

Обзор

Segment Anything

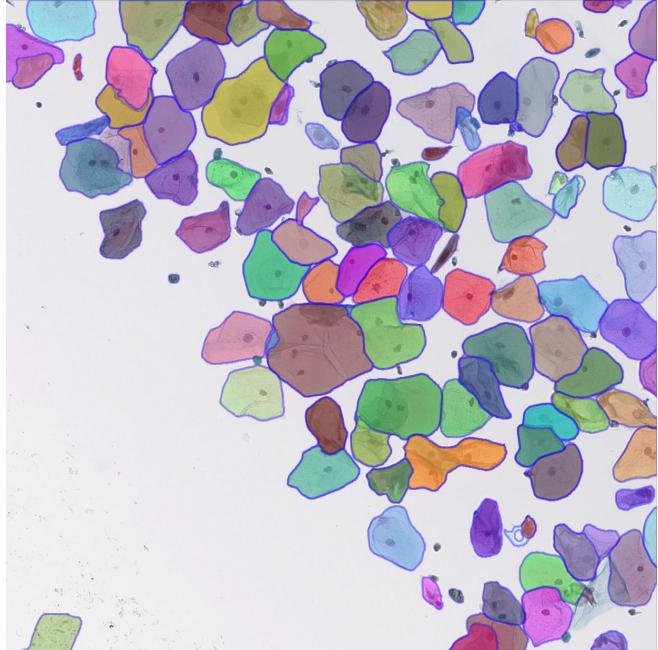
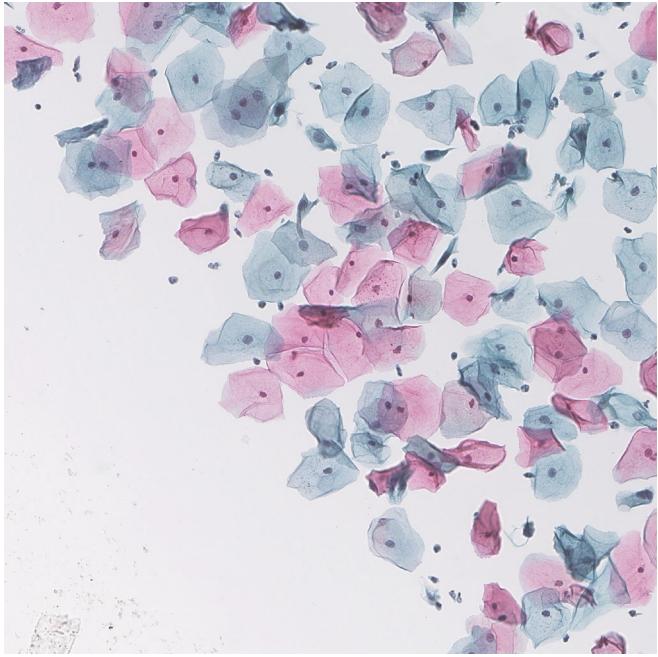
Арсений Литвинов

Segment Anything Model (SAM)

- SAM появился в апреле 2023 от Meta
- Foundation Model для сегментации изображений: модель может решать много задач сегментации на любых данных без дополнительного обучения.
- “SAM is a promptable segmentation system with zero-shot generalization to unfamiliar objects and images, without the need for additional training”
 - Два режима работы: интерактор и Automatic Mask Generator
- Опубликован крупнейший открытый датасет SA-1B с 11 миллионами картинок и 1+ миллиардом масок без классов, который размечен SAM
- Доступно онлайн: segment-anything.com/demo, статья: arxiv.org/abs/2304.02643

Содержание

1. Promptable Segmentation
2. Архитектура Segment Anything
 - a. Image Encoder
 - b. Prompt Encoder
 - c. Mask Decoder
3. Процесс обучения и разметки (SAM Data Engine)
4. Automatic Mask Generator
5. Сравнение с другими моделями и задачами (Zero-Shot Transfer)



Мотивация

Создать foundation model для сегментации изображений

- Foundation Model – модель которая может решать много различных задач без дополнительного обучения. (Пример: ChatGPT в NLP, в CV - CLIP)
- Цель создать модель, которая будет работать на различных данных в разных доменах, решать различные задачи без изменения архитектуры
- Была выбрана задача promptable segmentation. “task that is general enough to provide a powerful pretraining objective and to enable a wide range of downstream applications”
- От модели требуется поддержка быстрых ответов на промпты, так как композиция разных автоматических промптов будет использоваться для сведения ответов к другим задачам

Сегментация

Маски строятся по изображению



(a) image



(b) semantic segmentation

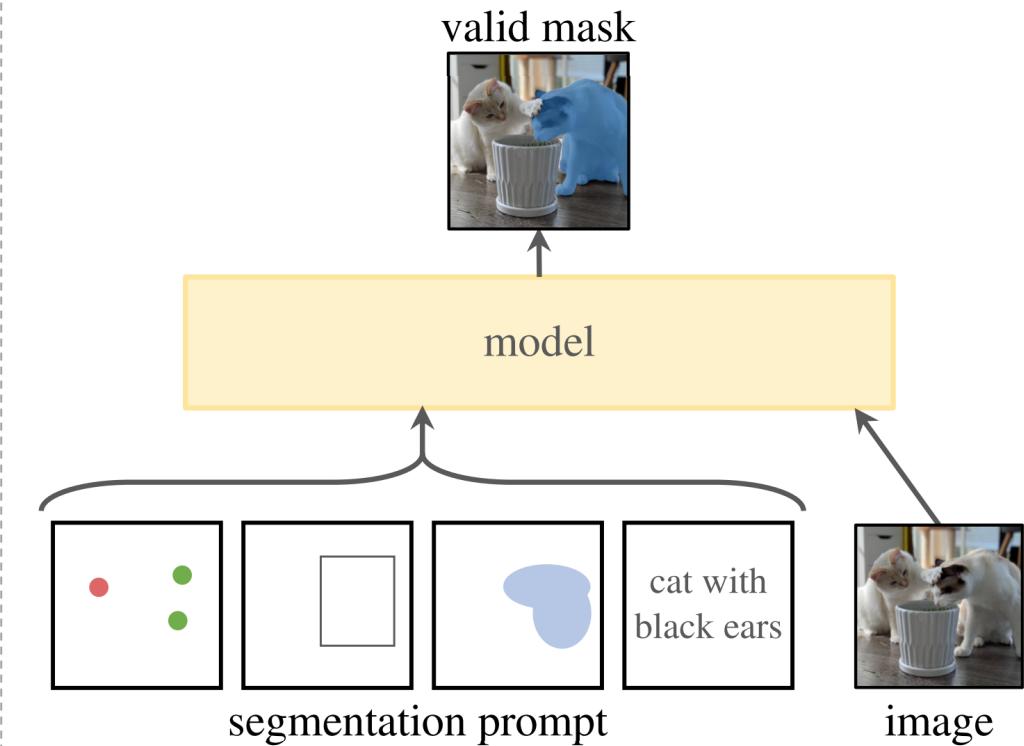


(c) instance segmentation



(d) panoptic segmentation

Маски строятся по изображению и промпту

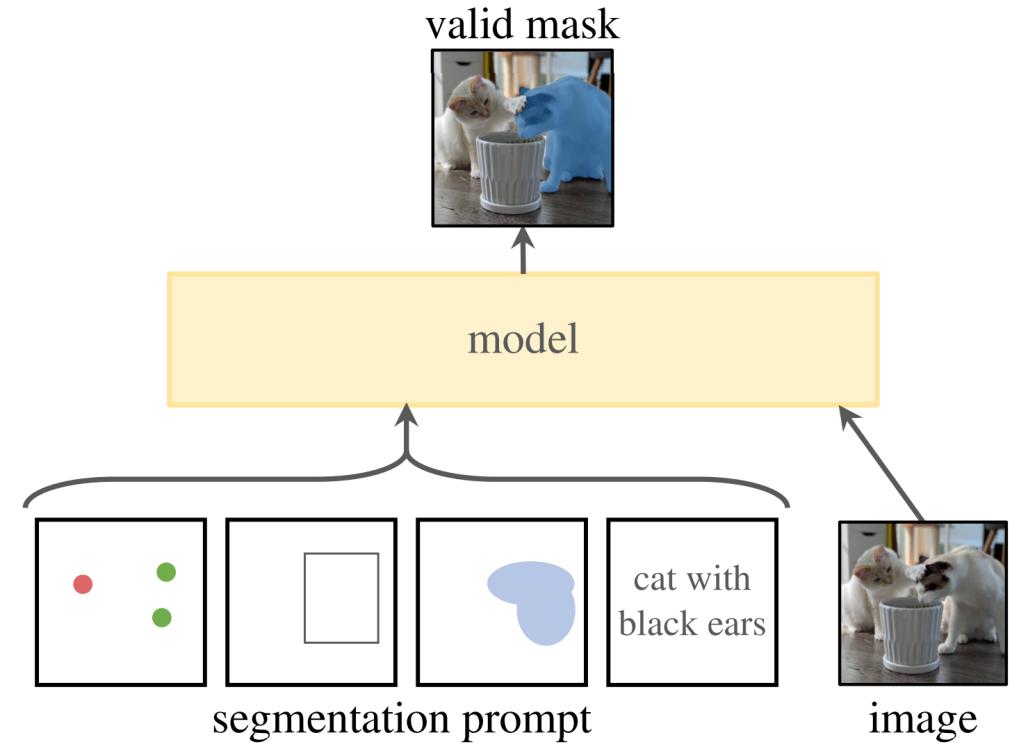


Promptable Segmentation в SAM

Промпт может состоять из непустой комбинации

- Точек (**позитивные** и **негативные**)
- Box (не более одного)
- Маски
- Текста (не выложено в коде, но исследовано в статье)

