

**ИНТЕЛЛЕКТ** ➤

# BERT, Sentence BERT, векторные БД, RAG

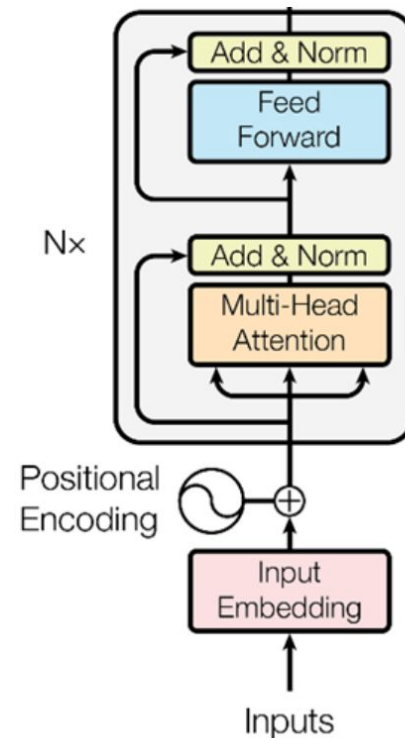
к.ф.-м.н. Тихомиров Михаил Михайлович

научный сотрудник  
НИВЦ МГУ имени М. В. Ломоносова

# BERT (2018)



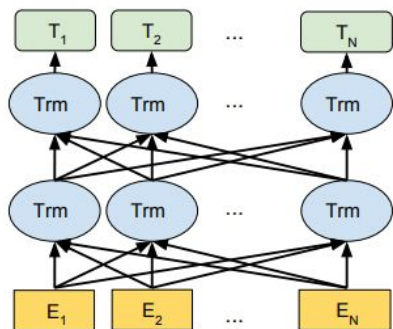
- Base: 12 слоев **Transformer encoder** (~110 млн. параметров),
- Обучение в **2 этапа!**:
  - **Предобучение (pre-training)** на задачах:
    - маскированного языкового моделирования (**MLM**)
    - предсказания следующего предложения (**NSP**).
  - **Дообучение (fine-tuning)** на целевые задачи
- **3.3 миллиарда** слов English Wikipedia + BooksCorpus.



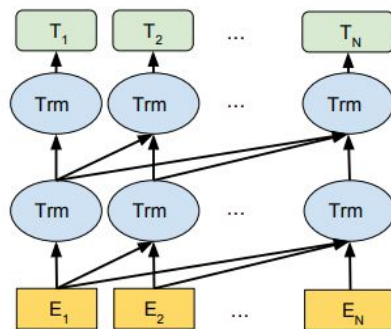
# BERT: визуальное сравнение архитектур



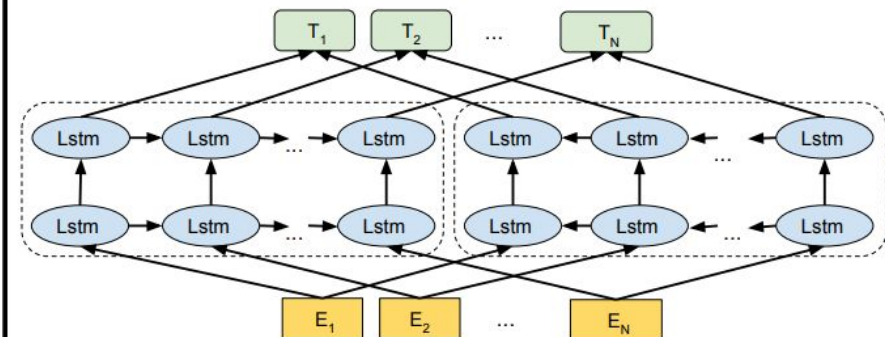
## BERT (Ours)



