M. Krainin, H. Chang, R. Zabih, W. T. Freeman, and C. Liu. AutoFlow: Learning a better training set for optical flow. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 8, 21

[75] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 8, 20

[76] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2014. 3

[77] D. Svenstrup, J. M. Hansen, and O. Winther. Hash embeddings for efficient word representations. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2017. 7

[78] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 3

[79] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2020. 19

[80] Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, and C. Z. Zhe Zhao. Synthesizer: Rethinking self-attention in Transformer models. In *Proceedings of International Conference on Machine Learning (ICML)*, 2021. 3

[81] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler. Long range arena: A benchmark for efficient Transformers. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021. 3

[82] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient Transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020. 4

[83] Y. Tay, V. Q. Tran, S. Ruder, J. Gupta, H. W. Chung, D. Bahri, Z. Qin, S. Baumgartner, C. Yu, and D. Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672*, 2021. 3

[84] Z. Teed and J. Deng. RAFT: Recurrent All-pairs Field Transforms for optical flow. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2020. 8, 20

[85] A. Toshev and C. Szegedy. DeepPose: Human pose estimation via deep neural networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 3

[86] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image Transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020. 10, 23

[87] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou. Going deeper with image Transformers. *arXiv preprint arXiv:2003.10580*, 2021. 10

[88] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2017. 1, 2, 3, 19

[89] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, Dani Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, A. Chris, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. 1, 2, 9, 10, 22

[90] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2015. 10

[91] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2019. 6, 7

[92] H. Wang, Y. Zhu, H. Adam, A. Yuille, and L.-C. Chen. Max-deeplab: End-to-end panoptic segmentation with mask Transformers. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 3

[93] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020. 3

[94] Y. Xiong, Z. Zeng, R. Chakraborty, M. Tan, G. Fung, Y. Li, and V. Singh. Nyströmformer: A Nyström-based algorithm for approximating self-attention. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2021. 3

[95] L. Xue, A. Barua, N. Constant, R. Al-Rfou, S. Narang, M. Kale, A. Roberts, and C. Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *arXiv preprint arXiv:2105.13626*, 2021. 3

[96] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 6

[97] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo. Image captioning with semantic attention. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[98] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021. 19, 21

[99] S. Yun, D. Han, S. J. Oh, J. C. Sanghyuk Chun, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2019. 23

[100] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3

[101] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *Proceedings of European Conference on Computer Vision (ECCV)*, 2014. 3

[102] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. *mixup*: Beyond empirical risk minimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018. 23

Domain	Input Modality	Encoder KV input	Encoder KV channels	Decoder query input	Decoder query channels
Language (MLM)	Text	byte/token encoding + learned pos	768	learned pos	1280
Language (Perceiver IO++ MLM)	Text	byte/token encoding + learned pos	768	learned pos	1536
Language (GLUE)	Text	byte/token encoding + learned pos	768	Class query (per-task)	1280
Language (Perceiver IO++ GLUE)	Text	byte/token encoding + learned pos	768	Class query (per-task)	1536
Optical Flow	Video (image pairs)	[2 × conv features, 3D FFs]	450	[2 × conv features, 3D FFs]	450
Optical Flow (pixels)	Video (image pairs)	[Linear(2 × RGB), 2D FFs]	322	[Linear(2 × RGB), 2D FFs]	322
Kinetics	Video, Audio, Label	[Linear(RGB), 3D FFs, learned modality feat.] [sound pressure, 1D FF, learned modality feat.] [one-hot label, learned modality feat.]	704 704 704	[3D FFs, learned modality feat.] [1D FF, learned modality feat.] [learned modality feat.]	1026 1026 1026
StarCraft II	SC2 entities	Entity features	128	Entity features	128
ImageNet	Image	[RGB, 2D FFs]	261	Class query (single)	1024
ImageNet (learned pos)	Image	[Linear(RGB), learned pos]	512	Class query (single)	1024
ImageNet (conv)	Image	[Conv features, 2D FFs]	322	Class query (single)	1024

Table 7: The structure and size of the positional and task embeddings used to construct Perceiver IO's encoder key-value inputs and decoder query inputs, for each domain described in the main text. "[x, y]" indicates that x's and y's features are concatenated, while "x + y" indicates that x's and y's features are added to produce the full featurization. "FF" = Fourier features, as in [35].

