ЕЩЁ О ТРАНСФОРМЕРАХ

Сергей Николенко

СПбГУ — Санкт-Петербург

31 октября 2024 г.

Random facts:

- 31 октября во всём мире — Хэллоуин, ночь накануне Дня всех святых, а в России — День работников СИЗО и тюрем; 31 октября 1963 года были созданы первые СИЗО в СССР
- «31 октября» (Club Deportivo 31 de Octubre) — боливийский футбольный клуб из Ла Паса; 31 октября 1952 г. в Боливии национализировали горнорудную промышленность, а 31 октября 1954 г. там открылся съезд Первого национального конгресса трудящихся
- 31 октября 1517 г. Мартин Лютер прибил к двери церкви в Виттенберге 95 тезисов
- 31 октября 1811 г. был открыт Царскосельский лицей; среди первых воспитанников были не только Пушкин с Горчаковым, но и, например, полярный исследователь и адмирал Фёдор Матюшкин, тверской губернатор Александр Бакунин и российский посланник в Бразилии, Португалии и Нидерландах Сергей Ломоносов
- 31 октября 1905 г. была провозглашена Марковская республика, крестьянское самоуправление в Марковской волости Волоколамского уезда Московской губернии; казаки прекратили существование республики только в июне 1906 года
- 31 октября 2000 г. был остановлен последний компьютер, работавший под управлением операционной системы Multics, первый выпуск которой состоялся в 1965 году

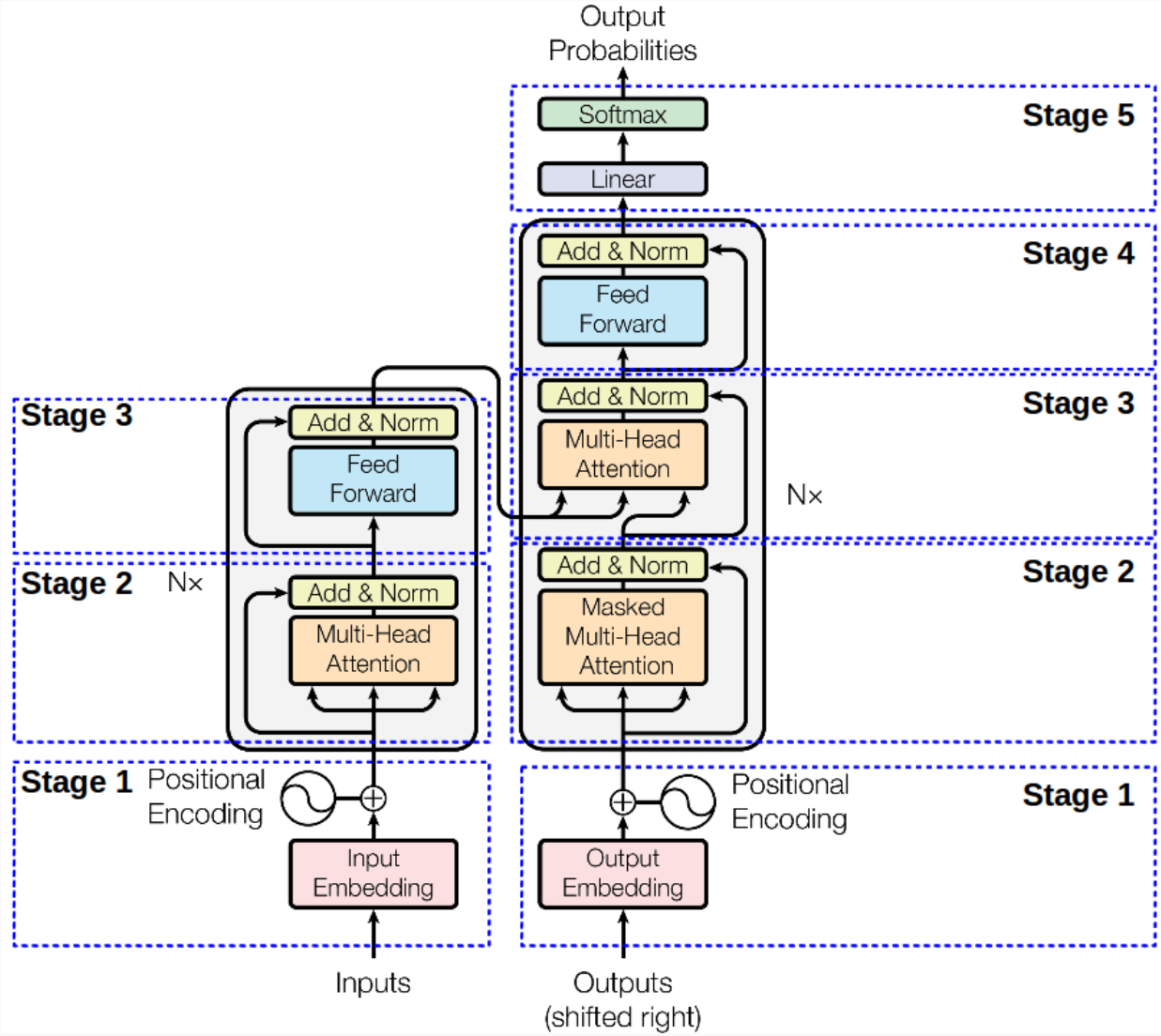
Ещё о трансформере

Transformer

- Мы изучали перевод на рекуррентных сетях и дошли до архитектуры Google NMT
- Но в 2017 году оказалось, что всё может быть ещё проще и интереснее
- Google: «Attention is all you need» (Vaswani et al., 2017)
- Основная идея – self-attention; оказывается очень плодотворной для всевозможных seq2seq задач
- Главная мотивация – попробовать всё-таки уйти от кодирования вектором постоянной длины

Transformer

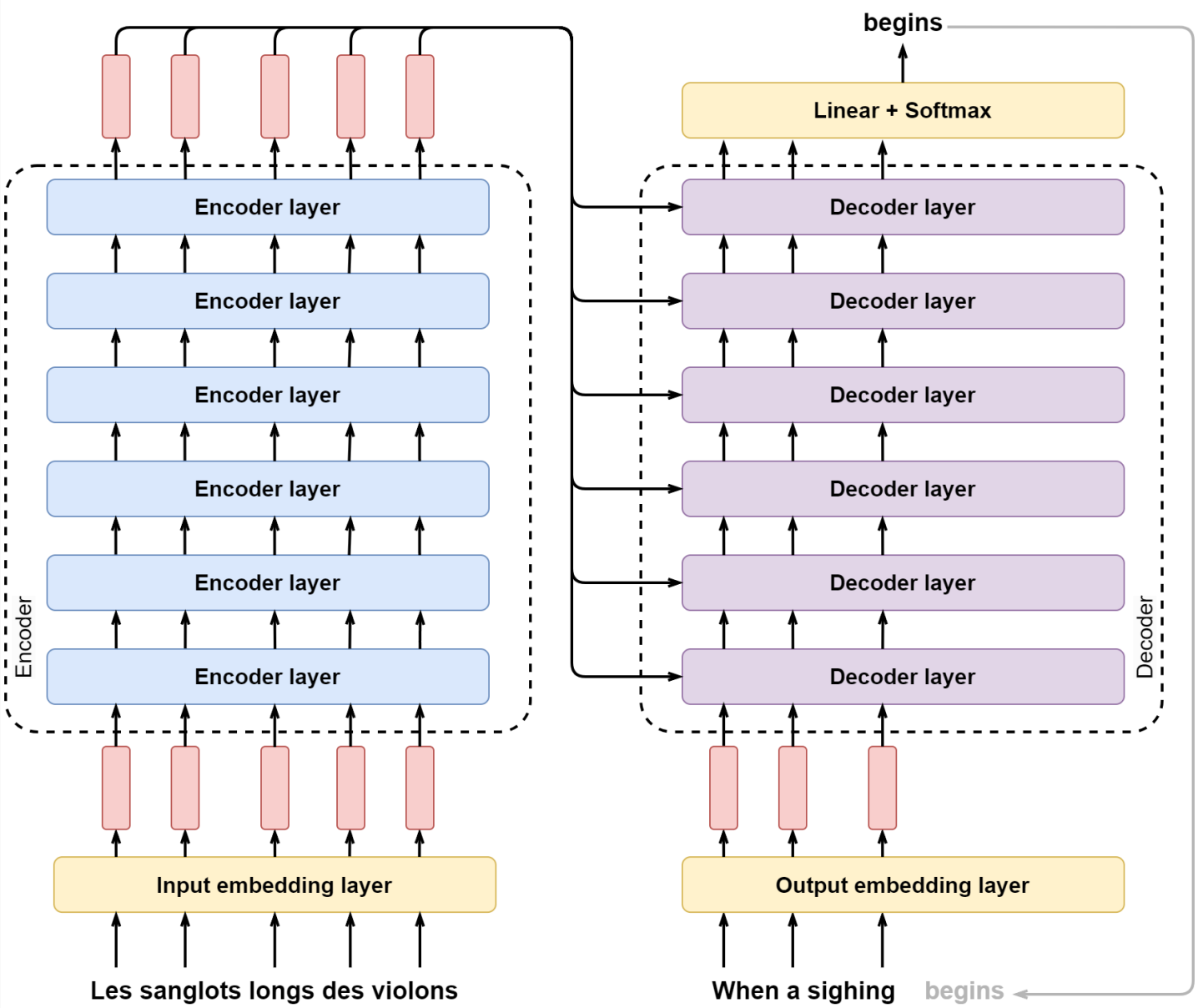
- Общая схема:



- Теперь подробнее...

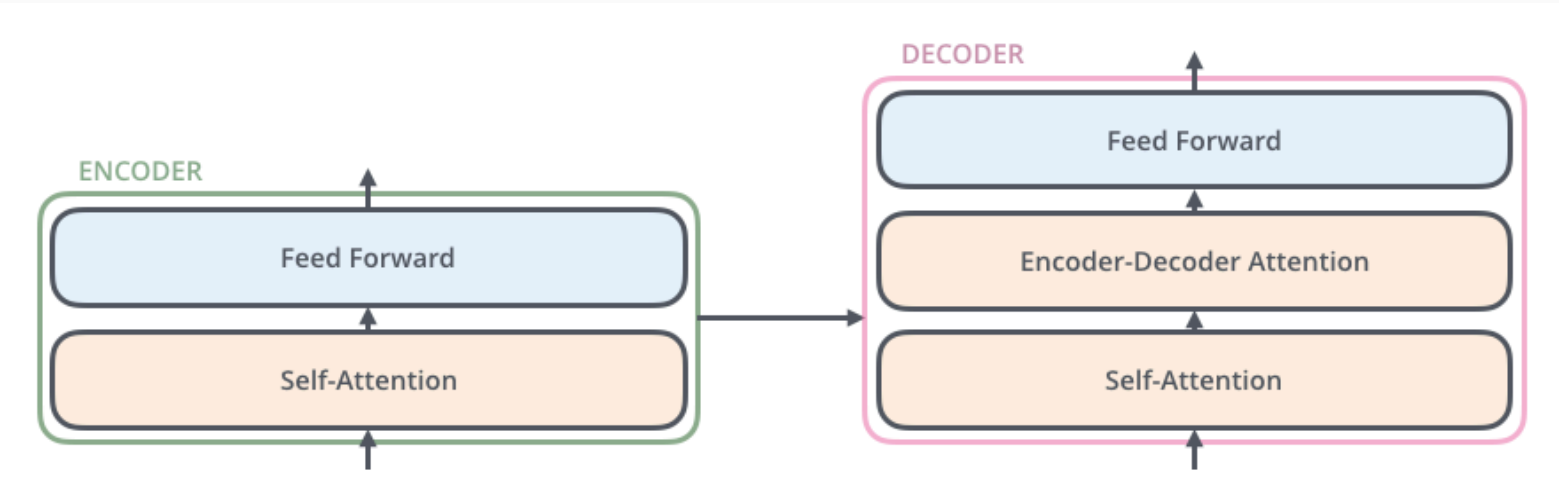
Transformer

- Суть, как и раньше, – encoder-decoder:



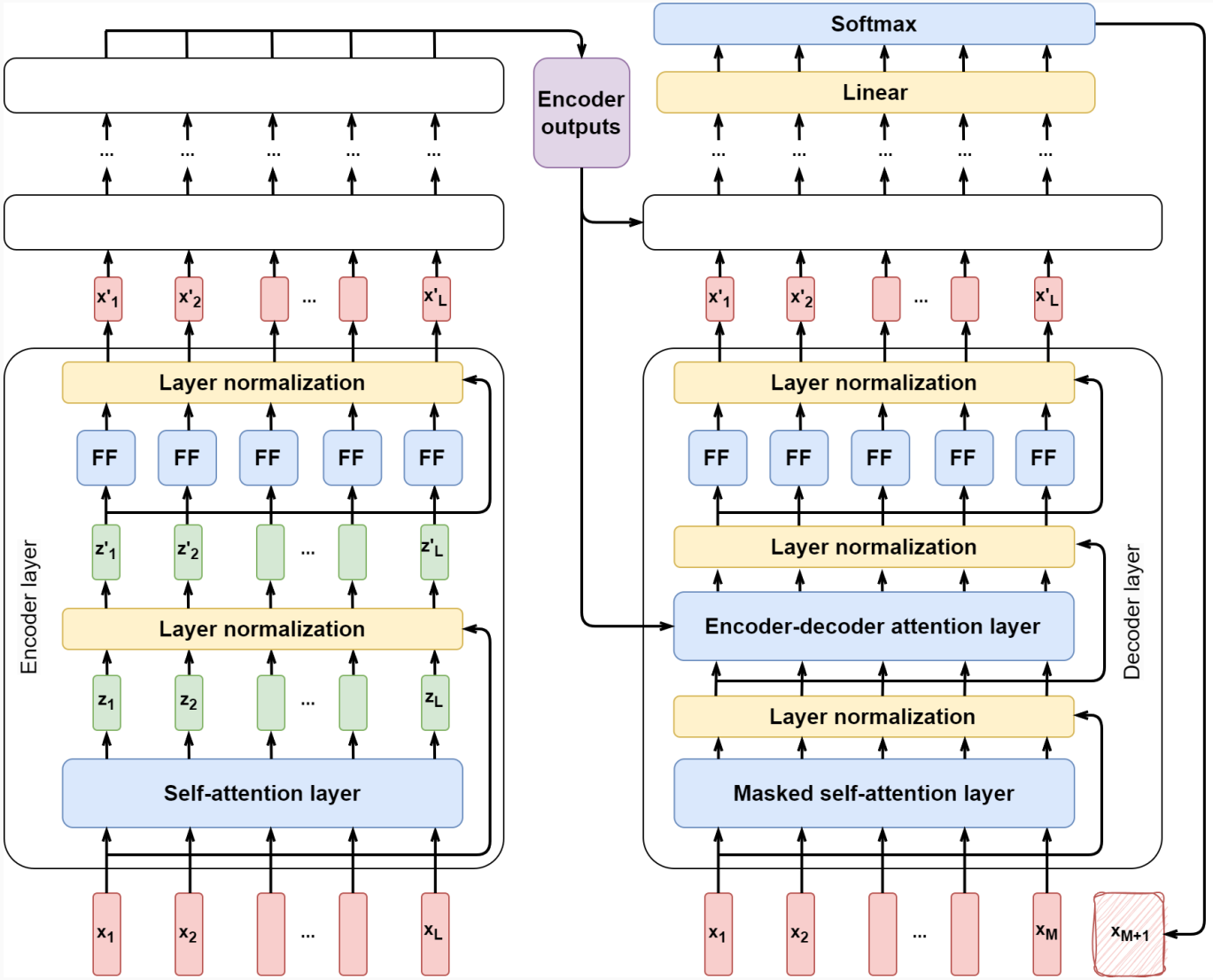
Transformer

- В каждом слое – слой self-attention, а потом feedforward layer, который независимо применяется к каждой позиции входа. У декодера ещё есть attention между ними:



Transformer

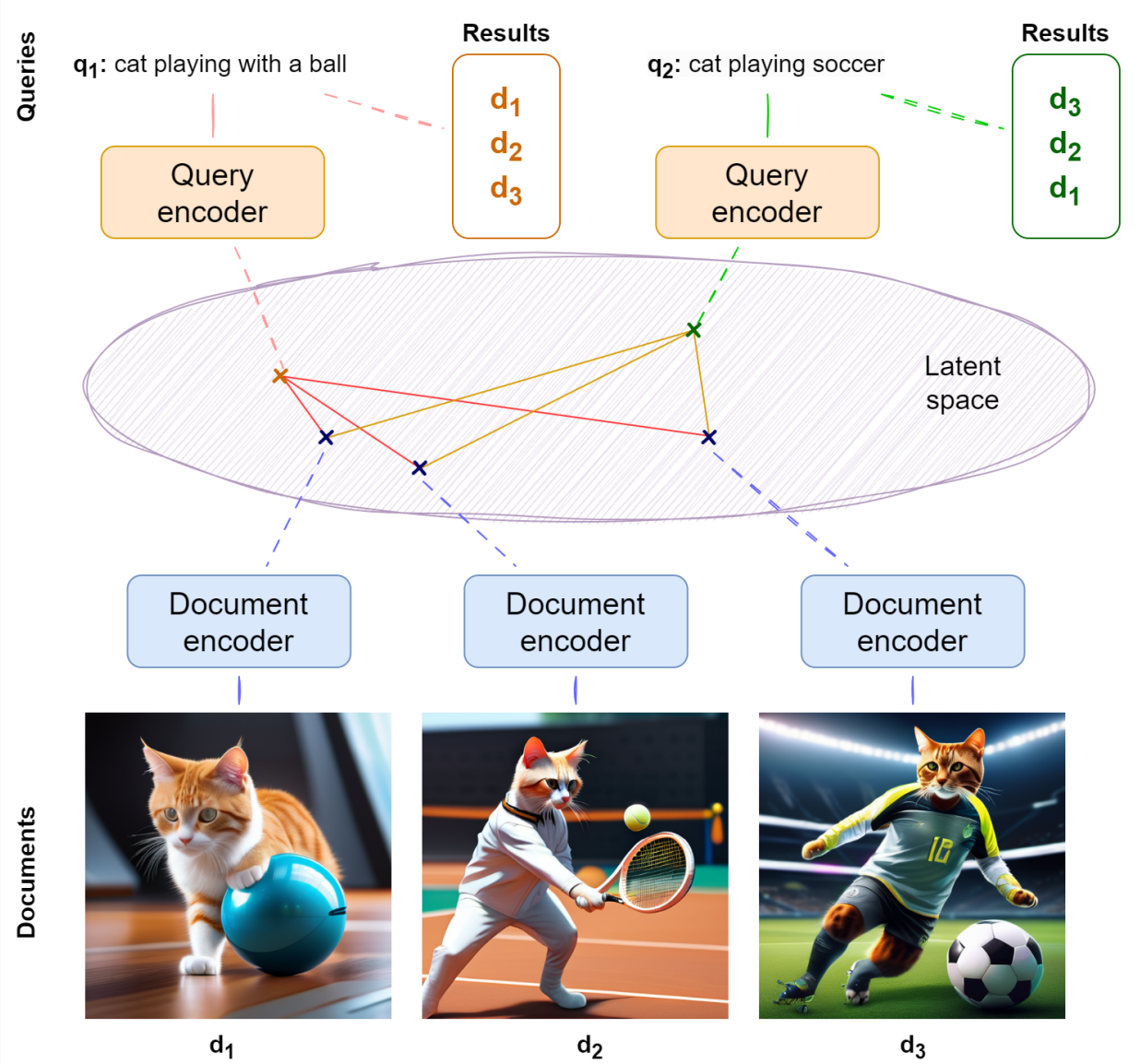
- Слова, естественно, представляются векторами, в feedforward слое всё параллельно:



- Но что же это такое — self-attention?

Self-attention

- Чтобы понять самовнимание (self-attention), начнём с информационного поиска (information retrieval)



Self-attention

- Запросы и документы отображаются в одно и то же латентное пространство (но разными кодировщиками — это могут быть вообще объекты разной природы)
- Запрос идёт через query encoder
- Документы (на иллюстрации это картинки) — через другой encoder, но в то же пространство
- Чтобы найти самые релевантные документы, мы ищем ближайших соседей в латентном пространстве среди документов; часто предполагают, что латентное пространство линейно, и расстояние там — это просто скалярное произведение $\mathrm{dist}(q, d) = \mathrm{Enc}_q(q)^\top \mathrm{Enc}_d(d)$.