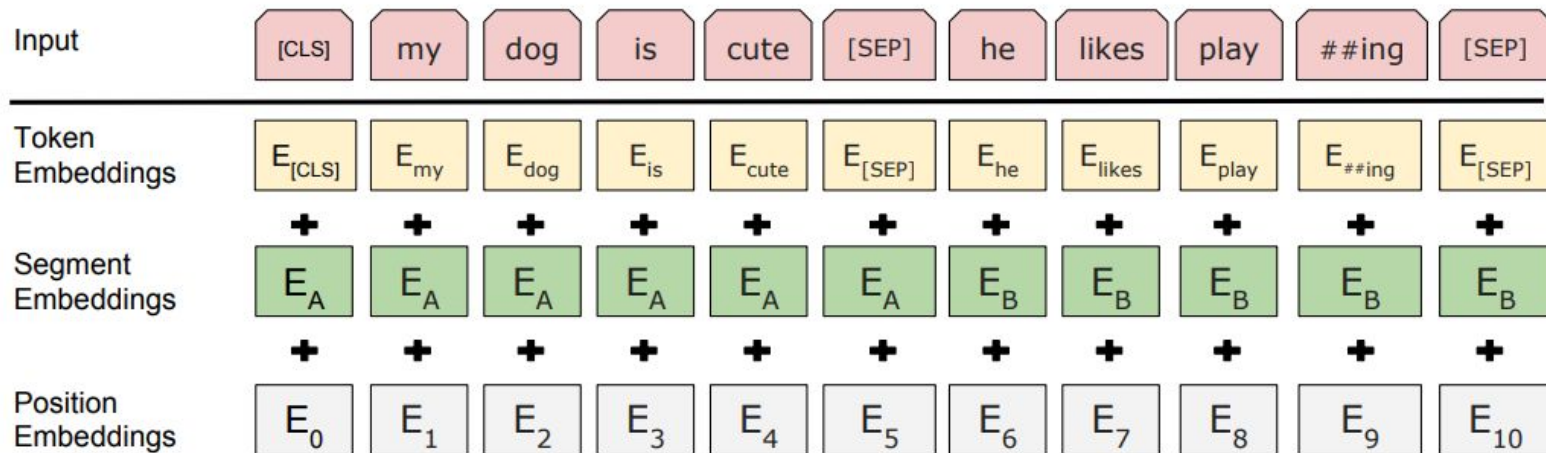
## OpenAI GPT



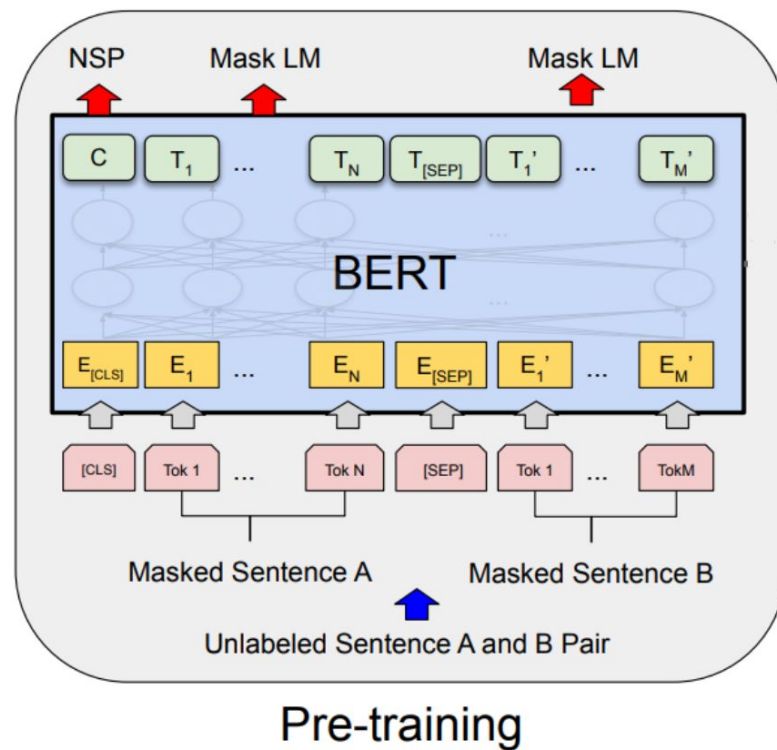
## ELMo



# BERT: формирование входа



# BERT: Предобучение



Pre-training

# BERT: Masked Language Modeling



BERT (англ. Bidirectional Encoder Representations from Transformers) — языковая модель, основанная на архитектуре **[MASK]**, предназначенная для ...

- Маскировалось 15% случайных токенов в тексте.
- Модель во время претрейна обучалась предсказывать исходный токен, который был спрятан.

# BERT: Masked Language Modeling [2]



## Инициализация:

```
import torch
from transformers import AutoTokenizer, AutoModelForMaskedLM

tokenizer = AutoTokenizer.from_pretrained('FacebookAI/xlm-roberta-base', use_fast=False)
model = AutoModelForMaskedLM.from_pretrained('FacebookAI/xlm-roberta-base')
model.eval()
```

## Функция предсказания:

```
def predict_masked_sent(text, top_k=5):
    # Tokenize input
    text = "<s>%s</s>"%text
    tokenized_text = tokenizer.tokenize(text)
    print(tokenized_text)
    masked_index = tokenized_text.index("<mask>")
    indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)
    tokens_tensor = torch.tensor([indexed_tokens])
    tokens_tensor = tokens_tensor.to(model.device)

    # Predict all tokens
    with torch.no_grad():
        outputs = model(tokens_tensor)
        predictions = outputs[0].detach().cpu()

    probs = torch.nn.functional.softmax(predictions[0, masked_index], dim=-1)
    top_k_weights, top_k_indices = torch.topk(probs, top_k, sorted=True)

    for i, pred_idx in enumerate(top_k_indices):
        predicted_token = tokenizer.decode([pred_idx])
        token_weight = top_k_weights[i]
        print("<mask>: '%s'%predicted_token, " | weights:", float(token_weight))
```

# BERT: Masked Language Modeling [3]



```
predict_masked_sent("Я пришел в кофейню и выпил вкусный <mask>. Он оказался горячим.", top_k=5)
```

```
['<s>', '_Я', '_пришел', '_в', '_кофе', 'й', 'ню', '_и', '_вы', 'пил', '_вкус', 'ный', '<mask>', '_', '.', '_Он', '_оказался', '_горя', 'чим', '.', '</s>']  
<mask>: 'кофе' | weights: 0.8137695789337158  
<mask>: 'чай' | weights: 0.15855158865451813  
<mask>: 'суп' | weights: 0.0038443673402071  
<mask>: 'салат' | weights: 0.00360748334787786  
<mask>: 'сок' | weights: 0.0035875297617167234
```

```
[5] predict_masked_sent("После выезда на МКАД в меня въехал красный <mask>. Мы вызвали ГИБДД и мирно разрешили ситуацию.", top_k=5)
```

```
['<s>', '_После', '_выезд', 'а', '_на', '_', 'МК', 'АД', '_в', '_меня', '_въ', 'е', 'хал', '_крас', 'ный', '<mask>', '_', '.', '_Мы', '_вызвал', 'и', '_', 'ГИБ', 'ДД', 'Д', '_и', '_мирно']  
<mask>: 'автомобиль' | weights: 0.20946624875068665  
<mask>: 'автомобус' | weights: 0.20619282126426697  
<mask>: 'мотоцикл' | weights: 0.0780663713812828  
<mask>: 'свет' | weights: 0.07340569794178009  
<mask>: 'автомат' | weights: 0.04334369674324989
```

