VQ-VAE, DALL-E и Gumbel-Softmax

Сергей Николенко

СПбГУ — Санкт-Петербург

13 ноября 2026 г.





Random facts:

- 13 ноября 1833 г. над Северной Америкой наблюдали мощный метеорный шторм Леонид, ставший важной вехой в изучении метеоров
- 13 ноября 1851 г. по подводному телеграфному кабелю Дувр — Кале передали первое сообщение; кабель прослужил до 1875 года
- 13 ноября 1927 г. туннель Холланда под Гудзоном открылся для автомобильного движения между Нью-Йорком и Нью-Джерси
- 13 ноября 1940 г. в Нью-Йорке состоялась премьера «Фантазии» Диснея, соединившей анимацию и классическую музыку
- 13 ноября 1971 г. Mariner 9 вышел на орбиту Марса и стал первым искусственным спутником другой планеты
- 13 ноября 1982 г. в Вашингтоне торжественно открыли Мемориал ветеранов Вьетнама по проекту Майи Лин

Дискретные латентные пространства в вариационных автокодировщиках

VQ-VAE

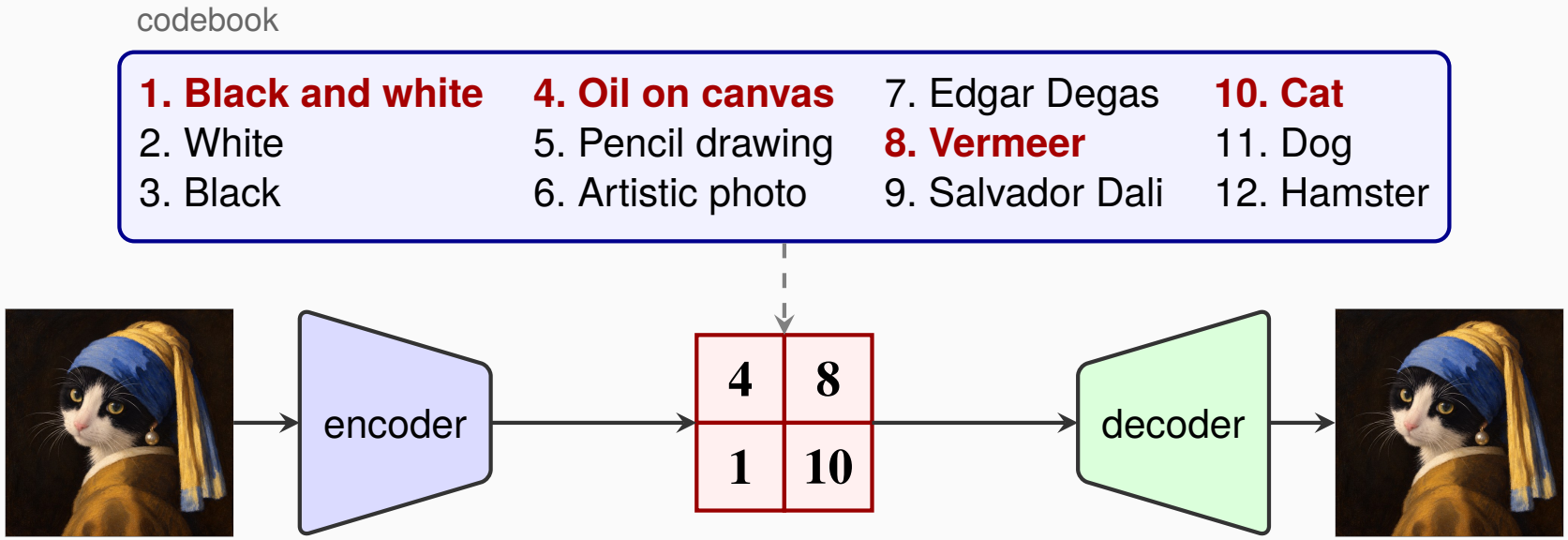
- Куда пришла идея VAE сегодня? Для этого нужна ещё одна модификация в нашу конструкцию
- VQ-VAE (Vector Quantized VAE; Razavi et al., 2019, DeepMind):



- Как такая красота получается?

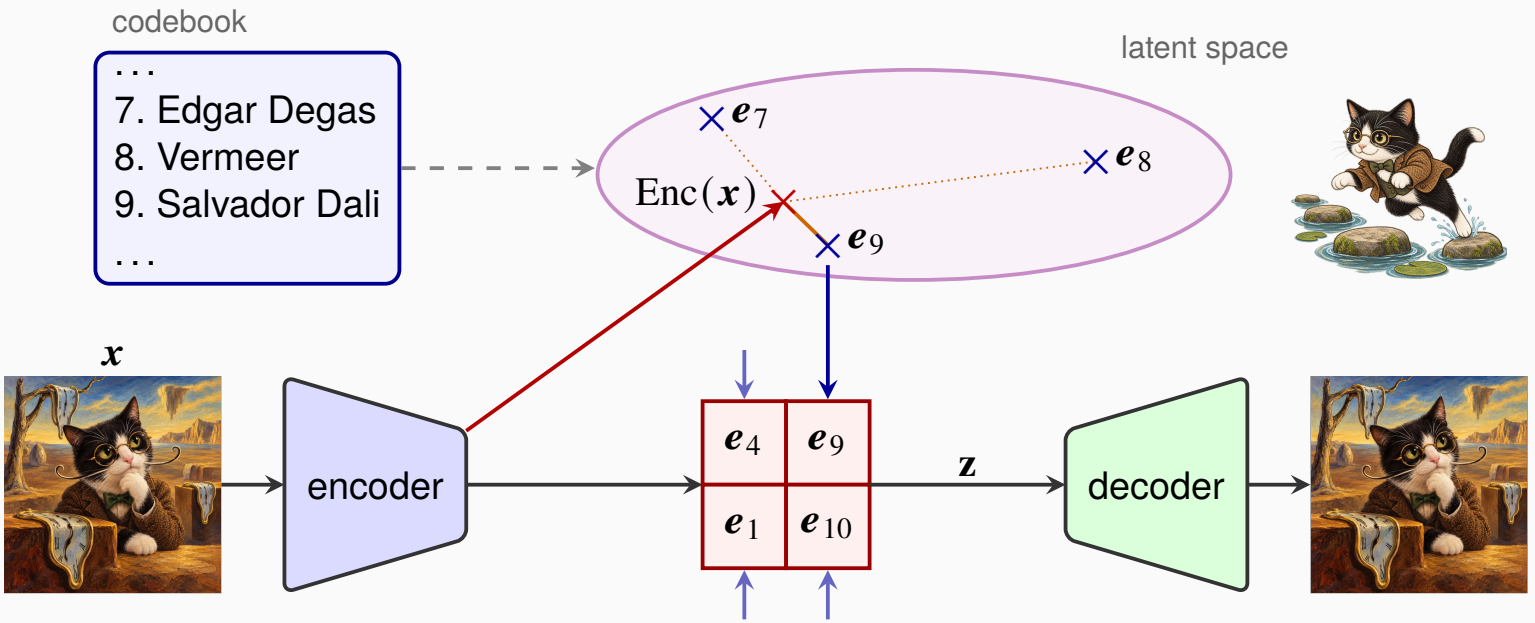
VQ-VAE

- Идея VQ-VAE (van den Oord et al., 2017): давайте квантизуем латентный код, т.е. код – это как бы «язык», на котором переговариваются кодировщик и декодировщик:



VQ-VAE

- Для этого:
  - кодировщик, как обычно, по $\mathbf{x}$ строит тензор из латентных кодов $\mathrm{Enc}(\mathbf{x})$ (теперь целый тензор);
  - для каждого $\mathbf{z}$ мы находим ближайшего соседа в словаре и заменяем на него;
  - декодер получает тензор уже из словарных векторов, с конечным множеством возможных вариантов.

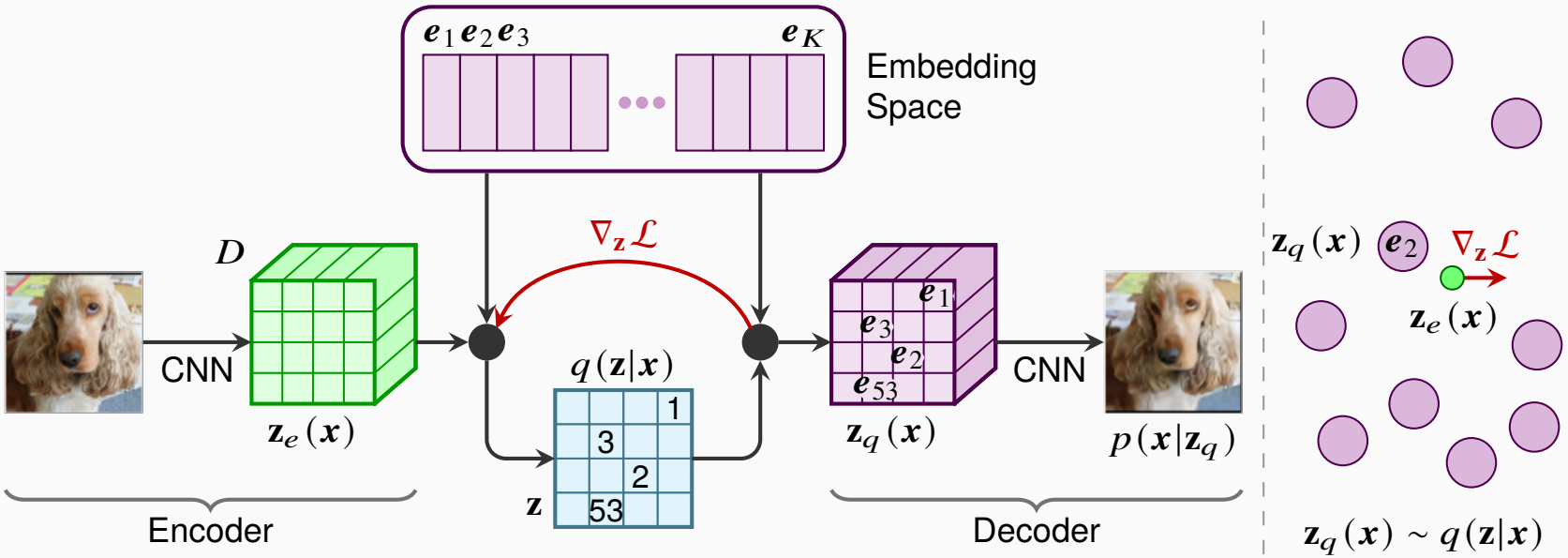


VQ-VAE

- Формально говоря,

$$\text{Quant}(\mathbf{z}_e(\mathbf{x})) = \mathbf{e}_k, \qquad k = \arg\min_j \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_j\|_2.$$

- Через это градиент не пропустишь, но давайте градиент просто скопируем:



VQ-VAE

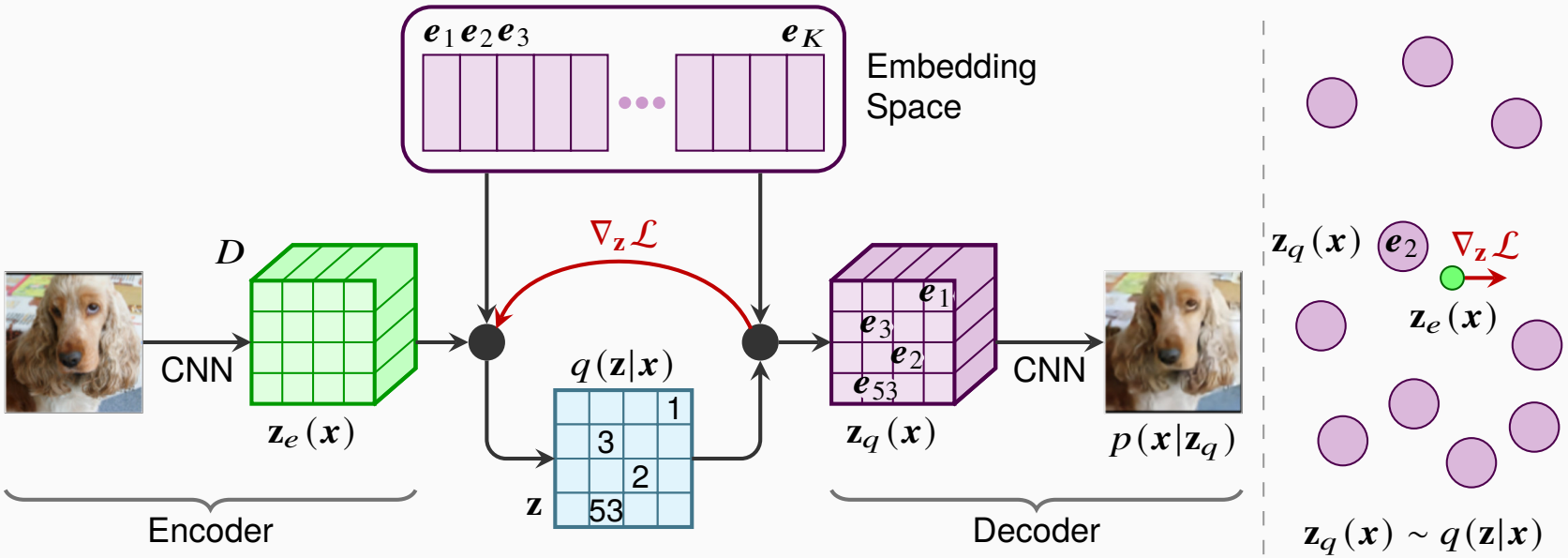
- Как обучать тогда эти embeddings, если там нет градиентов от реконструкции?
- Давайте просто добавим обычным vector quantization алгоритмом, т.е. будем подтягивать $\mathbf{e}$ к выходам кодировщика $\mathbf{z}_e(\mathbf{x})$.
- Чтобы кодировщик не убегал неограниченно от embeddings, его тоже к embeddings подтягиваем.
- Т.е. функция ошибки получается такая:

$$L = \log p\left(\mathbf{x}\middle|\mathbf{z}_q(\mathbf{x})\right) + \left\|\mathrm{sg}[\mathbf{z}_e(\mathbf{x})] - \mathbf{e}\right\|_2^2 + \beta\left\|\mathbf{z}_e(\mathbf{x}) - \mathrm{sg}[\mathbf{e}]\right\|_2^2,$$

где sg (stopgradient) – это оператор, который останавливает градиент (id в прямую сторону, 0 в обратную); т.е. декодер оптимизирует первое слагаемое, кодировщик первое и третье, а сами embeddings оптимизируются вторым слагаемым.