Self-attention

- В самовнимании эта интуиция используется очень абстрактно. Слой получает на вход последовательность векторов $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L$, т.е. матрицу $X \in \mathbb{R}^{d \times L}$.
- Запросы, ключи и документы берутся из самих $\mathbf{x}_i$:
  - умножая $\mathbf{x}_1$ на матрицу весов W^Q, мы получаем вектор *запроса* $\mathbf{q}_1 = W^Q \mathbf{x}_1$; обратите внимание, что размерность q векторов запросов, $\mathbf{q}_i \in \mathbb{R}^q$, может отличаться (обычно меньше) от входной размерности d, $\mathbf{x}_i \in \mathbb{R}^d$;
  - умножая $\mathbf{x}_i$ на матрицу весов W^K, мы получаем векторы *ключей* $\mathbf{k}_i = W^K \mathbf{x}_i$ для $i = 1, \dots, L$; поскольку мы хотим, чтобы запросы и ключи находились в одном латентном пространстве, ключи имеют ту же размерность, что и запросы, $\mathbf{k}_i \in \mathbb{R}^q$, таким образом $W^V \in \mathbb{R}^{q \times d}$;
  - третья матрица весов W^V даёт векторы *значений* $\mathbf{v}_i = W^V \mathbf{x}_i$ для $i = 1, \dots, L$; это те *документы*, которые мы будем «извлекать» с помощью ключей $\mathbf{k}_i$; формально мы имеем другую размерность v для значений, $\mathbf{v}_i \in \mathbb{R}^v$ и $W^V \in \mathbb{R}^{v \times d}$; на практике обычно $v = q$.

Self-attention

- Затем мы выполняем поиск, вычисляя оценки внимания как скалярные произведения между запросами и документами
- Нормализуем $\mathbf{q}_i^\top \mathbf{v}_j$, деля на $\sqrt{q}$, и получить оценки через softmax, чтобы преобразовать их в вероятности:

$$\alpha_{ij} = \mathrm{softmax}\left(\frac{1}{\sqrt{q}}\mathbf{q}_i K^\top\right)_j,$$

где $K \in \mathbb{R}^{q \times L}$ — это все ключи, объединенные в матрицу, $K = W^K X$.

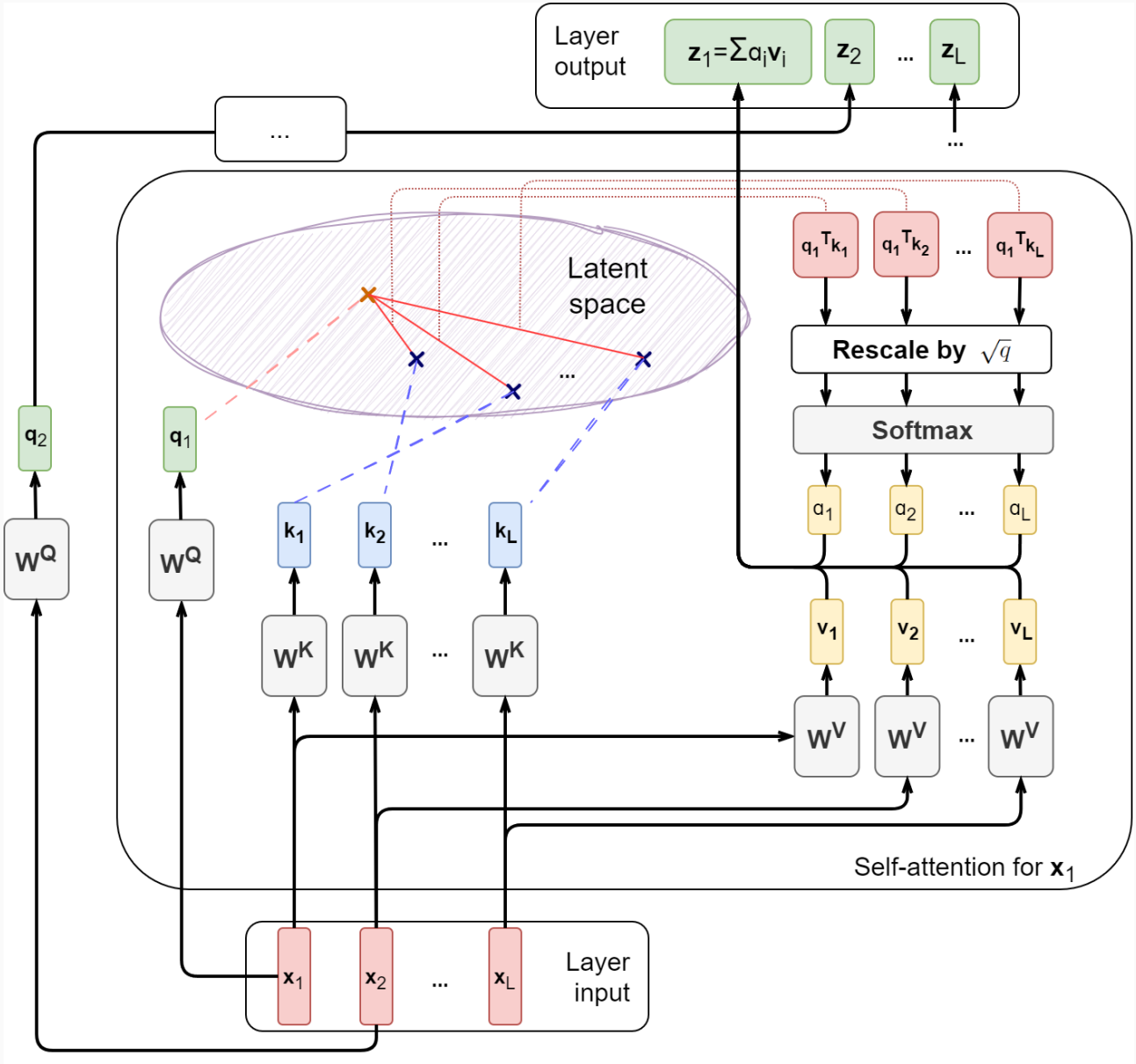
- Затем используем результат в качестве коэффициентов для выпуклой комбинации *значений* $\mathbf{v}_j$:

$$\mathbf{z}_i = \mathrm{softmax}\left(\frac{1}{\sqrt{q}}\mathbf{q}_i K^\top\right) V,$$

где $V \in \mathbb{R}^{q \times L}$ — значения (values), $V = W^V X$.

Self-attention

- Общая структура слоя самовнимания (self-attention):



Self-attention

- И это вся интуиция! Мы можем объединить вычисления каждого $\mathbf{z}_i$ в одну формулу в матричном виде:

$$Z = \mathrm{softmax}\left(\frac{1}{\sqrt{q}} Q K^\top\right) V.$$

- Но это только один способ «смотреть» на входные векторы; то, что мы сейчас определили, — это не полный слой самовнимания, а только одна *голова* (self-attention head)
- Можно распараллелить эти вычисления вдоль H различных голов, используя разные матрицы весов $W_1^Q, \dots W_H^Q$, $W_1^K, \dots W_H^K$ и $W_1^V, \dots W_H^V$. Это *многоголовое внимание* (multi-head attention)

Self-attention

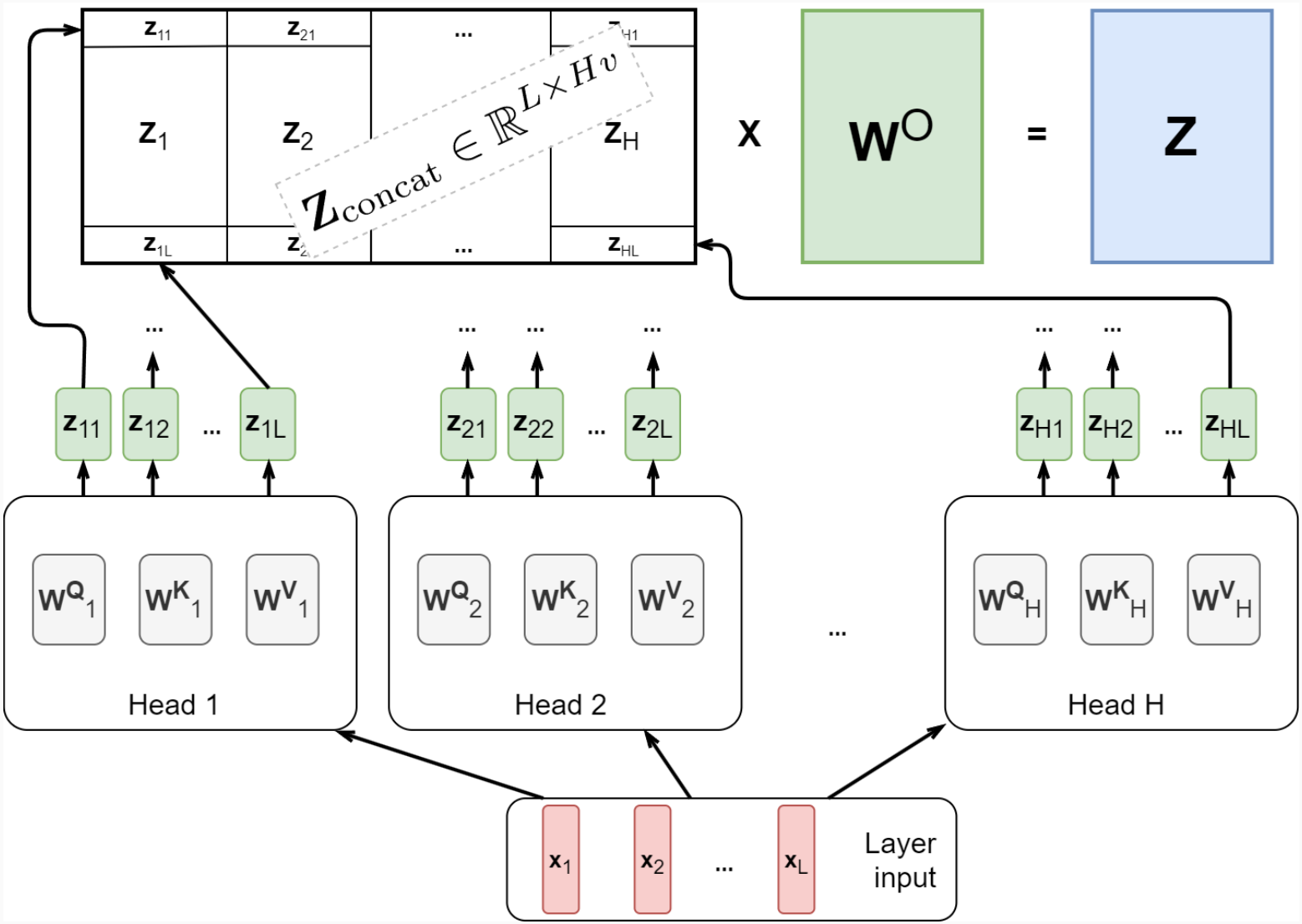
- После H параллельных голов мы получаем матрицы выходов $Z_1, Z_2, \ldots, Z_H$ размерности $v \times L$, объединяем их все в $Z_{\text{concat}} \in \mathbb{R}^{L \times Hv}$ и добавляем ещё одну матрицу весов $W^O \in \mathbb{R}^{Hv \times d}$, чтобы сжать результат обратно до необходимой размерности:

$$Z = \left(Z_{\text{concat}} W^O\right)^\top.$$

- С помощью W^O слой самовнимания может комбинировать представления, полученные из различных голов внимания; это также важный способ добавить гибкость и выразительность в архитектуру.

Self-attention

- Многоголовое внимание:



- Это самая важная часть, но это ещё не всё.

Декодер

- Декодер включает два дополнительных типа слоёв: *маскированное самовнимание* (masked self-attention) и *encoder-decoder attention*
- Маскированное внимание в основном означает, что, поскольку декодер работает автокоррективно, он не должен смотреть на токены, которые ещё не породил
- Для этого проще всего ввести всю последовательность и замаскировать будущие позиции внутри слоёв самовнимания
- Формально это означает, что мы задаем аргументы softmax как $-\infty$ для будущих токенов, что означает, что их веса внимания всегда будут нулевыми

Декодер

- Encoder-decoder attention — это вариация механизма самовнимания, которая опирается на результаты кодировщика
- Звучит сложно... но на самом деле это почти тривиальное изменение! Мы используем разные векторы в качестве входных данных в том же самовнимании:
  - для создания запросов используем векторы из предыдущего слоя, то есть текущие представления уже порождённых выходных токенов;
  - но для «документов» в задаче «поиска», то есть для векторов ключей и значений, мы используем векторы из декодера
- Неформально это значит, что мы выполняем «поиск» на выходе энкодера с запросами, состоящими из уже порождённых токенов
- Формально все размерности хорошо совпадают: в аргументе softmax есть L элементов для каждого вектора, но число запросов и, следовательно, число выходов совпадает с числом входов

Токенизация

- Мы используем вложения (embeddings), но как мы делим текст на токены?
- *Byte-pair encoding*: интересная идея, основанная на кодировании Хаффмана
- Давайте начнем с построения словаря:

$$\{\text{cat} : 10, \text{pet} : 12, \text{mat} : 5, \text{rat} : 8, \text{eats} : 4\}.$$

Original words:	cat		pet		mat		rat	eats
Frequencies:	10		12		5		8	4
Characters:	c	a	t	p	e	m	r	s
Frequencies:	10	27	39	12	16	5	8	4
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts
Frequencies:	10	**27**	12	12	5	8	4	4

Токенизация

- Затем разделим его на символы и подсчитаем пары символов:

$$\{\text{cat} : 10, \text{pet} : 12, \text{mat} : 5, \text{rat} : 8, \text{eats} : 4\},$$

$$\{c : 10, a : 27, t : 39, p : 12, e : 16, m : 5, r : 8, s : 4\},$$

$$\{ca : 10, at : 27, pe : 12, et : 12, ma : 5, ra : 8, ea : 4, ts : 4\}.$$

Original words:	cat		pet		mat		rat		eats
Frequencies:	10		12		5		8		4
Characters:	c	a	t	p	e	m	r	s	
Frequencies:	10	27	39	12	16	5	8	4	
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts	
Frequencies:	10	**27**	12	12	5	8	4	4	

Токенизация

- Теперь возьмем наиболее частую пару («*at*») и перекодируем её в новый символ (Z), который добавляется в словарь:

$$\{\mathrm{cZ}:10, \mathrm{pet}:12, \mathrm{mZ}:5, \mathrm{rZ}:8, \mathrm{eZs}:4\},$$
$$\{c:10, Z:27, t:12, p:12, e:16, m:5, r:8, s:4\},$$
$$\{cZ:10, pe:12, et:12, mZ:5, rZ:8, eZ:4, Zs:4\}.$$

Original words:	cat		pet		mat		rat		eats
Frequencies:	10		12		5		8		4
Characters:	c	a	t	p	e	m	r	s	
Frequencies:	10	27	39	12	16	5	8	4	
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts	
Frequencies:	10	**27**	12	12	5	8	4	4	

Токенизация

- Теперь мы можем выбрать новую наиболее частую пару — в данном случае *pe* или *et* — и заменить её на другой новый символ, например *Y*:

$$\{\mathrm{cZ}:10, \mathrm{pet}:12, \mathrm{mZ}:5, \mathrm{rZ}:8, \mathrm{eZs}:4\},$$
$$\{c:10, Z:27, t:12, p:12, e:16, m:5, r:8, s:4\},$$
$$\{cZ:10, pe:12, et:12, mZ:5, rZ:8, eZ:4, Zs:4\}.$$

Original words:	cat		pet		mat		rat	eats
Frequencies:	10		12		5		8	4
Characters:	c	a	t	p	e	m	r	s
Frequencies:	10	27	39	12	16	5	8	4
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts
Frequencies:	10	**27**	12	12	5	8	4	4

Positional encodings

- В трансформере каждый входной токен может “посмотреть” на любой другой *напрямую*, через любое число других токенов
- Это большое преимущество по сравнению с RNN!
- Но это значит, что у нас фиксированное ограниченное *context window* и, главное, поскольку веса внимания покрывают все токены равномерно, мы *теряем последовательность*: слой self-attention “не знает”, что некоторые токены стоят рядом, а некоторые далеко друг от друга
- Нужно как-то дать понять трансформеру, что у последовательности есть порядок, и это делается через *positional encodings*

Positional encodings

- Positional encodings are vectors added to the embedding that reflect where in the sequence the current token is.
- How could we encode the position? We could have a number that is increasing with the position but it is hard to get right:
  - if we use an increasing sequence like $1, 2, 3, ...$, it will not generalize to sequences longer than the usual length in the training set, and the network's behaviour would be much less well defined for large values;
  - if we use a given interval, say $[0, 1]$, and break it down into the necessary number of pieces, then the positional encoding would have no idea how many words are actually there between two tokens: the distance from 0 to $\frac{1}{2}$ could be 1 token or 100.

Positional encodings

- So we use something like positional numbers (where each digit forms a periodic function with different periods), but with continuous periodic functions, sine waves:

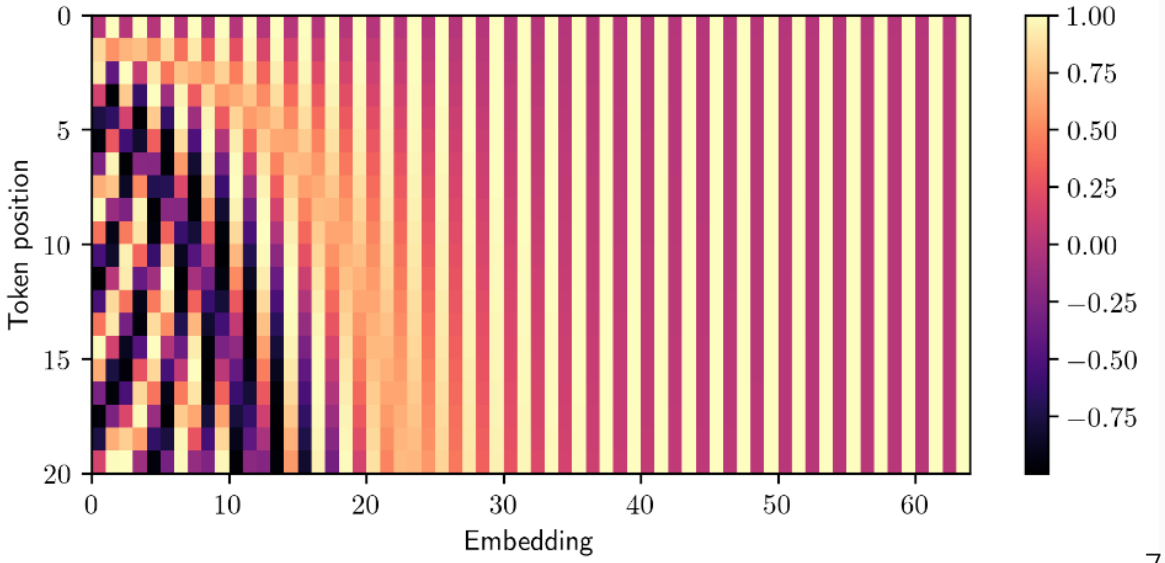
$$\mathrm{PE}\,(\mathrm{pos}, 2i) = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right), \quad \mathrm{PE}\,(\mathrm{pos}, 2i+1) = \cos\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right),$$

where pos is the token position, d is the embedding dimension.