Get mask(s) by picture and prompt

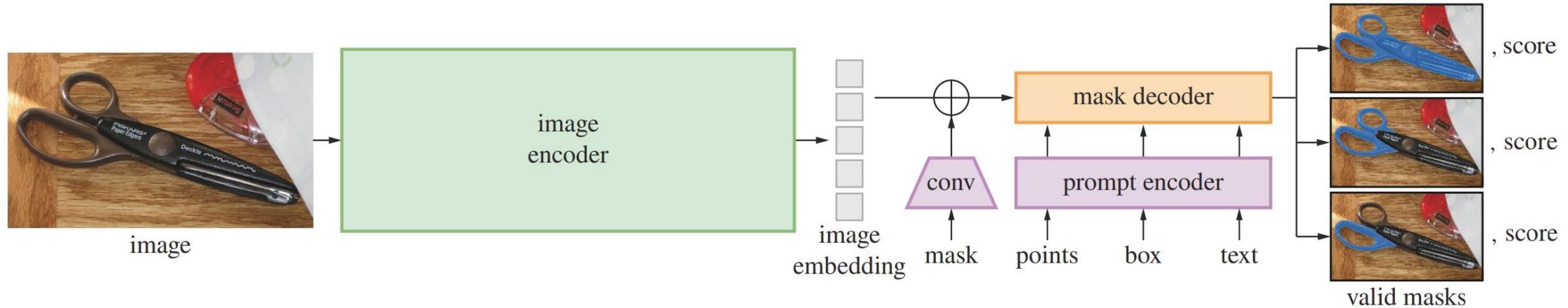


Неоднозначные промпты



- На неоднозначные промпты выдаётся несколько масок – основное отличие от интерактивной сегментации.
- “...interactive segmentation models are designed with human users in mind, a model trained for promptable segmentation can also be composed into a larger algorithmic system...”.
- “general method for zero-shot transfer to downstream segmentation tasks via prompting”
- Гиперпараметр – 3 неоднозначных маски: whole, part, subpart

Архитектура

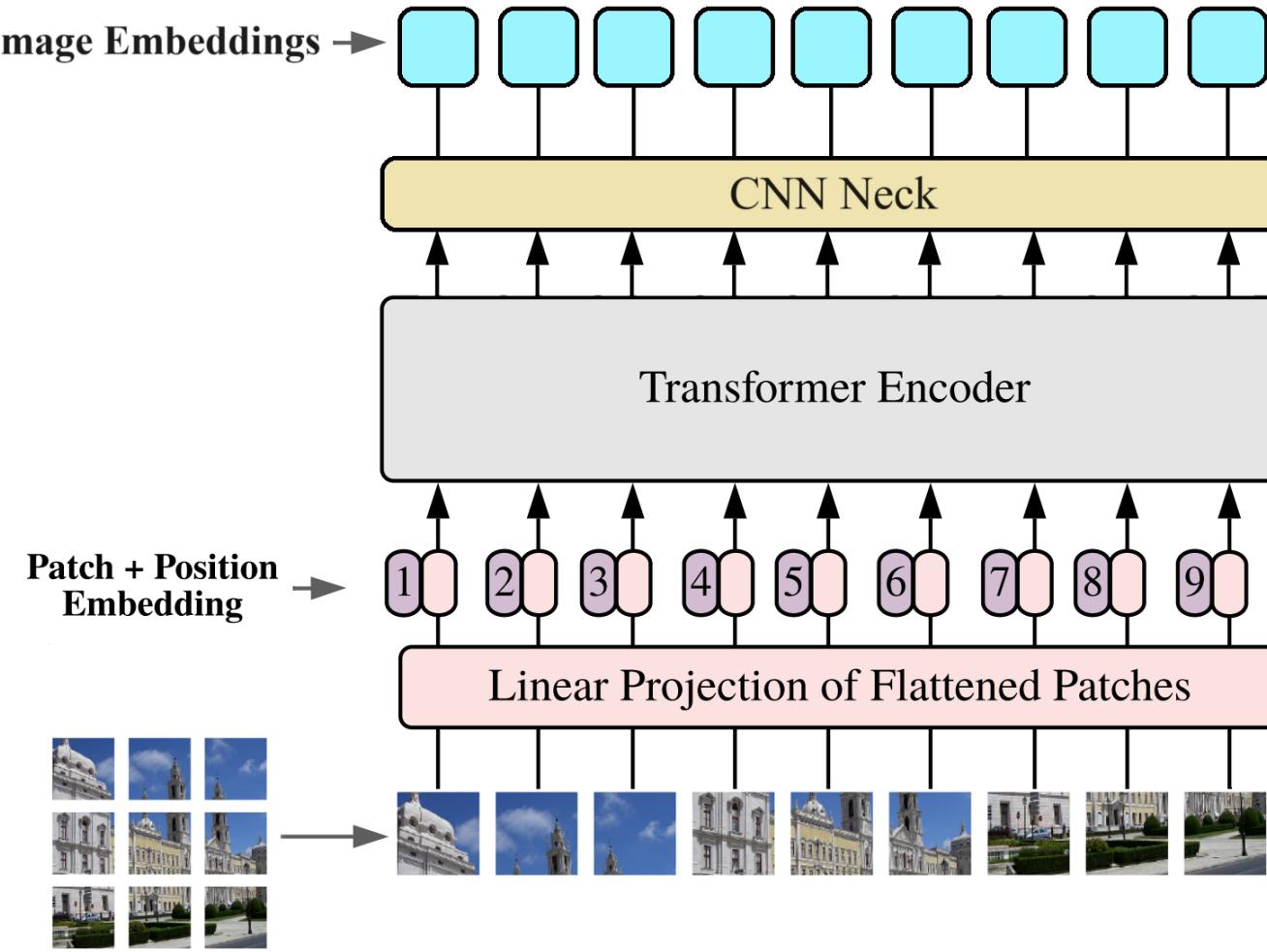


- Огромный (512мб/1ГБ/2ГБ, 632М) **Image Encoder** – чтобы извлечь признаки из изображения
- Лёгкий (32кб) **Prompt Encoder** – чтобы закодировать запросы
- Лёгкий (17мб) **Mask Decoder** – чтобы выдавать маски по вычисленным эмбеддингам в ответ на промпты

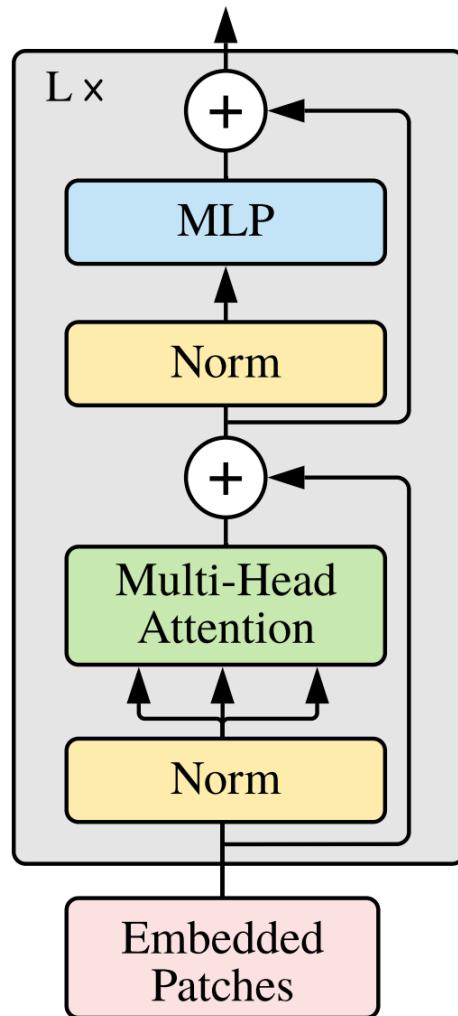
Image Encoder

- Большая сеть для извлечения признаков из входного изображения
- Изображение проходит через неё перед началом обработки промптов
- Авторы SAM используют ViTDet, но можно использовать практически любую архитектуру
- После прохождения через неё изображение отображается в “image эмбеддинги”: в карту признаков размера (256, 64, 64)
 - После этого оригинальное RGB изображение больше не используется, используется только построенная карта признаков (и оригинальные размеры картинки для ресайза масок)

ViTDet



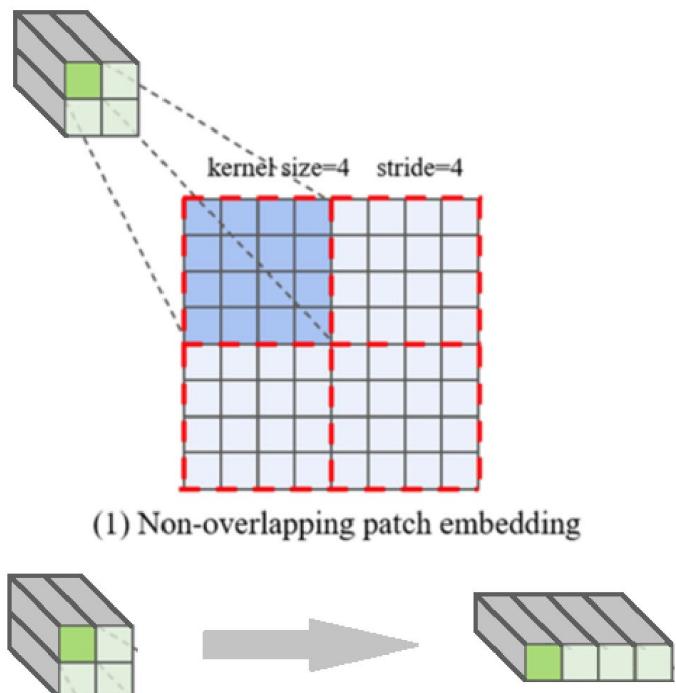
Transformer Encoder



Exploring Plain Vision Transformer Backbones for Object Detection: arxiv.org/abs/2203.16527

Patch Embedding

Применяем к изображению свёртку с шагом = patch_size и размером ядра = patch_size.



```
class PatchEmbedding(nn.Module):
    def __init__(self,
                 patch_size=(16, 16),
                 embedding_dim=512):
        super().__init__()
        self.conv = nn.Conv2d(
            in_channels=3,
            out_channels=embedding_dim,
            kernel_size=patch_size,
            stride=patch_size
        )

    def forward(self, x):
        x = self.conv(x)
        # B C H W -> B H W C (далее B Hw C)
        x = x.permute(0, 2, 3, 1)
```

Learnable Absolute Position Embedding

```
class ImageEncoderViT(nn.Module):
    def __init__(...):
        self.img_size = img_size # Фиксирован для модели
        self.patch_embed = PatchEmbedding(...)
        self.abs_pos_embed = nn.Parameter(
            torch.zeros(1, img_size // patch_size, img_size // patch_size, embed_dim)
        )
        self.blocks = nn.Sequential([Block(), ..., Block()]) # Сам трансформер
        self.neck = SmallCNN(...) # Чтобы изменить количество каналов у image_embeddings

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.patch_embed(x)
        x = x + self.abs_pos_embed

        for blk in self.blocks:
            x = blk(x)

        x = self.neck(x)
        return x
```

Из-за требования фиксированного размера
любая картинка ресайзится к (1024, 1024).

Неквадратное изображение, перед этим
шагом, дополняется паддингом (снизу или
справа) до квадрата.