# BERT: Next Sentence Prediction



- Задача классификации, следует ли следующее предложение за текущим.
- Разметка создается автоматически на основе корпуса текстов:
  - В 50% случаев берем пару предложений, идущих вместе.
  - В 50% случаев берем случайное предложение из корпуса.
- Обучается слой NSP над представлением специального токена [CLS], идущего в начале последовательности.
- В оригинальной работе заявлялось, что помогает в задачах QA и NLI, но в работе про RoBERTa было показано, что NSP не обязателен.



# BERT: Next Sentence Prediction [3]



Функция предсказания:

```
def predict_nsp(text1, text2):
    model_input = tokenizer(text1, text2, max_length=512, return_tensors="pt", truncation=True)
    model_input = {k: v.to(model.device) for k, v in model_input.items()}
    with torch.no_grad():
        logits = model(**model_input).logits.detach().cpu()
        probs = torch.nn.functional.softmax(logits, dim=-1)[0]

    return probs.argmax(), probs
```

Примеры:

```
[28] predict_nsp('Вчера я вышел на улицу ранним утром.', 'Самолеты летают не так, как птицы.')
```

```
(tensor(0), tensor([0.5476, 0.4524]))
```

```
[29] predict_nsp('Вчера я вышел на улицу ранним утром.', 'Там по лужайке бегала собака, ей тоже нравилось это утро, как и мне.')
```

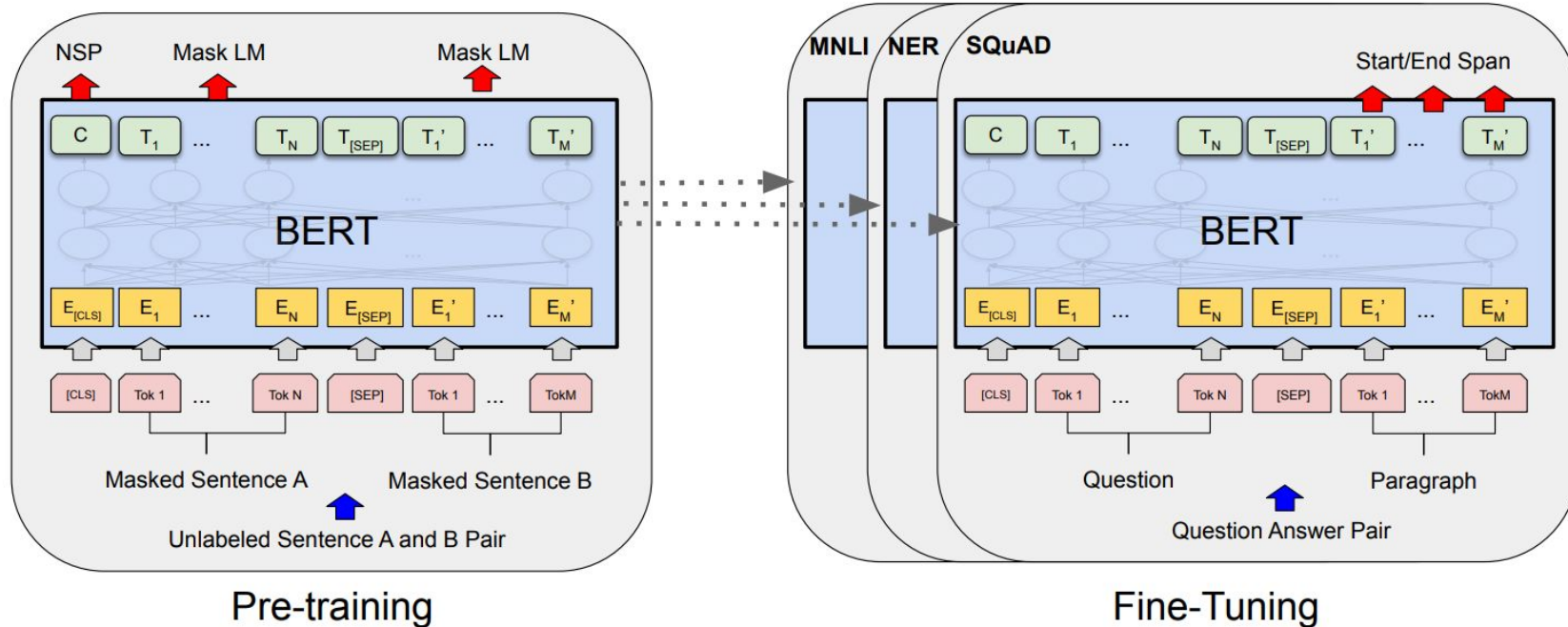
```
(tensor(1), tensor([0.3761, 0.6239]))
```

# Контекстуализированные эмбединги



- Модель учит “язык”, решая задачу MLM.
- BERT может рассматриваться как модель, порождающая **контекстуализированные эмбединги** для токенов предложения!
- Раньше: статические эмбединги на вход, нейросеть поверх.  
Теперь: BERT на вход, нейросеть поверх.