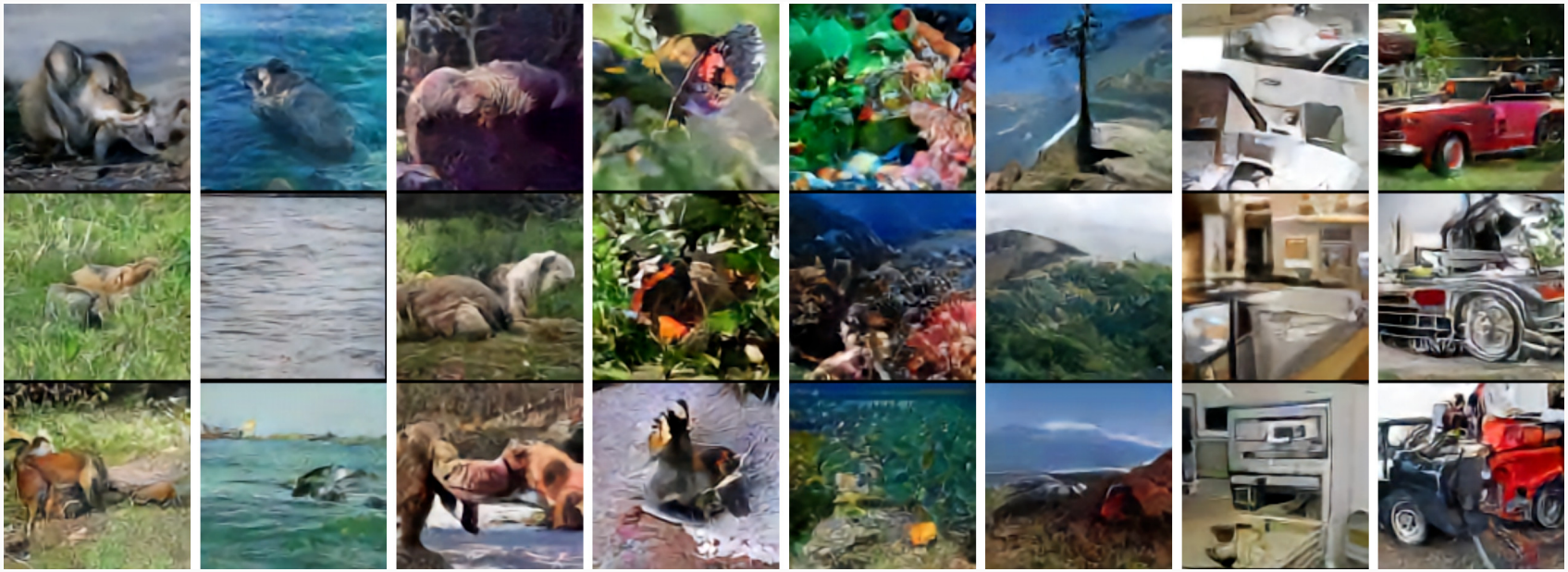
VQ-VAE

- Вот такая конструкция:



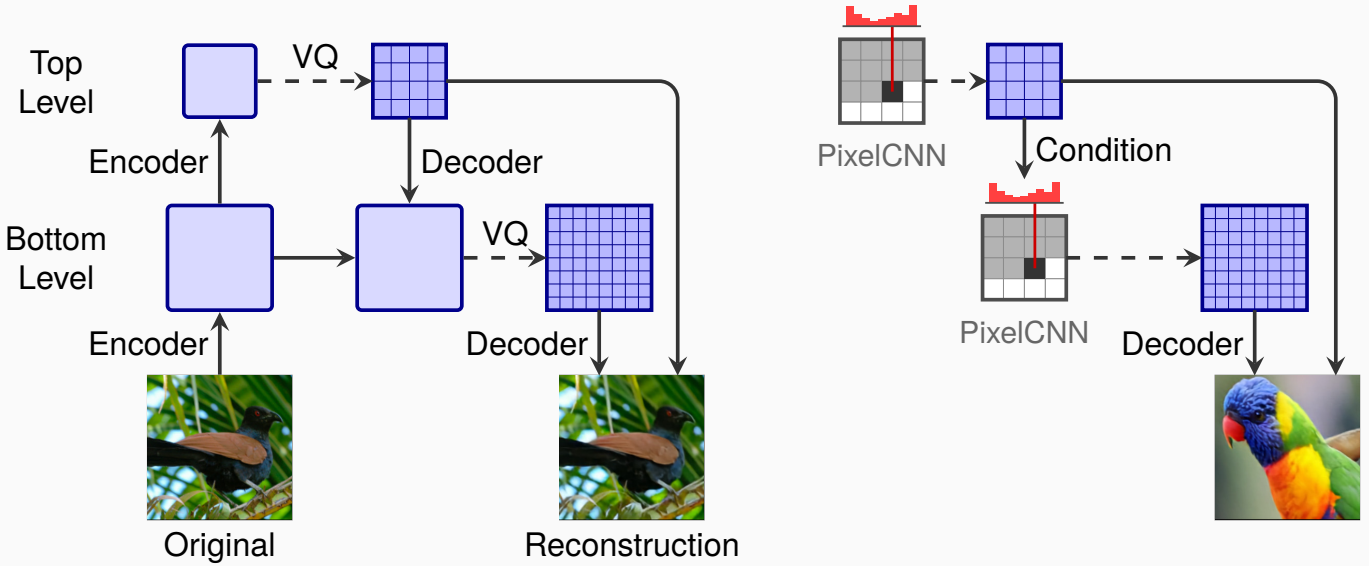
VQ-VAE

- Уже исходный VQ-VAE на основе PixelCNN неплохо выглядел для 2017 года:



VQ-VAE

- А в VQ-VAE-2 процесс сделали иерархическим:



- Пример реконструкций по разным латентным картам:



VQ-VAE

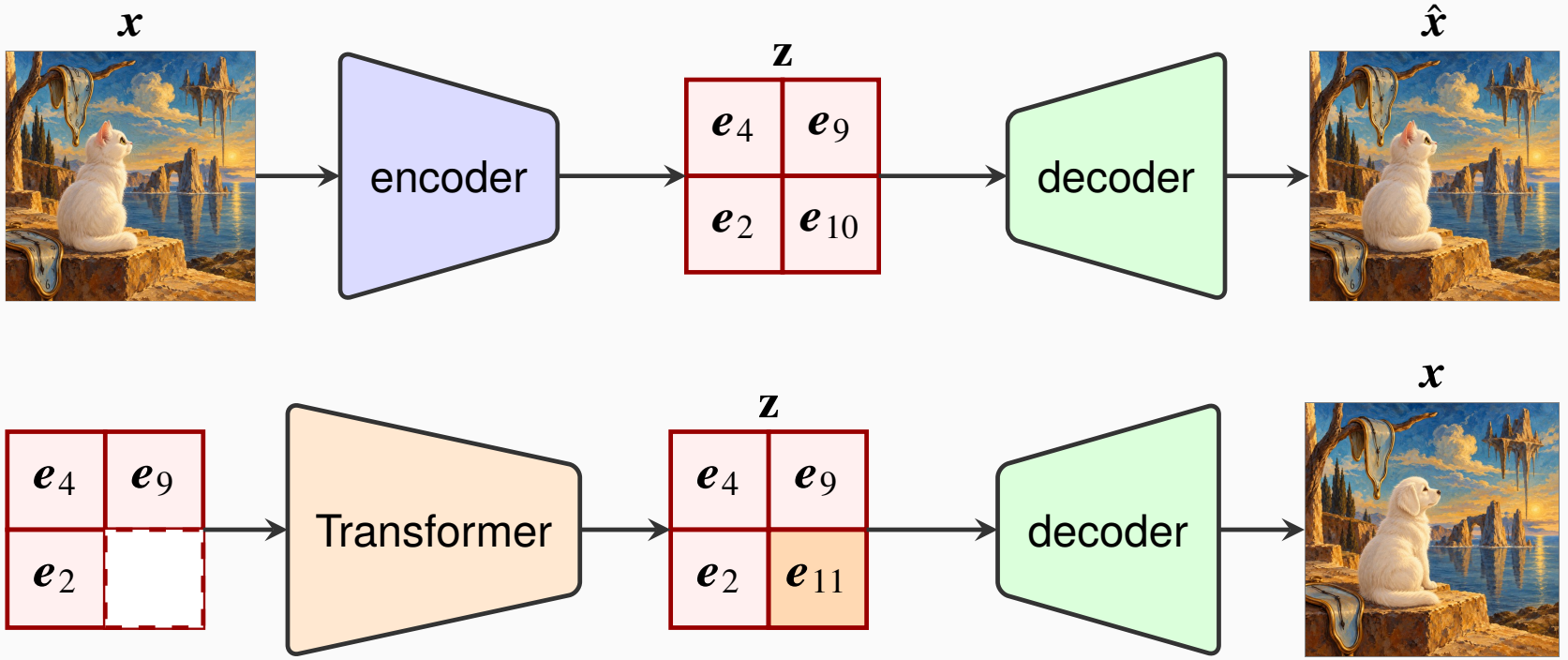
- Получается круто:





VQ-GAN

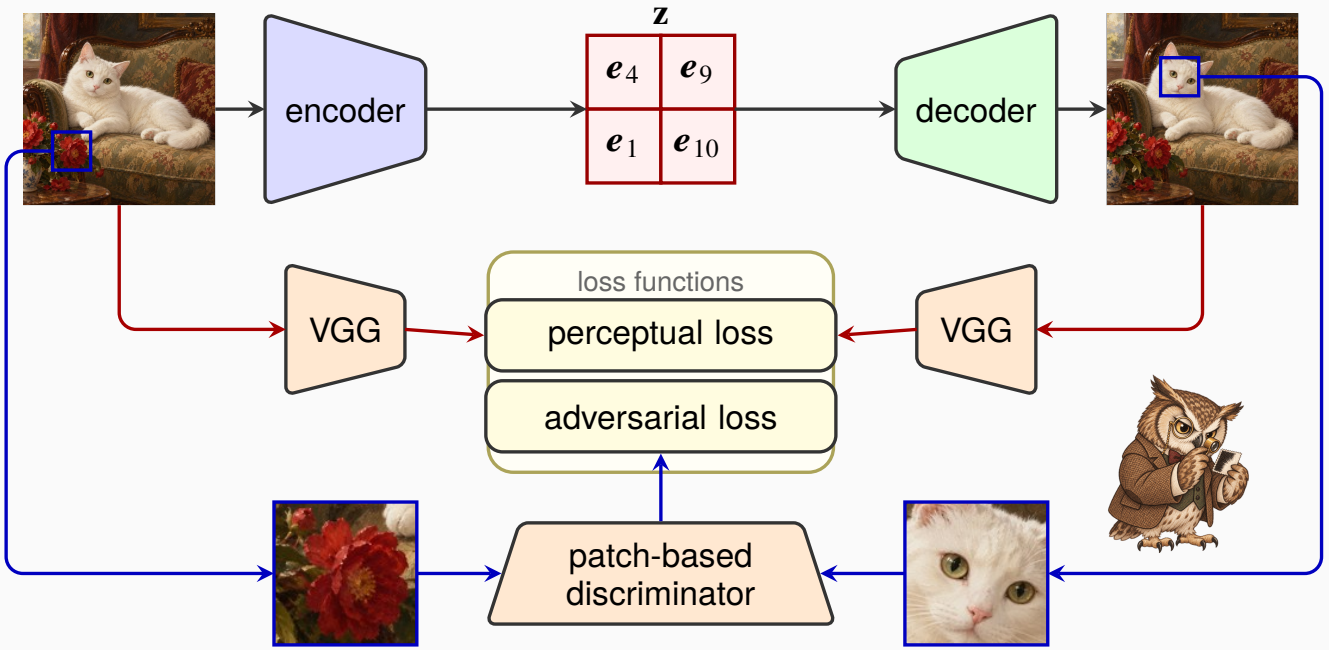
- Но всё-таки в итоге VQ-VAE победил, особенно когда превратился в VQ-GAN.
- Мы бы хотели производить латентные коды не PixelCNN, а трансформером:



- Но для этого нужен очень выразительный словарь

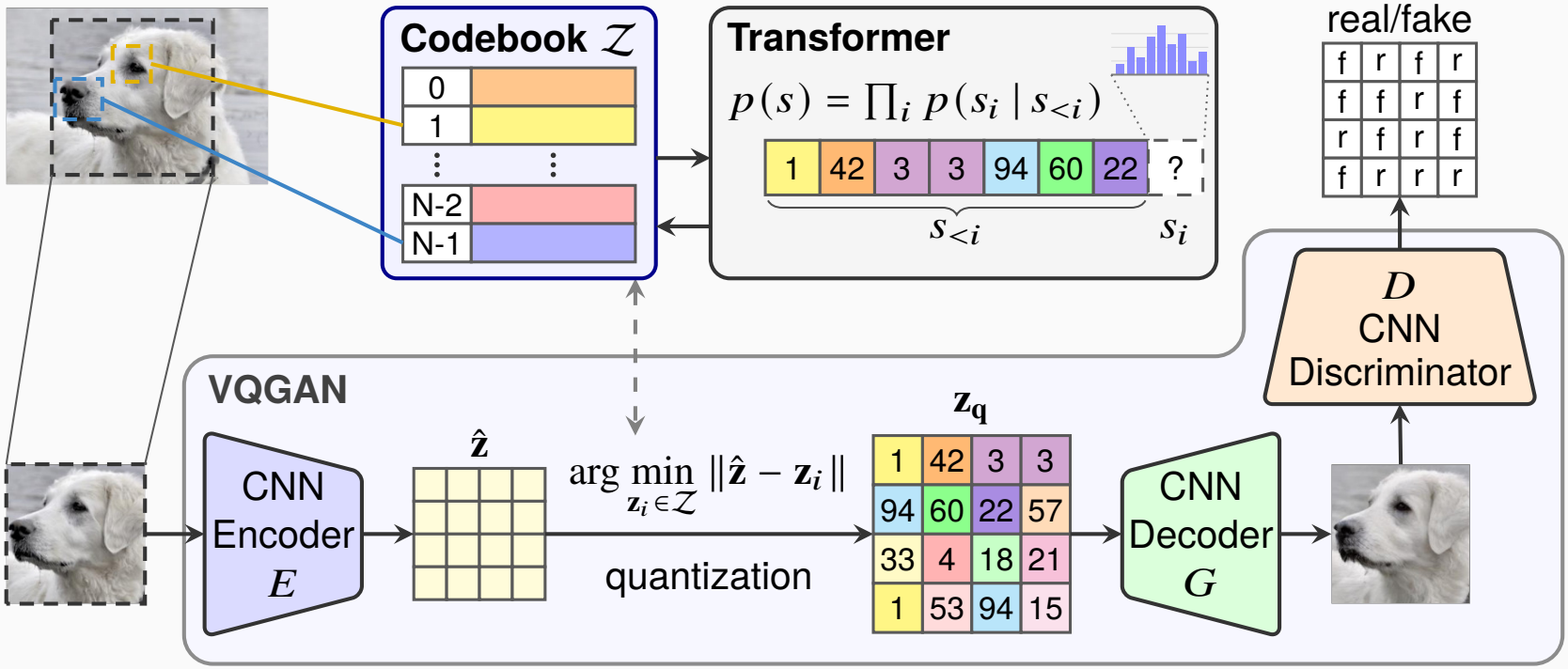
VQ-GAN

- Поэтому вместо обычной reconstruction loss VQ-GAN (Esser et al., 2021) добавляет patch-based discriminator, а базовая ошибка превращается в perceptual loss



VQ-GAN

- Дискриминатор теперь занимается локальной структурой порождённой картинки, а perceptual loss — собственно контентом
- Исходная иллюстрация:



VQ-GAN

- Получается ещё лучше, в том числе на базовом ImageNet:



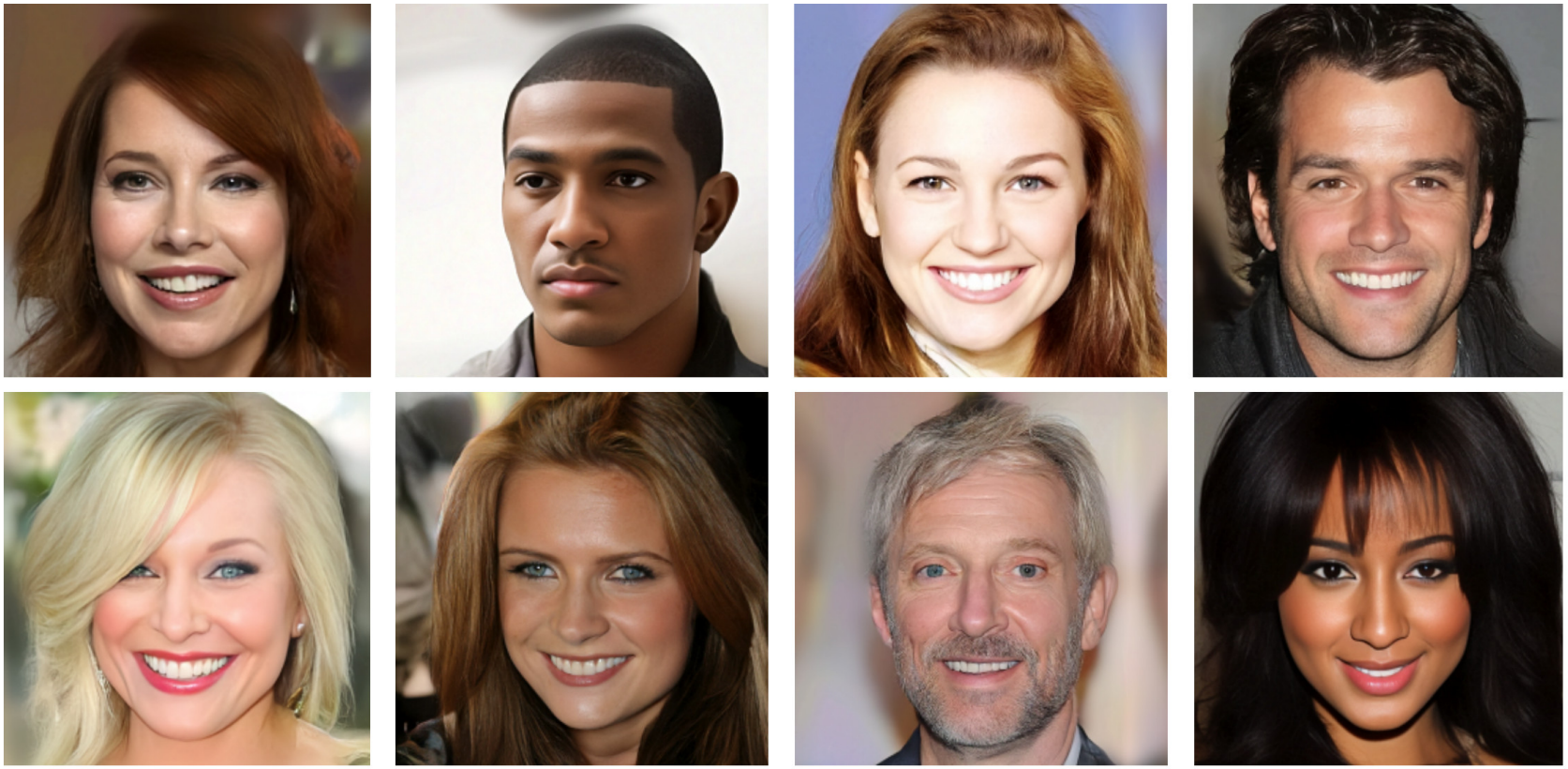
VQ-GAN

- А если добавить условие, например грубую карту сегментации, то с этим условием VQ-GAN уже мог создавать большие почти фотореалистичные изображения (ниже пример из 1280×832 пикселей):



VQ-GAN

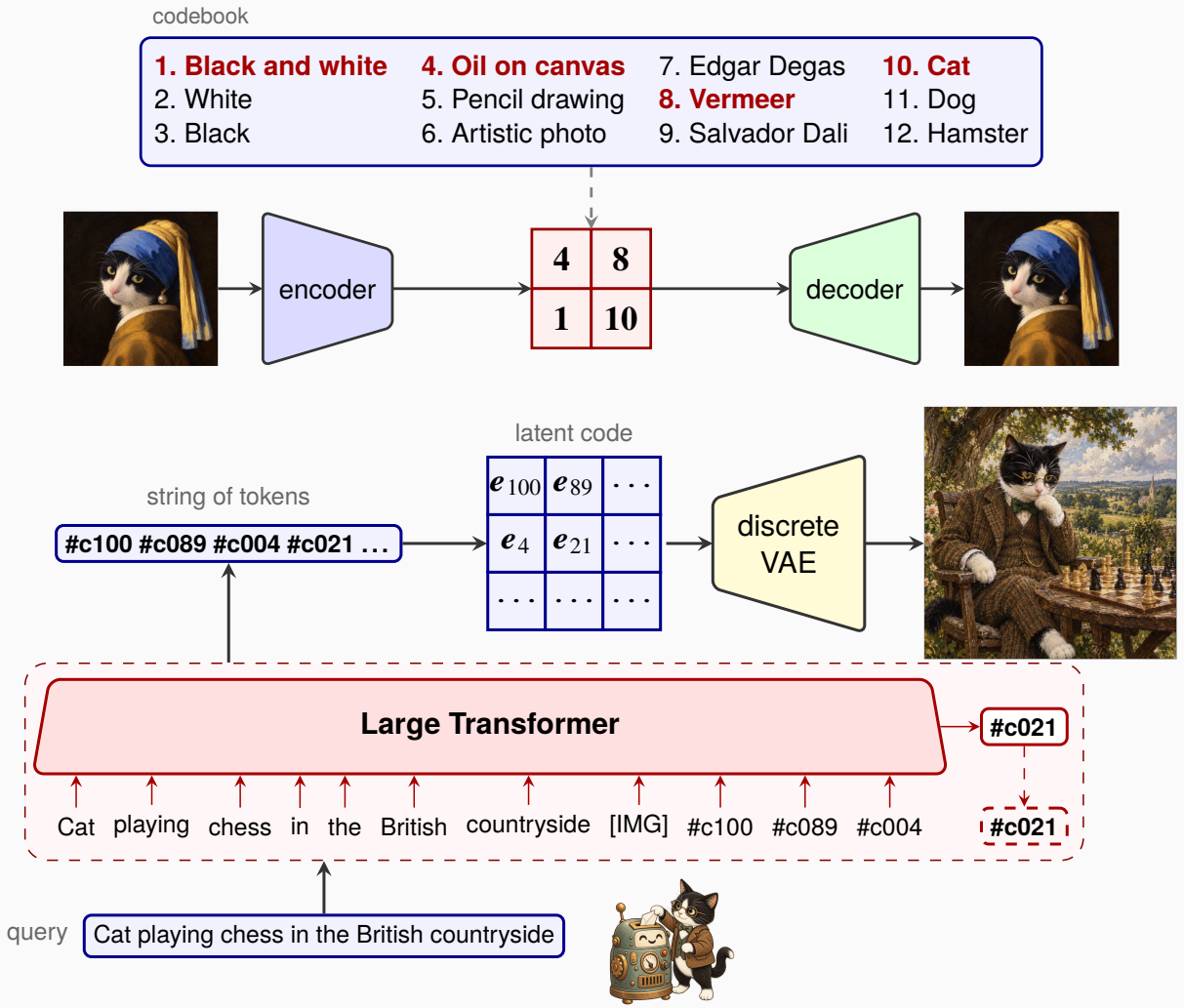
- В общем, хорошо получается:



DALL-E

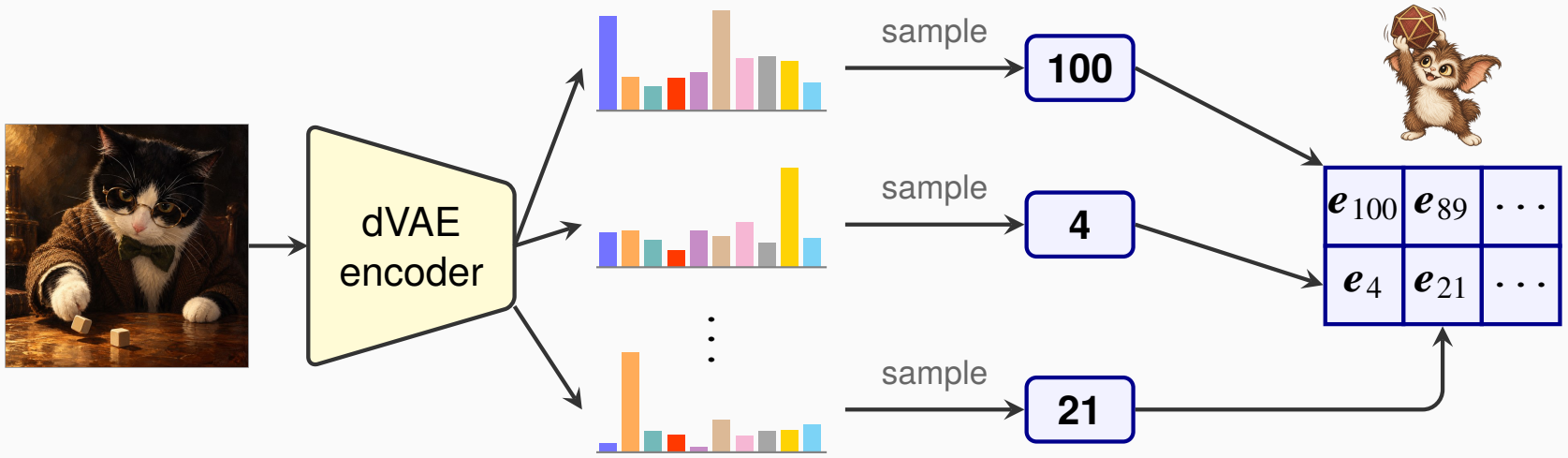
dVAE + Transformer = DALL-E

- Мы уже обсуждали VQ-VAE:



dVAE + Transformer = DALL-E

- dVAE – это VQ-VAE, который выдаёт не один вектор из словаря, а распределение на словаре:

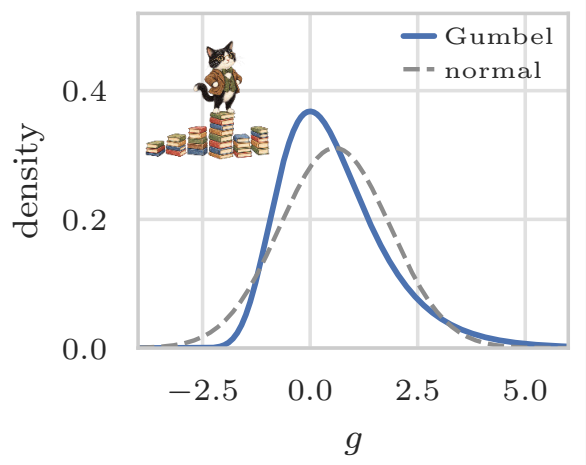


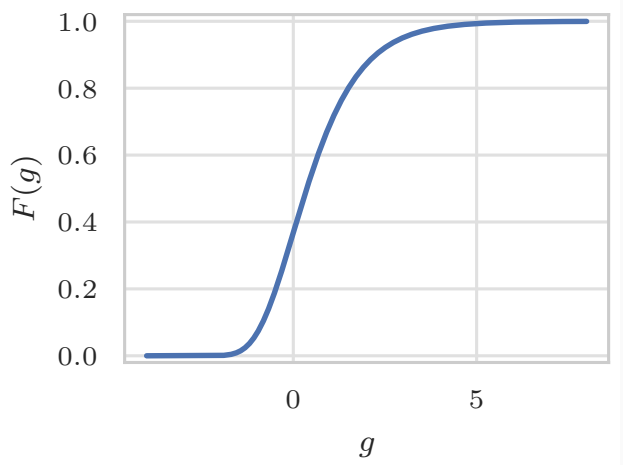
- Как протащить градиент через сэмплирование?

dVAE + Transformer = DALL-E

- dVAE считает градиенты не так, как VQ-VAE; для дискретизации служит *Gumbel-Softmax distribution*:
- Gumbel-Max trick: можно сэмплировать из дискретного распределения с вероятностями классов π_i как $z = \arg\max_i (g_i + \log \pi_i)$, где g_i берутся из распределения Гумбеля:

$$p(g_i) = e^{-(g_i + e^{-g_i})}, \quad F(g_i) = e^{-e^{-g_i}}.$$





dVAE + Transformer = DALL-E

- Объяснение трюка:

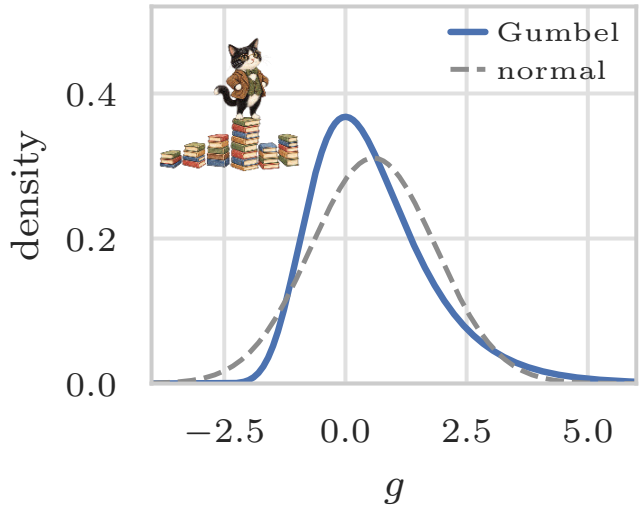
$$
\begin{aligned}
p(z=k) &= p\left(g_k + \log \pi_k \geq g_j + \log \pi_j \text{ для всех } j\right) = \\
&= \int_{-\infty}^{\infty} \prod_{j \neq k} p\left(g_k + \log \pi_k \geq g_j + \log \pi_j \middle| g_k\right) p(g_k)\, \mathrm{d}g_k = \\
&= \int_{-\infty}^{\infty} \prod_{j \neq k} e^{-e^{-g_k - \log \pi_k + \log \pi_j}} e^{-(g_k + e^{-g_k})} \mathrm{d}g_k = \\
&= \int_{-\infty}^{\infty} e^{-\sum_{j \neq k} \pi_j e^{-g_k - \log \pi_k}} \pi_k e^{-\left(g_k + \log \pi_k + \pi_k e^{-g_k - \log \pi_k}\right)} \mathrm{d}g_k = \\
&= \pi_k \int_{-\infty}^{\infty} e^{-g_k - \log \pi_k - \left(\pi_k + \sum_{j \neq k} \pi_j\right) e^{-g_k - \log \pi_k}} \mathrm{d}g_k = \pi_k.
\end{aligned}
$$

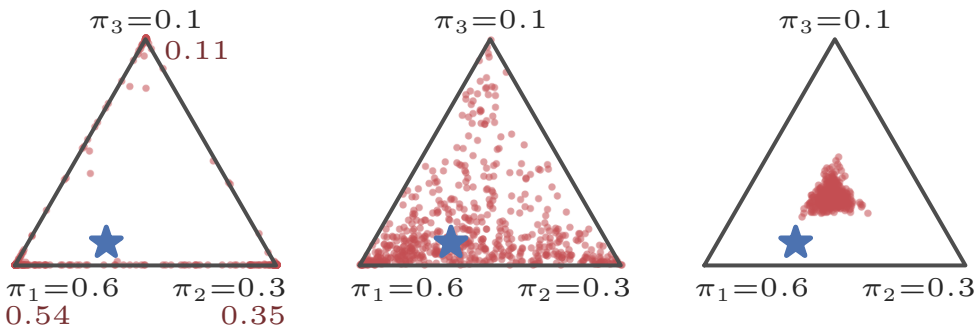
dVAE + Transformer = DALL-E

- A Gumbel-Softmax — это релаксация дискретного распределения:

$$y_i = \mathrm{softmax}\left(\frac{g_i + \log \pi_i}{\tau}\right) = \frac{e^{\frac{1}{\tau}(g_i + \log \pi_i)}}{\sum_j e^{\frac{1}{\tau}(g_j + \log \pi_j)}},$$

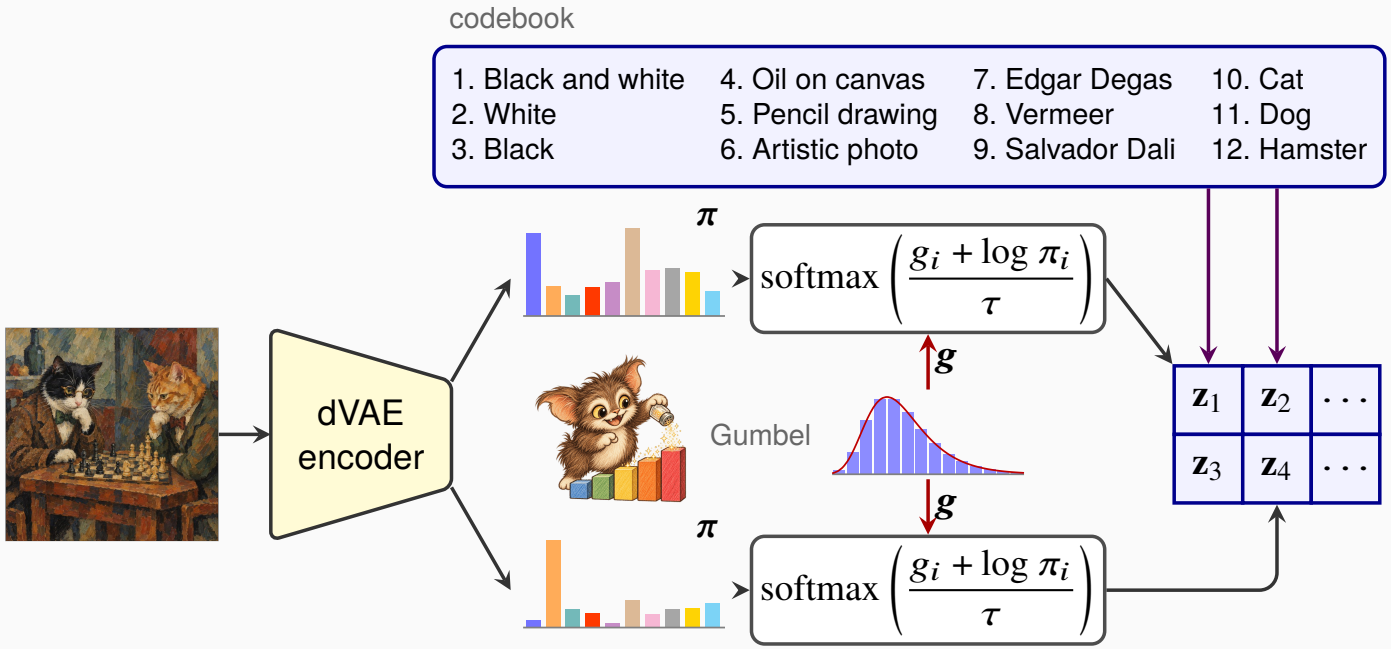
и при $\tau \to 0$ это стремится к дискретному распределению с вероятностями π_i.





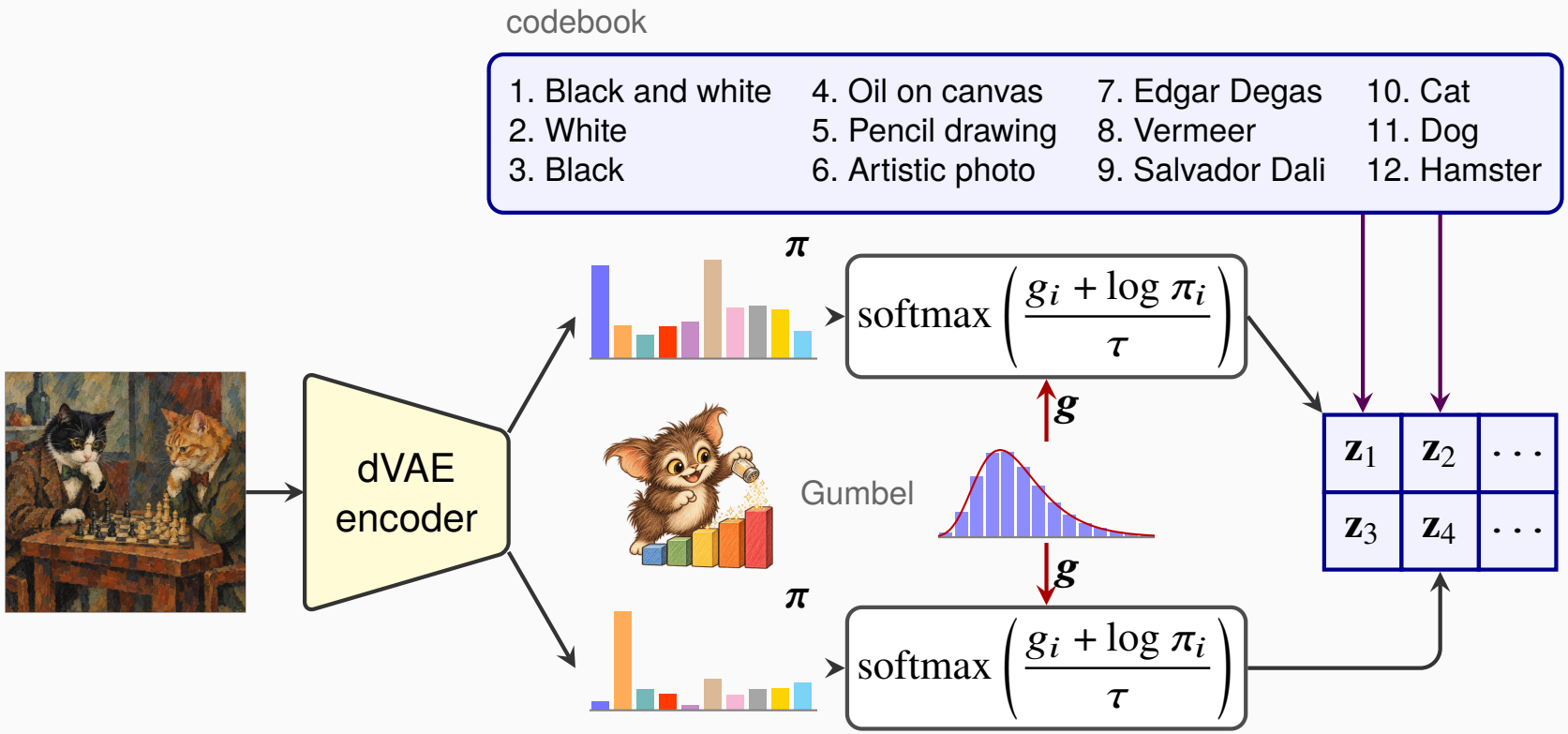
dVAE + Transformer = DALL-E

- Это то, что мы делаем с кодовыми векторами — сэмплируем выпуклую комбинацию $\mathbf{z} = \sum_{j=1}^{k} y_j \mathbf{e}_j$ через Gumbel-Softmax (это тоже, кстати, reparametrization trick: теперь g_i сэмплируются отдельно), и теперь все градиенты проходят, а во время обучения потихоньку устремляем $\tau \to 0$:



dVAE + Transformer = DALL-E

- Общая картина:

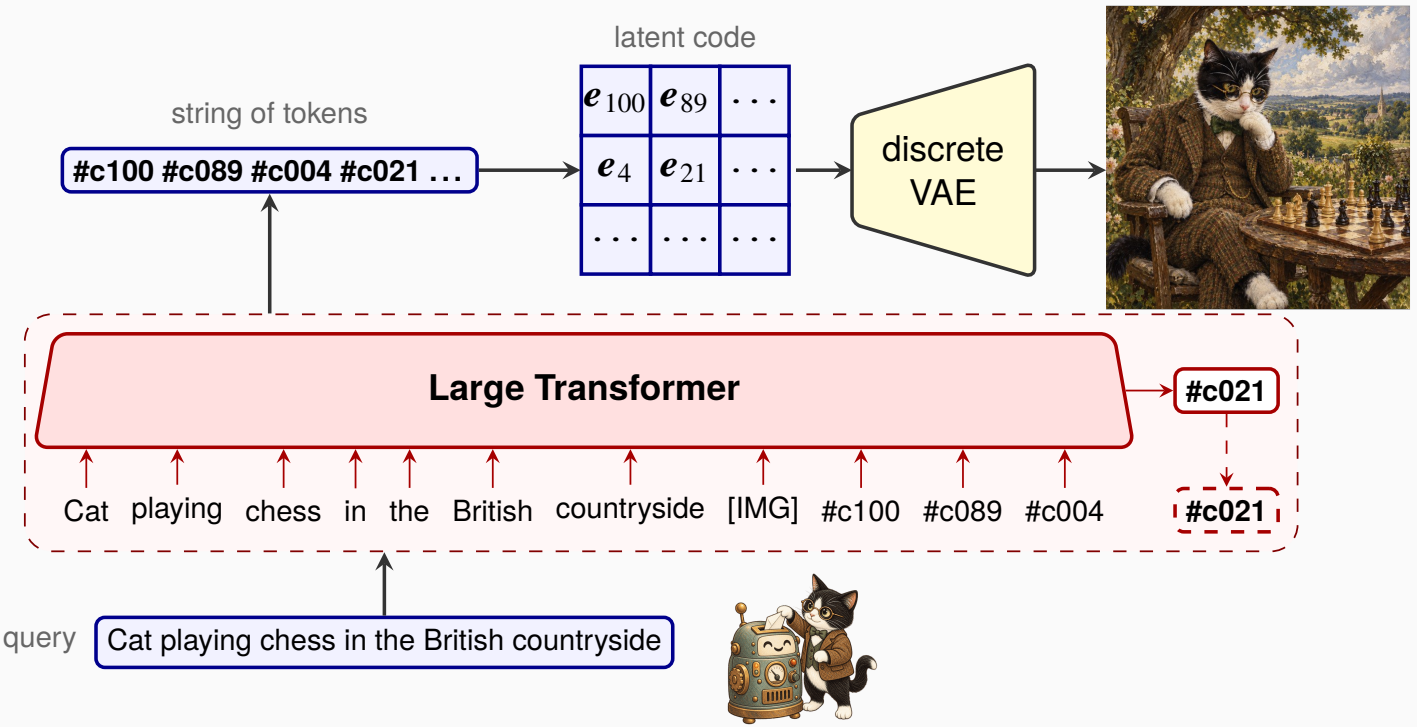


dVAE + Transformer = DALL-E

- И теперь DALL-E моделирует код $\mathbf{z}$ из текста $\mathbf{y}$, а потом использует его в dVAE, чтобы получить картинку $\mathbf{x}$:

$$p_{\theta,\psi}(\mathbf{x},\mathbf{y},\mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})\, p_\psi(\mathbf{y},\mathbf{z})\,.$$

- Трансформер порождает только $\mathbf{z}$, как последовательность:



dVAE + Transformer = DALL-E

- То есть фактически это одна большая вариационная оценка:

$$\ln p_{\theta,\psi}(\mathbf{x},\mathbf{y}) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x}|\mathbf{y},\mathbf{z}) - \beta\mathrm{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})\,\|\,p_\psi(\mathbf{y},\mathbf{z})\right)\right]:$$

  - $q_\phi(\mathbf{z}|\mathbf{x})$ — распределение над 32×32 токенами, порождённое энкодером dVAE по картинке $\mathbf{x}$;
  - $p_\theta(\mathbf{x}|\mathbf{y},\mathbf{z})$ — распределение на картинках, порождённое декодером dVAE по токенам $\mathbf{z}$; здесь мы предположим $p_\theta(\mathbf{x}|\mathbf{y},\mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})$;
  - $p_\psi(\mathbf{y},\mathbf{z})$ — совместное распределение на текстах и латентных кодах, которое моделирует трансформер.

dVAE + Transformer = DALL-E

- То есть фактически это одна большая вариационная оценка:

$$\ln p_{\theta,\psi}(\mathbf{x},\mathbf{y}) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x}|\mathbf{y},\mathbf{z}) - \beta\mathrm{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})\,\|p_\psi(\mathbf{y},\mathbf{z})\right)\right].$$

- И обучение идёт в два приёма:
  - сначала максимизируем оценку по ϕ и θ, т.е. обучаем dVAE просто на картинках, без текстов; здесь считаем p_ψ равномерным, а q_ϕ релаксируем через Gumbel-Softmax;
  - потом фиксируем ϕ и θ и обучаем ψ, т.е. обучаем трансформер моделировать текст (BPE encoding) и картинки (их коды $\mathbf{z}$) совместно.
- И дальше очень, очень много трюков с распределённым обучением, потому что модели очень большие.

dVAE + Transformer = DALL-E

- Получается очень круто (для 2021), и с выдумкой:



dVAE + Transformer = DALL-E

- Получается очень круто (для 2021), и с выдумкой:



dVAE + Transformer = DALL-E

- В том числе сложные запросы (`https://openai.com/blog/dall-e/`):



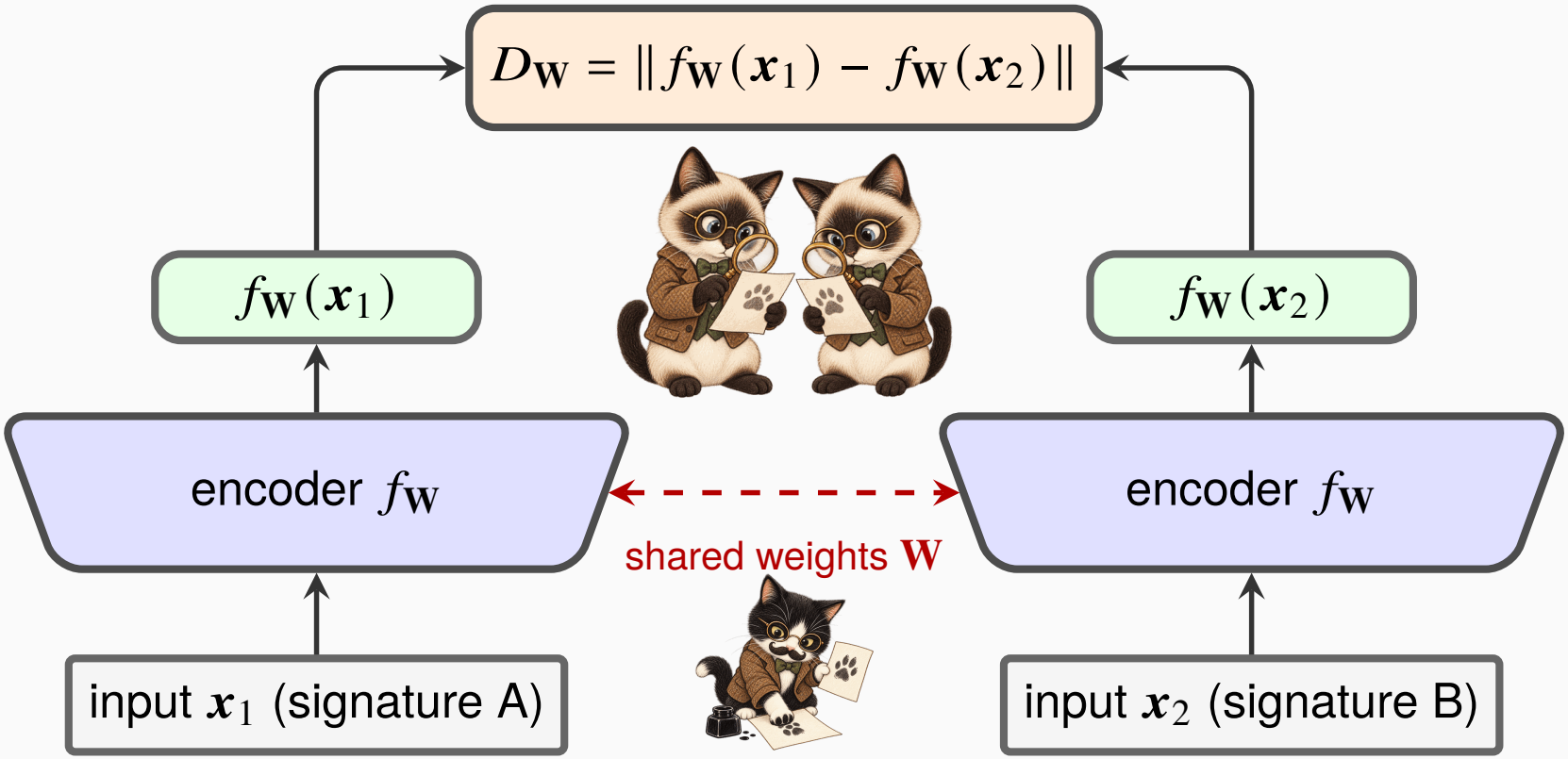
Контрастивное обучение

От порождения данных к представлениям

- VAE учит порождать данные; теперь будем учить кодировщик, представления которого пригодятся в следующих задачах.
- В self-supervised learning обучающий сигнал получаем из самих данных: берём два представления одного объекта и учим согласовывать их в латентном пространстве.
- Нужно решить три вопроса: откуда берутся пары, как сравнивать их представления и как не дать всем объектам схлопнуться в одну точку.