Scaled Dot-Product Attention

Q: Queries (I_q, d_q), K: Keys (I_k, d_k), V: Values (I_v, d_v)

$$\text{Attention}(Q, K, V) = \text{Softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

- $I_k = I_v$. Равенство числа строк в K и V: каждому key соответствует соответствующий вектор value
- $d_q = d_k$. Равенство числа столбцов в Q и K. Размерности вектора-запроса и векторов-ключей одинаковые
- На каждый запрос из Q внимание строит взвешенное среднее всех векторов-значений из V
- Чем ближе query к key_i, тем выше вес перед соответствующим value_i во взвешенной сумме
- Для определения близости между запросами и ключами используется масштабированное на $\sqrt{d_k}$ скалярное произведение

Self-Attention

$$\text{Attention}(Q, K, V) = \text{Softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

Shapes:
X: $(I_x, d_{embedding_dim})$
 W_Q : $(d_{embedding_dim}, d_q)$
Q: (I_q, d_q)

Self-Attention: Queries, K, V получаются по формулам $Q = XW_Q$, $K = XW_K$ и $V = XW_V$, W_Q , W_K , W_V - 3 обучаемых nn.Linear слоя (ещё есть bias)

$$\text{SelfAttention}_{W_Q, W_K, W_V}(X) = \text{Softmax} \left(\frac{XW_Q(XW_K)^T}{\sqrt{d_k}} \right) XW_V$$

- Каждый эмбеддинг x в последовательности X обновляется с учётом значений всех остальных эмбеддингов
- Обработка последовательностей любой длины, неограниченное receptive поле, parameter sharing, нет catastrophic forgetting как у RNN, квадратичная сложность от количества патчей

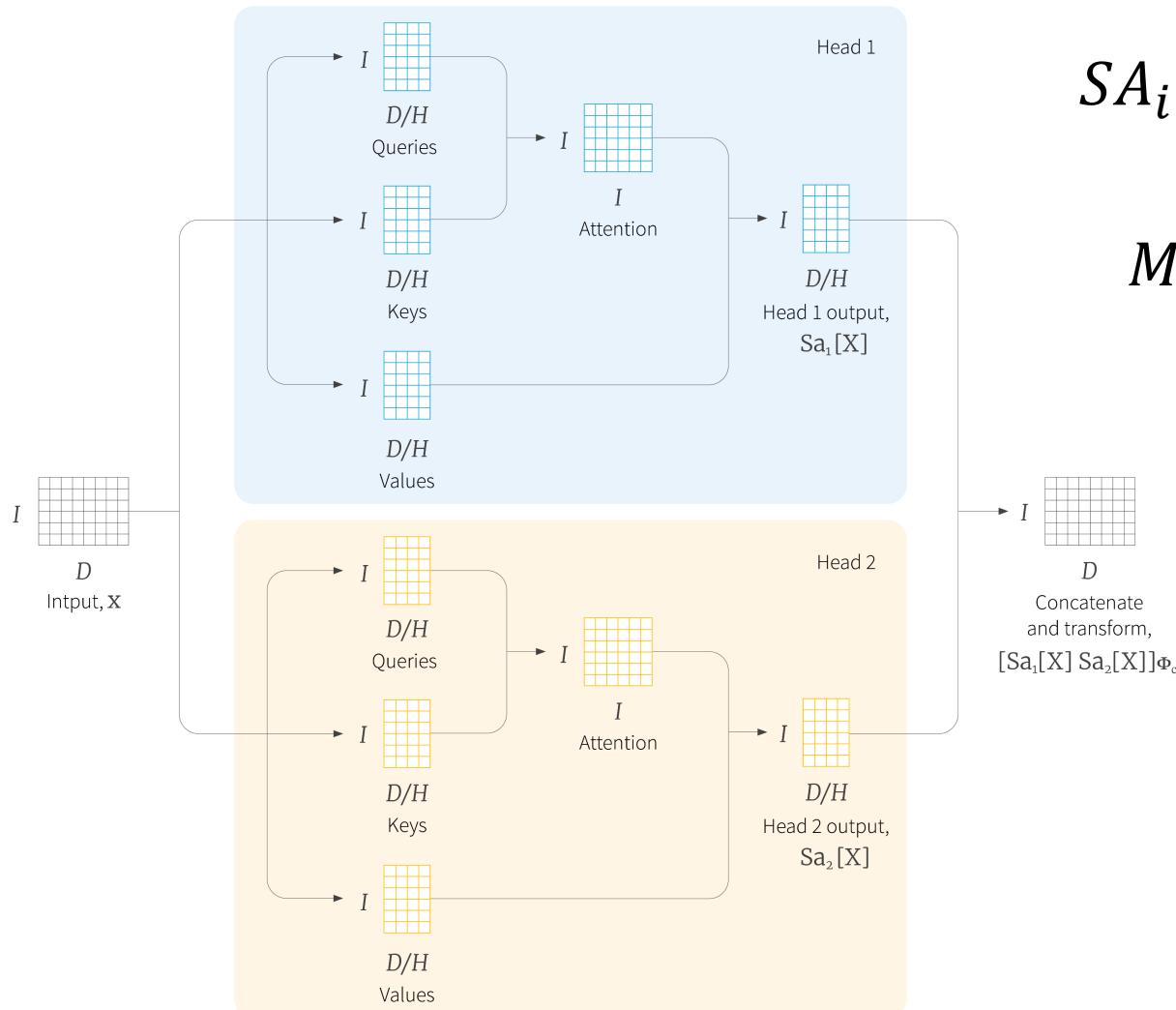
Self-Attention + Relative Position Embedding

$$\text{SelfAttention}_{W_Q, W_K, W_V}(X) = \text{Softmax} \left(\frac{XW_Q(XW_K)^T}{\sqrt{d_k}} \right) XW_V$$

- Проблема permutation equivariance у внимания: взаимное расположение эмбеддингов в последовательности не влияет на ответ
- Используются относительные обучаемые эмбеддинги позиций, которые добавляются к матрице внимания. Каждая ячейка матрицы R кодирует расстояние между query_i и key_j. R получается из двух обучаемых параметров относительных сдвигов по вертикали и горизонтали и ещё учитывается XW_Q
 - Размеры матрицы R равны (число_патчей, число_патчей)

$$\text{SelfAttention}_{W_Q, W_K, W_V, R}(X) = \text{Softmax} \left(\frac{XW_Q(XW_K)^T}{\sqrt{d_k}} + R \right) XW_V$$

Multi-head Self-Attention



$$SA_i(X) = \text{Softmax} \left(\frac{XW_Q^i(XW_K^i)^T}{\sqrt{d_k}} + R \right) XW_V^i$$

$$MSA(X) = \text{Concat}(SA_1(X), \dots, SA_N(X))W_C$$

- Дополнительный обучаемый параметр W_C
- Ансамблирование разных голов, улавливающих разные зависимости
- Легко параллелится

borealisai.com/research-blogs/tutorial-14-transformers-i-introduction

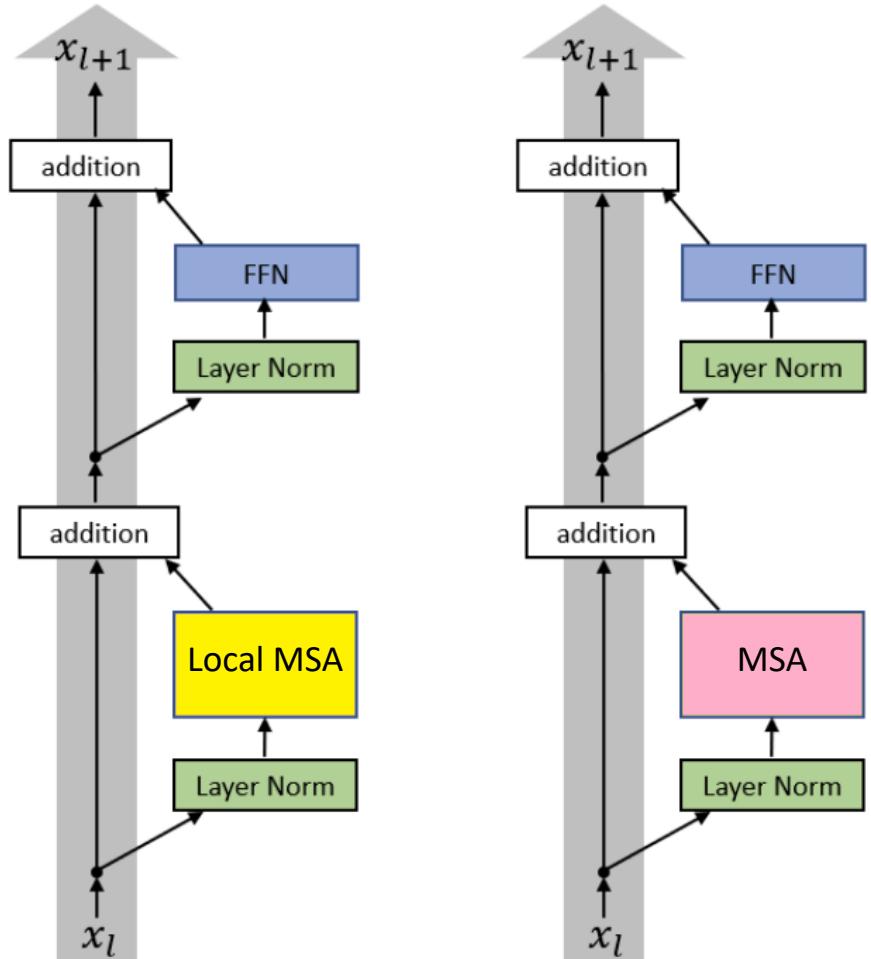
Local Windowed Self-Attention



Картина и идея из Swin

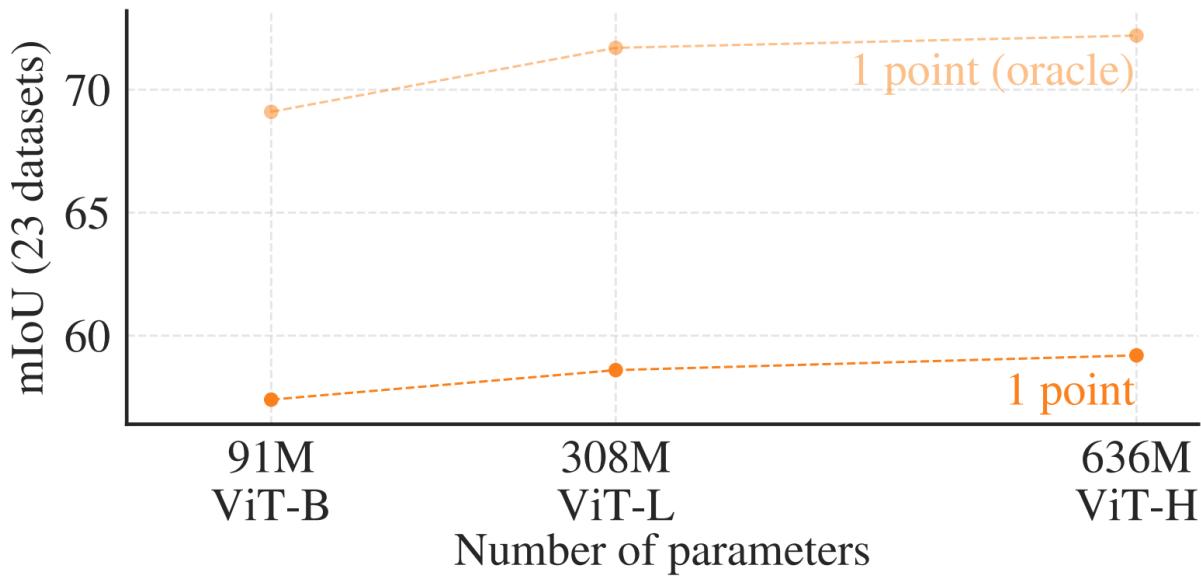
- Картинка состоит из окон, окна состоят из патчей (на примере 4 окна, 64 патча в сумме)
- Слой MSA применяется не к картинке (один раз) целиком, а к каждому из окон
- Патчи из разных окон никак на друга не влияют
- Поскольку сложность внимания квадратична от количества патчей, то получается выигрыш во временной сложности в `num_of_windows` раз (на примере в 4 раза: I^2 vs $4 \left(\frac{I}{4}\right)^2$, ($I = 64$))
- Теряется возможность захвата глобального контекста изображения