# BERT: pre-training и fine-tuning



# BERT: результаты



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
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>92.7</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>82.1</b>

# Sentence BERT



# Векторизация текста

- tf-idf векторизация очень разреженная,
  - Хотя существуют эффективные реализации пакетов для работы с подобными матрицами,
- Существует проблема синонимов, перефразирования и тп,
- Возможные решения:
  - Словари синонимов (тезаурусы, графы знаний),
  - Векторизация слов:
    - word2vec,
    - fasttext и др,
  - Тематическое моделирование:
    - PLSA,
    - LDA,
    - BigARTM,
  - Энкодеры текста на основе архитектуры трансформер, BERT.



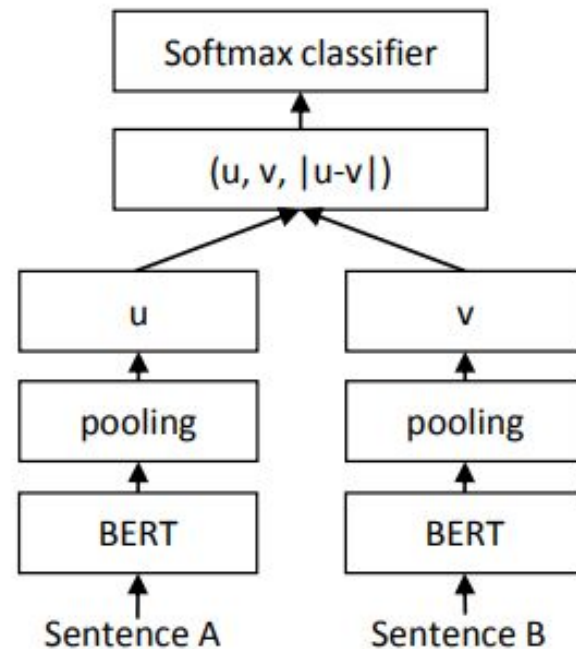
# Векторизация текста + word2vec

- **Word2Vec** позволяет получить сжатое **векторное представление** слова,
- Текст - **мешок слов** (Bag Of Words),
- Решение 1: **Усреднять** векторные представления слов текста!
  - Может работать на коротких текстах,
- Решение 2: **Взвешенное** усреднение на основе **tf-idf**:
  - Часто может выступать как хороший быстрый **бейзлайн** для задач **классификации**, где поверх **tf-idf + word2vec** идет **логистическая регрессия \ SVM** или иные алгоритмы,
- В целом не решает проблему векторизации текстов.

# Sentence BERT (sbert)



- В качестве инициализации уже **предобученный BERT**,
- **bi-encoder** подход,
- Операция **пулинга**:
  - На входе последовательность векторов,
  - На выходе вектор предложения.
- Обучение на **задаче логического следования** (NLI, natural language inference).



Mean pooling в данном случае показал себя лучше других вариантов (max pooling, cls token)

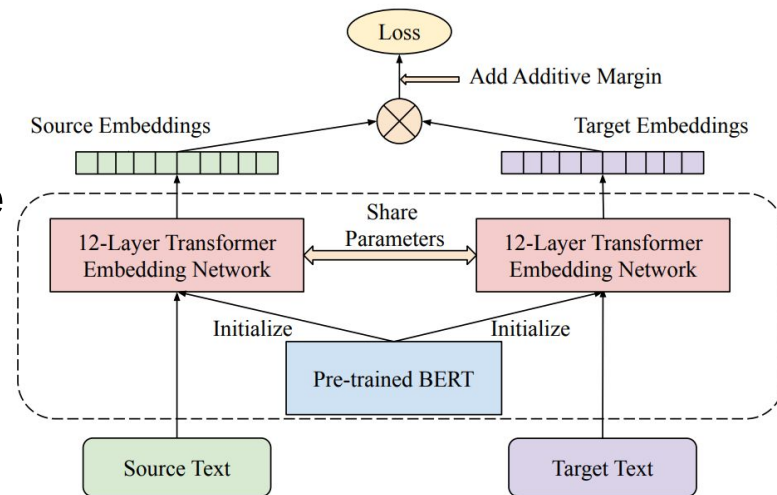
# Sentence BERT: pooling стратегия



- Исследовали, как лучше формировать вход для классификатора.
- Качество пулинга +/- одинаковое у CLS и MEAN.
- В качестве входа для слоя классификатора удачно сработала конкатенация  $v$ ,  $u$  и  $|u - v|$ .

	NLI	STSb
<i>Pooling Strategy</i>		
MEAN	<b>80.78</b>	<b>87.44</b>
MAX	79.07	69.92
CLS	79.80	86.62
<i>Concatenation</i>		
$(u, v)$	66.04	-
$( u - v )$	69.78	-
$(u * v)$	70.54	-
$( u - v , u * v)$	78.37	-
$(u, v, u * v)$	77.44	-
$(u, v,  u - v )$	<b>80.78</b>	-
$(u, v,  u - v , u * v)$	80.44	-

- Также **bi-encoder** подход,
- В качестве пулинга **CLS** токен,
- Основные особенности:
  - Дообучение на корпусе **параллельных текстов на более чем 100 языках**,
  - Additive Margin Loss,
  - In-batch negative sampling,
- **6 миллиардов пар предложений**, полученных **автоматически** для 109 языков,
- Модель **многоязычная!**