A FLOPs calculation

In all cases, we report theoretical FLOPs with multiplies and accumulates counted as separate operations. This is the strategy used in [39] and elsewhere in the literature. We use this strategy consistently here to allow comparisons between the models we propose and develop (including our BERT reimplementation). Note that some papers in the literature report FLOPs using fused multiply-accumulates: using this strategy will cut the figures reported here in half.

B Architectural details

Perceiver IO is constructed from GPT-2-style [62] Transformer attention modules, which consist of QKV attention followed by an MLP, along with linear projection layers to ensure inputs to and outputs from the QKV attention and MLP take on desired sizes. Using the array sizes of the encoder attention, the QKV attention takes in two two-dimensional arrays, a key-value input array $X_{KV} \in \mathbb{R}^{M \times C}$ and a query input array $X_Q \in \mathbb{R}^{N \times D}$, and maps them to an array $X_{QKV} \in \mathbb{R}^{N \times D}$, sharing the shape of the query input (after projection). X_{QKV} is used as input to an MLP, which is applied independently to each element of the index dimension,[3] producing a final array $X_{MLP} \in \mathbb{R}^{N \times D}$.

While we describe attention as taking two inputs, in standard Transformers it is typically described as mapping one input to an output of the same size. This is because all modules of a standard Transformer use *self*-attention, where the same input is used for both key-value inputs and query inputs. The view of attention that we describe encompasses both cross-attention and self-attention, both of which are specific ways of using QKV-attention. Perceiver IO uses cross-attention for encoder and decoder attention modules and uses self-attention for the latent processing modules. These modules differ primarily in terms of what shape inputs they expect and what shape outputs they produce (see Fig. 5).

We now describe the structure of QKV attention and the MLP in more detail.

B.1 Attention module internals

QKV attention takes in two two-dimensional arrays, a query input $X_Q \in \mathbb{R}^{N \times D}$ and a key-value input $X_{KV} \in \mathbb{R}^{M \times C}$. The output of QKV attention is an array with the same index (first) dimension as the query input and a channel (second) dimension determined by an output projection:

$$Q = f_Q(X_Q); K = f_K(X_{KV}); V = f_V(X_{KV}) \tag{1}$$

$$X_{QK} = \text{softmax}(QK^T/\sqrt{F}) \tag{2}$$

$$\text{Attn}(X_Q, X_{KV}) = X_{QKV} = f_O(X_{QK}V), \tag{3}$$

[3] i.e. by convolving the MLP with its input along the first dimension

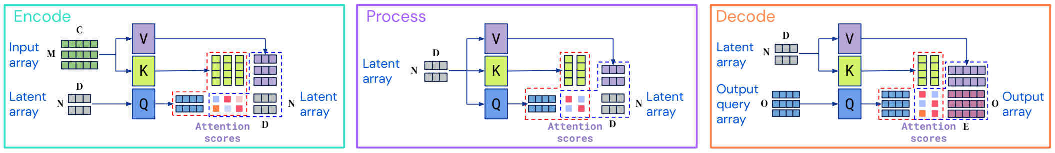


Figure 5: Schematic depiction of encode, process, and decode attention. Each attention module uses the same operations, but differs in which inputs are used to generate key/values or queries and in the output shape. Encode attention can be viewed as mapping an input to a latent space, typically with a smaller index dimension (fewer elements). Decode attention can be viewed as mapping a latent to an output space, often with a larger index dimension (more elements). Both of these are forms of cross-attention. Process attention (self-attention) preserves the input index dimension (same elements). Red and blue dashed lines are used to highlight the two matrix multiplications used in QKV attention, as described in the text.

where X_{QK} is an array of attention maps $\in \mathbb{R}^{N \times M}$, and X_{QKV} is an array $\in \mathbb{R}^{N \times D}$. The functions $f_{\{Q,K,V\}}$ are linear layers mapping each input to a shared feature dimension F and f_O is a linear layer projecting the output to a target channel dimension, which is often the same size as X_Q's. All linear layers are applied convolutionally over the index dimension (the first dimension of their inputs). We have omitted batch and head dimensions (in the case of multi-headed attention) for readability.