Трансформер-блок



- Большая часть блоков имеют локальное внимание: считаются независимые внимания на уровне окон
- 4 трансформер-блока имеют глобальное внимание: считается одно внимание на уровне всего изображения, все патчи влияют друг на друга
- Transformer Encoder – это стек трансформер-блоков

Image Encoder: размеры

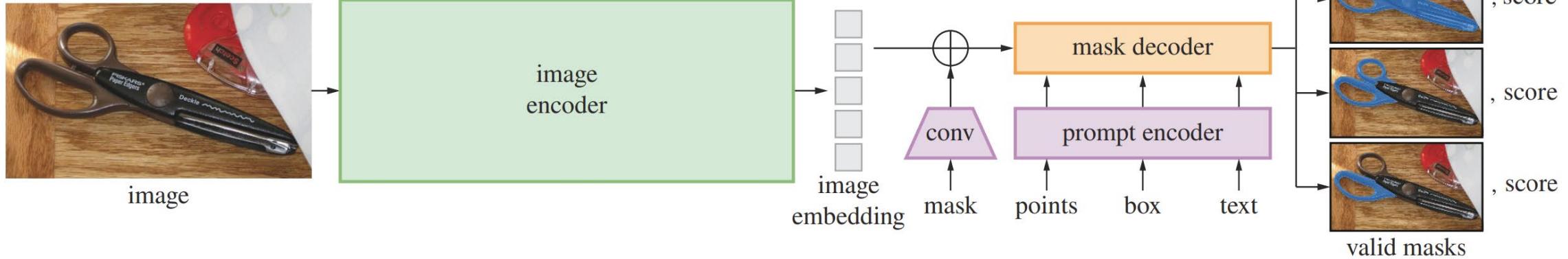


```
configurations = {
    "vit_base": {
        "embed_dim": 768,
        "depth": 12,
        "num_heads": 12,
        "global_attn_indexes": [2, 5, 8, 11],
    },
    "vit_large": {
        "embed_dim": 1024,
        "depth": 24,
        "num_heads": 16,
        "global_attn_indexes": [5, 11, 17, 23],
    },
    "vit_huge": {
        "embed_dim": 1280,
        "depth": 32,
        "num_heads": 16,
        "global_attn_indexes": [7, 15, 23, 31],
    },
}
```

Image Encoder

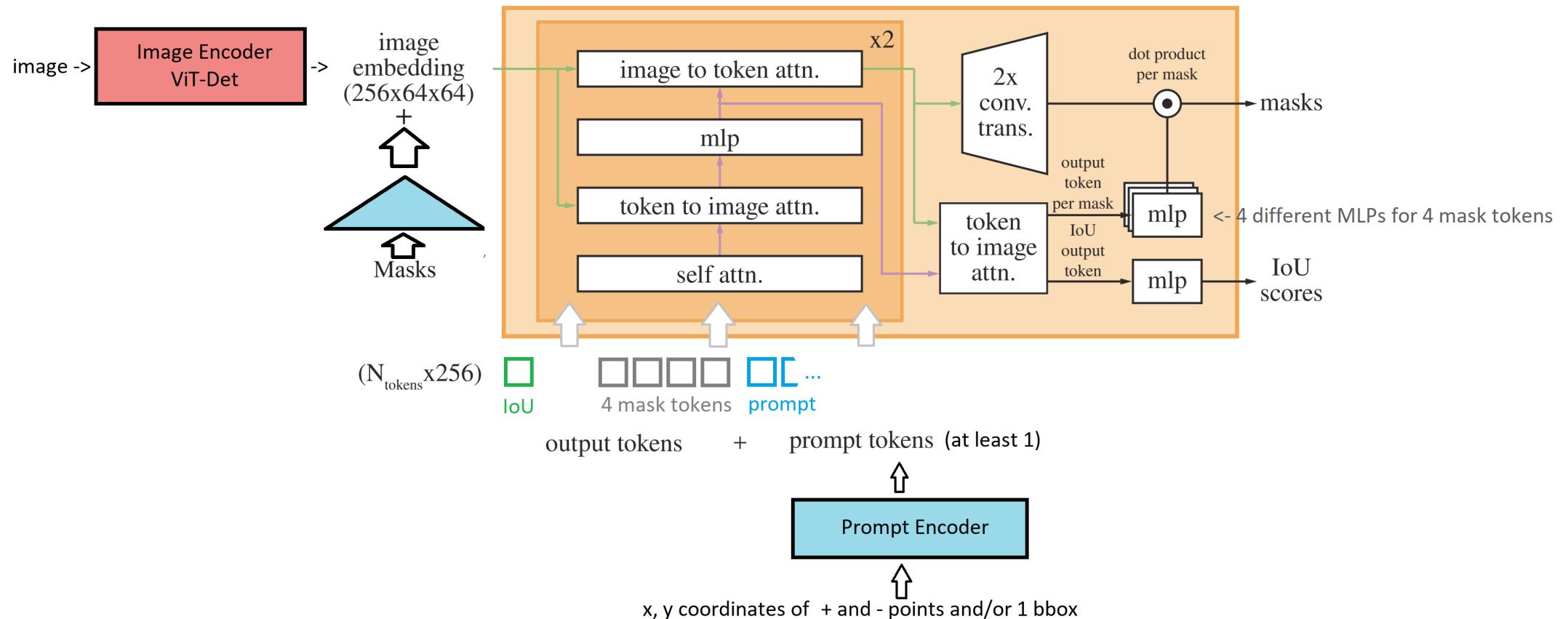
- В качестве Image Encoder используется vision трансформер ViTDet, который является ускорением ViT путём замены в большей части трансформер-блоков глобального self-attention на локальное windowed и исключением <CLS> токена.
- Картинка (1024, 1024) нарезается на патчи, далее строятся эмбеддинги патчей с помощью свёртки + стэк из трансформер-блоков и небольшая свёрточная neck.
- В результате получается feature map / image embedding – тензор размера (C=256, H=64, W=64).
 - Доступны 3 размера: base, large и huge: чем больше размер – тем выше качество.
 - Image Encoder является самой тяжёлой частью модели Segment Anything.
 - Части, которые обрабатывают промпты, существенно меньше, что сделано для эффективности выполнения большого количества разных промпов на одном изображении

Prompt Encoder



dense prompts	sparse prompt		
mask (bool tensor (h, w))	points: [xy₊/₋, ...]	box: x_yx_y	text
Проводится ресайз и маска проходит через небольшую CNN	Строится эмбеддинг для каждой точки	Строится два эмбеддинга позиций top_left и bottom_right box-а	Encoder из CLIP строит эмбеддинг текста
После этого маска поэлементно добавляется к image embeddings	Получается список векторов эмбеддингов позиций для каждой точки	Получается ровно 2 вектора токенов позиции г	Получается вектор эмбеддинг текста
Полученный обновлённый image_embedding передаётся в mask_decoder со стороны image_embedding	Полученные эмбеддинги передаются в mask_decoder со стороны "токенов"		

Mask Decoder



Cross-Attention

$$\text{Attention}(Q, K, V) = \text{Softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

Самовнимание: $Q = XW_Q$, $K = XW_K$ и $V = XW_V$:

$$\text{SelfAttention}(X) = \text{Softmax} \left(\frac{XW_Q(XW_K)^T}{\sqrt{d_k}} \right) XW_V$$

Перекрёстное внимание: $Q = XW_Q$, $K = \textcolor{red}{Y}W_K$ и $V = \textcolor{red}{Y}W_V$:

$$\text{CrossAttention}(X, \textcolor{red}{Y}) = \text{Softmax} \left(\frac{XW_Q(\textcolor{red}{Y}W_K)^T}{\sqrt{d_k}} \right) \textcolor{red}{Y}W_V$$

Multi-head Cross-Attention

$$CA_i(X, Y) = \text{Softmax} \left(\frac{(X + \Pi_X)W_Q^i((Y + \Pi_Y)W_K^i)^T}{\sqrt{d_k}} \right) Y W_V^i$$

$$MCA(X, Y) = \text{Concat}(CA_1(X, Y), \dots, CA_N(X, Y))W_C$$

В тексте обозначается: X to Y attention

В трансформер-блоке Mask Decoder SAM будет сразу два разных блока МСА:

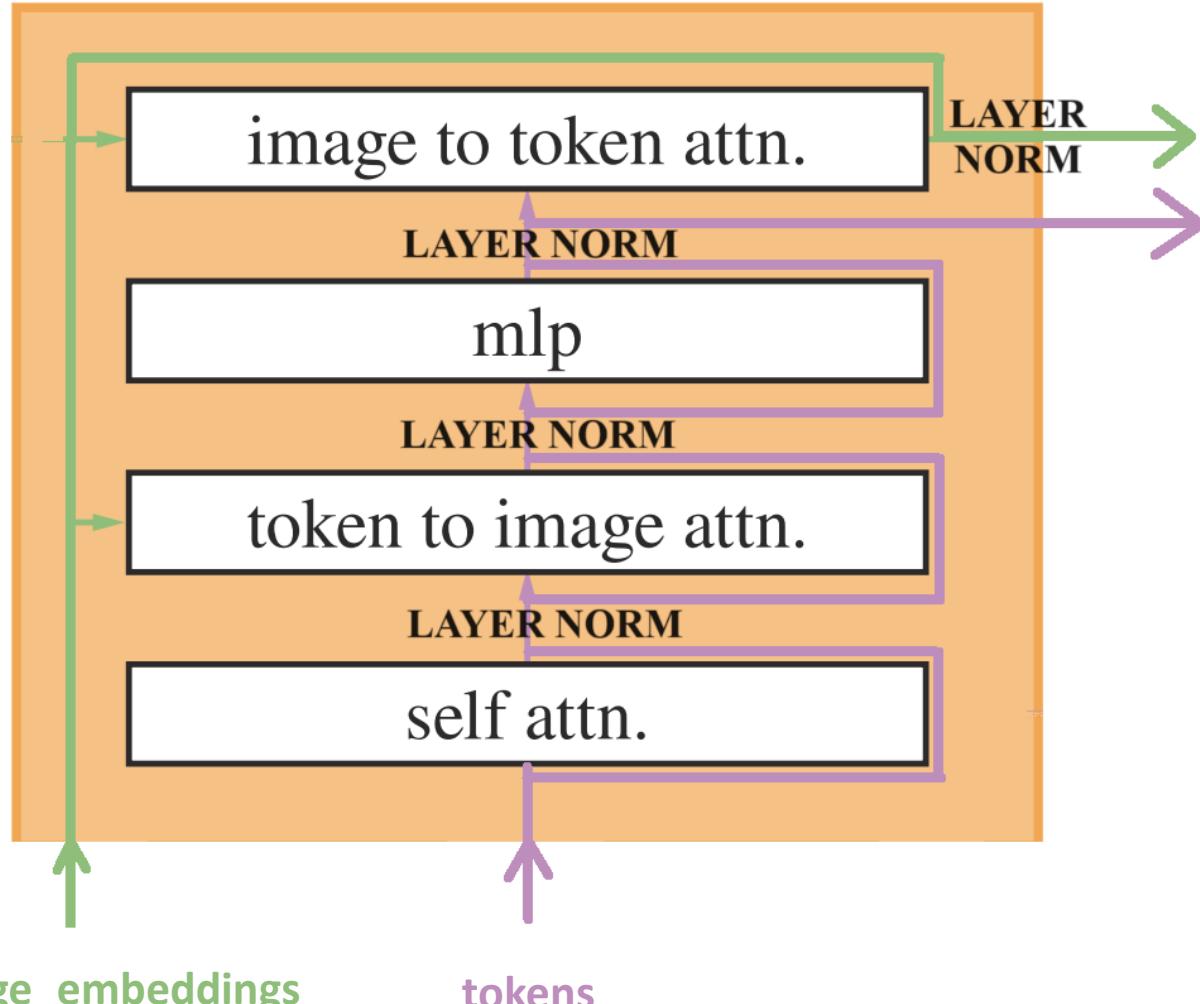
1) Внимание image embeddings на **токены промптов и ответов**

$$\text{image_embeddings} = MCA(\text{image_embeddings}, \text{tokens})$$

2) Внимание токенов промптов и ответов на **image embeddings**

$$\text{tokens} = MCA(\text{tokens}, \text{image_embeddings})$$

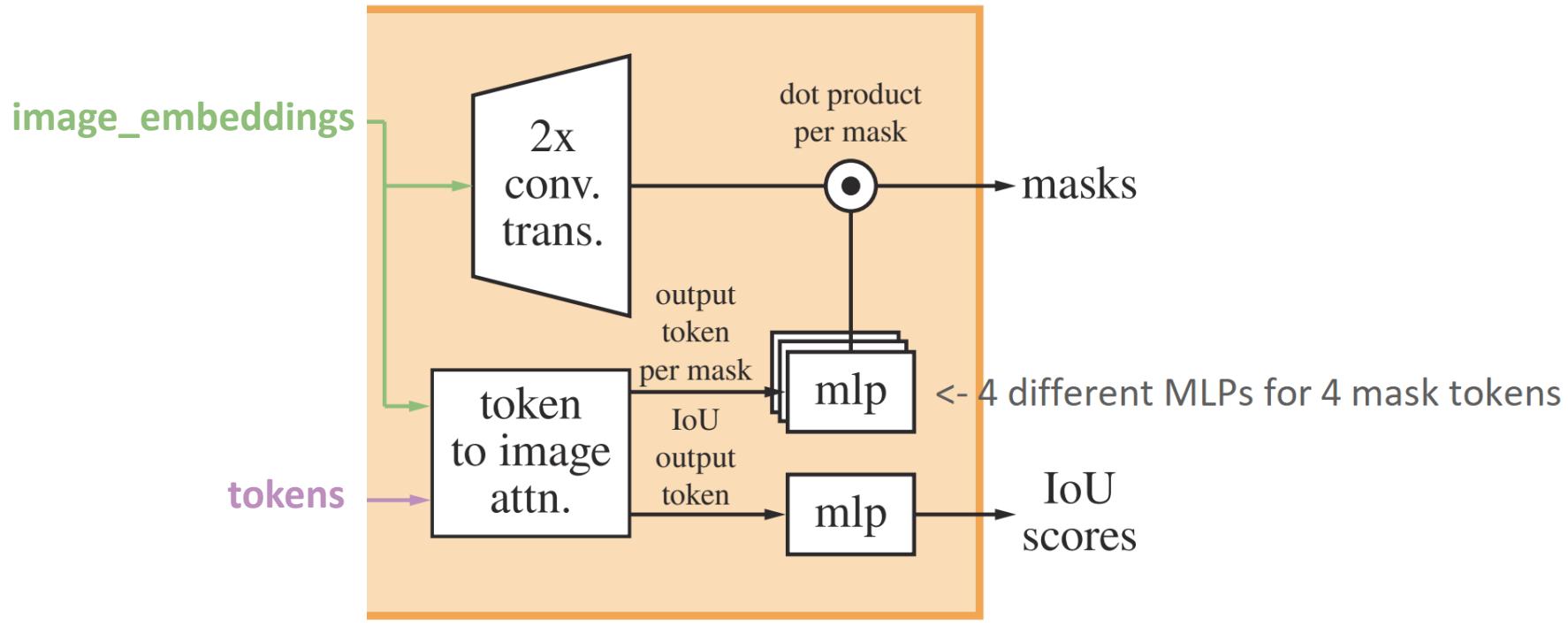
Трансформер-блок декодера



- Mask Decoder состоит из двух трансформер-блоков
- Каждый трансформер-блок обновляет и `image_embeddings`, и токены

Построение масок

Ещё одно внимание



5 MLP сетей
обрабатывают токены
ответов: токен IoU и 4
токена маски

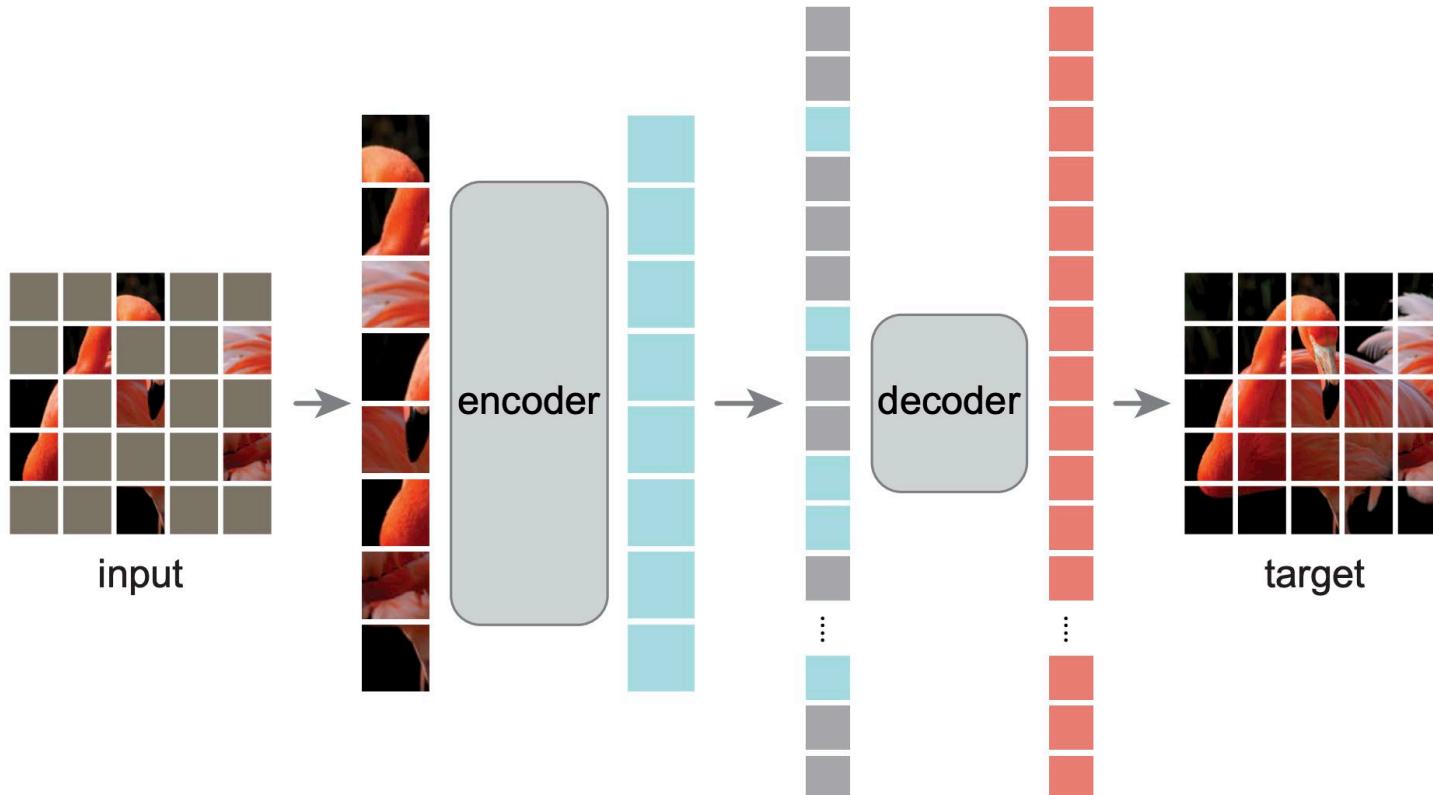
Свёрточная
увеличивает
разрешение
`image_embeddings`

Маска получается “скалярным произведением” соответствующего токена (C_i) на `image_embeddings` (C, H, W): сначала умножаем i -й канал `image_embeddings` на i -й элемент вектора, а далее вычисляем сумму вдоль оси каналов