- Основные идеи по сравнению с другими работами:
  - Обучение на крупном датасете **CCPairs**
    - Состоит из **пар (запрос, текст)** построенных автоматически из Reddit, Common Crawl и других ресурсов (с правилами фильтрации),
    - **Хитрая фильтрация**: сначала датасет состоял из **1.3 млрд. пар**, на которых обучили модель -> с помощью модели откинули пары “плохого” качества,
    - Итого **270 млн. пар**,
  - Mean pooling,
  - **InfoNCE loss** + in batch negative sampling,
  - Дополнительно обучали на MS-MARCO (ранжирование), NQ (вопросно-ответный) и NLI (логическое следование),
- Есть мультязычная версия!

$$L_{\text{cont}} = -\frac{1}{n} \sum_i \log \frac{e^{s_{\theta}(q_i, p_i)}}{e^{s_{\theta}(q_i, p_i)} + \sum_j e^{s_{\theta}(q_i, p_{i_j}^-)}}$$

$$s_{\theta}(p, q) = \cos(\mathbf{E}_q, \mathbf{E}_p) / \tau$$



# E5: использование в huggingface

- Mean Pooling
- Префиксы “query: “ и “passage: “ перед векторизацией.
  - Для “симметричного” поиска и запрос и базу с “query: “.
  - Для более классического поиска, “query: “ для запроса и “passage: “ для базы.
- max\_length = 512!
- Близость по косинусу.

```
import torch.nn.functional as F

from torch import Tensor
from transformers import AutoTokenizer, AutoModel

def average_pool(last_hidden_states: Tensor,
                  attention_mask: Tensor) -> Tensor:
    last_hidden = last_hidden_states.masked_fill(~attention_mask[..., None].bool(), 0.0)
    return last_hidden.sum(dim=1) / attention_mask.sum(dim=1)[..., None]

# Each input text should start with "query: " or "passage: ", even for non-English
# For tasks other than retrieval, you can simply use the "query: " prefix.
input_texts = ['query: how much protein should a female eat',
               'query: 南瓜的家常做法',
               "passage: As a general guideline, the CDC's average requirement of",
               "passage: 1.清炒南瓜丝 原料:嫩南瓜半个 调料:葱、盐、白糖、鸡精 做法: 1、南瓜"]

tokenizer = AutoTokenizer.from_pretrained('intfloat/multilingual-e5-large')
model = AutoModel.from_pretrained('intfloat/multilingual-e5-large')

# Tokenize the input texts
batch_dict = tokenizer(input_texts, max_length=512, padding=True, truncation=True)

outputs = model(**batch_dict)
embeddings = average_pool(outputs.last_hidden_state, batch_dict['attention_mask'])

# normalize embeddings
embeddings = F.normalize(embeddings, p=2, dim=1)
scores = (embeddings[:2] @ embeddings[2:].T) * 100
print(scores.tolist())
```

# Бенчмарк MTEB



Overall Bitext Mining Classification Clustering Pair Classification Reranking Retrieval STS Summarization

English Chinese French Polish

Overall MTEB English leaderboard

- Metric: Various, refer to task tabs
- Languages: English

Rank	Model	Model Size (GB)	Embedding Dimensions	Max Tokens	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)	Pair Classification Average (3 datasets)	Reranking Average (4 datasets)	Retrieval Average (15 datasets)	STS Average (10 datasets)
1	<a href="#">SFR-Embedding-Mistral</a>	14.22	4096	32768	67.56	78.33	51.67	88.54	60.64	59	85.05
2	<a href="#">voyage-lite-02-instruct</a>		1024	4000	67.13	79.25	52.42	86.87	58.24	56.6	85.79
3	<a href="#">GritLM-7B</a>	14.48	4096	32768	66.76	79.46	50.61	87.16	60.49	57.41	83.35
4	<a href="#">e5-mistral-7b-instruct</a>	14.22	4096	32768	66.63	78.47	50.26	88.34	60.21	56.89	84.63
5	<a href="#">GritLM-8x7B</a>	93.41	4096	32768	65.66	78.53	50.14	84.97	59.8	55.09	83.26
6	<a href="#">echo-mistral-7b-instruct-latest</a>	14.22	4096	32768	64.68	77.43	46.32	87.34	58.14	55.52	82.56
7	<a href="#">mxbai-embed-large-v1</a>	0.67	1024	512	64.68	75.64	46.71	87.2	60.11	54.39	85
8	<a href="#">UAE-Large-V1</a>	1.34	1024	512	64.64	75.58	46.73	87.25	59.88	54.66	84.54
9	<a href="#">text-embedding-3-large</a>		3072	8191	64.59	75.45	49.01	85.72	59.16	55.44	81.73
10	<a href="#">voyage-lite-01-instruct</a>		1024	4000	64.49	74.79	47.4	86.57	59.74	55.58	82.93
11	<a href="#">Cohere-embed-english-v3.0</a>		1024	512	64.47	76.49	47.43	85.84	58.01	55	82.62
12	<a href="#">multilingual-e5-large-instruct</a>	1.12	1024	514	64.41	77.56	47.1	86.19	58.58	52.47	84.78