Binary numbers

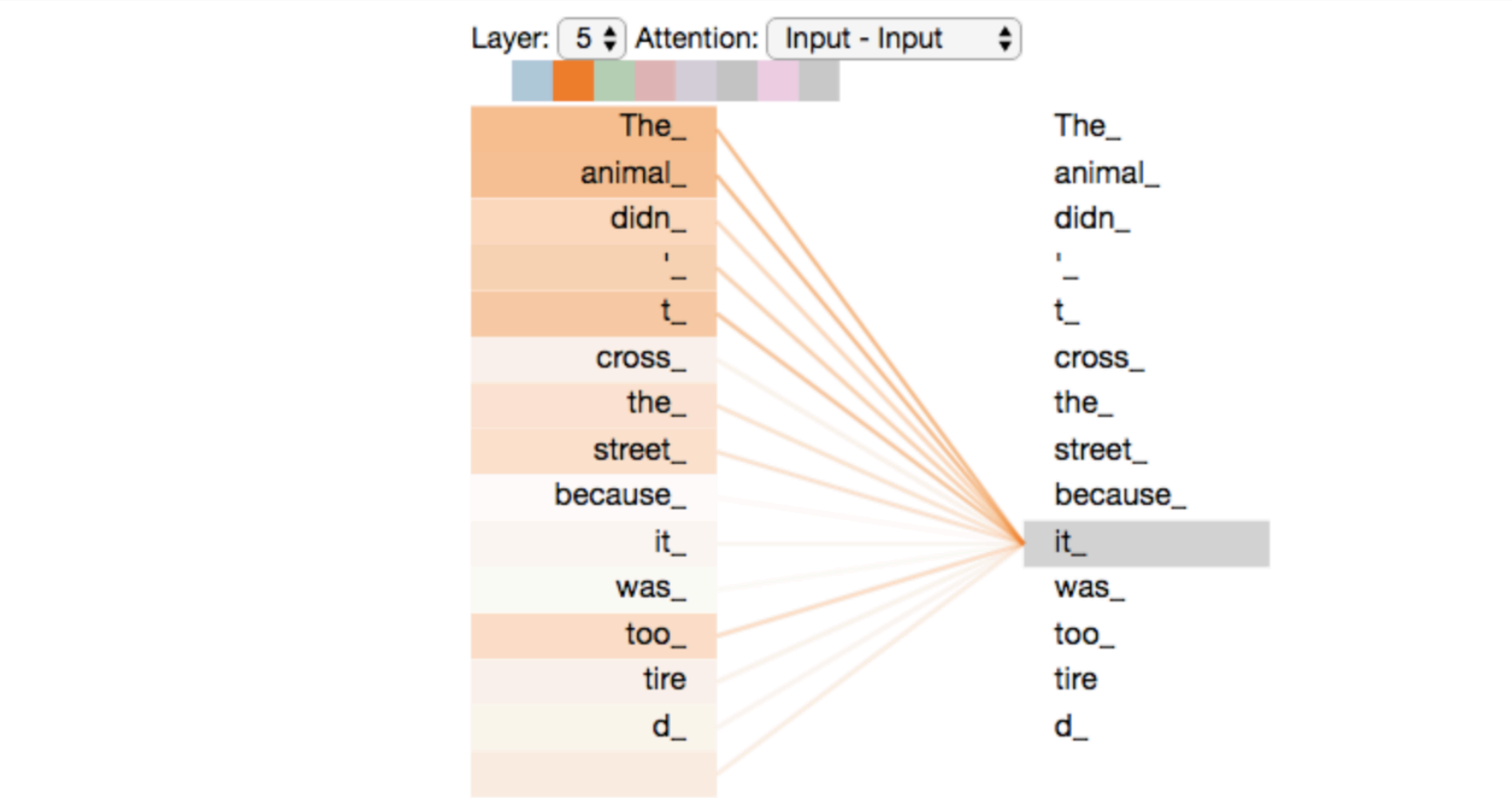
0000	1000
0001	1001
0010	1010
0011	1011
0100	1100
0101	1101
0110	1110
0111	1111

Positional encodings



Результаты

- В результате трансформер обучает веса того, как одни слова ”участвуют в обработке” других слов:



Результаты

- Бонус от self-attention – во-первых, вычислительный, во-вторых, сокращает пути между словами, в-третьих, потенциальная интерпретируемость:

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Результаты

- Работает лучше, обучается в сто раз быстрее:

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Семейство GPT

- Следующий шаг – OpenAI GPT (Generative Pretrained Transformer) (Radford et al., 2018)
- Используем Transformer в такой последовательности:
  - сначала обучаем обычную языковую модель

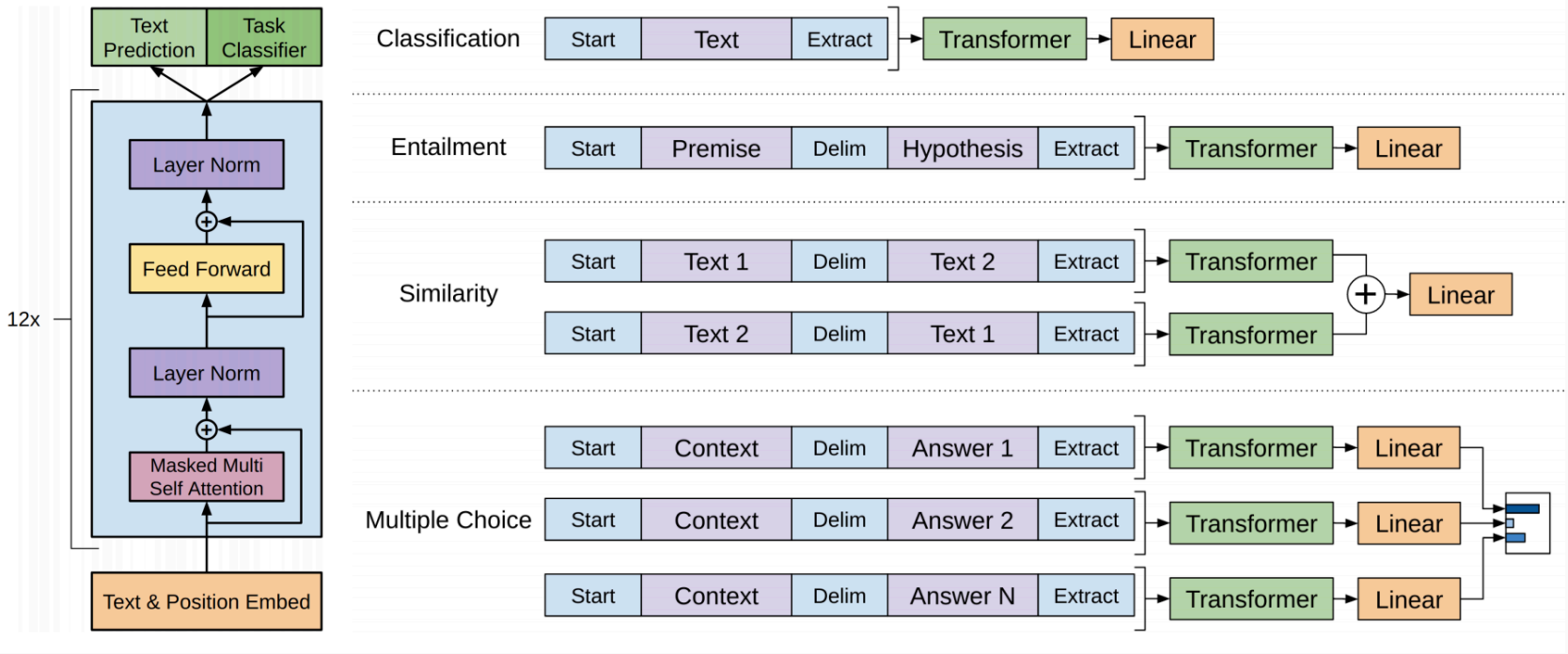
$$L_1(D) = \sum_i \log p(u_i \mid u_{i-k}, \dots, u_{i-1}; \theta);$$

  - потом добавляем новый линейный слой для каждой задачи и делаем fine-tuning уже с учителем:

$$L(C, D) = \sum_{(\mathbf{x}, y)} \log p(y \mid \mathbf{x}) + \lambda L_1(D).$$

Семейство GPT

- Идея в том, чтобы переиспользовать одну модель на много разных задач:



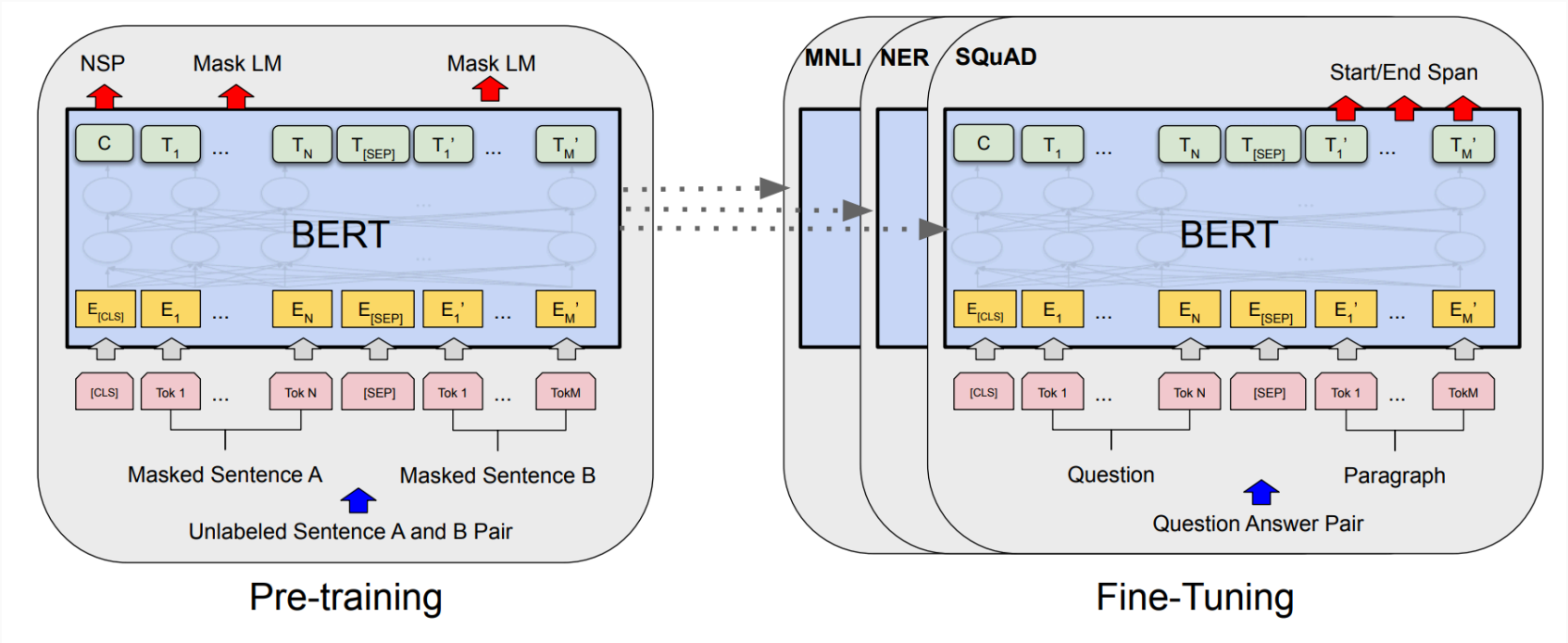
- Получают результаты гораздо лучше, чем были раньше, сразу для нескольких задач.

BERT-семейство

- (Devlin et al., 2018): BERT – Bidirectional Encoder Representations from Transformers
- Фактически тот же Transformer, но теперь двунаправленный, при условии и левого, и правого контекста во всех слоях.
- Т.е. та же языковая модель, но теперь работает с контекстом и слева, и справа.
- Или так не работает? Что делать?..

BERT-семейство

- Просто вместо обычной языковой модели будем маскировать случайные слова и пытаться их предсказывать.
- И вторая задача для pretraining – предсказание следующего предложения.



BERT-семейство

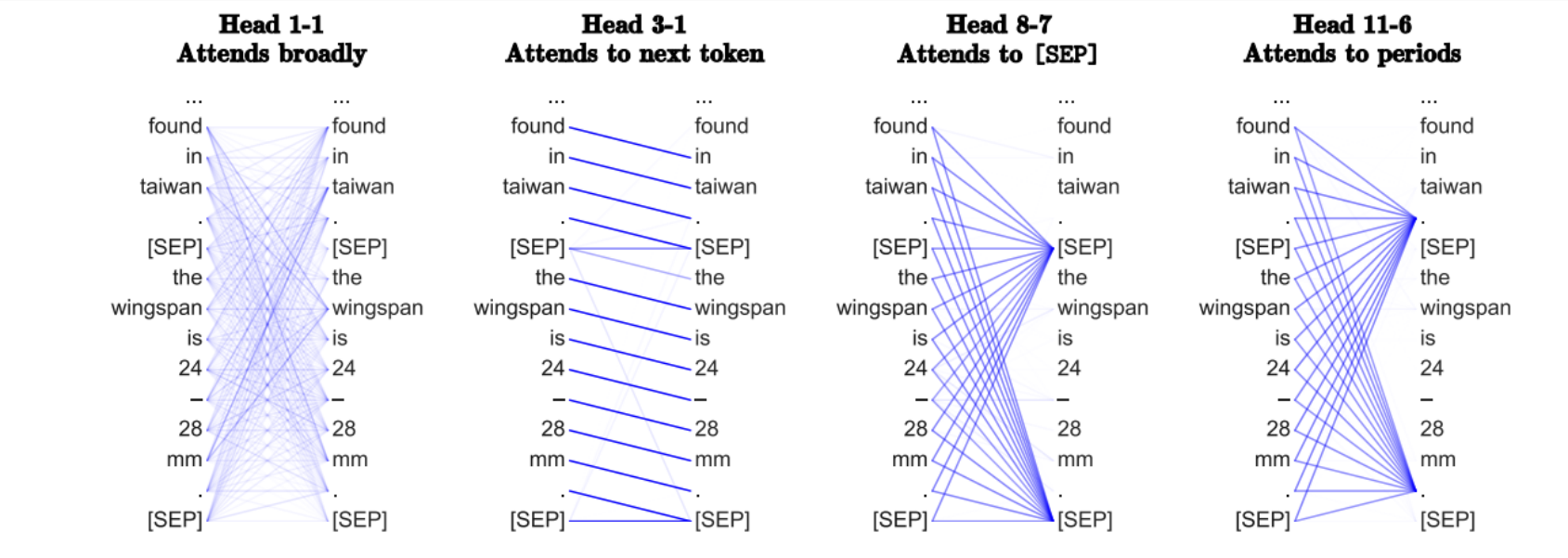
- И всё, в остальном обычный Transformer. Опять побили лучшие результаты для всех задач:

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

- Сейчас BERT – стандартная основа для conversational models и вообще чего угодно в NLP.

BERT-семейство

- Ещё важные детали о BERT:
  - wordpiece embeddings: используем фиксированный словарь подслов размером 30К, а слова делим на части: electrodynamics → electro# #dy# #nami# #cs
  - можно найти «головы», которые смотрят по-разному, дают разный attention:

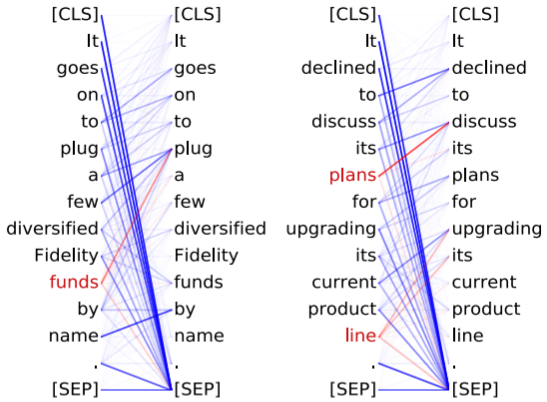


BERT-семейство

- И даже более грамматически (Clark et al., 2019):

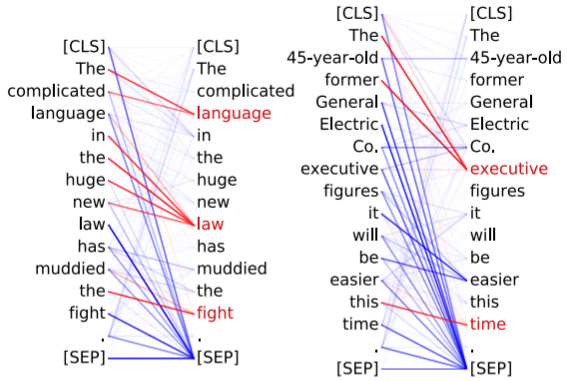
Head 8-10

- **Direct objects** attend to their verbs
- 86.8% accuracy at the `dobj` relation



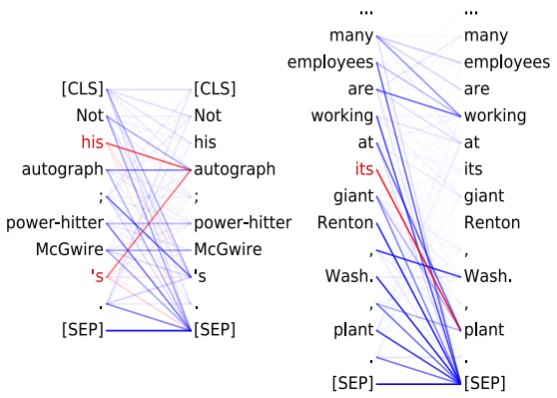
Head 8-11

- **Noun modifiers** (e.g., determiners) attend to their noun
- 94.3% accuracy at the `det` relation



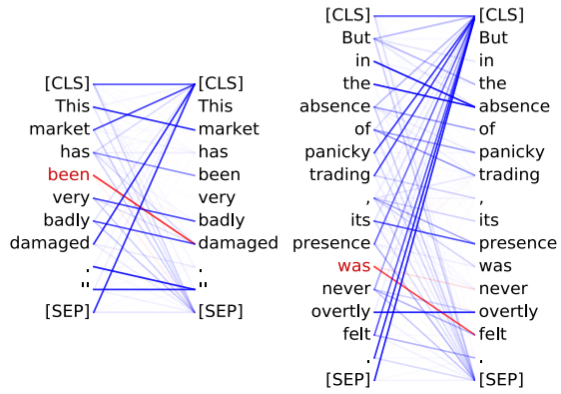
Head 7-6

- **Possessive pronouns** and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the `poss` relation



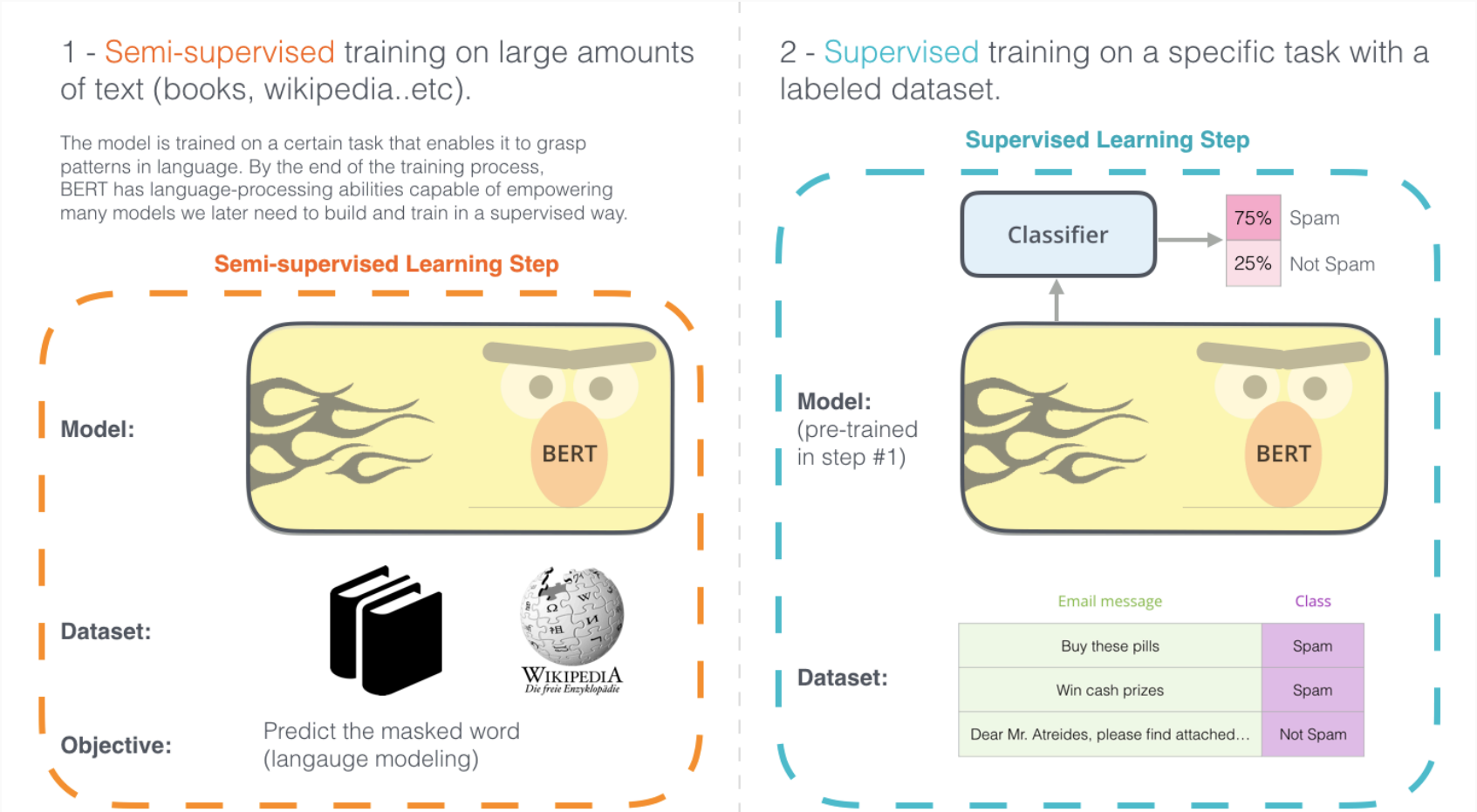
Head 4-10

- **Passive auxiliary verbs** attend to the verb they modify
- 82.5% accuracy at the `auxpass` relation