Mask Decoder

- Эмбеддинги промптов и эмбеддинги IoU и 4 масок образовывают последовательность “токенов”
 - И токены, и image embeddings обновляются после прохода через трансформер-декодер
 - Трансформер-декодер SAM обрабатывает image embeddings и токены за один проход
- Финальные маски получаются умножением (эмбеддинга) токена маски на image embedding вдоль оси каналов и суммирования вдоль этой же оси
 - Происходит ресайз масок к оригинальными размерам изображения
- Дополнительный (эмбеддинг) токена IoU используется для предсказания 4-х iou_predictions для 4-х масок
- iou_prediction (или score, confidence) маски может быть использован для оценки качества маски

Предобучение MAE: Masked Autoencoder



- **Image Encoder** учится извлекать из изображения наиболее важные признаки без разметки
- Эффективный self-supervised learning метод для трансформеров, который обогнал контрастные методы SimCLR, MoCo3
- Идея пришла маскирования эмбеддингов пришла NLP

Promptable Segmentation: обучение

Циклично: уточнение (создание) промпта, ответ модели и подсчёт ошибки



Случайно выбирается точка в области маски из равномерного распределения (или box примерно вокруг маски)

В области ошибки сэмплируется новая точка (в зависимости от ошибки либо **позитивная**, либо **негативная**), полученная маска на предыдущем шаге подаётся как `mask_prompt` дополнительно

Шаг оптимизатора (на первый промпт 3 маски: `backpropagation` идёт от маски с наименьшим `loss`-ом, на остальные 1 маска)

И так далее..., первый промпт (box или точка) + 8 точек уточнения + 2 итерации без сэмплирования для уточнения предсказания IoU

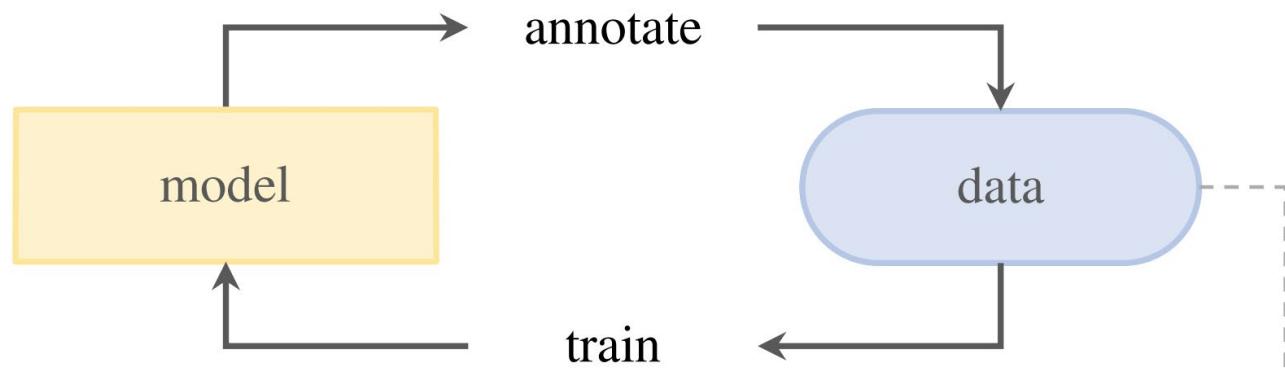
Mask Loss: $20 *$
 $focal + 1 * dice$

IoU Loss: MSE

Optimizer: AdamW

Data Engine

Основная идея – циклический процесс (до)разметки и (до)обучения модели



Segment Anything 1B (SA-1B):

- 1+ billion masks
- 11 million images
- privacy respecting
- licensed images



Этапы (stages):

1. assisted-manual
2. semi-automatic
3. fully automatic

Модель может сама размечать
данные с помощью
автоматического сэмплирования
промптов

Модель размечает, аннотаторы
подправляют

Assisted-manual Stage: все маски размечают аннотаторы

- Использовались публичные датасеты для начальной тренировки
- Далее использовались их изображения: высокого разрешения, responsible AI
- Аннотаторы выделяли все маски с помощью интерактивной сегментации, а также могли и вручную править маски с помощью кисти и ластика
- Когда было размечено много данных на SA-1B модель полностью переучилась исключительно на них.
- Было 6 итераций дообучения модели
- На разметку одного изображения тратилось 30 секунд – выделялись наиболее яркие объекты. На более поздних итерациях – меньше (14 секунд)
- Было размечено 4.3М масок на 120 тысяч изображений

Semi-automatic Stage

- Модель обучалась с помощью автоматического сэмплирования запросов в области маски
- Часть масок выделяла модель: с самым высоким predicted IoU
- Аннотаторы выделяли невыделенные маски с помощью интерактивной сегментации.
- Дообучение модели
- Было 5 итераций дообучения модели
- На разметку одного изображения тратилось 34 секунды – выделялись наиболее яркие объекты
- Было размечено новых 5.9M масок на 180 тысяч изображениях

Fully automatic Stage – только разметка

- Датасет был размечен SAM в режиме Automatic Mask Generator
- Выложили только полностью размеченную моделью часть датасета
- В итоге огромный датасет с 11 миллионами изображений и 1 миллиардом масок
- Картинки с высоким разрешением изображений (отличие от COCO)
- Responsible AI: на датасете лица и номера машин заблокированы, diversity в данных

Датасет SA-1B: responsible AI

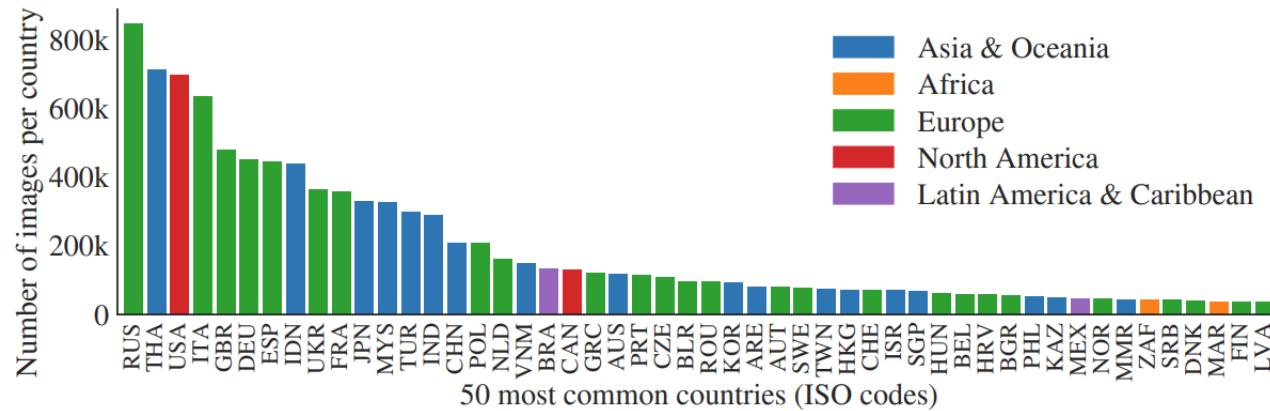
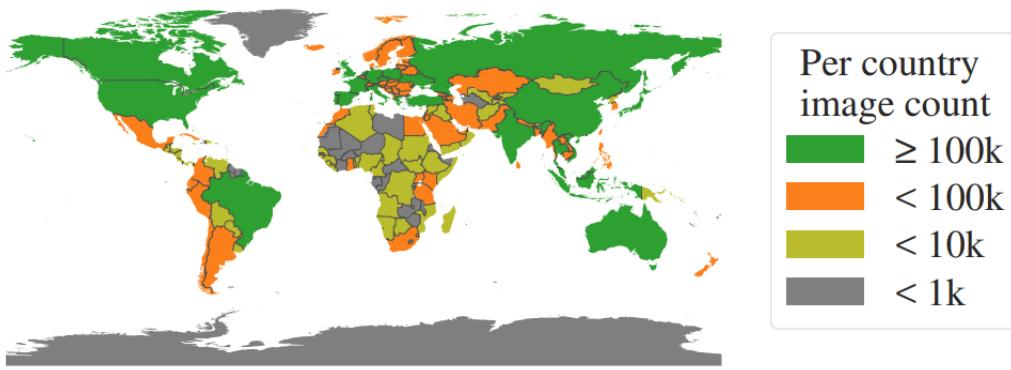


Figure 7: Estimated geographic distribution of SA-1B images. Most of the world's countries have more than 1000 images in SA-1B, and the three countries with the most images are from different parts of the world.

	mIoU at 1 point		mIoU at 3 points	
<i>perceived gender presentation</i>				
feminine	54.4 ± 1.7		90.4 ± 0.6	
masculine	55.7 ± 1.7		90.1 ± 0.6	

	mIoU at 1 point		mIoU at 3 points	
<i>perceived skin tone</i>				
1	52.9 ± 2.2		91.0 ± 0.9	
2	51.5 ± 1.4		91.1 ± 0.5	
3	52.2 ± 1.9		91.4 ± 0.7	
4	51.5 ± 2.7		91.7 ± 1.0	
5	52.4 ± 4.2		92.5 ± 1.4	
6	56.7 ± 6.3		91.2 ± 2.4	