Refresh

# E5: оценка на MTEB



# of datasets →	Class. 12	Clust. 11	PairClass. 3	Rerank 4	Retr. 15	STS 10	Summ. 1	Avg 56
<i>Unsupervised models</i>								
Glove	57.3	27.7	70.9	43.3	21.6	61.9	28.9	42.0
BERT	61.7	30.1	56.3	43.4	10.6	54.4	29.8	38.3
SimCSE-BERT-unsup	62.5	29.0	70.3	46.5	20.3	74.3	<b>31.2</b>	45.5
E5-PT <sub>small</sub>	67.0	41.7	78.2	53.1	40.8	68.8	25.2	54.2
E5-PT <sub>base</sub>	67.9	43.4	79.2	53.5	42.9	69.5	24.3	55.5
E5-PT <sub>large</sub>	69.0	<b>44.3</b>	80.3	54.4	44.2	69.9	24.8	56.4
<i>Supervised models</i>								
SimCSE-BERT-sup	67.3	33.4	73.7	47.5	21.8	79.1	23.3	48.7
BERT-FT <sub>base</sub>	68.7	33.9	82.6	50.5	41.5	79.2	29.0	55.2
Contriever	66.7	41.1	82.5	53.1	41.9	76.5	<u>30.4</u>	56.0
GTR <sub>large</sub>	67.1	41.6	<u>85.3</u>	55.4	47.4	78.2	29.5	58.3
Sentence-T5 <sub>large</sub>	72.3	41.7	85.0	54.0	36.7	<u>81.8</u>	29.6	57.1
E5 <sub>small</sub>	71.7	39.5	85.1	54.5	46.0	80.9	24.9	58.8
E5 <sub>base</sub>	<u>72.6</u>	42.1	85.1	<u>55.7</u>	<u>48.7</u>	81.0	26.0	<u>60.3</u>
E5 <sub>large</sub>	<b>73.1</b>	43.3	<b>85.9</b>	<b>56.5</b>	<b>50.0</b>	<b>82.1</b>	25.2	<b>61.3</b>
<i>Larger models</i>								
GTR <sub>xxl</sub>	67.4	42.4	86.1	56.7	48.5	78.4	30.6	59.0
Sentence-T5 <sub>xxl</sub>	73.4	43.7	85.1	56.4	42.2	82.6	30.1	59.5

# E5: оценка на Beir



	BM25	SimCSE	LaPraDor <sup>†</sup>	Contriever	cpt <sub>300M</sub>	E5-PT <sub>small</sub>	E5-PT <sub>base</sub>	E5-PT <sub>large</sub>
MS MARCO	22.8	9.4	16.9*	20.6	19.9	25.4	<u>26.0</u>	<b>26.2</b>
Trec-Covid	<b>65.6</b>	26.2	22.7	27.4	52.9	52.0	61.0	<u>61.8</u>
NFCorpus	32.5	9.9	31.1	31.7	32.0	29.3	<b>35.8</b>	<u>33.7</u>
NQ	32.9	11.7	18.1	25.4	-	37.3	<u>39.0</u>	<b>41.7</b>
HotpotQA	<b>60.3</b>	19.8	30.3	48.1	51.5	46.0	<u>52.4</u>	52.2
FiQA	23.6	9.8	20.3	24.5	34.1	38.3	<u>40.0</u>	<b>43.2</b>
ArguAna	31.5	38.3	<b>45.9</b>	37.9	38.7	42.5	42.2	<u>44.4</u>
Touche-2020	<b>36.7</b>	8.9	9.4	19.3	<u>21.0</u>	19.9	16.9	19.8
CQADupStack	29.9	13.2	22.0	28.4	-	35.0	<u>35.4</u>	<b>38.9</b>
Quora	78.9	78.0	78.7	83.5	68.1	<u>85.8</u>	85.7	<b>86.1</b>
DBPedia	31.3	15.0	25.0	29.2	27.2	34.5	<u>35.4</u>	<b>37.1</b>
Scidocs	15.8	5.5	13.3	14.9	-	19.9	<u>21.1</u>	<b>21.8</b>
Fever	<b>75.3</b>	21.1	36.8	68.2	57.1	62.5	63.4	<u>68.6</u>
Climate-Fever	<b>21.3</b>	11.8	13.8	15.5	<u>15.8</u>	14.5	15.4	<u>15.7</u>
Scifact	66.5	25.7	55.5	64.9	<u>65.4</u>	68.5	<b>73.7</b>	<u>72.3</u>
Average	41.7	20.3	29.3	36.0	-	40.8	<u>42.9</u>	<b>44.2</b>
Best on	<u>5</u>	0	1	0	0	0	<u>2</u>	<b>7</b>

# Векторные БД



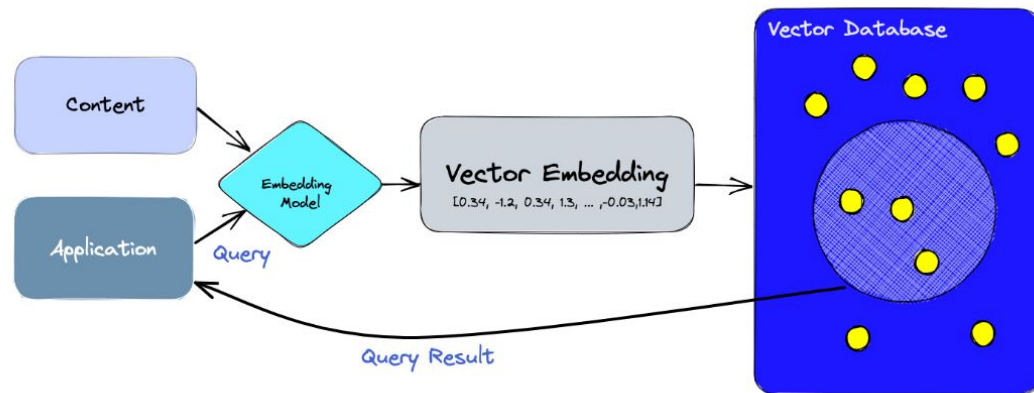
# Векторные базы данных

- Векторная база данных — это **тип базы данных**, в которой **хранятся** векторные представления объектов (**эмбеддинги**),
- Все данные не обязаны помещаться в оперативной памяти и даже на одной машине,
- Работает поверх разных алгоритмов неточного поиска соседей **Approximate Nearest Neighbor (ANN)**,
- Известные реализации:
  - Milvus,
  - vektonn,
  - qdrant,
  - FAISS (не векторная БД, но популярный пакет для быстрого поиска).