QKV attention is followed by a two-layer MLP with a GELU [31] nonlinearity following the first layer. The full module has the following structure:

$$
\begin{aligned}
X_{QKV} &= \text{Attn}(\text{layerNorm}(X_Q), \text{layerNorm}(X_{KV})) && (4)\\
X_{QKV} &= X_{QKV} + X_Q && (5)\\
X_{QKV} &= X_{QKV} + \text{MLP}(\text{layerNorm}(X_{QKV})), && (6)
\end{aligned}
$$

slightly abusing notation for simplicity and to emphasize the residual structure. "Attn" refers to QKV as described above.

In the context of decoder attention, we sometimes find it helpful to omit the second step ($X_{QKV} = X_{QKV} + X_Q$), as it involves adding the model output with a query. Queries sometimes include features inherited from the input space (see Tab. 7), and this residual connection may make learning unnecessarily difficult. For example, for optical flow, including this residual connection forces the network to produce optical flow output by adding RGB and Fourier features to the model's output.

B.2 Computational complexity

The computational complexity of each attention module is dominated by the two matrix multiplications in QKV attention. Still using the shapes of the encoder attention, these two matrix multiplies involve matrices of shape $M \times F$ and $N \times F$ and $M \times N$ and $N \times F$, giving overall time and memory complexity of $\mathcal{O}(MNF)$. Let M, N, and O be the index dimensions for the input, latent, and output arrays, and to simplify the analysis let F be the feature size for all layers. The KV and Q sizes for the encoder, latent transformer, and decoder will then be $M \times F$ and $N \times F$ (for the encoder), $N \times F$ and $N \times F$ (for the latent transformer), and $N \times F$ and $O \times F$ (for the decoder). A model with L latent attention blocks has complexity $\mathcal{O}([M + O + LN]NF)$. In other words, Perceiver IO has complexity linear in the size of the input and output arrays and it decouples the depth of the latent transformer from the input and output sizes. Both of these properties contribute to Perceiver IO's efficiency: while many proposals for efficient attention modules or architectures include linear or sub-quadratic scaling with input/output size, Perceiver IO is unusual in also decoupling depth from input/output size (without requiring domain-specific strategies like 2D convolution). For a longer discussion of these points, see section 2 and Appendix Section A of [35].

C Language: additional details

C.1 Architecture details

The architecture hyper-parameters for the Perceiver IO used in the language experiments are given in Table 8.

Model	Perceiver IO Base	Perceiver IO	Perceiver IO++
Tokenizer	SentencePiece	UTF-8 bytes	UTF-8 bytes
Number of inputs (M)	512	2048	2048
Input embedding size (C)	768	768	768
Number of Process layers	26	26	40
Number of latents (N)	256	256	256
Latent size (D)	1280	1280	1536
FFW hidden dimension for latents	1280	1280	1536
Number of output queries during pretraining (O)	512	2048	2048
Dimension of learned queries (E)	768	768	768
FFW hidden dimension for outputs	768	768	768

Table 8: Perceiver IO architecture details for Language experiments

C.2 MLM pretraining

We pretrain all models on a mixture of the C4 dataset [63] and English Wikipedia, where 70% of the training tokens are sampled from the C4 dataset and the remaining 30% from Wikipedia. We concatenate 10 documents before splitting into crops to reduce wasteful computation on padding tokens. We use the same masking strategy for SentencePiece and byte-level experiments: each word is masked independently with probability 15% where word boundaries are defined using white-space boundaries.

The pretraining hyper-parameters are given in Table 9. For the BERT (matching FLOPs) model trained on bytes, we reduce the model width from 768 to 512, the feed-forward hidden size from 3072 to 2048, the number of layers from 12 to 6 and the number of attention heads from 12 to 8. Given the longer sequence length of 2048 bytes, this model has about the same number of inference FLOPs as a BERT Base model on a sequence length of 512 tokens.

In order to decode, we use learned queries of the same dimension of the input array (see Table 8). We have as many output queries as inputs to be able to predict the masked token at all positions in the sentence (M=O).