Сиамские сети

- Bromley et al. (1993): сравниваем две подписи с помощью двух одинаковых сетей с общими весами.
- Вместо классификации по заранее известным именам учим расстояние между представлениями:



Контрастивная функция потерь

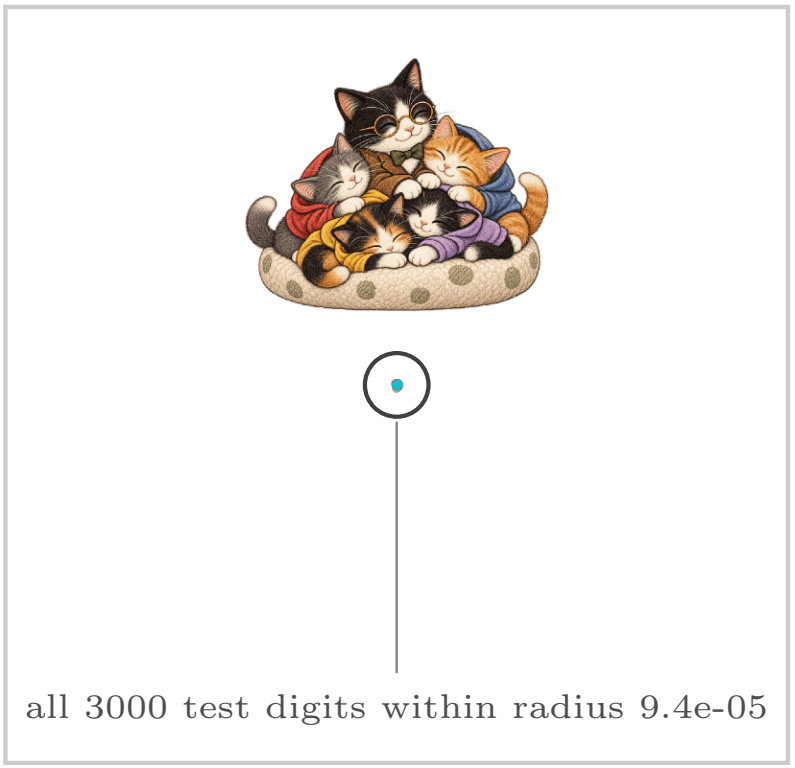
- DrLIM (Hadsell et al., 2006): пусть $D_{\mathbf{W}}$ – расстояние между представлениями, $Y = 0$ для похожих пар и $Y = 1$ для непохожих.

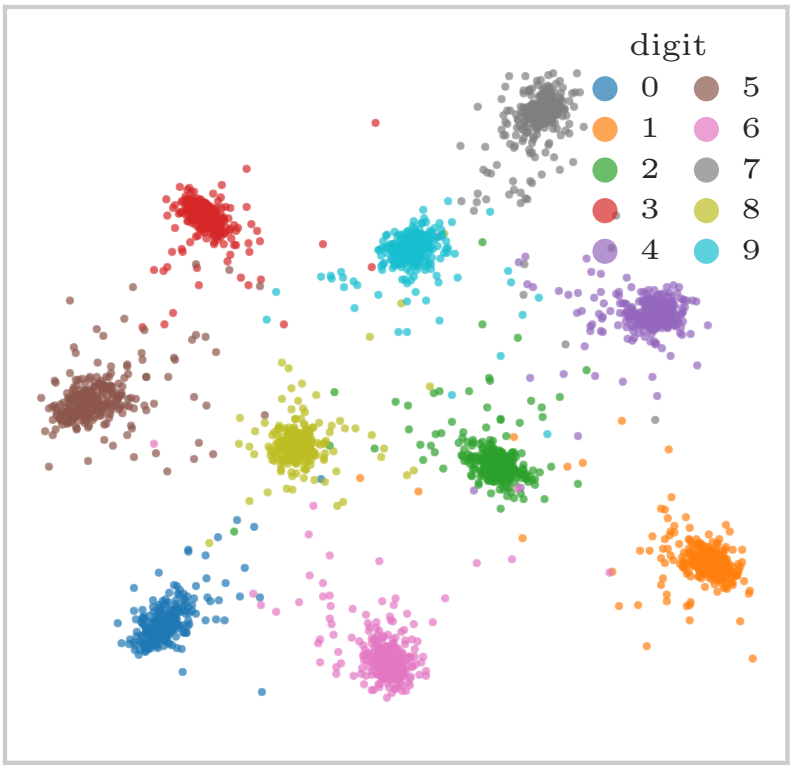
$$\mathcal{L}(\mathbf{W}) = (1 - Y)\frac{1}{2}D_{\mathbf{W}}^2 + Y\frac{1}{2}\big(\max(0, m - D_{\mathbf{W}})\big)^2.$$

- Похожие объекты притягиваем друг к другу; непохожие отталкиваем, пока расстояние не станет хотя бы m.
- Если оставить только притяжение, получится идеальное бесполезное решение: кодировщик переводит все объекты в одну точку. Это *коллапс представлений*.

Зачем нужно отталкивание

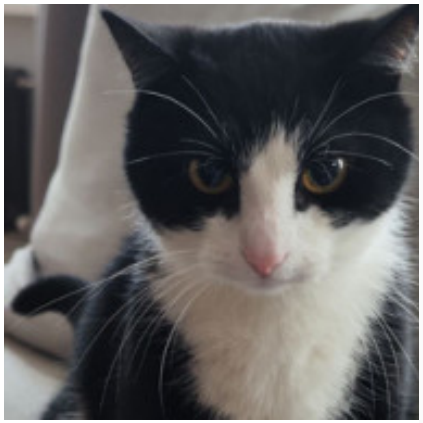
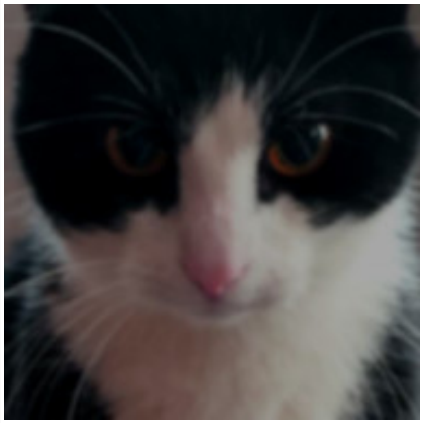
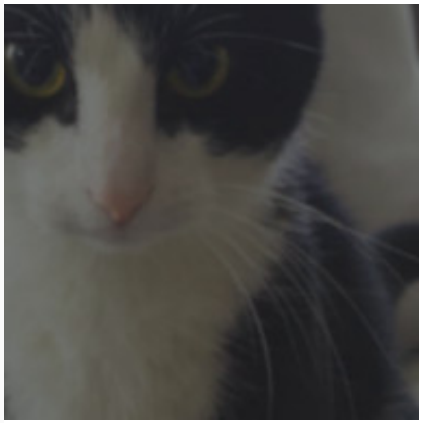
- Сиамская сеть на MNIST: слева только притяжение, справа контрастивная ошибка с отступом $m = 1$.
- Обучались на метках пар «похожи / не похожи», а не на номерах классов:





Позитивные пары без ручной разметки

- Два случайных преобразования одного изображения дают позитивную пару; представления других изображений используем как негативные примеры.
- Обрезка, отражение, изменение цвета, размытие должны сохранять важное для задачи содержание.

InfoNCE: найти свою пару

- Есть представление $\mathbf{z}$, его позитивный партнёр $\mathbf{z}^+$ и набор кандидатов $\{\mathbf{z}_j\}$, включающий и позитивный пример, и негативные:

$$\mathcal{L}_{\text{InfoNCE}} = -\log \frac{\exp\big(\text{sim}(\mathbf{z}, \mathbf{z}^+)/\tau\big)}{\sum_j \exp\big(\text{sim}(\mathbf{z}, \mathbf{z}_j)/\tau\big)}.$$

- Это обычная softmax-классификация: нужно выбрать правильного партнёра среди кандидатов; sim обычно означает косинусное сходство, τ – температуру.
- Без негативов числитель совпадёт со знаменателем и ошибка всегда будет нулевой. Негативные примеры не дают представлениям просто схлопнуться.

SimCLR и источник негативных примеров

- SimCLR (Chen et al., 2020): сиамская сеть, сильные аугментации, InfoNCE и большие батчи; другие изображения в батче дают негативные примеры.
- После кодировщика во время обучения ставят небольшой MLP, *projection head*; для следующих задач его убирают и используют представления кодировщика.
- MoCo (He et al., 2020) вместо огромного батча хранит очередь негативных представлений; кодировщик для них обновляется скользящим средним весов основной сети.
- Позитивная пара может состоять и из разных модальностей: изображение и его подпись. Так мы приходим к CLIP.

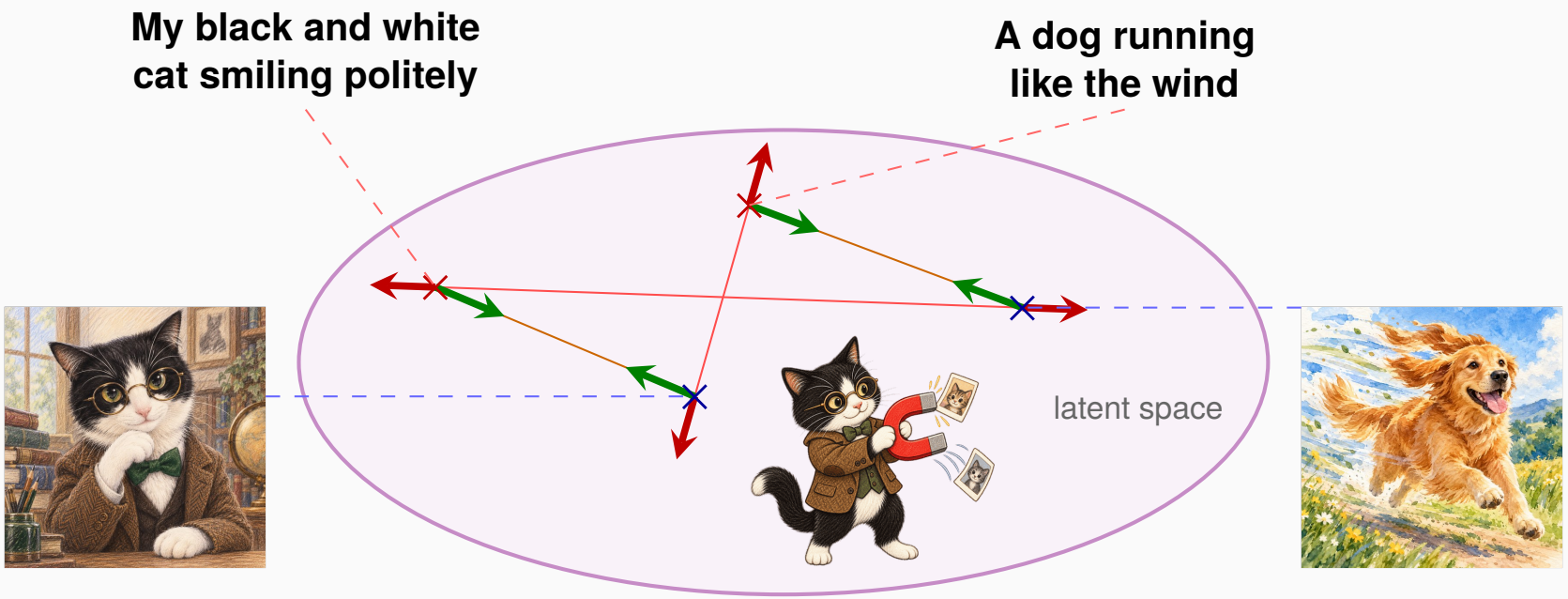
CLIP и BLIP

Мультимодальные латентные пространства

- У нас есть много картинок и текстов, свободно доступных в интернете
- Как использовать текст в паре с изображениями так, чтобы получить хорошее мультимодальное латентное пространство?
- Авторы CLIP пишут, что их первой идеей было обучить свёрточную сеть для изображений и трансформер для текста, чтобы предсказывать подпись к изображению
- Но это не сработало: предсказывать точную подпись очень сложно (практически безнадёжно), и это не совсем то, что нам нужно в этой модели, нам просто нужны хорошие мультимодальные представления