# Векторные базы данных: схема работы



1. Векторизация корпуса для последующей индексации в БД.
2. В БД создается индекс и заполняется векторами и связанной с ними информацией, например, самим текстом.
3. При поступлении запроса, запрос векторизуется и из индекса извлекаются ближайшие соседи на основе некоторой меры близости.



# Отличия векторной БД от векторного индекса



Векторные индексы, такие как FAISS тоже позволяют осуществлять быстрый поиск среди векторов эффективным способом (ANN), но:

- Дата менеджмент:
  - В БД удобный способ добавления и удаления данных.
  - Real-time обновления.
- Удобство поиска и использования:
  - В БД есть метаданные, такие как сам текст, название документа и другие, которые можно учитывать при поиске (фильтры).
  - В случае популярных фреймворков легкая интеграция в такие системы, как llamaindex, langchain и другие.
- Масштабируемость:
  - БД позволяет работать с данными превышающими объем RAM.
  - Многие решения позволяют масштабировать на несколько машин.

# Faiss пример



```
from datasets import load_dataset
```

```
news_dataset = load_dataset('zloelias/lenta-ru', split='test')
```

```
Downloading data: 100% ██████████ 240M/240M [00:15<00:00, 17.1MB/s]
```

```
Downloading data: 100% ██████████ 26.7M/26.7M [00:02<00:00, 15.7MB/s]
```

```
Generating train split: ██████████ 185972/0 [00:02<00:00, 70871.14 examples/s]
```

```
Generating test split: ██████████ 20664/0 [00:00<00:00, 47022.76 examples/s]
```

```
[7] sentences = [s.replace('\xa0', ' ').strip() for s in news_dataset['title']]  
sentences[:10]
```

```
['Брак в умных часах Watch вынудил Apple сократить их поставки',  
 'Пятой серии «Пиратов Карибского моря» придумали название',  
 'Английские болельщики выбрали лучшего футболиста',  
 'Прокуратура перечислила виновные в транспортном коллапсе авиакомпании',  
 'Мыши на ИКС доказали вредность космоса для кожи',  
 'С героями «Людей Икс» снимут сериал',  
 '«ЖКОС» поможет «Роснефти» порвать с «Газпромом»',  
 'Индонезия модернизирует транспортники Hercules',  
 'Результаты расследования смерти Хьюстон огласят после ее похорон',  
 'Нефть Brent побил антирекорд 2004 года']
```

```
[15] sentences = sentences[:1000]
```

```
[63] from tqdm import tqdm  
bs = 8  
vectors = []  
for i in tqdm(range(0, len(sentences), bs)):  
    vectors += vectorize_batch(sentences[i:i+bs], False)
```

```
100% ██████████ | 125/125 [06:08<00:00, 2.94s/it]
```

```
[64] import numpy as np  
vectors = [v.numpy() for v in vectors]  
vectors = np.vstack(vectors)  
vectors.shape
```

```
(1000, 1024)
```

```
import faiss
```

```
index = faiss.IndexFlatIP(vectors.shape[1])  
index.add(vectors)  
index.ntotal
```

```
1000
```

```
[91] D, I = index.search(vectorize_single('Киберспорт в России'), 10) # search  
print(I)
```

```
[[133 812 578 923 783 836 84 682 230 904]]
```

```
for j, i in enumerate(I[0]):  
    print(D[0][j], sentences[i])
```

```
0.83408076 Российского солдата будущего срисовали с компьютерной игры  
0.81235623 Российская сеть игровых автоматов выходит на биржу  
0.80241823 Жеребьевку Кубка России по футболу провели с нарушением регламента  
0.80022424 Хоккеист сборной России избежал дисквалификации  
0.79809797 "Авиалинии России" переименованы  
0.7978493 Сборная России по футболу сыграла вничью с Кот-д'Ивуаром  
0.7969447 Российский рынок подготовился к санкциям  
0.7965601 Михаил Гершкович: К чемпионату мира сборная России прибавит  
0.7964196 «Звездные войны» стали самой популярной игрой для iPad в России  
0.7959726 Бывший тренер «Зенита» рассказал о работе в России
```

# Milvus пример: создание коллекции и индекса



```
import numpy as np
import time

from pymilvus import (
    connections,
    utility,
    FieldSchema, CollectionSchema, DataType,
    Collection,
)
```

connections.connect("default", host="localhost", port="19530")

✓ 0.1s

```
fields = [
    FieldSchema(name="id", dtype=DataType.VARCHAR, is_primary=True, auto_id=False, max_length=100),
    FieldSchema(name="csv_name", dtype=DataType.VARCHAR, max_length=10),
    FieldSchema(name="sentence", dtype=DataType.VARCHAR, max_length=1024),
    FieldSchema(name="embeddings", dtype=DataType.FLOAT_VECTOR, dim=1024)
]
```

```
schema = CollectionSchema(fields, "lenta_ru_e5")
news = Collection("lenta_ru", schema, consistency_level="Strong")
```