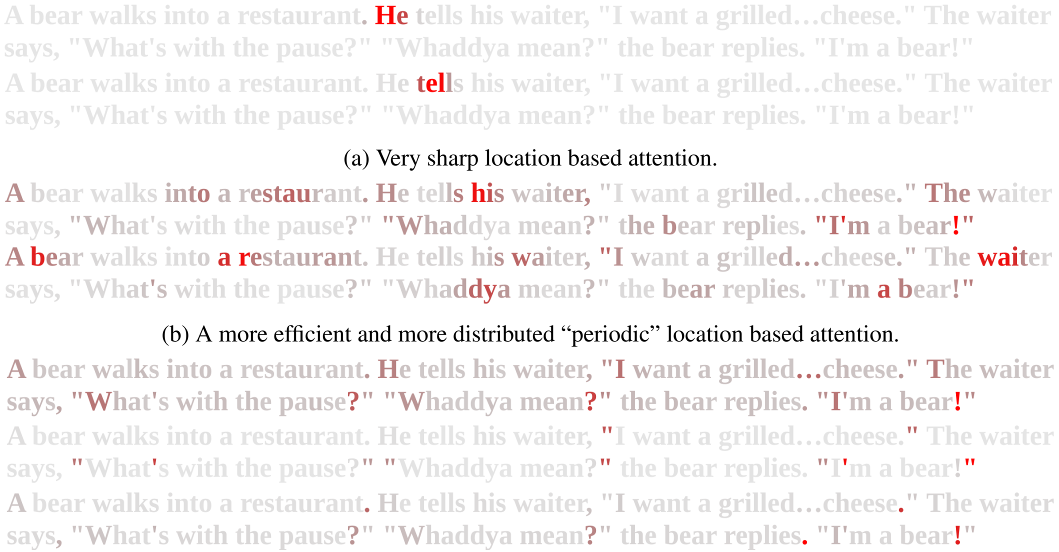
To get an insight into the learnt queries we visualize the attention weights in the first cross attention layer on a small paragraph (Figure 6). We discover that the model has learnt both position and content based look-ups. The position-based look-ups can be either very sparse and precise or more distributed and periodic. This second mode appears somewhat less often and is more efficient because more data is being attended to at the same time, but also more distributed, since the values are subsequently averaged: this acts as a learned pooling. The content based retrievals focus mostly on syntactic elements like capital letters and punctuation (colon, exclamation marks, quotation marks, etc). This is probably because these are good word delimiters and can help the model reduce prediction uncertainty.

C.3 GLUE Finetuning

Following [21], we specify a fixed-size hyper-parameter grid and select the best dev performance across that grid for each task independently (Table 10). The full GLUE results are shown in Table 11. Following [21] we exclude the WNLI task. We use accuracy for all tasks expect STS-B and CoLA where we use Pearson correlation and Matthews correlation respectively. The average is computed by first averaging the results of MNLI-matched and MNLI-mismatched, which is then counted as a single task in the overall average.

Training steps	500,000
Batch size	512
Masking strategy	Words
Optimizer	LAMB [98]
Learning rate	0.00125
Linear warmup steps	1,000
Cosine cycle decay	500,000
Weight decay	0.01

Table 9: Hyper-parameters for masked language modelling (MLM) pre-training experiments



(a) Very sharp location based attention.

(b) A more efficient and more distributed "periodic" location based attention.

(c) Content based attention for syntactic elements like punctuation and capital letters.

Figure 6: Visualization of attention weights for a few queries in the initial cross-attention layer. We use the color to convey the weight of the attention and normalize by the maximum weight to make them easier to visualize. Best viewed in color.

For single-task experiments, we do not require a [CLS] token as we use a single decoding query vector. In both single-task and multi-task experiments an extra 2-layer MLP with a hidden size of E and a tanh activation is used to map the the Perceiver IO outputs to the class logits (or regression target for STS-B).

Training epochs	10
Batch size	$\{16, 32, 64\}$
Optimizer	LAMB
Learning rate	$\{1\times10^{-4}, 5\times10^{-5}, 2\times10^{-5}, 1\times10^{-5}\ \}$
Linear warmup steps	200
Weight decay	0.01

Table 10: Perceiver IO Hyper-parameters for GLUE finetuning experiments. The values in brackets are swept over.