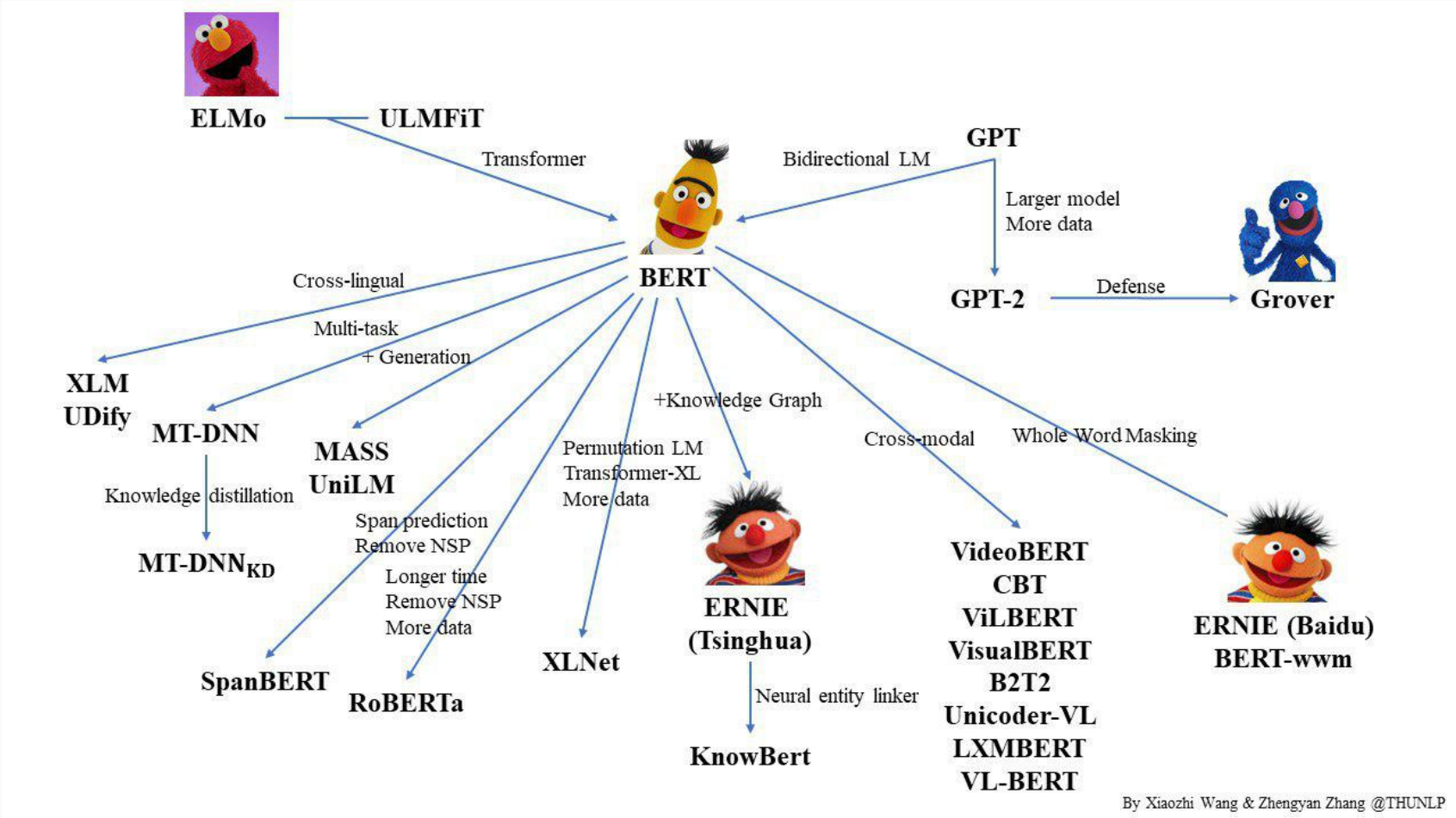
BERT-семейство

- Процесс обучения: сначала semi-supervised, потом fine-tuning



BERT-семейство

- BERT-подобных моделей уже очень много:

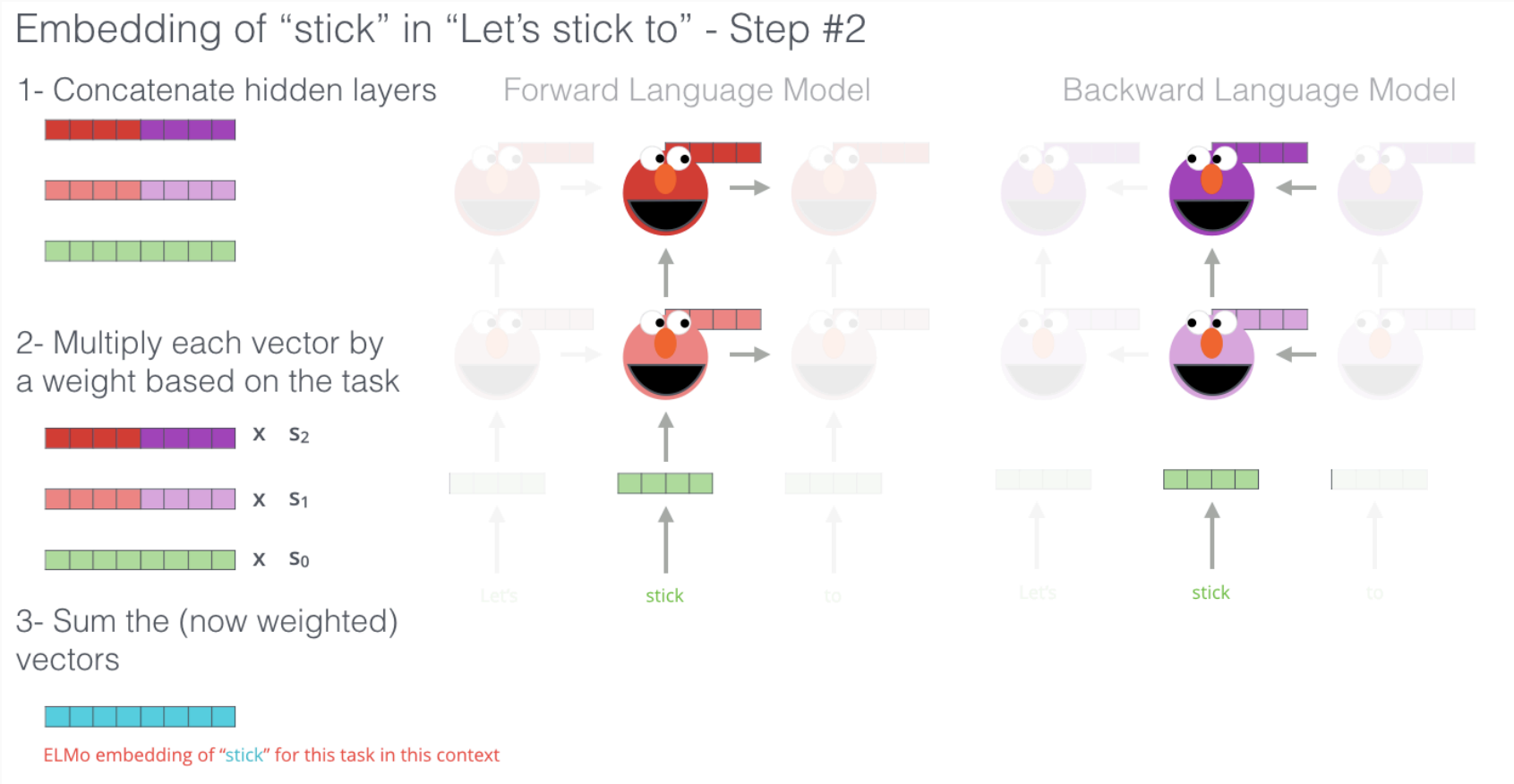


BERT-семейство

- ELMo: embeddings с контекстом; ведь у слова много смыслов:

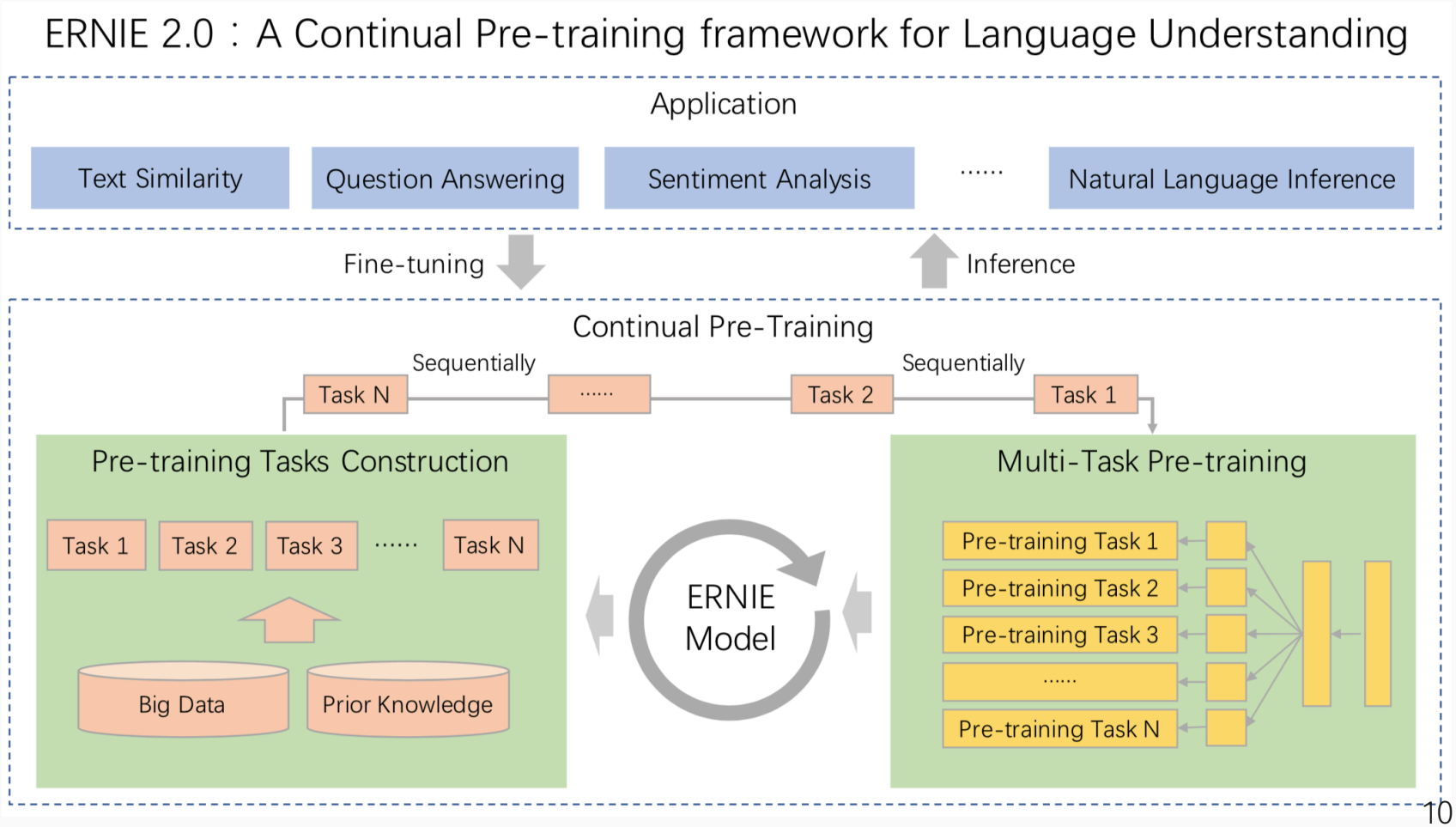
BERT-семейство

- ELMo берёт линейную комбинацию состояний двунаправленного LSTM с весами, зависящими от конкретной задачи:



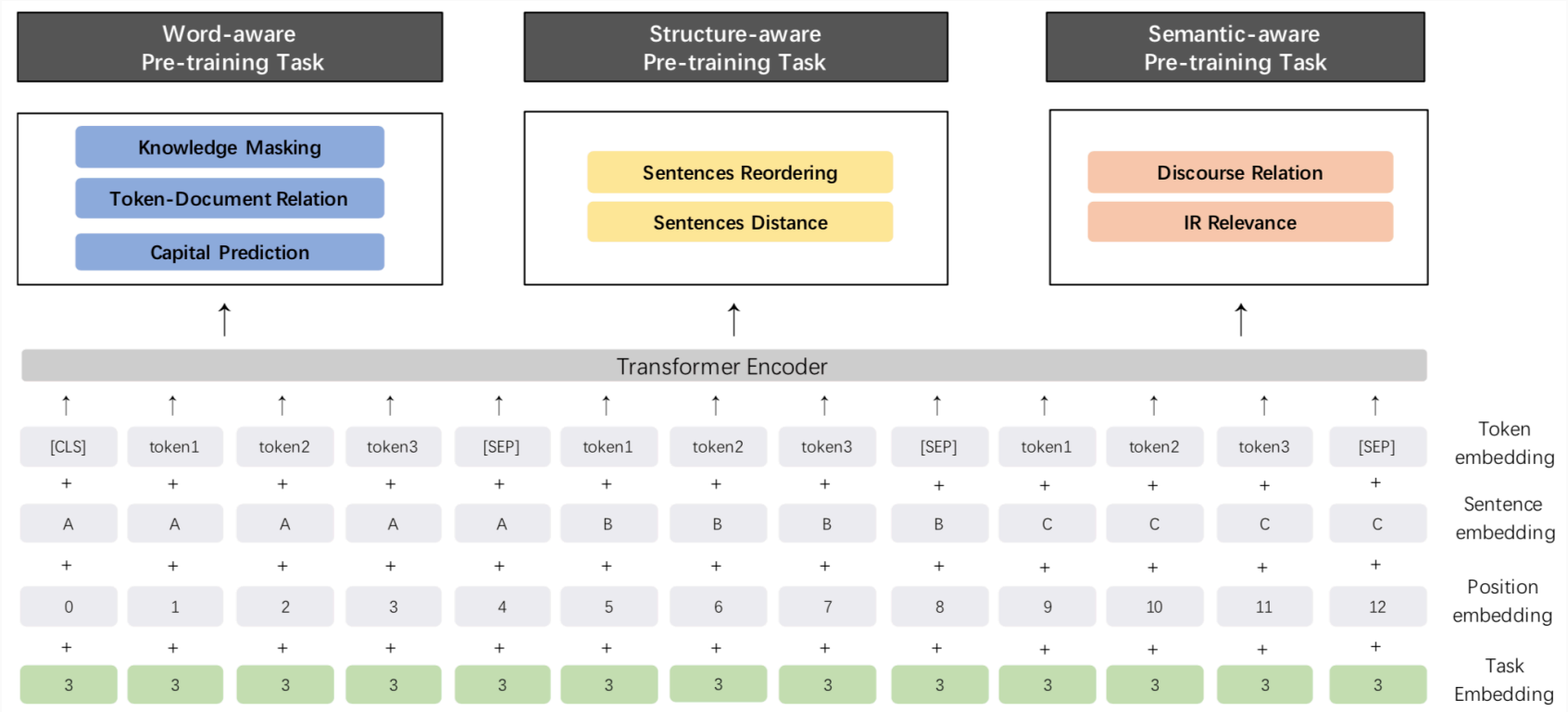
BERT-семейство

- ERNIE (Zhang et al., 2019) строит пайплайн предобучения на основе разных задач, к которым можно легко породить примеры:



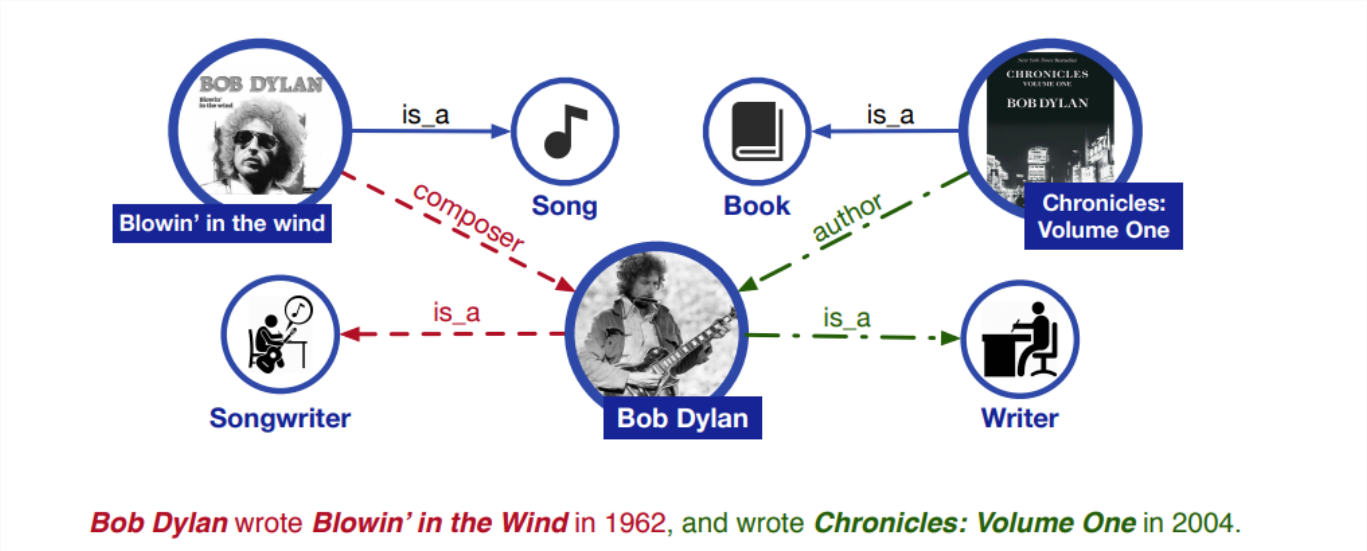
BERT-семейство

- Задачи типа найти в контексте что-то, предсказать капитализацию:



BERT-семейство

- Но не только, ещё добавляем знания из knowledge graph:



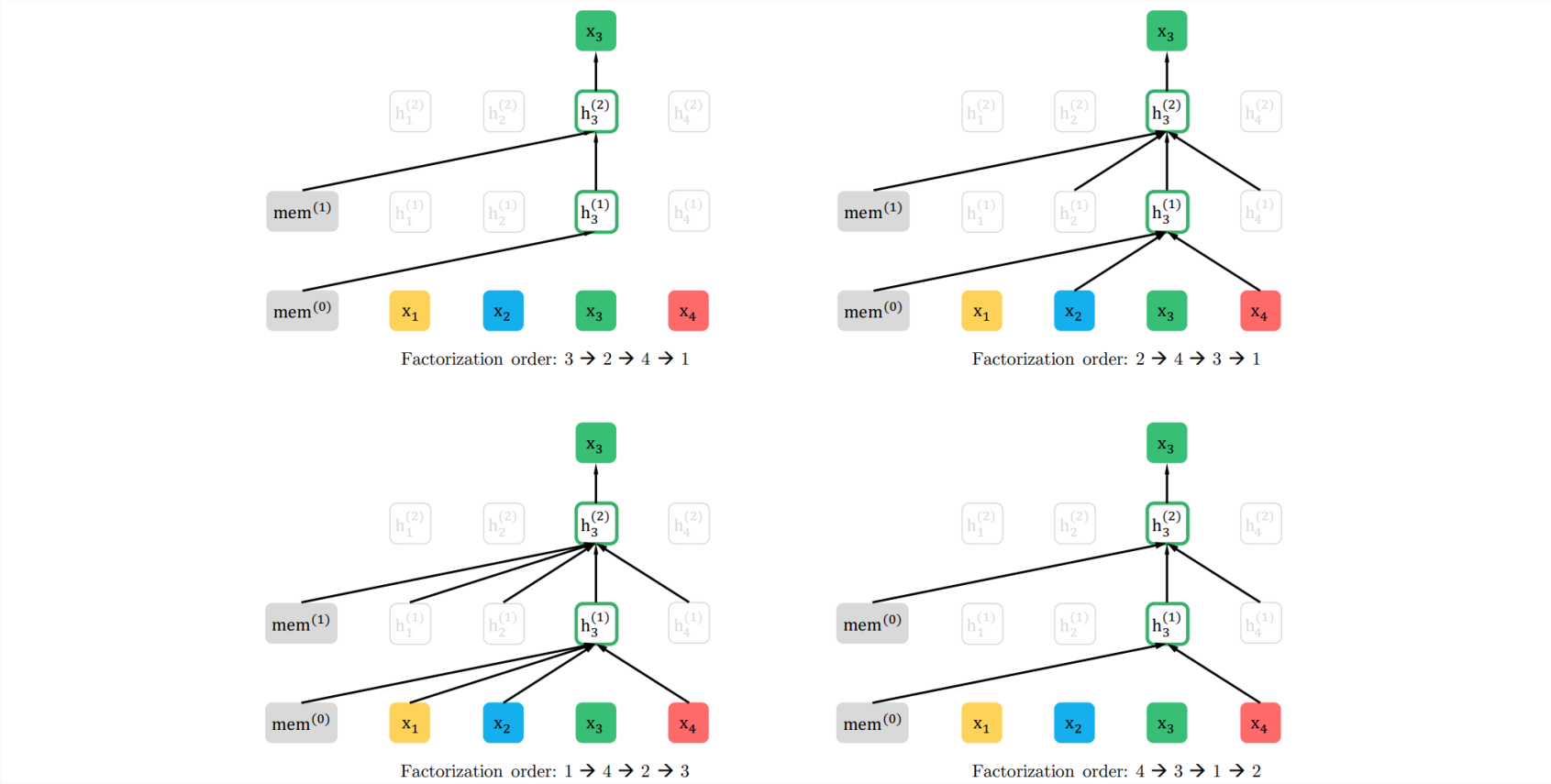
- В реальности это значит, что маски по-умному строятся:

```
- Learned by BERT :[mask] Potter is a series [mask] fantasy novel [mask] by J. [mask] Rowli

- Learned by ERNIE:Harry Potter is a series of [mask] [mask] written by [mask] [mask] [mask
```