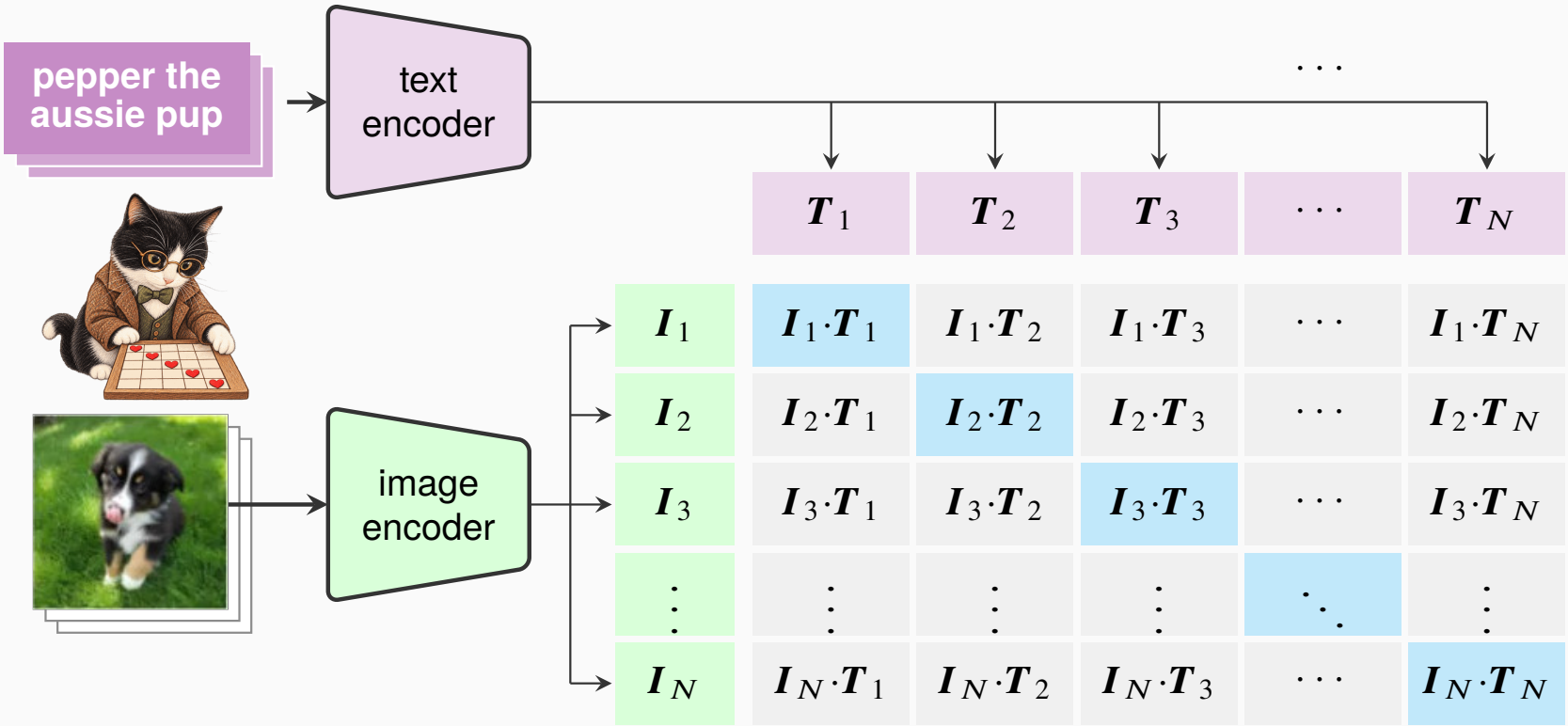
CLIP

- Контрастивное предобучение (contrastive pretraining): моделируем силы притяжения и отталкивания в совместном латентном пространстве



CLIP

- CLIP (Radford et al., 2021; OpenAI), Contrastive Language–Image Pretraining; использовали пары изображение-текст, которые были просто собраны из интернета:



CLIP: симметричная InfoNCE

- Для нормированных представлений изображений $\mathbf{u}_i$ и текстов $\mathbf{v}_j$ положим $s_{ij} = \mathbf{u}_i^\top \mathbf{v}_j / \tau$.

$$\mathcal{L}_{I \to T} = -\frac{1}{N} \sum_i \log \frac{e^{s_{ii}}}{\sum_j e^{s_{ij}}},$$

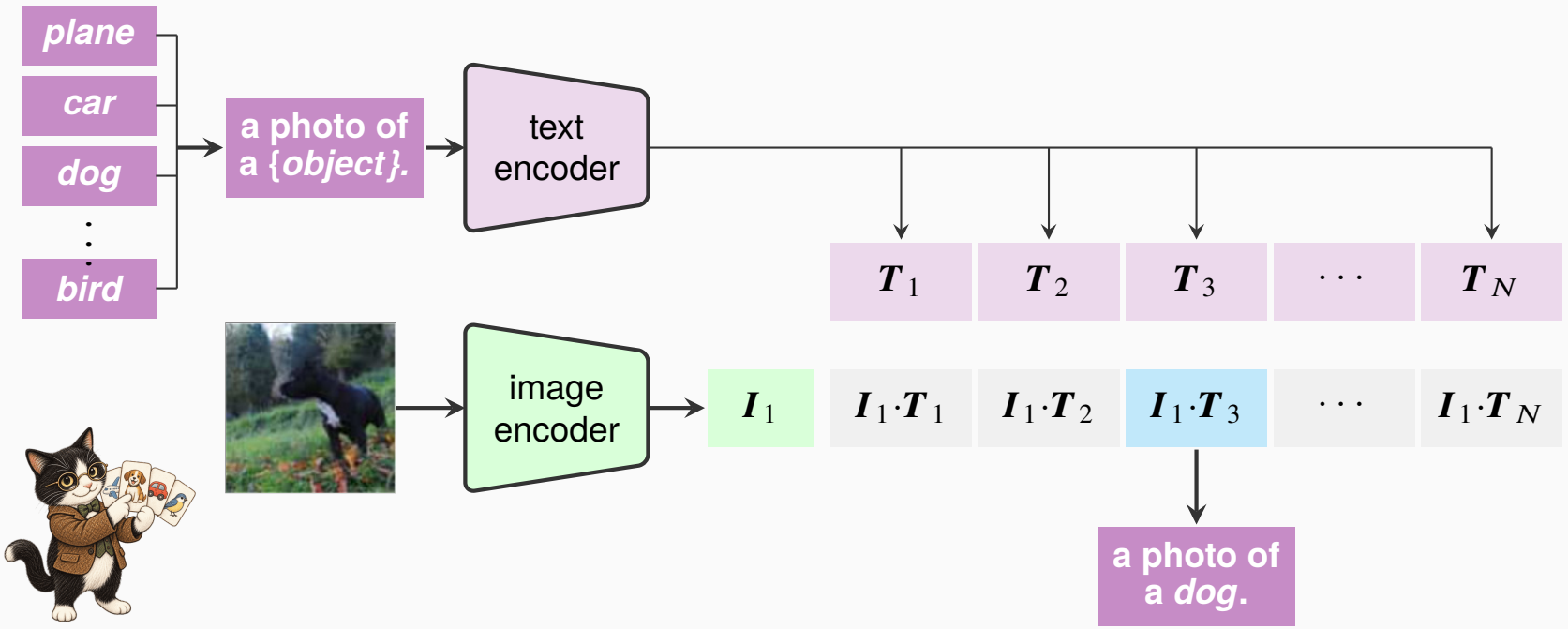
$$\mathcal{L}_{T \to I} = -\frac{1}{N} \sum_i \log \frac{e^{s_{ii}}}{\sum_j e^{s_{ji}}},$$

$$\mathcal{L}_{\text{CLIP}} = \frac{1}{2}(\mathcal{L}_{I \to T} + \mathcal{L}_{T \to I}).$$

- Верные пары лежат на диагонали матрицы; остальные тексты и изображения батча служат негативными примерами.
- В исходной работе: 400 миллионов пар, батчи из 32768 пар и обучаемая температура.

CLIP

- В итоге работала даже zero-shot классификация – преобразуем метки классов в простые запросы:

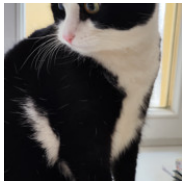


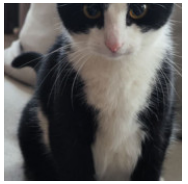
CLIP: zero-shot на фотографиях кота













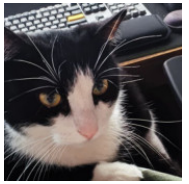
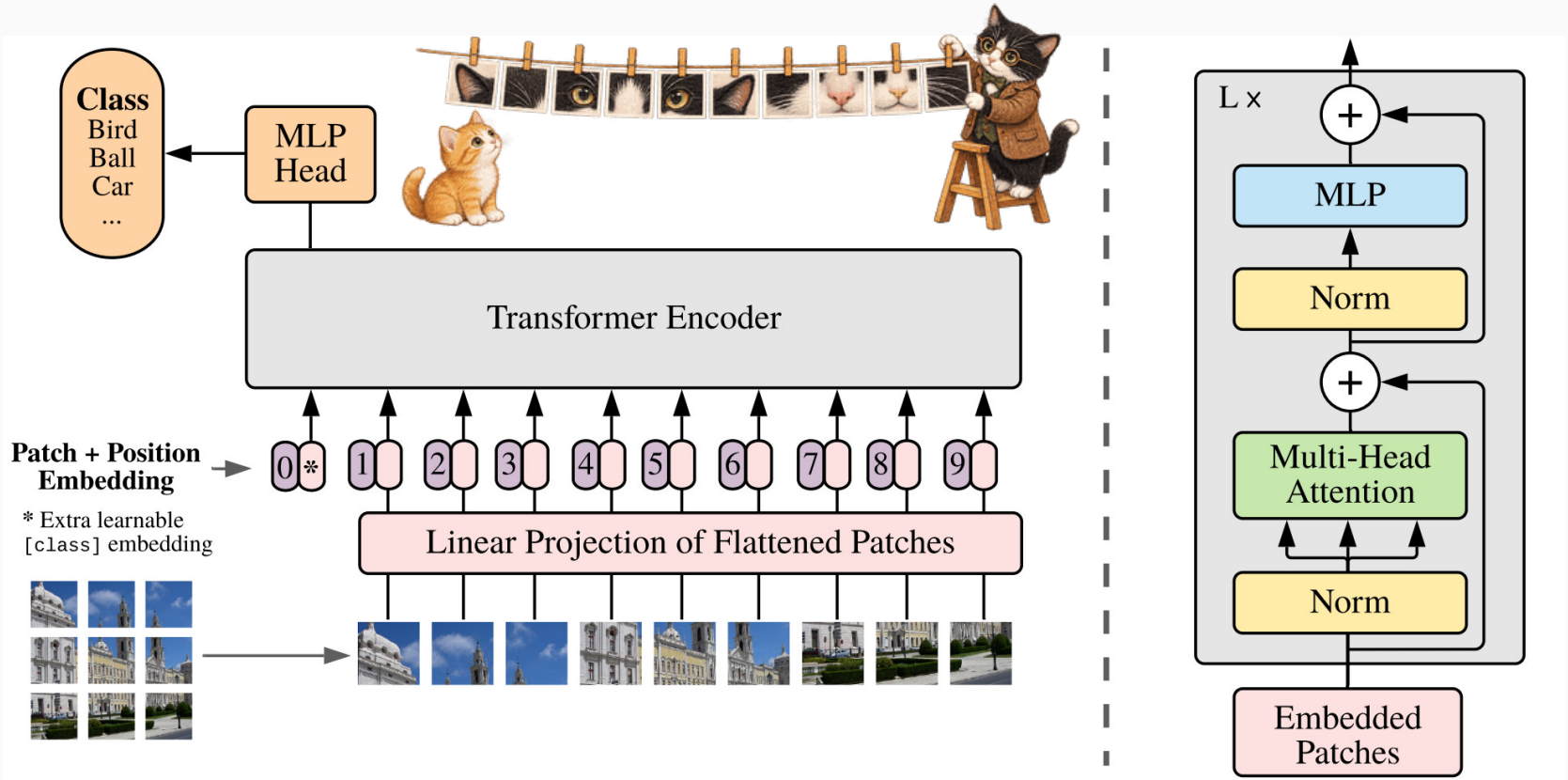


	image 1	image 2	image 3	image 4	image 5	image 6
“a photo of a cat”	0.96	0.89	0.97	0.98	0.95	0.95
“a photo of a dog”	0.01	0.00	0.01	0.01	0.01	0.00
“a photo of a tiger”	0.01	0.00	0.01	0.00	0.01	0.02
“a photo of a rabbit”	0.00	0.00	0.01	0.01	0.00	0.00
“a photo of a LEGO set”	0.00	0.04	0.00	0.00	0.00	0.00
“a photo of a cardboard box”	0.01	0.06	0.00	0.00	0.00	0.00
“a photo of a computer keyboard”	0.00	0.00	0.00	0.00	0.00	0.02
“a photo of a wooden floor”	0.01	0.00	0.00	0.00	0.03	0.00

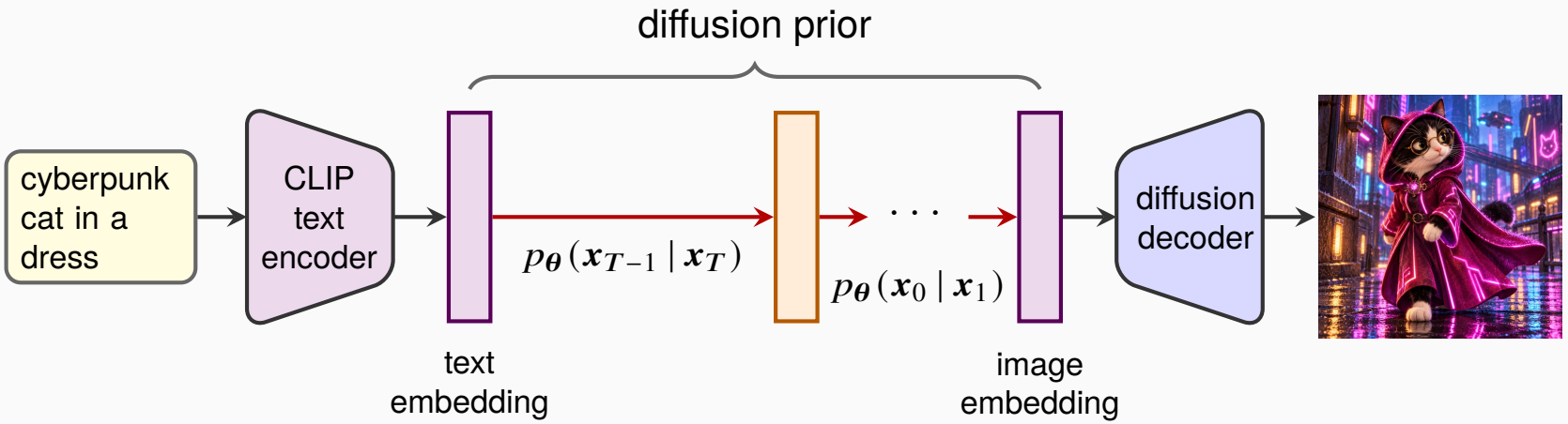
CLIP

- Кодировщик изображений был ViT (Vision Transformer, Dosovitsky et al., 2020)