D Positional encodings for image and audio experiments

For all image experiments (with the exception of the ImageNet experiment that uses learned positions, see Section H), we use the same position encoding strategy as in [35]. We use a 2D Fourier feature positional encoding [88, 72, 55, 79] using a sine and cosine bands with frequencies spaced linearly

Model	Tokenizer	Multi-task	CoLA	MNLI-m/mm	MRPC	QNLI	QQP	RTE	SST-2	STS-B	Average
Bert Base (test) [21]	SentencePiece	No	52.10	84.60/83.40	84.80	90.50	89.20	66.40	93.50	87.10	80.95
Bert Base (ours)	SentencePiece	No	50.28	85.56/85.68	85.75	92.67	91.05	61.72	93.98	88.04	81.14
Perceiver IO Base	SentencePiece	No	47.11	84.53/85.03	87.25	92.12	90.22	65.23	94.38	88.18	81.16
BERT (matching FLOPs)	UTF-8 Bytes	No	20.06	74.11/75.55	77.00	85.75	88.23	53.91	89.00	82.84	71.45
Perceiver IO	UTF-8 Bytes	No	50.19	83.22/83.89	87.24	91.71	90.12	64.84	93.17	86.81	80.95
Perceiver IO++	UTF-8 Bytes	No	52.54	84.13/84.91	86.03	92.06	90.46	66.54	93.98	87.93	81.76
Perceiver IO (Shared input token)	UTF-8 Bytes	Yes	47.43	82.03/82.65	89.58	90.18	89.20	82.03	93.17	77.95	81.49
Perceiver IO (Task specific input token)	UTF-8 Bytes	Yes	49.06	82.14/82.64	89.84	90.53	89.40	79.69	93.17	80.02	81.76
Perceiver IO (Multitask query)	UTF-8 Bytes	Yes	47.88	82.05/82.77	90.36	90.37	89.49	80.08	93.75	79.95	81.79

Table 11: Full GLUE results (higher is better). The first 3 models use SentencePiece tokens, the latter 3 use UTF-8 bytes directly.

Method	Patch size	Concat. frames	Downsample	Depth	Latents	Sintel.clean	Sintel.final	KITTI
PWCNet [75]	-	-	-	-	-	2.17	2.91	5.76
RAFT [84]	-	-	-	-	-	1.95	2.57	4.23
Perceiver IO	3×3	Yes	No	24	2048	1.81	2.42	4.98
Perceiver IO	3×3	No	No	24	2048	1.78	2.70	6.19
Perceiver IO	1×1	Yes	No	24	2048	1.91	2.56	5.39
Perceiver IO	1×1	No	No	24	2048	1.72	2.63	5.93
Perceiver IO	N/A	Yes	Yes	24	2048	1.84	2.52	4.83
Perceiver IO	N/A	No	Yes	24	2048	1.90	2.53	6.66
Perceiver IO	N/A	Yes	Yes	16	1024	2.06	2.67	6.12

Table 12: Ablated Optical Flow results (end-point error, lower is better). The top Perceiver IO results show the configuration from the main paper. We ablate 1) patch size for the context surrounding each pixel, 2) whether the two frames are concatenated or input separately to the Perceiver, 3) whether the inputs and queries are downsampled by a factor of 4 using a convolution, and then subsequently upsampled with RAFT, and finally a the number of self-attention modules (depth) and number of elements in the latent array, resulting in a bottom-row network which is substantially less expensive than the original model.

from a minimum frequency to a maximum frequency. We use 64 sine/cosine bands per dimension in all settings. The minimum frequency is always set to the minimum frequency of the input signal, corresponding to a single full oscillation over the input dimension. The maximum frequency is typically set to the input's Nyquist frequency (e.g. 112 cycles for an image with 224 pixels per dimension). As in [35], the input position used to construct the Fourier frequencies is scaled to [-1, 1] for each input dimension. For example, the upper left corner of an image is at position [-1, -1] while the bottom right corner is at position [1, 1]. We follow the same strategy using 1D and 3D Fourier feature positional encoding for audio's time and video's spatiotemporal inputs, respectively. See each section for details.

E Optical Flow: additional details and results

Pre- and post-processing can provide non-trivial inductive biases when processing image data and also change computation time. Therefore, in this section, we ablate these choices. The network in the main paper concatenates the two frames frames before extracting 3D patches around each pixel, each of size 3×3×2. Table 12 shows a few alternative designs for patch extraction. 1×1 means that only a single pixel (or pair of pixels) is used for each input element. 'Separate frames' means that the frames are not concatenated, but rather, input array elements are extracted independently from the two frames (thereby doubling the number of input elements). In the case of separate frames, 1×1 means essentially no preprocessing: each pixel becomes its own element with no spatio-temporal context whatsoever.