BERT-семейство

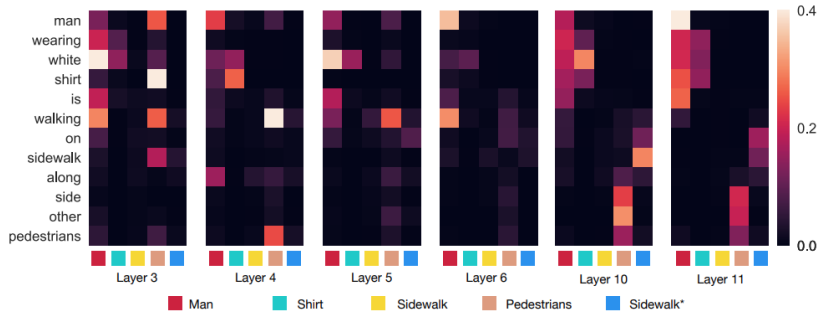
- XLNet (Yang et al., 2019): BERT предсказывает маскированные слова, а XLNet пытается предсказывать данное слово сразу во всех перестановках входного предложения:



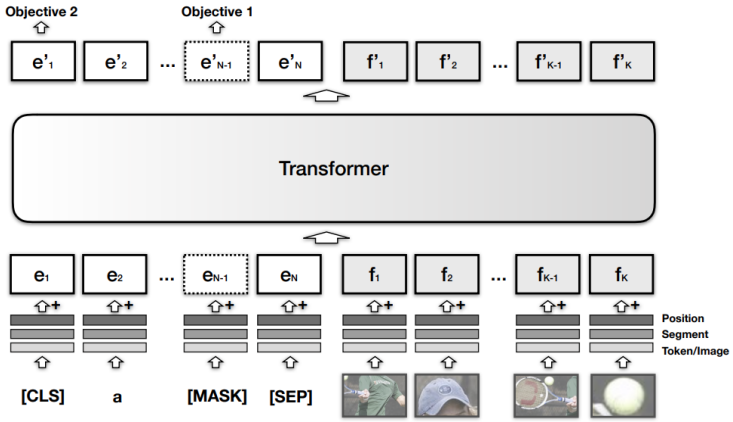
- На самом деле сэмплируют при обучении один случайный порядок, но параметры остаются общими

BERT-семейство

- Visual BERT (Li et al., 2019) – Show, Attend, and Tell, только через BERT:

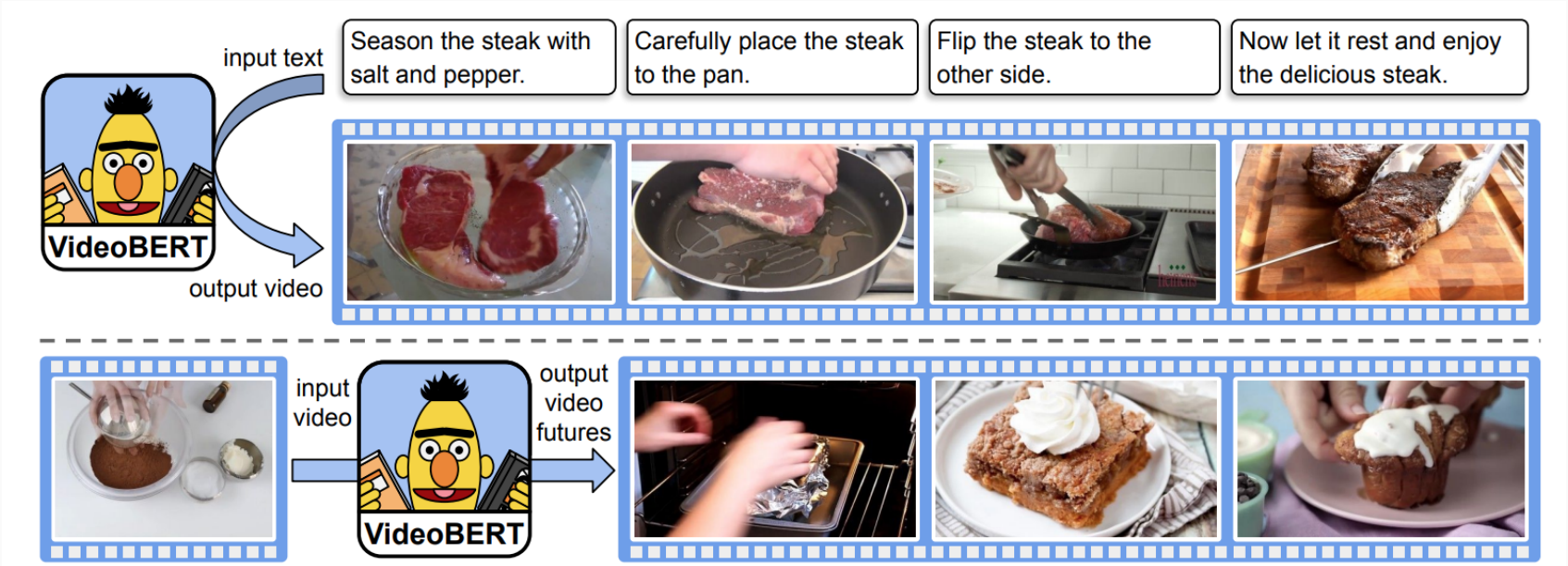


A person hits a ball with a tennis racket



BERT-семейство

- VideoBERT (Sun et al., 2019) – давайте к предложениям ещё видео приложим:



BERT-семейство

- В результате получается, например, video captioning:


GT: add some chopped basil leaves into it

VideoBERT: chop the basil and add to the bowl

S3D: cut the tomatoes into thin slices


GT: cut the top off of a french loaf

VideoBERT: cut the bread into thin slices

S3D: place the bread on the pan


GT: cut yu choy into diagonally medium pieces

VideoBERT: chop the cabbage

S3D: cut the roll into thin slices


GT: remove the calamari and set it on paper towel

VideoBERT: fry the squid in the pan

S3D: add the noodles to the pot

BERT-семейство

- Следующая новость – GPT-2 (Radford et al., 2019).
- Это тот же GPT по сути, но:
  - гораздо больше размером: 1.5B параметров (у GPT было 110M, у BERT 340M);
  - обученный на огромном датасете: WebText – все ссылки с Reddit с кармой ≥ 3, 40GB текста;
  - без всякого fine-tuning и вообще без supervision, проверялось качество в zero-shot контексте.

BERT-семейство

- Результаты:

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**	**40.31**	**0.97**	**1.02**	22.05	44.575
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**	42.16

- Конечно, гораздо хуже специализированных моделей, но работает прямо само по себе, из небольшого контекста.
- `https://www.reddit.com/user/GPT-2_Bot/`
- `https://openai.com/blog/better-language-models/#sample1`

BERT-семейство

- Вопросы (есть на BERT хорошая модель, Alberti et al., 2019):

Question	Generated Answer	Correct	Probability
Who wrote the book the origin of species?	Charles Darwin	✓	83.4%
Who is the founder of the ubuntu project?	Mark Shuttleworth	✓	82.0%
Who is the quarterback for the green bay packers?	Aaron Rodgers	✓	81.1%
Panda is a national animal of which country?	China	✓	76.8%
Who came up with the theory of relativity?	Albert Einstein	✓	76.4%
When was the first star wars film released?	1977	✓	71.4%
What is the most common blood type in sweden?	A	✗	70.6%
Who is regarded as the founder of psychoanalysis?	Sigmund Freud	✓	69.3%
Who took the first steps on the moon in 1969?	Neil Armstrong	✓	66.8%
Who is the largest supermarket chain in the uk?	Tesco	✓	65.3%
What is the meaning of shalom in english?	peace	✓	64.0%
Who was the author of the art of war?	Sun Tzu	✓	59.6%
Largest state in the us by land mass?	California	✗	59.2%
Green algae is an example of which type of reproduction?	parthenogenesis	✗	56.5%
Vikram samvat calender is official in which country?	India	✓	55.6%
Who is mostly responsible for writing the declaration of independence?	Thomas Jefferson	✓	53.3%
What us state forms the western boundary of montana?	Montana	✗	52.3%
Who plays ser davos in game of thrones?	Peter Dinklage	✗	52.1%
Who appoints the chair of the federal reserve system?	Janet Yellen	✗	51.5%
State the process that divides one nucleus into two genetically identical nuclei?	mitosis	✓	50.7%
Who won the most mvp awards in the nba?	Michael Jordan	✗	50.2%
What river is associated with the city of rome?	the Tiber	✓	48.6%
Who is the first president to be impeached?	Andrew Johnson	✓	48.3%
Who is the head of the department of homeland security 2017?	John Kelly	✓	47.0%
What is the name given to the common currency to the european union?	Euro	✓	46.8%
What was the emperor name in star wars?	Palpatine	✓	46.5%
Do you have to have a gun permit to shoot at a range?	No	✓	46.4%
Who proposed evolution in 1859 as the basis of biological development?	Charles Darwin	✓	45.7%
Nuclear power plant that blew up in russia?	Chernobyl	✓	45.7%
Who played john connor in the original terminator?	Arnold Schwarzenegger	✗	45.2%

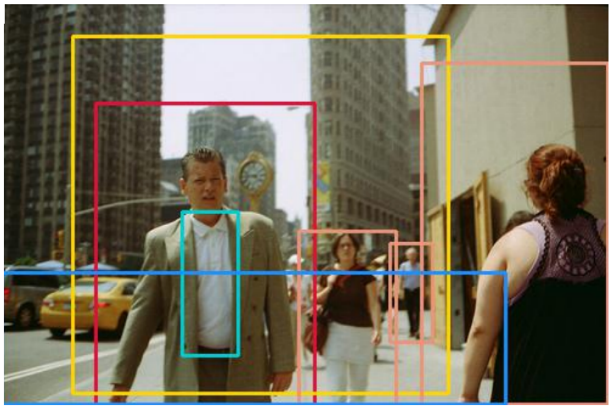
Vision Transformers

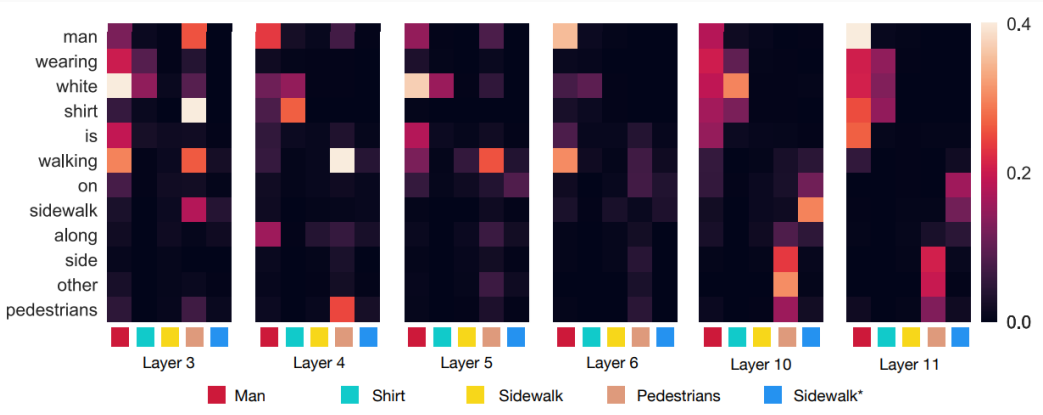
Visual BERT

- Images are not sequences, right? Well...
- Visual BERT: let's model captions and images jointly
- Use a fixed pretrained object detection system such as Faster R-CNN, cut the objects out, embed them into vectors via CNNs and special positional embeddings, and feed into a single Transformer with the caption

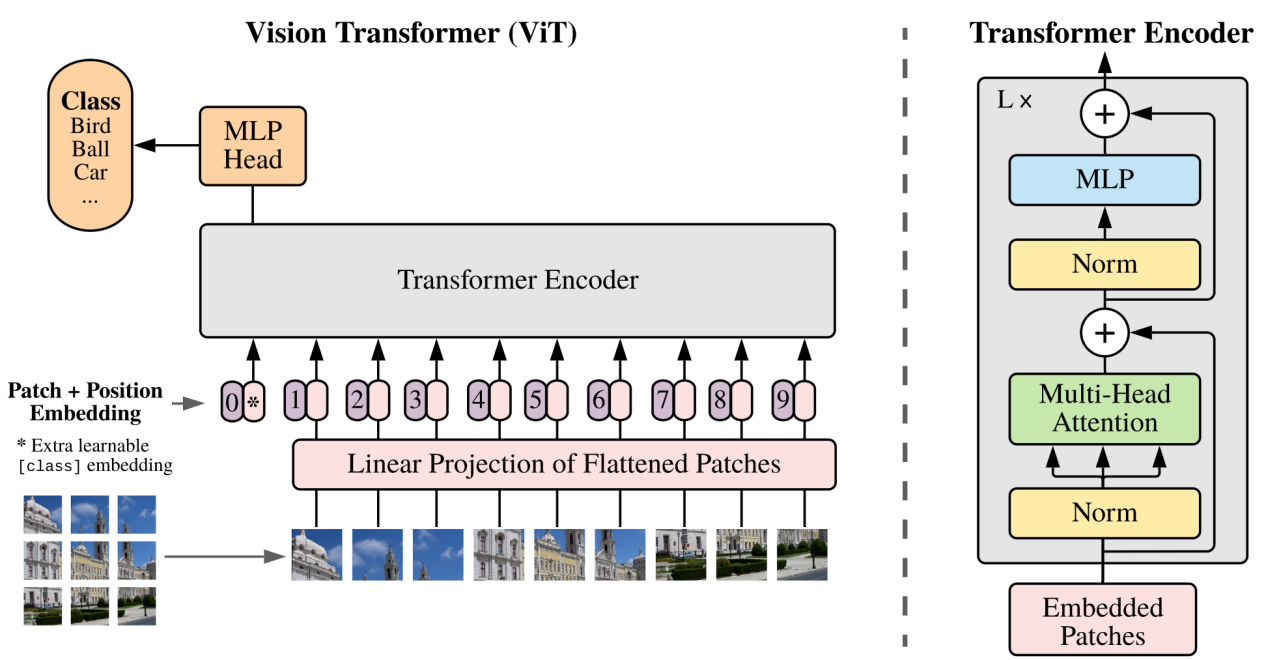
Visual BERT

- Pretraining: masked language modeling + sentence-image prediction (distinguish whether a given caption matches the image)
- Sample attention heads show how words from the caption actually do attend to the corresponding objects



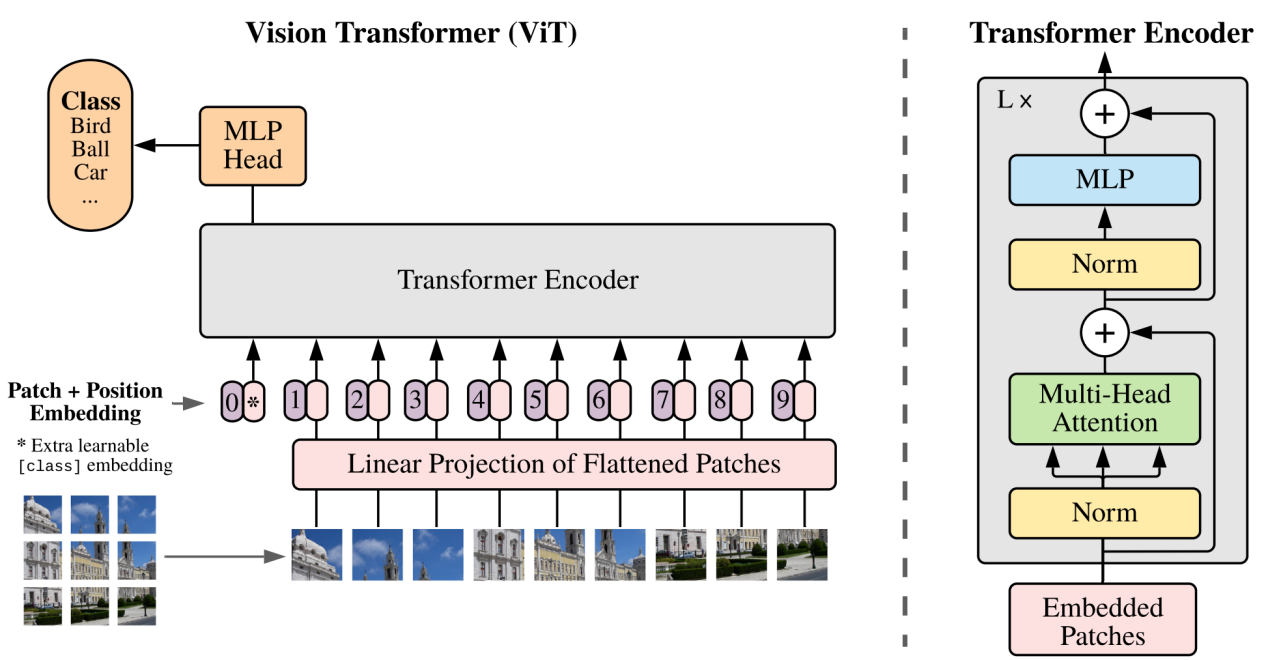


Visual Transformer (ViT)



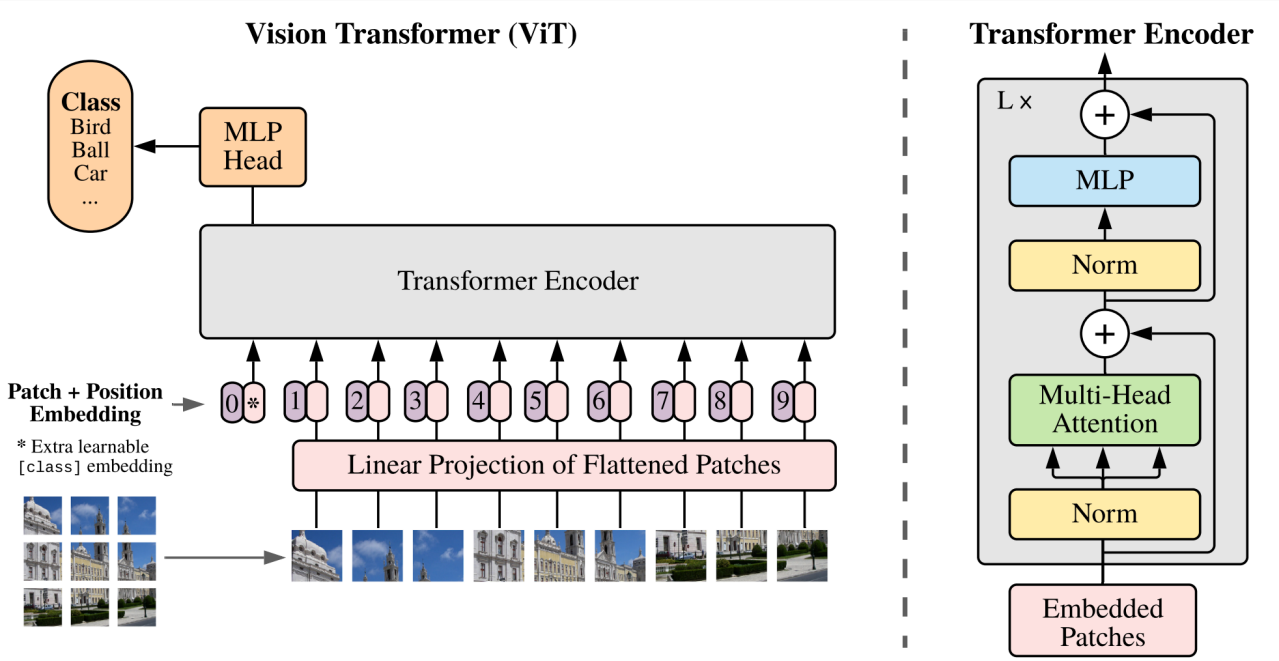
- The most successful Transformer-based architecture for images: *Vision Transformer* (ViT; Dosovitsky et al., 2020)
- ViT is again a straighforward modification of BERT, but now there is no text at the input, only image-based tokens