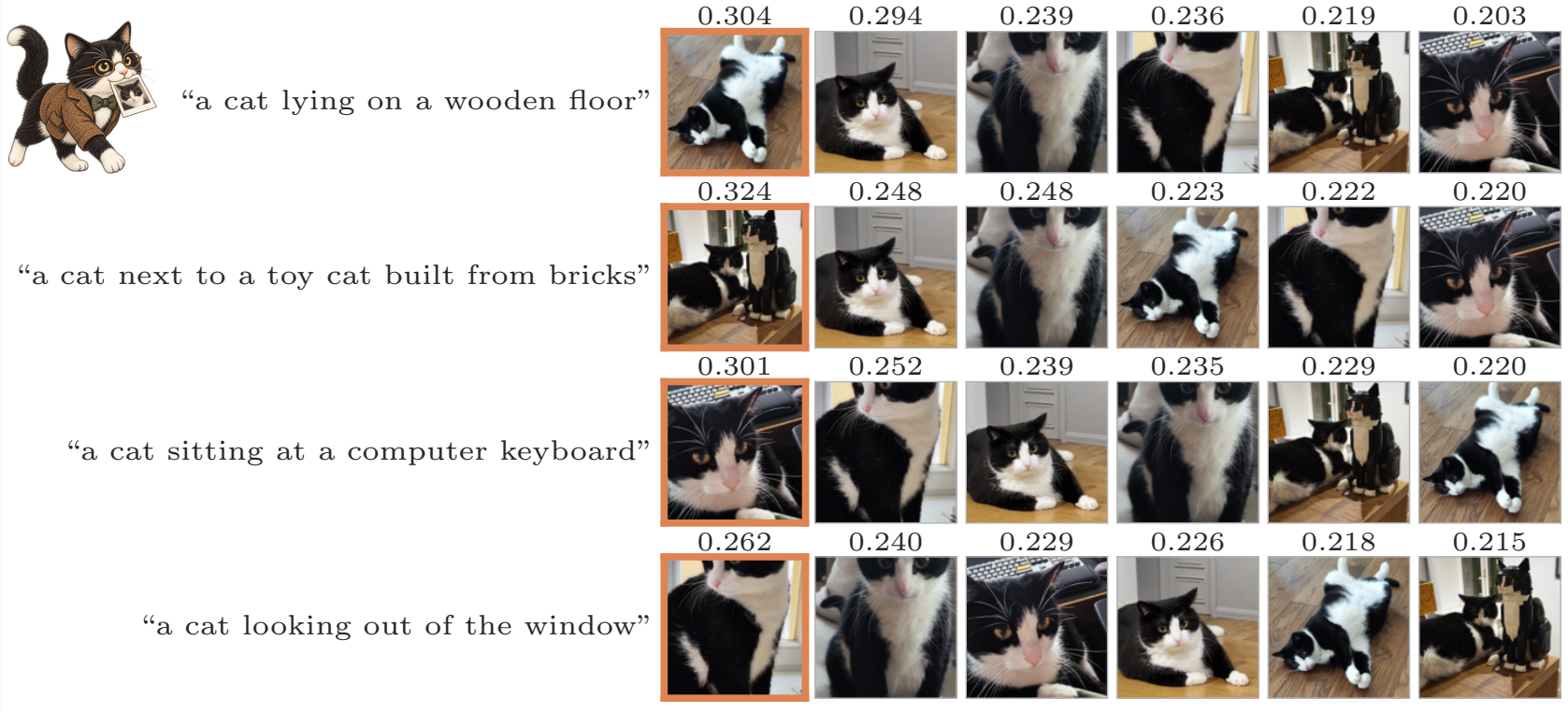
CLIP

- CLIP стал популярным источником совместных представлений; например, DALL-E 2 рисовал картинки в латентном пространстве CLIP:



CLIP

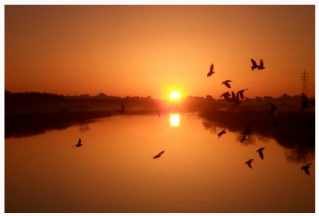
- Основное применение CLIP — это обеспечение информационного поиска и порождающих моделей:



BLIP

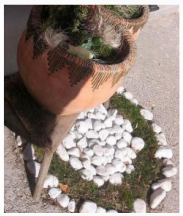
- BLIP (Li et al., 2022) – Bootstrapping Language-Image Pretraining; порождаем синтетические подписи и фильтруем их





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"

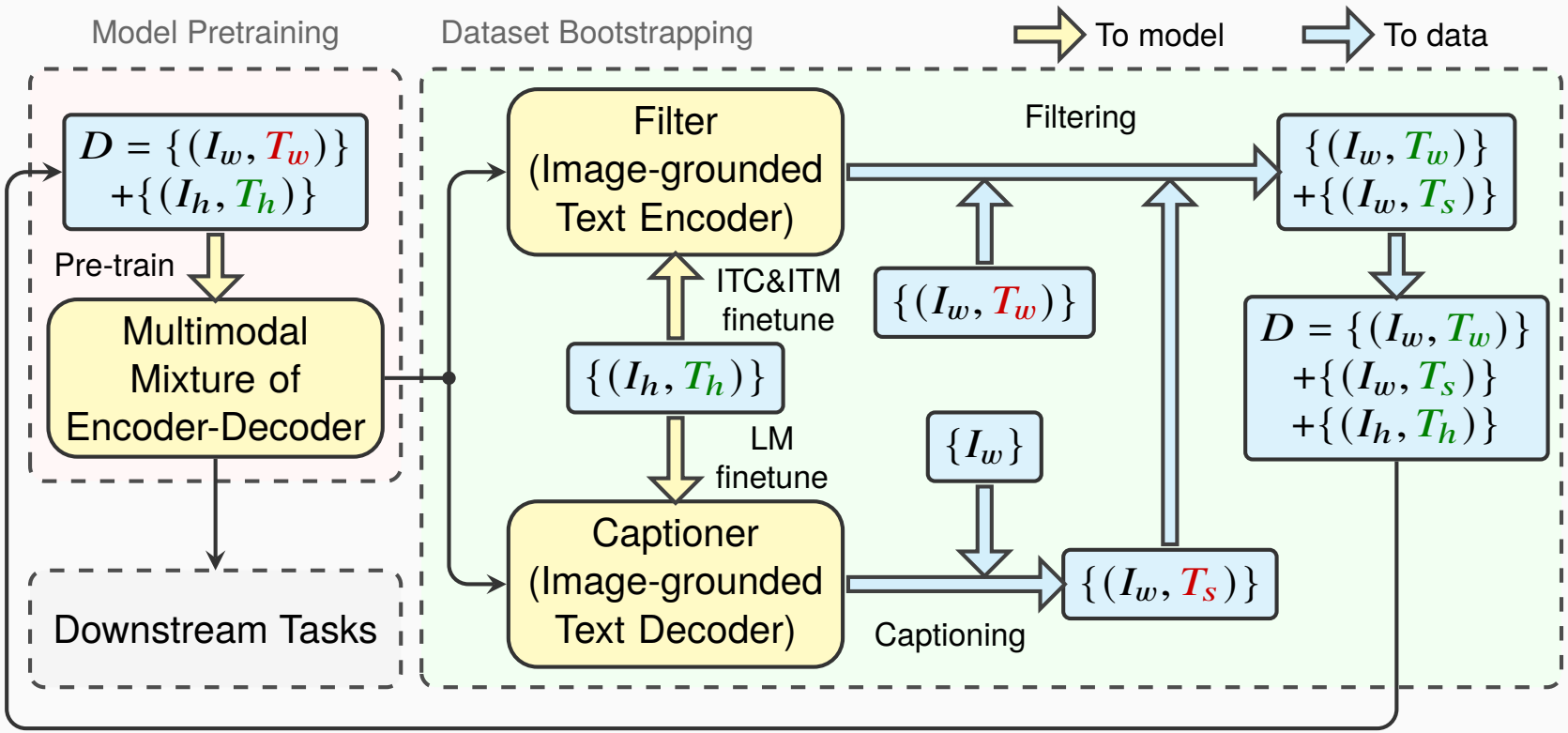


T_w: "the current castle was built in 1180, replacing a 9th century wooden castle"

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

BLIP

- Структура обучения:



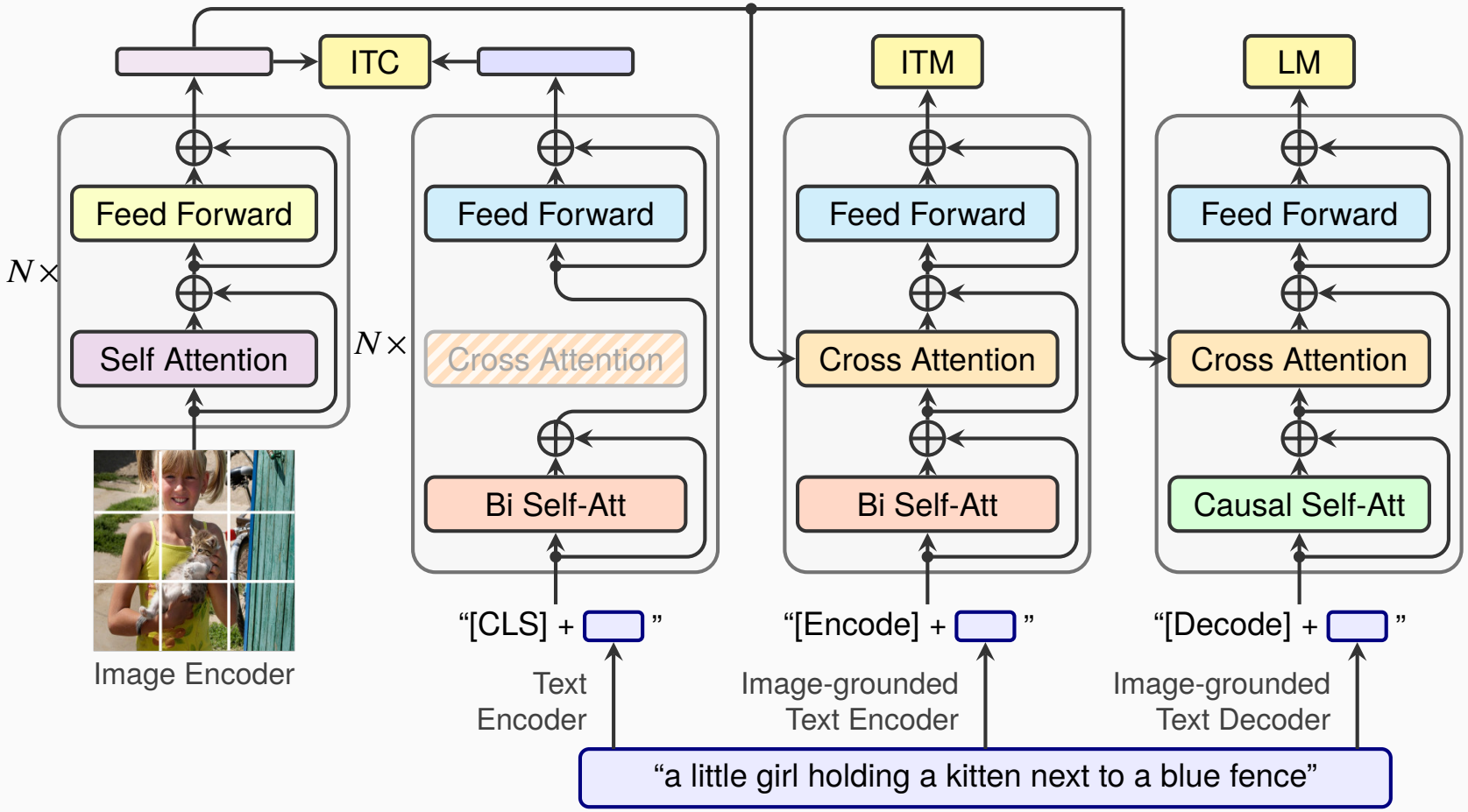
I_w: web images I_h: human-annotated images T_h: human-annotated texts

T_w: web texts T_w: filtered web texts T_s: synthetic texts T_s: filtered synthetic texts

BLIP

- Три разных кодировщика и декодер: ITC – контрастивная функция потерь для изображения и текста (сопоставление представлений изображения и текста), ITM – функция потерь на соответствие изображения и текста (различение положительных и отрицательных пар), LM – языковое моделирование (авторегрессивное порождение подписей)
- Один ViT для изображений и три текстовых режима; текстовые модули разделяют большую часть весов.

BLIP: архитектура и три задачи обучения



Спасибо за внимание!