We also performed experiments with a less expensive input model which uses a 7×7 convolution to 64 channels, followed by a max pool, similar to the one used in our ImageNet experiments. After feeding this through the Perceiver IO architecture (including querying with the same convolutional features used as input), we have an output a feature grid with stride 4 and 64 channels, on top of which we apply a RAFT upsampling layer. This involves a linear projection from 64 dimensions to 2, which is the coarse-resolution optical flow estimate. We then upsample this flow for a given pixel in the high-resolution flow map by applying attention over a neighboring 3x3 block of the low-resolution flow map, following the uppsampling approach in RAFT [84].

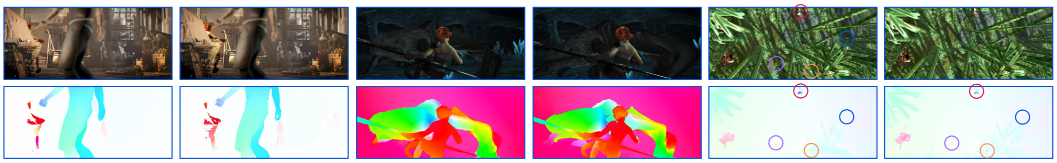


Figure 7: Qualitative examples of optical flow. For each image pair, we show the two frames (top), and then the estimated flow (bottom left) and the ground-truth flow (bottom right). In the left example, we see one person under heavy occlusion where the correct flow is propagated into a region with few details. Another person in the foreground has clothes with little texture and substantial blur, and yet the algorithm can propagate the flow across the entire region. In the center example, we see very large motions from both the dragon and the person, yet many fine structures are preserved like the pole. On the right, we see a forest scene with a few extremely small objects with very subtle motions (circled) which our algorithm is able to detect and segment correctly.

We found that concatenating frames led to a non-trivial performance improvement across the more difficult Sintel.final and KITTI Flow 2015 [52] datasets. Spatial context helps, and the impact of frame concatenation is larger when more context is available, suggesting that the algorithm is comparing spatial and temporal gradients. Convolutional downsampling and RAFT upsampling provide even more spatial context for both the input features and the queries, but this doesn't seem to provide any performance benefits, although it is less expensive computationally. Finally, we tested a shallower model with a smaller number of latents, which substantially reduces the latent computation. This model runs at close to real time (20.5 frames/sec) on our input resolution (368 × 496) on a 2017 NVIDIA Titan XP graphics card (while the original runs at 2.4 frames/sec). We see that the resulting model is only modestly worse than the full model, and still outperforms PWCNet on Sintel.

Figure 7 shows some results on example image pairs from the Sintel.final dataset. We see that the algorithm is capable of dealing with heavy occlusion, and can propagate optical flow across large regions with very little texture. The network can also deal with very large motions and very small objects.

Implementation details: Our experiments with pixels and patches use a sine and cosine position encoding with 64 bands for both X and Y, plus the raw X and Y values resulting in 258 extra features concatenated to the pixel or patch values. For experiments without concatenated frames, we have an additional time dimension which must be encoded with positional encoding, and for this we also use 64 sine and cosine bands (which are highly redundant, as there's only two frames). For this version, only the elements associated with the first frame are included as queries for the decoder. For both input and query, we project these concatenated features to 64 dimensions before inputting them into the transformer. We use a latent array with 2048 elements and 512 channels and 24 self-attention modules, each with 16 self-attention heads, unless otherwise noted. Our experiments with convolutional downsampling and RAFT upsampling use settings that are mostly similar, although we use no additional projection as the output of the convolutional network is already 64 channels. For these experiments, the output of the perceiver decoder's cross attend is 64 channels, which is fed into a RAFT-style upsampling operation. For the pixel- and patch-based models, total computational complexity for a forward pass on a 368 × 496 image is roughly 987 billion FLOPs, and there are roughly 27.9 million parameters.

In all cases, we train on the AutoFlow dataset [74], which consists of $400,000$ image pairs, for 480 epochs using a cosine learning rate schedule which starts at a learning rate of 4e-4. We use a batch size of 512. We use the LAMB [98] optimizer. We also use the default curriculum for AutoFlow, which gradually increases the severity of the augmentations over time. We find that naïve training on AutoFlow does not train, so we use an additional phase in this curriculum, where we completely disable all augmentations. Furthermore, for this phase, we feed every image pair twice in a batch: once forward, and once reversed. As the inverse flow is not currently available for AutoFlow, this inverse flow was computed via an approximation which averages all the flows terminating at a given pixel.