Visual Transformer (ViT)



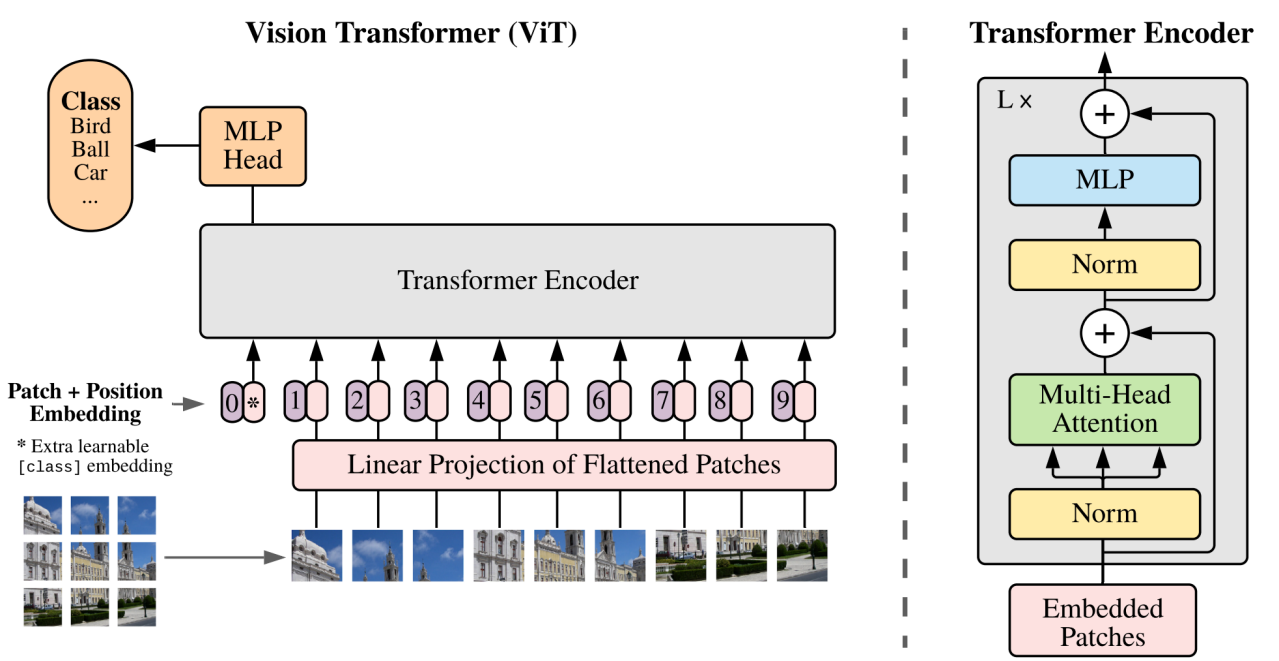
- The input image is cut into small patches: an $H \times W$ image with C channels $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ becomes a sequence of patches $\mathbf{x}_p \in \mathbb{R}^{N \times P^2 \times C}$, where $N = HW/P^2$ is the number of $P \times P$ patches that fit

Visual Transformer (ViT)



- Patches are turned into embeddings via a simple linear projection, and then the sequence is fed into a Transformer encoder just like BERT
- Pretraining: masked patch modeling just like BERT, replacing half of the input embeddings with the same learnable `[mask]` embedding

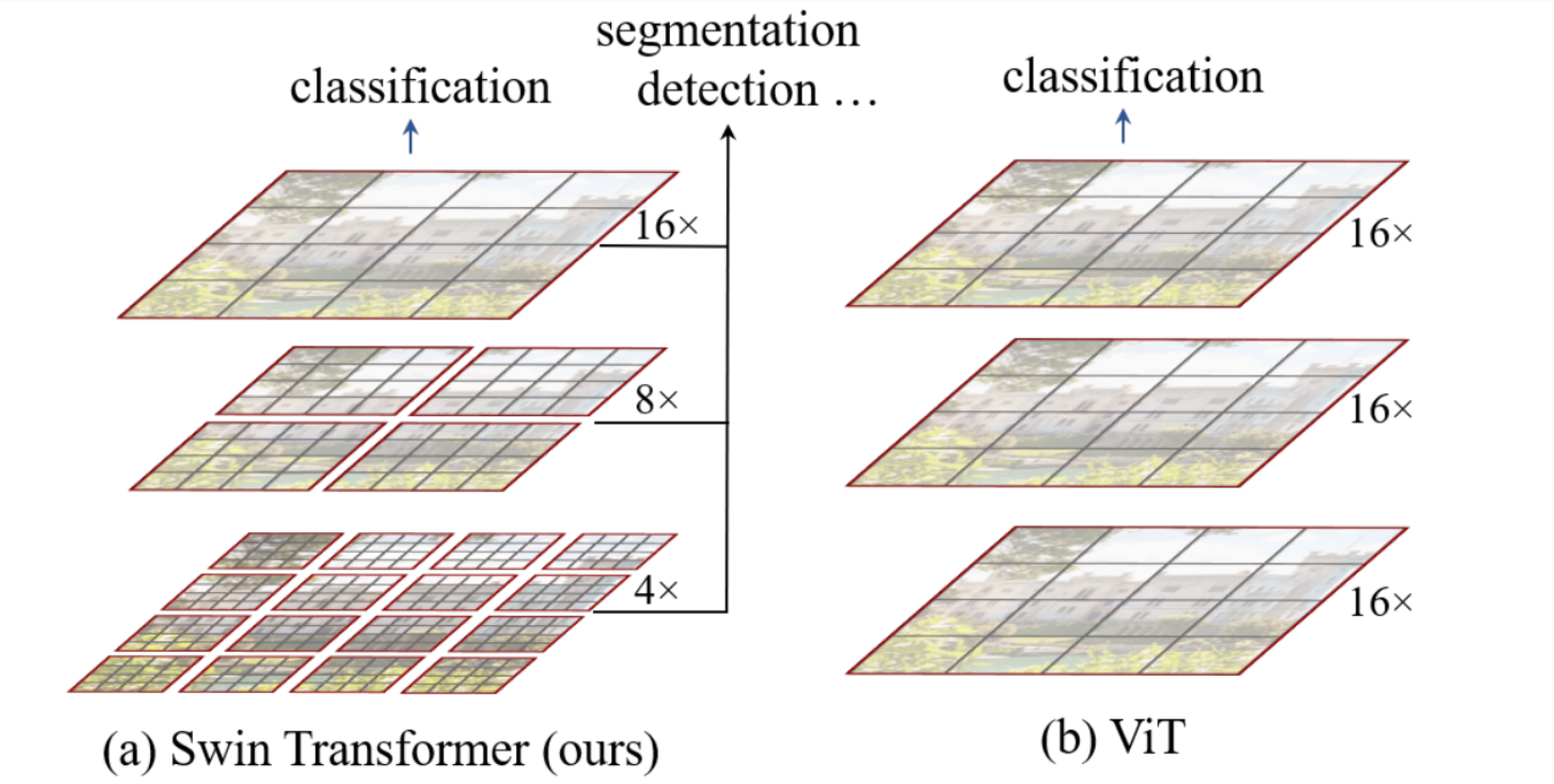
Visual Transformer (ViT)



- ViT uses positional encodings to add information about the sequence; interestingly, it is the same positional encoding as in the original Transformer even though the geometry is 2D now
- Dosovitsky et al. experimented with 2D encodings, and it did not make any difference

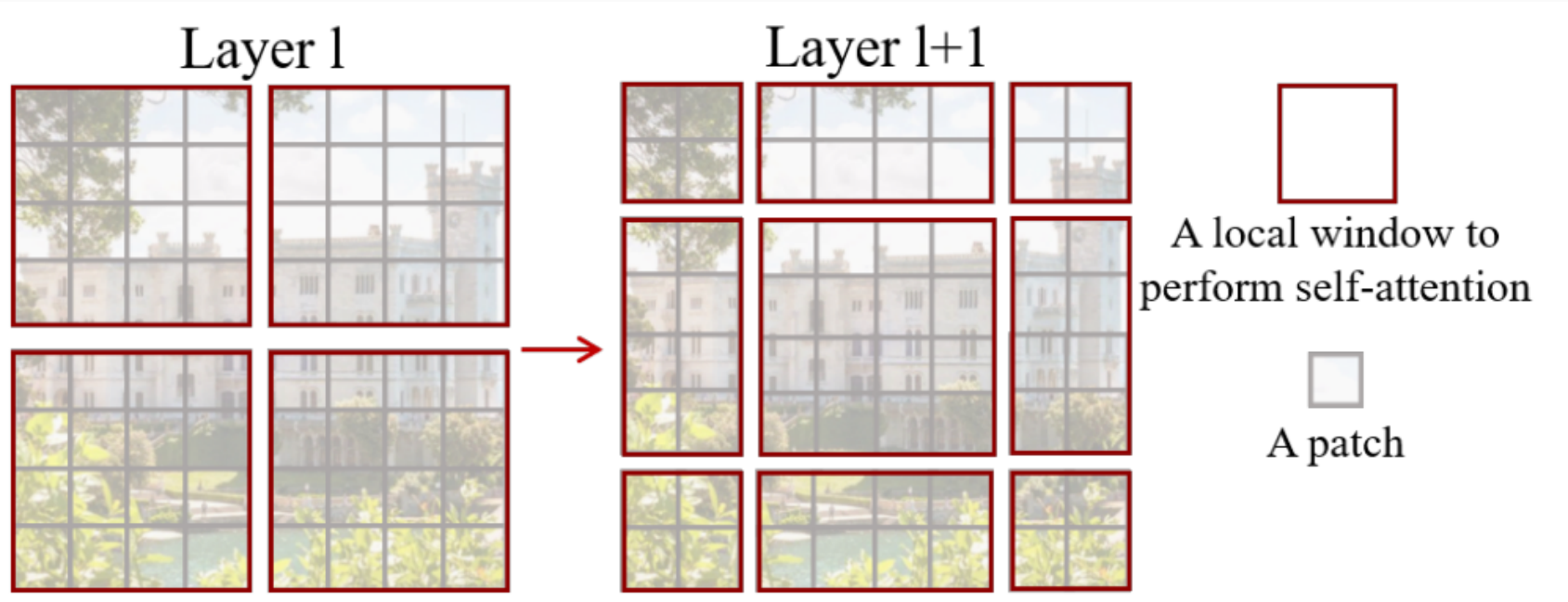
Swin Transformers

- *Swin Transformer* (Liu et al., 2021; **s**hifted **win**dows): similar to ViT but in a hierarchical fashion



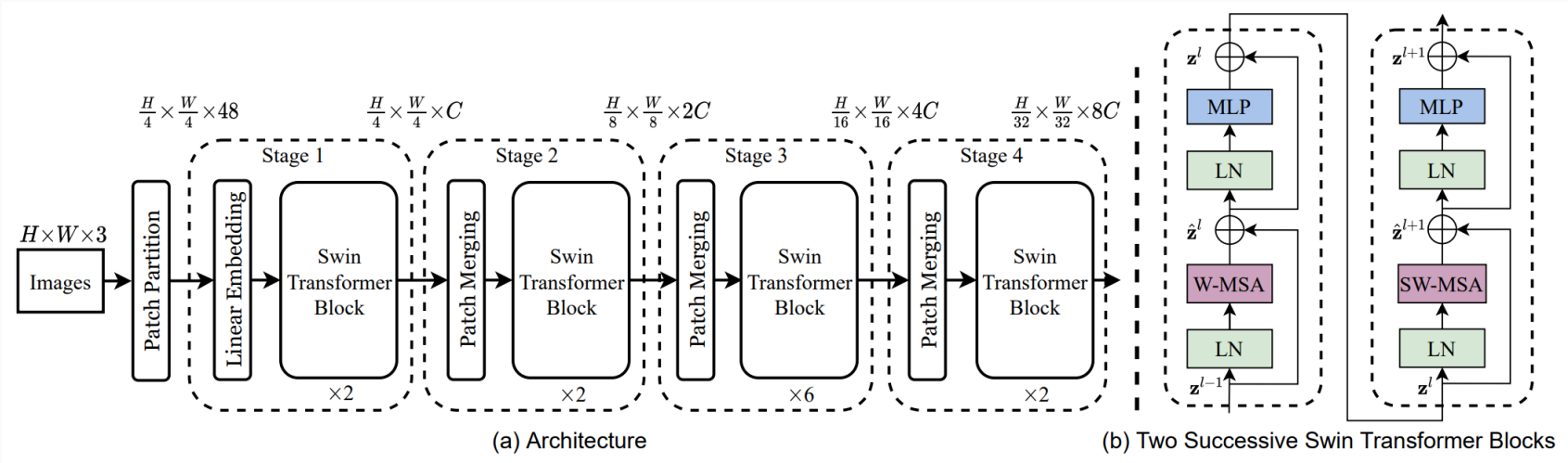
Swin Transformers

- It processes image patches on several scales, computing self-attention across patches in a convolutional-like architecture
- The windows shift between self-attention layers, providing connections between parts of the image



Swin Transformers

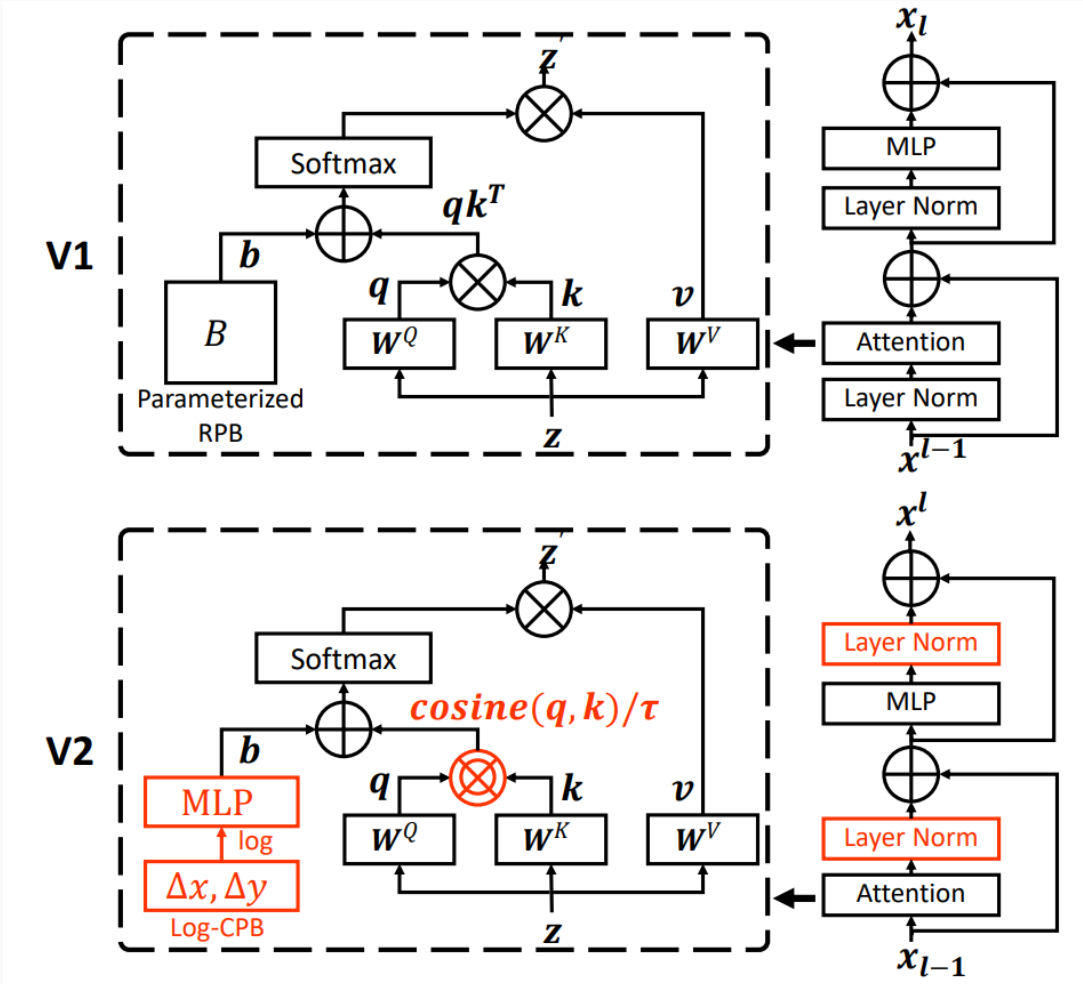
- The architecture itself goes on reducing the geometry, like classical CNN backbones:



- Swin can scale up to larger input resolutions and can be used for dense recognition tasks such as image segmentation

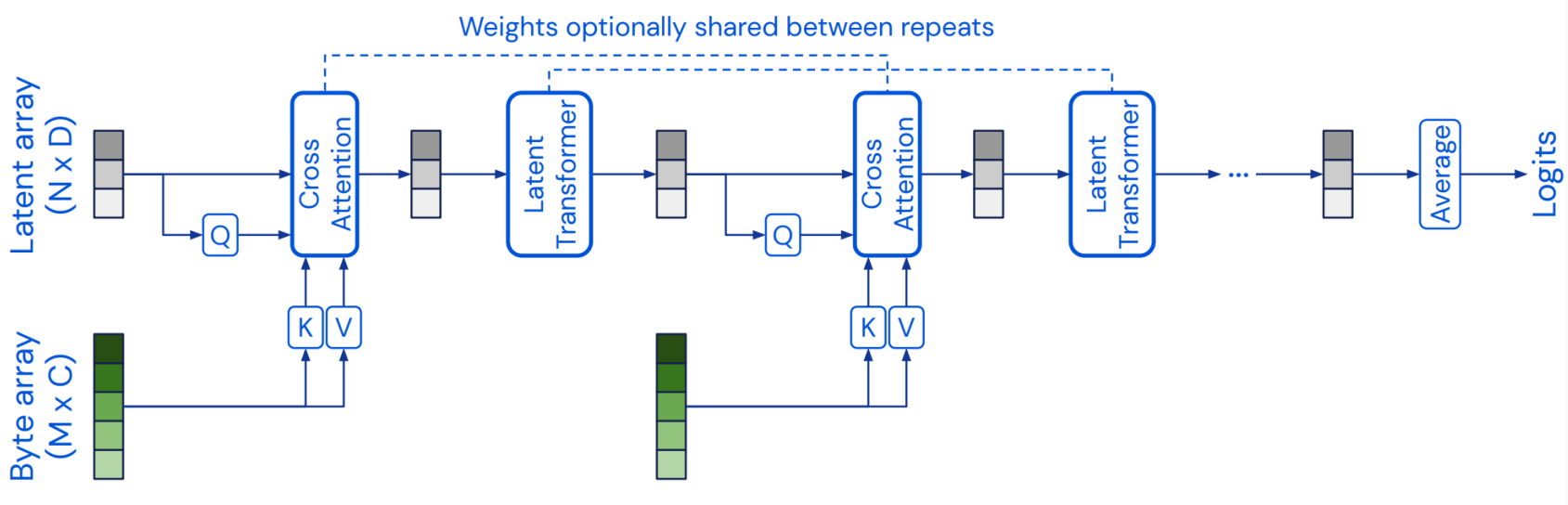
Swin Transformers

- A later iteration, *Swin Transformer v2* (Liu et al., 2022), scaled the Swin Transformer up to 3 billion parameters and allowed for training with images up to 1536×1536 pixels, further improving state of the art in image processing problems across the board



Perceiver

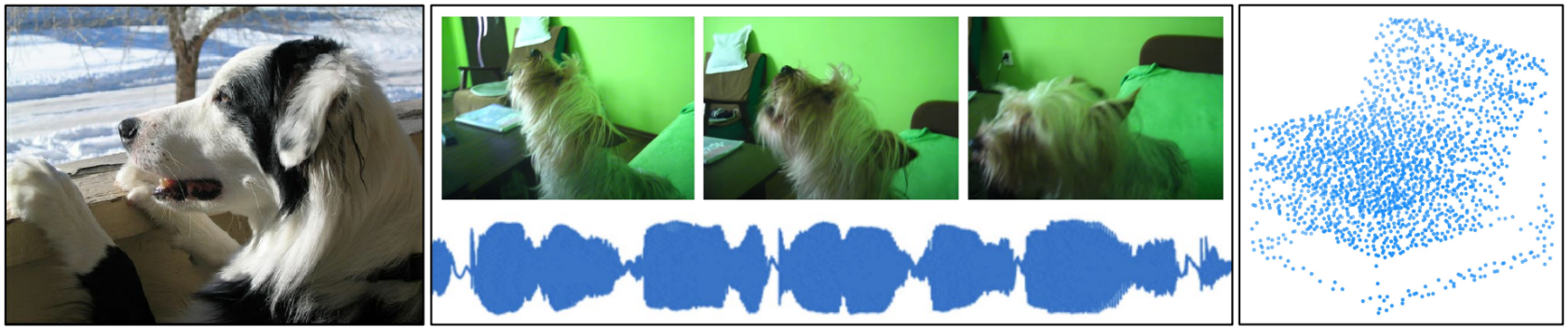
- *DeepMind*'s *Perceiver* (Jaegle et al., 2021a) is a general-purpose architecture that can process numerous different modalities



- The main idea is to avoid the quadratic bottleneck of self-attention by using lower-dimensional latent units
- It's quadratic in the number of queries, so we use a small vector of latents for queries and add large byte arrays for K and V

Perceiver

- Trained on images, point clouds, audio, and video, with no modality-specific encoders and no change to the architecture!