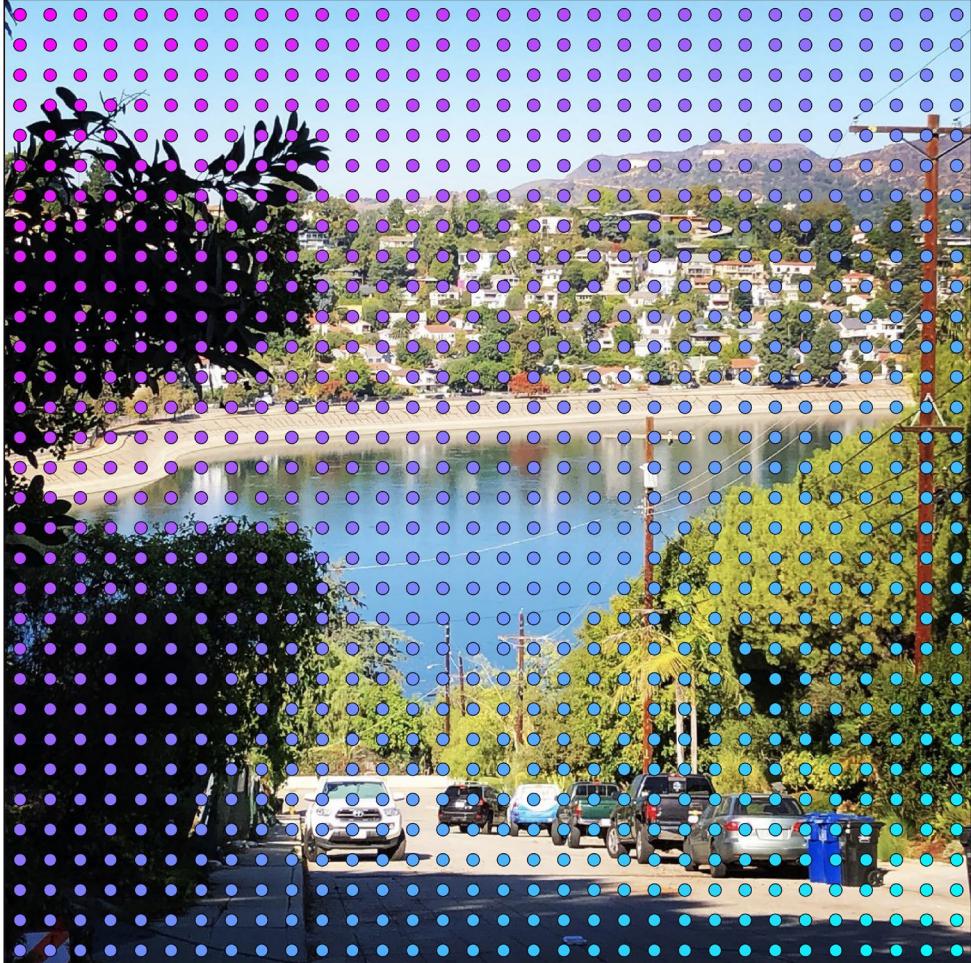
# countries	SA-1B		% images		
	#imgs	#masks	SA-1B	COCO	O.I.
Africa	54	300k	28M	2.8%	3.0%
Asia & Oceania	70	3.9M	423M	36.2%	11.4%
Europe	47	5.4M	540M	49.8%	34.2%
Latin America & Carib.	42	380k	36M	3.5%	3.1%
North America	4	830k	80M	7.7%	48.3%
high income countries	81	5.8M	598M	54.0%	89.1%
middle income countries	108	4.9M	499M	45.0%	10.5%
low income countries	28	100k	9.4M	0.9%	0.4%
				0.5%	

Automatic Mask Generator

- Ещё один режим работы SAM
- В ходе которого строятся маски всех объектов на изображении без ручных промптов – instance сегментация без классов
 - У масок отсутствуют классы
- Не требует дополнительного обучения
- Получается путём автоматической генерации одноточечных запросов и дальнейшего объединения ответов на них классическими методами CV



Cropping и генерация запросов



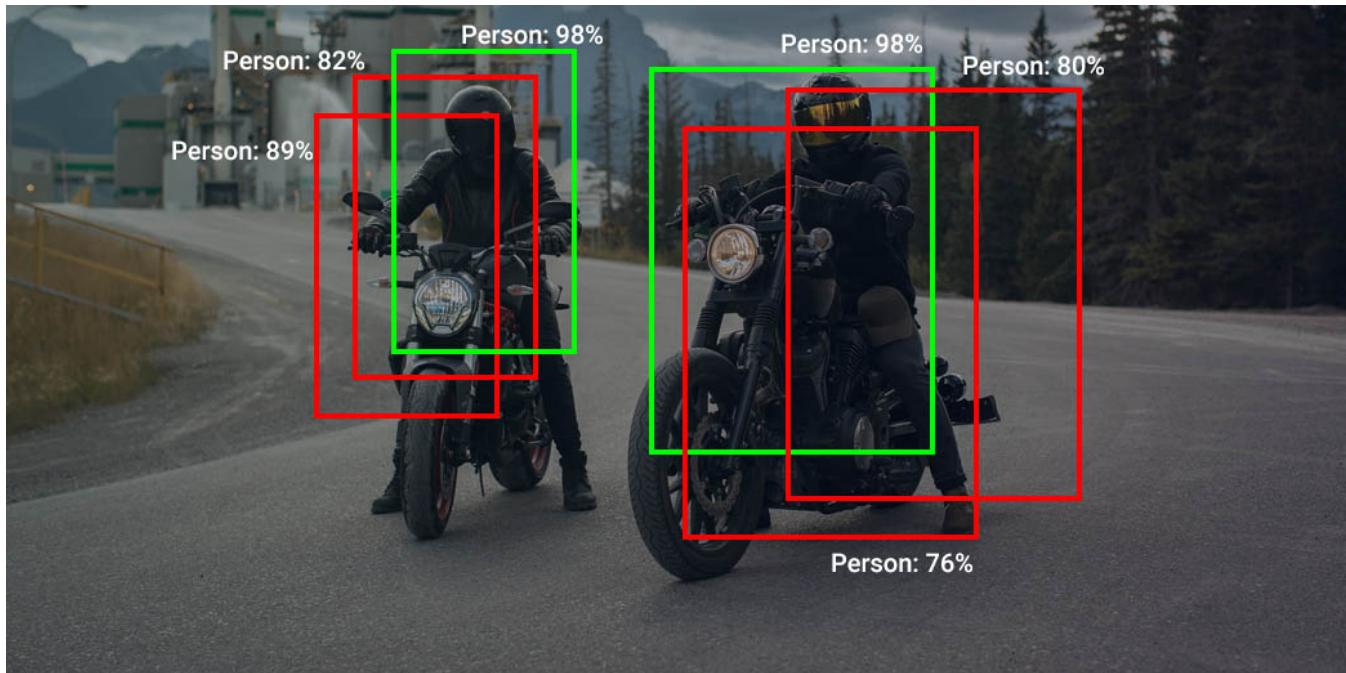
Сэмплируется сетка
32 x 32 точки

Выполняется
 $32 \times 32 = 1024$
независимых
одноточечных
промпта на
изображении
 $16 \times 16 = 256$
независимых
запросов на $\frac{1}{4}$
кропах и 4×4 на $\frac{1}{16}$
кропах

Получается много
масок, которые надо
отфильтровать

Non-Maximum Suppression (NMS)

Самый уверенный bbox подавляет остальные bbox-ы, с которыми у него близость IoU выше определённого порога (например, 0.7)



1. Сортируем список *worklist boxes* по убыванию их scores
2. Пока *worklist* не пуст, извлекаем первый элемент из списка и отправляем в ответ
3. Удаляем из *worklist* те box-ы, у которых IoU с выбранным на шаге 2 выше порога
4. Идём на шаг 2.

analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation

Фильтрация некачественных масок

- Удаляем **неуверенные маски**: маски, у которых `predicted_iou < 0.88`
- Удаляем **нестабильные маски**: маски, которые нестабильны и сильно меняют очертания при варьировании порогов (с расплывчатыми границами)



- Удаляем **огромные маски**: если маска покрывает больше 95% изображения

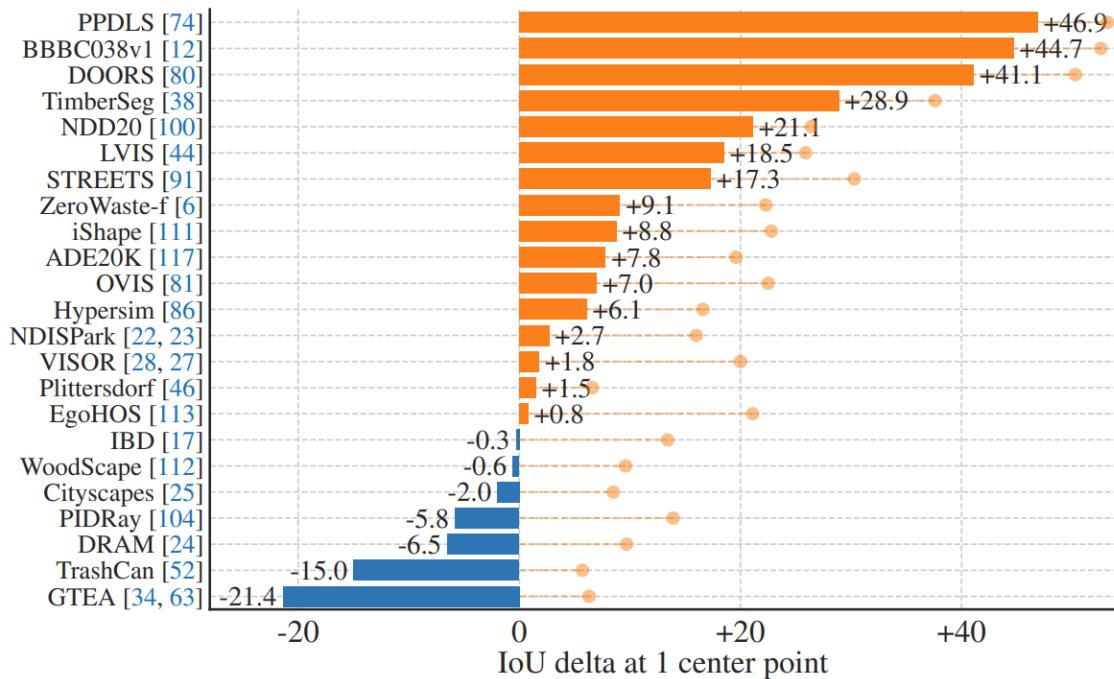
Постобработка

- **Удаление маленьких масок и маленьких несвязанных компонент на масках:** около 4% оставшихся масок содержат маленькие регионы размера меньше 100 пикселей.
- **Заполнение дыр:** ещё 4% масок содержат дыры, они заполняют дыры, площадь которых меньше чем 100 пикселей

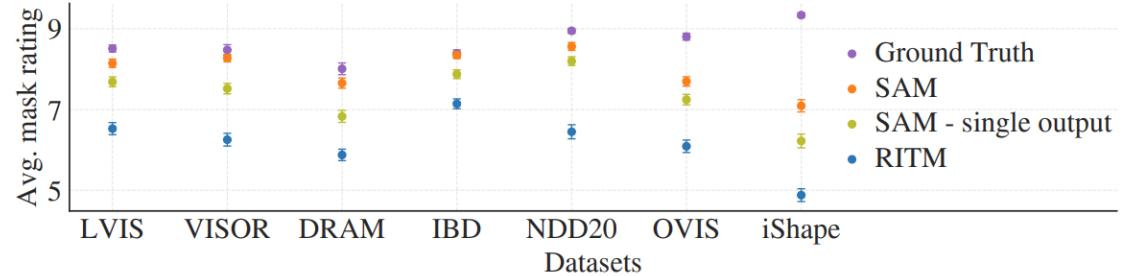
Automatic Mask Generator

- Заранее сэмплируются точечные запросы в разных регионах изображения
- Для разномасштабных масок подаются не только целые изображения, но ещё и части изображения
- Ответы за запросы объединяются, теперь надо удалить дублирующиеся маски
 - Неуверенные дубликаты подавляются с помощью NMS
- Далее происходит фильтрация: неуверенных, нестабильных и больших масок
 - Постобработка заполняет дыры на масках