The evaluation datasets have a different resolution, so we evaluated in a tiled manner, using six evenly-spaced tiles. For pixels that are covered by multiple tiles, we average the predictions, weighted proportional the distance to the nearest edge of the respective tile (as we expect predictions nearer to the tile edges to be less accurate). We leave the possibility of making Perceiver IO invariant to input shape to future work.

F Multimodal auto-encoding: additional details

For the multimodal auto-encoding experiments, we patch preprocessing for both images and audio, and we embed the labels as one-hot labels. The patch size is $1 \times 4 \times 4$ for video and 16 for audio. The audio is sampled at 48kHz, or 1920 samples per frame. The decoder outputs $16 \times 224 \times 224 + 16 \times 1920/16 + 1$ vectors with 512 channels, that is, one element for each pixel in the video, one element for each audio patch, and one element for the classification label. These are then linearly projected to the appropriate channel size for each modality: 3 for videos, 16 for audio and 700 for classification (the logits for each of the 700 classes in Kinetics700). Finally, we un-patch the audio to arrive at the output audio. We note that we read and generate the audio waveform directly in the time domain; we do not transform first to a spectrogram.

We use a 387 dimensional 3D Fourier position embedding for each input video patch and a 385 dimensional 1D Fourier position embedding for each audio patch (385 to ensure the input dimensions to Perceiver IO match for all elements). In addition, we pad all input elements with a learned vector representing the modality; inputs from the same modality share the same token. In particular, we add a 317 dimensional modality embedding to video elements, a 319 dimensional modality embedding to audio elements, and a 4 dimensional modality embedding to the label, so that all elements have 704 features.

The decoder queries are also constructed from Fourier position embeddings for video and audio and a learned positional embedding for label: 387 features for video, 385 features for audio, and 1024 learned features for the label. We pad the queries for each modality with a different learned vector for each modality, so that the final feature size for the queries is 1026.

We train on Kinetics 700 [71]. We use batch size of 1024, and learning rate of 1e-3. The training loss is a weighted sum of the L1 loss for video, the L1 loss for audio, and the cross entropy loss for the label. The weightings are 0.03 for video, 1 for audio, and 0.0001 for the label; the loss weights are imbalanced in favor of audio because it is more difficult to obtain audio of high perceptual quality by directly outputting the waveform. We also tried a different weighting (0.03 for video, 1 for audio, and 1 for the label) to obtain higher classification accuracy.

G StarCraft: additional details

StarCraft results were obtained by using Perceiver IO instead of a Transformer for the AlphaStar entity encoder. The entity encoder takes as input a set of 512 entities (referred to as `embedded_entity` in [89]) and produces as output an embedding for each entity (`entity_embeddings`) and a 1D embedding reduced over entities (`embedded_entity`). These 512 entities represent the units and other entities that are present in the game: unused entity slots are masked. `entity_embeddings` is produced by passing the outputs of the entity encoder through a ReLU and a 1D convolution with 256 channels. `embedded_entity` is produced by averaging the (unmasked) entity encoder outputs and passing it through a linear layer with 256 units and a ReLU.

In the original AlphaStar system, the entity encoder consisted of a Transformer with 3 attention layers, each of which used 2 heads, and a feature dimension of 128. The output of each attention layer is projected to 256 and followed by an 2-layer MLP with hidden size 1024 and output size 256. This architecture was arrived by an extensive tuning process as reported in [89]. We replaced this Transformer with a 3-layer Perceiver IO with a latent of index dimension 32. We tuned only the size of the index dimension (sweeping values of 32 and 64), but otherwise used the same hyperparameters as ImageNet.

H ImageNet: additional details

H.1 Details of ImageNet training

Model	FLOPs
Perceiver (pixels)	404B
Perceiver IO (pixels)	407B
Perceiver (conv)	367B
Perceiver IO (conv)	369B
Perceiver IO (pretrained)	213B

Table 13: ImageNet model FLOPs, in billions. The choice of positional encoding does not significantly change the FLOPs requirements of the model, so we do not show these numbers separately. The model used for pretraining uses only 16 process modules.