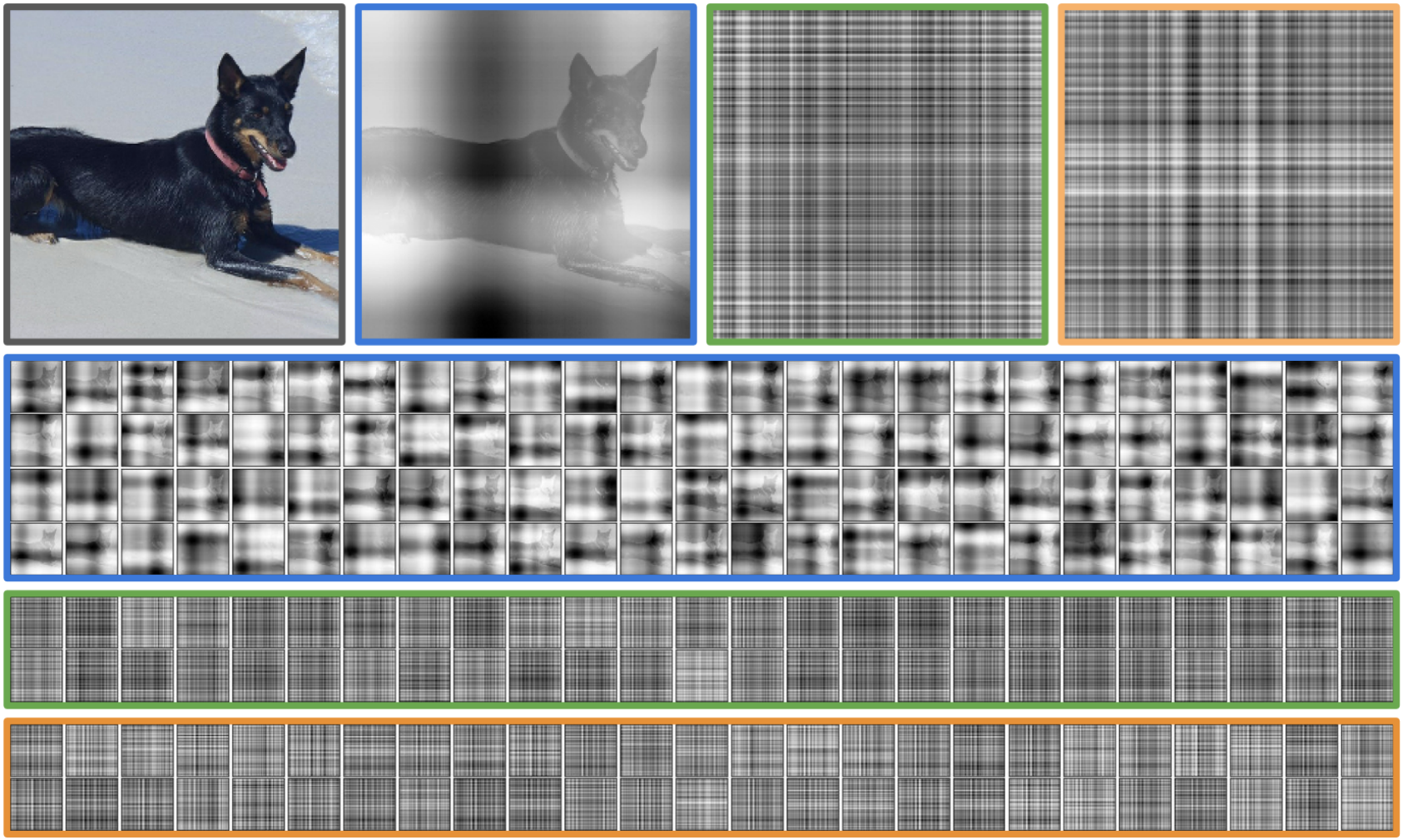


ResNet-50 (He et al., 2016)	77.6
ViT-B-16 (Dosovitskiy et al., 2021)	77.9
ResNet-50 (FF)	73.5
ViT-B-16 (FF)	76.7
Transformer (64x64, FF)	57.0
Perceiver (FF)	78.0

	Raw	Perm.	Input RF
ResNet-50 (FF)	73.5	39.4	49
ViT-B-16 (FF)	76.7	61.7	256
Transformer (64x64) (FF)	57.0	57.0	4,096
Perceiver:			
(FF)	78.0	78.0	50,176
(Learned pos.)	70.9	70.9	50,176

Perceiver

- Attention maps go down to individual pixels:

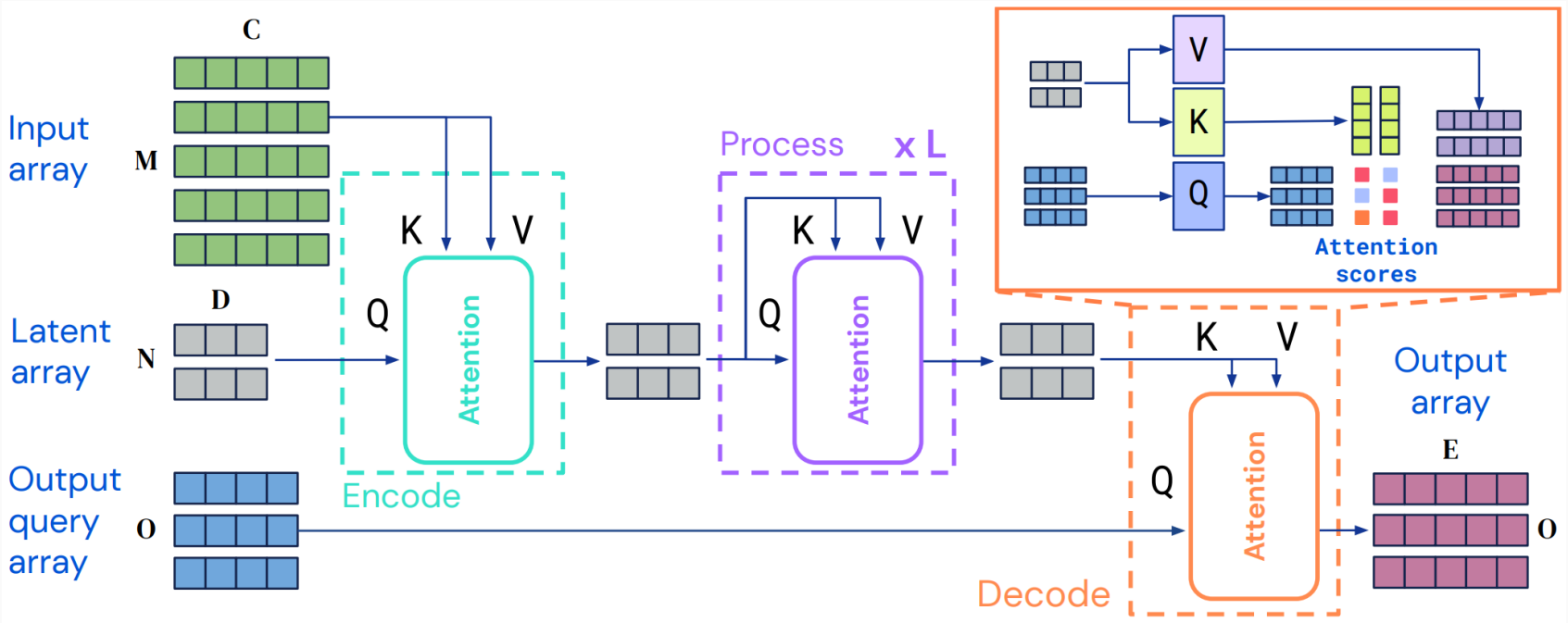


Perceiver

- *Perceiver IO* (Jaegle et al., 2021b), the next version that concentrates on high-dimensional *outputs* as well
- Can process even more various structured data: 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

Perceiver

- Perceiver IO architecture:



- The actual output queries are constructed automatically by combining a set of vectors that describe properties of the current output such as position coordinates

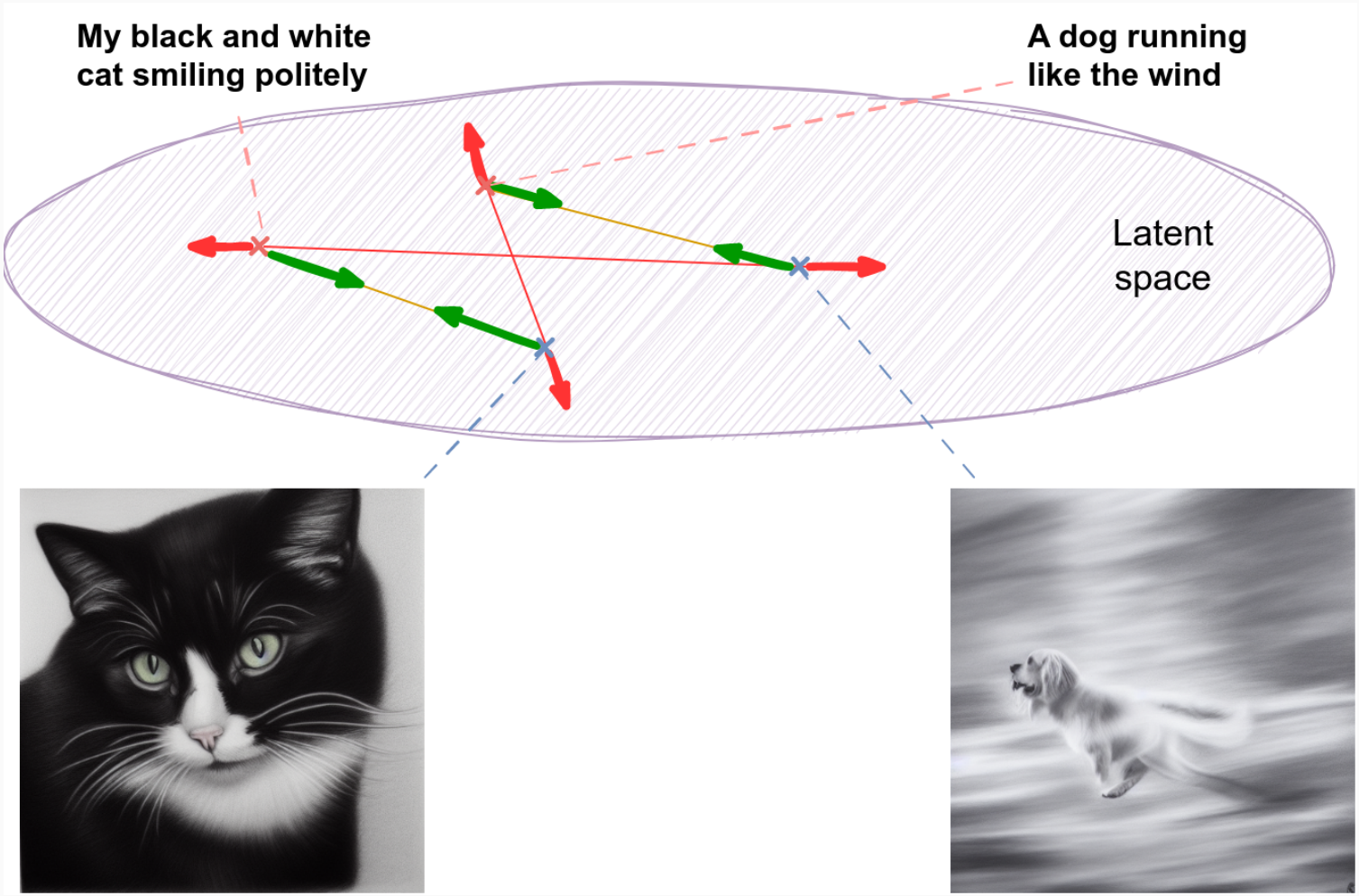
CLIP and BLIP

Multimodal latent spaces

- We have a lot of data freely available on the Web.
- How do we use text paired with images to get a good multimodal latent space?
- The authors of CLIP reported that their first instinct was to train an image CNN and a text Transformer to predict a caption of an image.
- But that didn't work: predicting the exact caption is very hard (basically hopeless), and it's not really what we need in this model, we just need good multimodal embeddings.

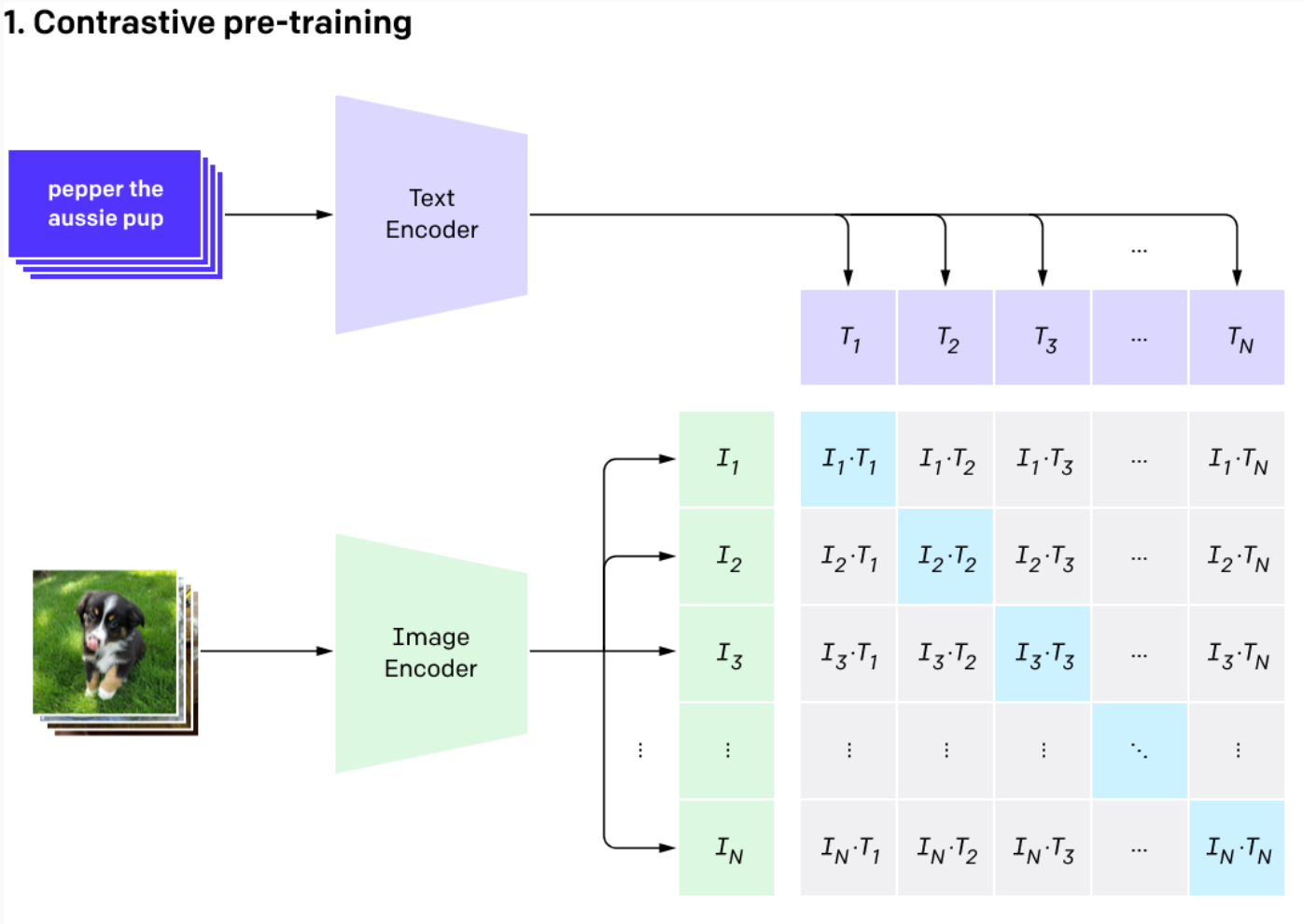
CLIP

- Contrastive pretraining: to get a multimodal latent space, let us model attractive and repulsive forces in the joint latent space for matching texts and images



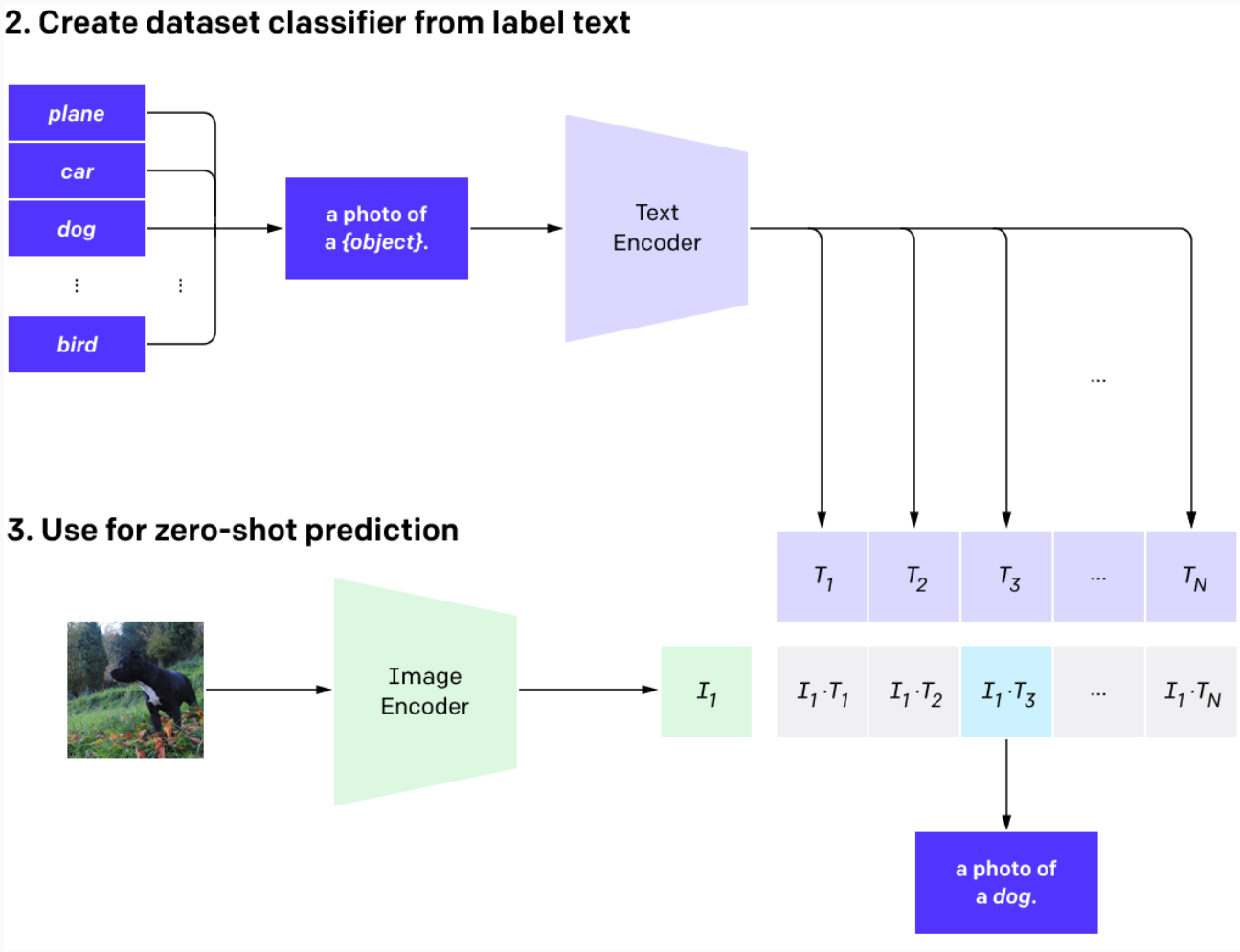
CLIP

- CLIP (Radford et al., 2021; OpenAI) stands for Contrastive Language–Image Pre-training
- They used image-text pairs that were just scraped off the Internet:



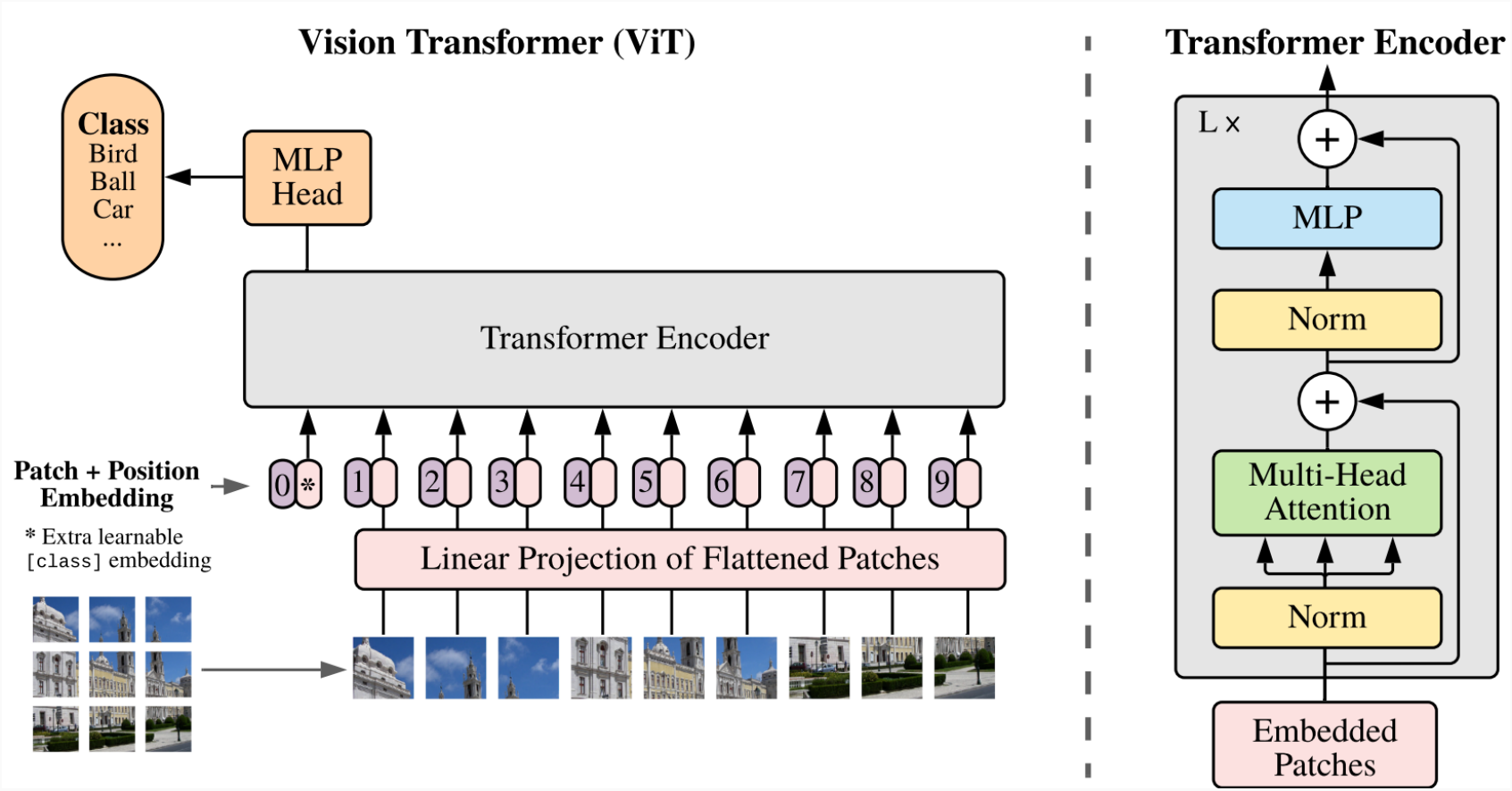
CLIP

- And then one can do zero-shot classification by converting class labels into simple queries:



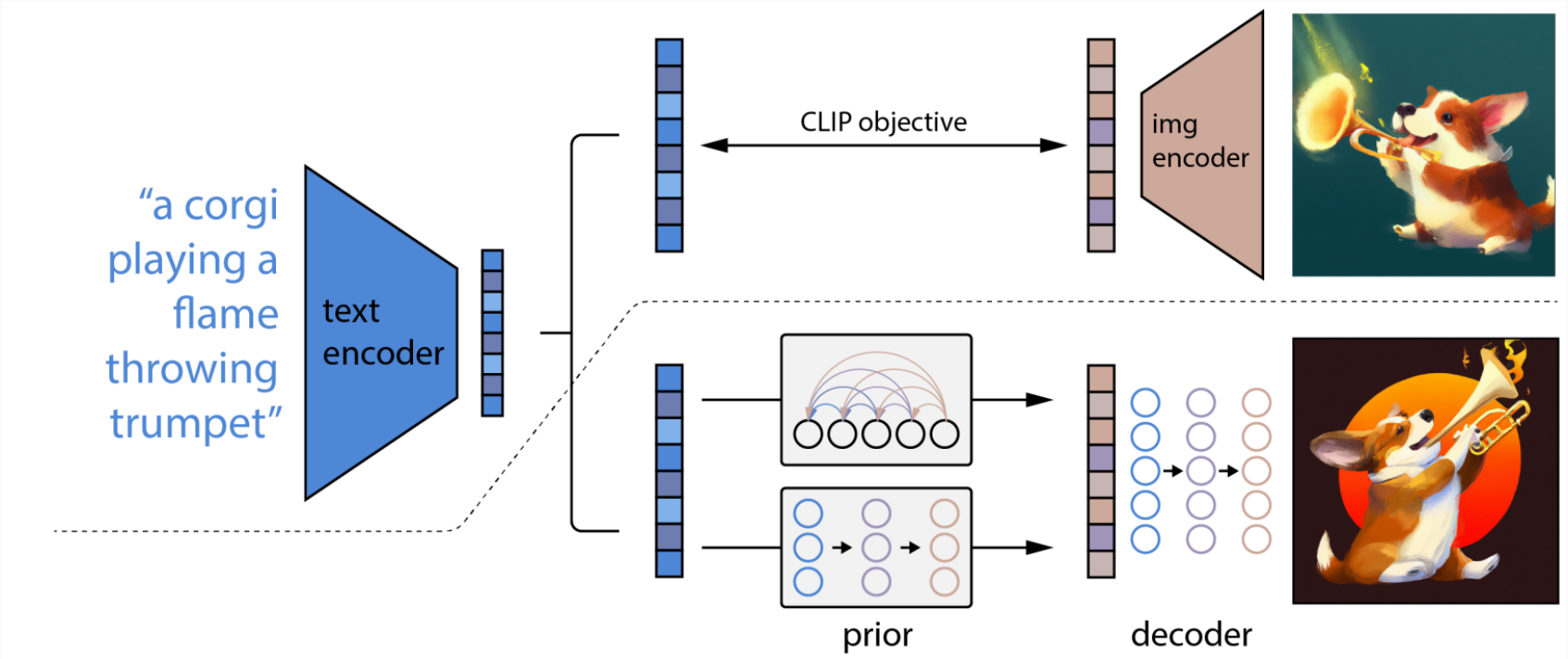
CLIP

- The image encoder was ViT (Vision Transformer) (Dosovitsky et al., 2020)



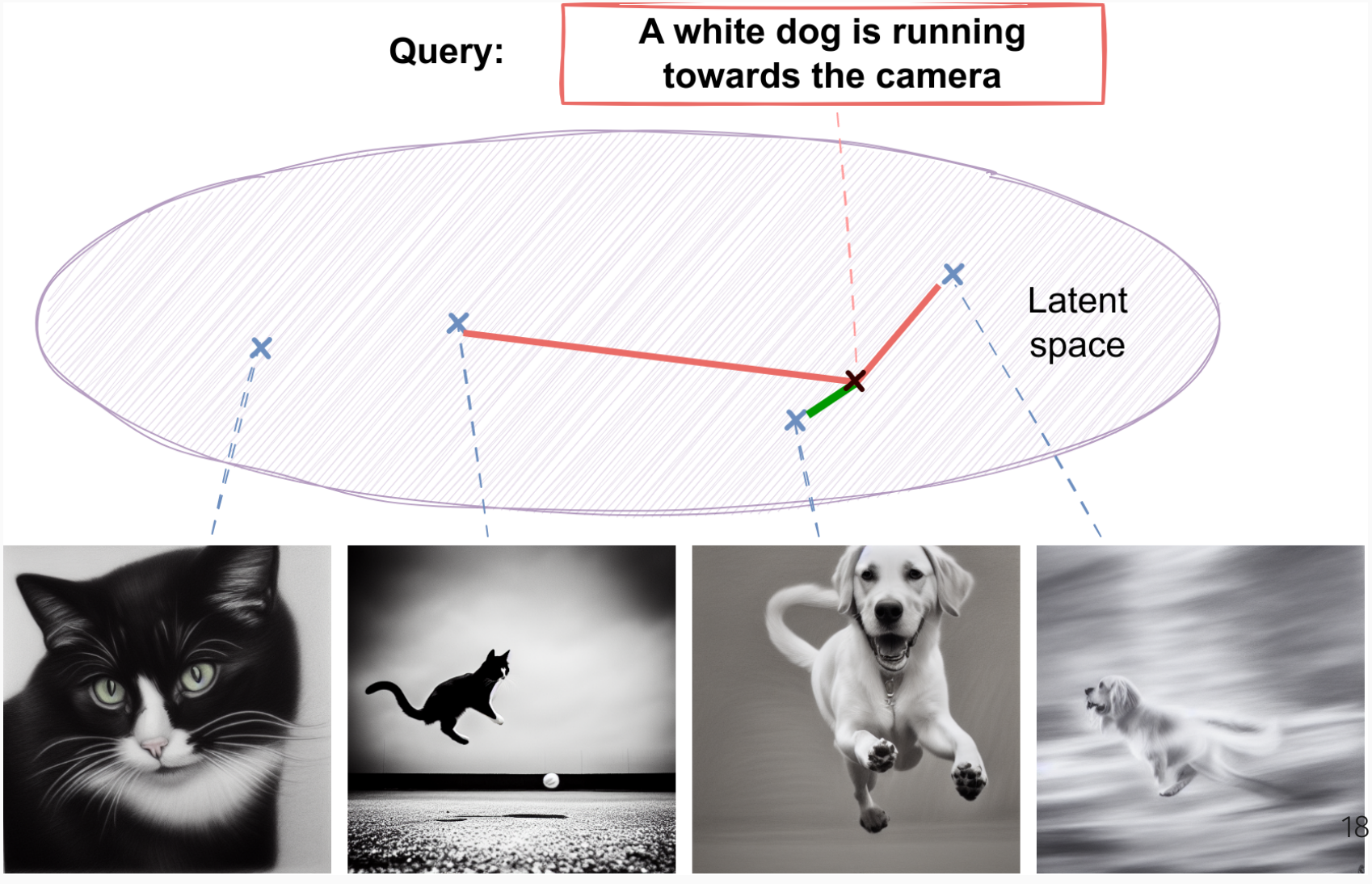
CLIP

- CLIP became a popular source of joint embeddings; e.g., DALL-E 2 does its magic in CLIP's latent space:



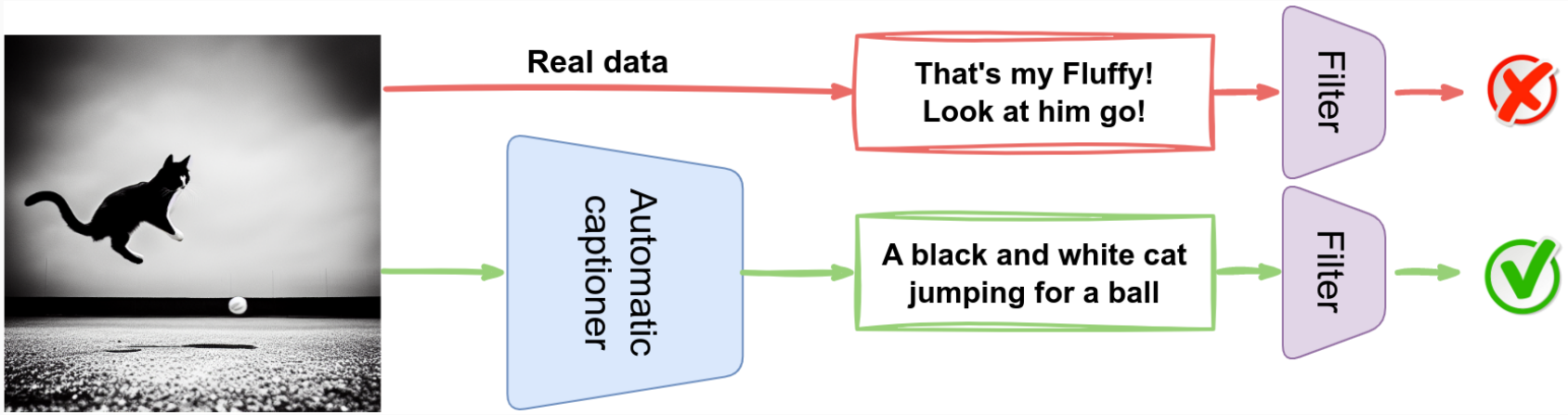
CLIP

- The main use of CLIP has been for enabling text-image retrieval and generative AI models:



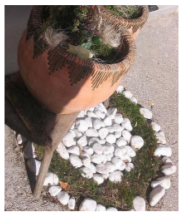
BLIP

- BLIP (Li et al., 2022) stands for Bootstrapping Language-Image Pre-training; they generate synthetic captions and filter them



T_w: "from bridge near my house"

T_s: "a flock of birds flying over a lake at sunset"



T_w: "in front of a house door in Reichenfels, Austria"

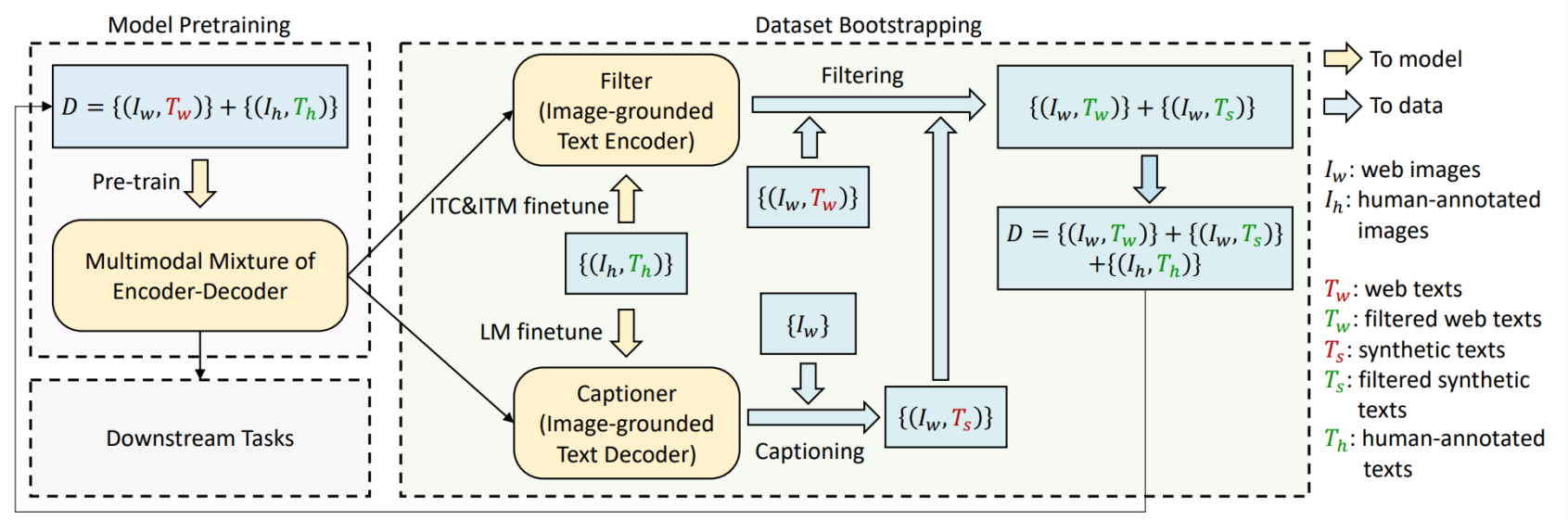
T_s: "a potted plant sitting on top of a pile of rocks"

T_w: "the current castle was built in 1180, replacing a 9th century wooden castle"

T_s: "a large building with a lot of windows on it"

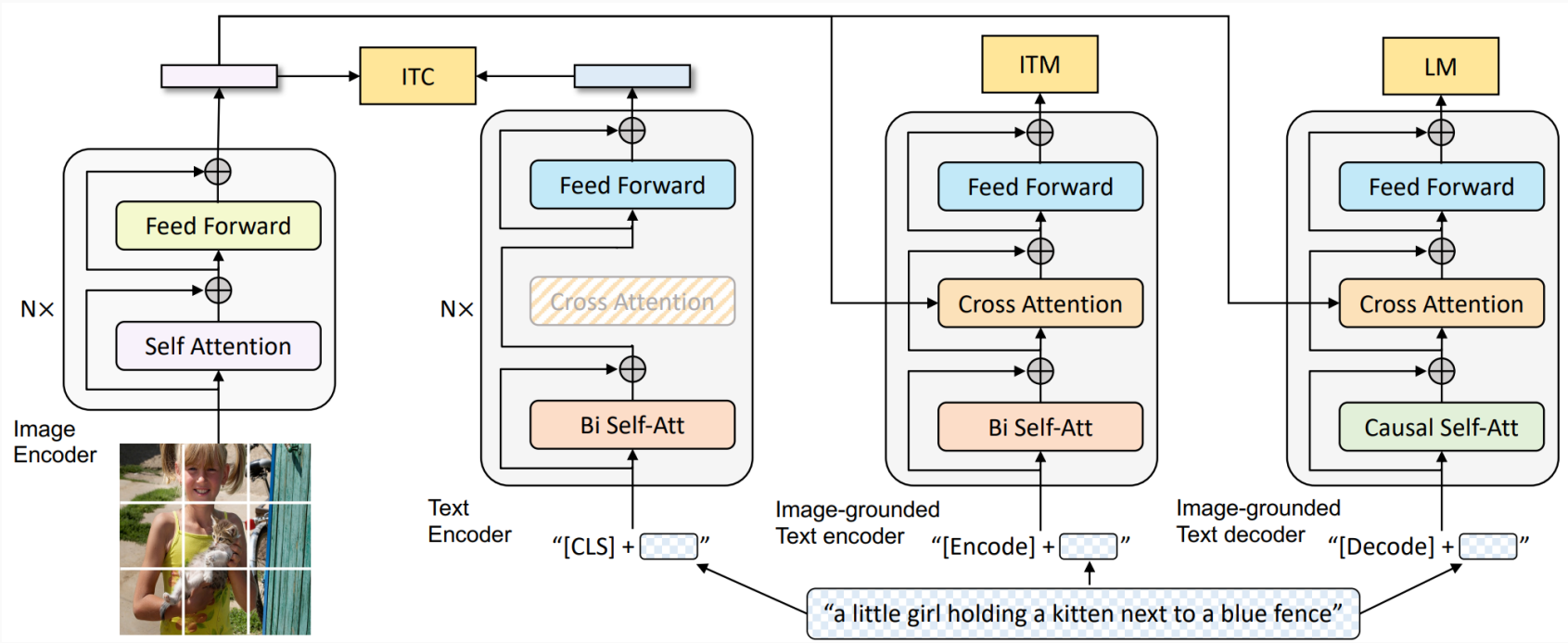
BLIP

- Training structure:



BLIP

- Three different encoders and a decoder: ITC – image-text contrastive loss (match image and text representations), ITM – image-text matching loss (distinguish positive and negative pairs), LM – language modeling (write captions autoregressively)



Спасибо за внимание!