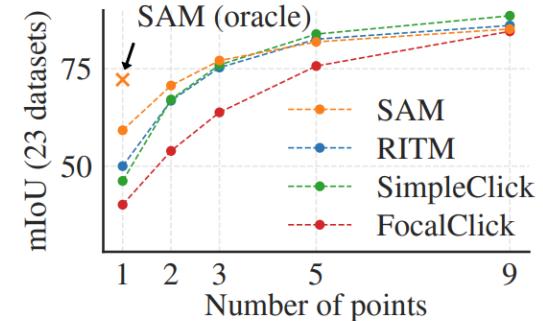
Zero-Shot интерактивная сегментация



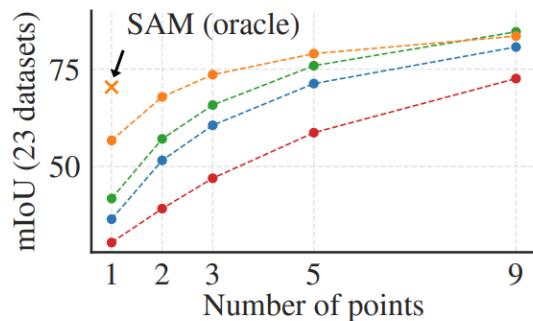
(a) SAM vs. RITM [92] on 23 datasets



(b) Mask quality ratings by human annotators



(c) Center points (default)



(d) Random points

SAM (oracle) – выбор из 3 масок наиболее близкой к ground truth: “we also present an “oracle” result, in which the most relevant of SAM’s 3 masks is selected by comparing them to the ground truth, rather than selecting the most confident mask. This reveals the impact of ambiguity on automatic evaluation. In particular, with the oracle to perform ambiguity resolution, SAM outperforms RITM on all datasets”

Zero-Shot Object Proposals на LVIS

method	all	mask AR@1000					
		small	med.	large	freq.	com.	rare
ViTDet-H [62]	63.0	51.7	80.8	87.0	63.1	63.3	58.3
<i>zero-shot transfer methods:</i>							
SAM – single out.	54.9	42.8	76.7	74.4	54.7	59.8	62.0
SAM	59.3	45.5	81.6	86.9	59.1	63.9	65.8

Table 4: Object proposal generation on LVIS v1. SAM is applied zero-shot, *i.e.* it was not trained for object proposal generation nor did it access LVIS images or annotations.

- SAM хуже: “In fact, SAM only underperforms ViTDet-H on small objects and frequent objects, where ViTDet-H can easily learn LVIS specific annotation biases since it was trained on LVIS, unlike SAM.”
- Также подкрутили некоторые пороги Automatic Mask Generator, чтобы получить лучшие результаты на конкретном датасете: “since this model produces fewer masks, we further increase the number of points sampled and NMS threshold to 128×128 and 0.95, respectively, obtaining ~ 950 masks per image on average.”

Zero-Shot instance сегментация на LVIS

method	COCO [66]				LVIS v1 [44]			
	AP	AP ^S	AP ^M	AP ^L	AP	AP ^S	AP ^M	AP ^L
ViTDet-H [62]	51.0	32.0	54.3	68.9	46.6	35.0	58.0	66.3
<i>zero-shot transfer methods (segmentation module only):</i>								
SAM	46.5	30.8	51.0	61.7	44.7	32.5	57.6	65.5

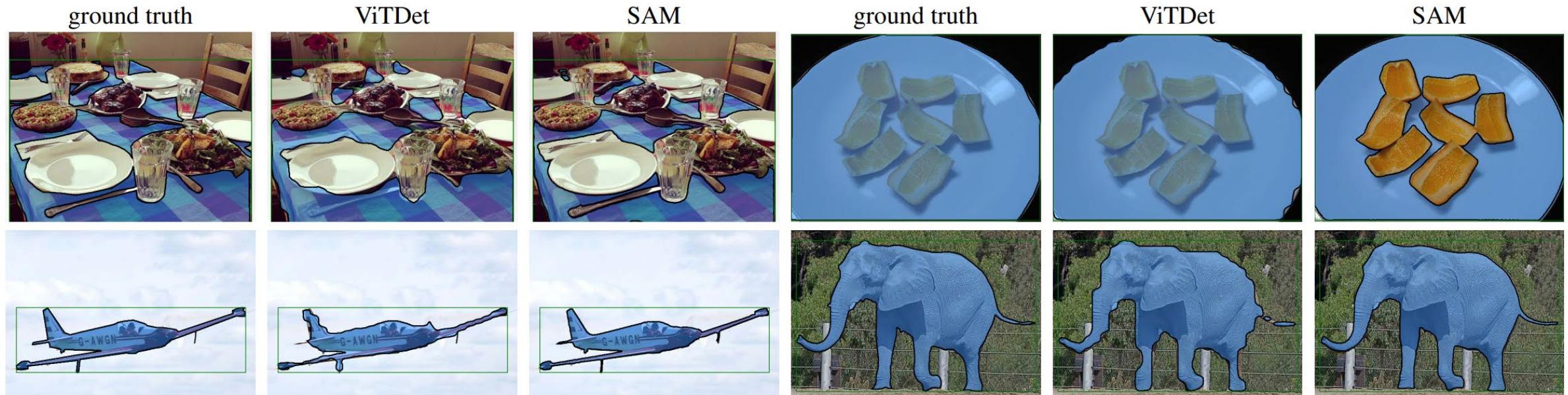


Figure 16: Zero-shot instance segmentation on LVIS v1. SAM produces higher quality masks than ViTDet. As a zero-shot model, SAM does not have the opportunity to learn specific training data biases; see top-right as an example where SAM makes a modal prediction, whereas the ground truth in LVIS is amodal given that mask annotations in LVIS have no holes.

Zero-Shot Text-to-Mask

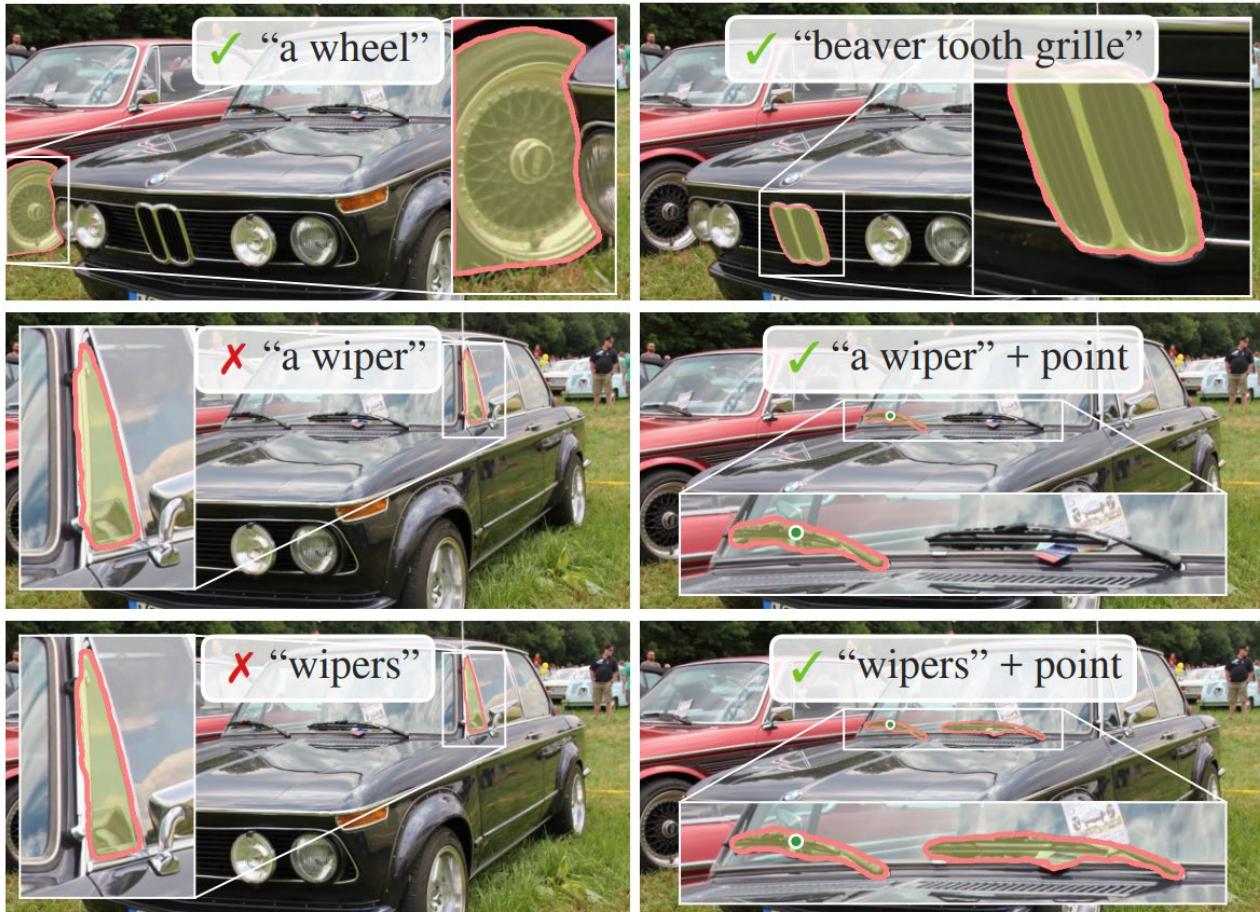


Figure 12: Zero-shot text-to-mask. SAM can work with simple and nuanced text prompts. When SAM fails to make a correct prediction, an additional point prompt can help.

Иногда необходима дополнительная точка, так как на только текстовый запрос нет правильного ответа: “we show qualitative results in Fig. 12. SAM can segment objects based on simple text prompts like “a wheel” as well as phrases like “beaver tooth grille”. When SAM fails to pick the right object from a text prompt only, an additional point often fixes the prediction...”

Заключение

- Как интерактор модель существенно обходит RITM и другие при промптах из 1 точки, но слегка уступает им при запросах с 5+ точками
- SAM may miss fine structures, hallucinate small disconnected components at times, and produce wrong boundaries
- Как Automatic Mask Generator + сведение к другим задачам, модель (без fine-tuning) работает чуть хуже чем SOTA решения
 - SAM не поддерживает предсказание классов
- SAM не способна решать задачи semantic segmentation и panoptic segmentation

Спасибо за внимание!