For ImageNet experiments, we use CutMix [99] and MixUp [102] regularization, in addition to RandAugment [17] as used in [35]. We observed only marginal improvements in performance from this change, but it brings the augmentation strategy more in line with the strategy used elsewhere in the literature [8, 86]. In all experiments, we use RandAugment with 4 layers at magnitude 10 (as in [35]) and CutMix with a ratio of 0.2. In early experiments, we found that higher weight decay and moderate gradient clipping contributed to better generalization: we use a weight decay of 0.1 and clip to a maximum global gradient norm of 10. We use no dropout. As in [35] we use an architecture with weight sharing in depth: the latent (processing) component of the architecture includes 8 blocks of 6 attention modules each, and weights are shared between the corresponding modules in each block. We omit the repeated encoder cross-attends used [35] as we found these to lead to relatively small performance improvements but to significantly slow down training: using 8 encoder cross-attention instead of 1 adds an additional 303 billion FLOPs. The FLOPs for all ImageNet models presented here are given in Table 13.

For all ImageNet experiments, we train for 110 epochs, using a batch size of 1024 and 64 TPUs. We use LAMB with a simple learning rate schedule consisting of a flat learning rate of 2×10^{-3} for 55 epochs, after which the learning rate is decayed to 0 over the final 55 epochs following a cosine decay schedule [47]. We found a cosine learning rate decay schedule simpler to tune than the step decay schedule used in [35] and that beginning the decay process halfway through training generally led to good performance without introducing instability. We found it important to omit an initial learning rate warm-up period, as this often prevented models from training when using LAMB.

H.2 Learned position encoding experiments

In addition to the main experiments, which use 2D Fourier feature positional encodings (as described in Section D), we also include results using a learned positional encoding. This positional encoding is an array of shape 50,176 × 256, which is randomly initialized using a truncated Gaussian distribution with scale 0.02. ImageNet networks that use this positional encoding are given no information about 2D image structure. For these experiments, we additionally use a 1D convolutional network to project the RGB at each point to 256 before concatenating it with the learned positional encoding.

H.3 Large-scale pretraining

As reported in [35], Perceiver models are able to easily overfit ImageNet-scale datasets without regularization. For this reason, we explored pretraining a model on JFT, a large-scale, multi-labeled internal dataset with 300 million images spanning approximately 18,000 classes [73]. We pretrain on this dataset at the same resolution used on ImageNet (224×224) using a base learning rate of 3×10^{-4} and a cosine decay schedule, decaying to 0 over 14 epochs. We omit all augmentation except basic cropping, resizing, and left-right flipping. We use a weight decay of 0.1. We use a larger batch size of 8192 and train on 256 TPUs. Images in this dataset come with a variable number of labels, so we use a cross-entropy loss with a multi-one-hot representation of the targets. Unlike in the other ImageNet experiments, we do not share weights in the latent self-attention process modules, but use a 16-layer latent network with no weight sharing in depth. Unlike the other ImageNet experiments, the process-module MLPs use a hidden layer with $4\times$ the number of channels (rather than $1\times$ as on other ImageNet experiments). As in the learned position encoding experiments, we use a 1D convolutional network to project input RGB at each point to 256 before concatenating it with the positional encoding (a 2D Fourier frequency positional encoding).

To evaluate transfer, we fine-tune our pre-trained model on ImageNet. We replace only the final linear layer of the decoder to produce the required 18,000 classes. For fine-tuning, we used similar optimizer and augmentation settings as with our from-scratch ImageNet training: 1024 batch size on

64 TPUs, 131K steps with LAMB using a flat base LR of 0.002 for the first 70K steps and a cosine learning rate decay for the last 61K steps,

H.4 2D convolutional preprocessing on ImageNet

In other image settings discussed here, we optionally use simple pre- and post-processing steps to reduce the size of very large inputs and outputs. Because ImageNet data points are relatively small (Tab. 1), we are able to process full images without convolutional pre- and post-processing. Consequently, we can use this dataset to probe the sensitivity of the model to convolutional pre-processing. Incorporating a single convolution + max pooling leads to a moderate improvement in the performance of the architecture: this is perhaps unsurprising, as convolutional pre-processing injects information about the 2D structure of images into the architecture. By comparison ViT first processes images by applying a 2D convolution with matched kernel and stride to downsample its inputs (referred to as a "linear projection of flattened patches" in that work). As in other experiments, we find that incorporating an attention-based decoder (Perceiver IO) leads to better results than averaging and pooling the output (Perceiver). Using convolutional preprocessing leads to a moderate reduction in the number of FLOPs used by the model, as shown in Table 13. The input to the network after preprocessing is 56×56 instead of 224×224 as in the experiments directly on pixels.