✓ 0.0s

```
for i in tqdm(range(0, len(sentences), bs)):
    sentences_batch = sentences[i:i+bs]
    vectors_batch = [v.numpy() for v in vectorize_batch(sentences_batch, False)]
    entities = [
        ['bs' + str(i//bs) + '_' + str(j) for j in range(len(sentences_batch))],
        ['0.csv'] * len(sentences_batch),
        sentences_batch, # field random, only supports list
        np.array(vectors_batch), # field embeddings, supports numpy.ndarray and list
    ]
    insert_result = news.insert(entities)
news.flush()
```

✓ 39.2s

100% | ██████████ | 2583/2583 [00:35<00:00, 72.38bit/s]

```
index = {
    "index_type": "IVF_FLAT",
    "metric_type": "COSINE",
    "params": {"nlist": 1024},
}
```

```
news.create_index("embeddings", index)
news.load()
```

Предварительно нужно поднять сам сервис с Milvus, например, используя docker

# Milvus пример: обращение базе



```
from tqdm import tqdm
search_params = {
    "metric_type": "COSINE",
    "params": {"nprobe": 10},
}

query = 'Киберспорт в России'
query_vec = vectorize_single(query)[0]
result = news.search([query_vec], "embeddings", search_params, limit=10, output_fields=["sentence"])
for hits in result:
    for hit in hits:
        print(hit)
```

✓ 0.3s

```
id: bs16_5, distance: 0.8340807557106018, entity: {'sentence': 'Российского солдата будущего срисовали с компьютерной игры'}
id: bs1144_3, distance: 0.8275099396705627, entity: {'sentence': 'В России опубликуют летопись видеоигр'}
id: bs454_5, distance: 0.8234059810638428, entity: {'sentence': 'Футбол назвали угрозой легкой атлетике в России'}
id: bs577_7, distance: 0.8228023052215576, entity: {'sentence': 'В России отреагировали на информацию о допинге среди футболистов'}
id: bs804_2, distance: 0.8198419809341431, entity: {'sentence': 'В России появился хоккейный телеканал'}
id: bs719_3, distance: 0.8143962621688843, entity: {'sentence': 'Украинский спортсмен стал выступать за Россию'}
id: bs1586_4, distance: 0.8136853575706482, entity: {'sentence': 'Футбольный "Спартак" можно будет потренировать в компьютерной игре'}
id: bs462_6, distance: 0.811881422996521, entity: {'sentence': 'Россия испытала робота-смертника'}
id: bs1774_6, distance: 0.811394453048706, entity: {'sentence': 'Пике обратился к российским спортсменам'}
id: bs1020_3, distance: 0.8112483620643616, entity: {'sentence': 'Российские борцы сильнее всех'}
```

# Qdrant пример: создание коллекции и индексация



```
from qdrant_client import QdrantClient
from qdrant_client.http import models
from qdrant_client.models import PointStruct
from tqdm import tqdm
```

```
client = QdrantClient('localhost', port=6333)
client.recreate_collection(
    collection_name='test',
    vectors_config=models.VectorParams(
        size=1024,
        distance=models.Distance.COSINE
    )
)
```

```
bs = 8
for i in tqdm(range(0, len(dataset), bs)):
    segments_batch = dataset[i: i+bs]
    vectors = vectorize(['passage: {s}' for s in segments_batch])
    vectors = [v.tolist() for v in vectors]
    ids = [j for j in range(i, i+bs)]
    client.upsert(
        collection_name='test',
        points=[
            PointStruct(
                id=idx,
                vector=vector,
                payload={'text': segment}
            )
            for idx, vector, segment in zip(*[ids, vectors, segments_batch])
        ]
    )
```

Предварительно нужно поднять сам сервис с Qdrant, например, используя docker.

# Qdrant: альтернативный способ



Без запуска Qdrant сервера

```
from qdrant_client import QdrantClient

client = QdrantClient(":memory:")
# or
client = QdrantClient(path="path/to/db")
```

# Qdrant пример: обращение базе



```
query = 'Киберспорт в России'  
query_vec = vectorize_single(query)[0]  
  
closest_vectors = qdrant_client.search(  
    collection_name='qdrant_news_lenta',  
    search_params=models.SearchParams(hnsw_ef=128, exact=True),  
    query_vector=query_vec,  
    limit=10,  
    with_vectors=False,  
    with_payload=True  
)  
closest_vectors
```

✓ 0.2s

```
[ScoredPoint(id=133, version=16, score=0.83408076, payload={'csv_name': '0.csv', 'sentence': 'Российского солдата будущего срисовали с компьютерной игры'}, vector=None, shard_key=None),  
ScoredPoint(id=9155, version=1144, score=0.8275099, payload={'csv_name': '0.csv', 'sentence': 'В России опубликуют летопись видеоигр'}, vector=None, shard_key=None),  
ScoredPoint(id=19818, version=2477, score=0.82631135, payload={'csv_name': '0.csv', 'sentence': 'Российские мотострелки получают виртуальный боевой тренажер'}, vector=None, shard_key=None),  
ScoredPoint(id=3637, version=454, score=0.8234061, payload={'csv_name': '0.csv', 'sentence': 'Футбол назвали угрозой легкой атлетике в России'}, vector=None, shard_key=None),  
ScoredPoint(id=4623, version=577, score=0.8228024, payload={'csv_name': '0.csv', 'sentence': 'В России отреагировали на информацию о допинге среди футболистов'}, vector=None, shard_key=None),  
ScoredPoint(id=11396, version=1424, score=0.8210906, payload={'csv_name': '0.csv', 'sentence': 'В России появился первый спортивный канал'}, vector=None, shard_key=None),  
ScoredPoint(id=20056, version=2507, score=0.8209353, payload={'csv_name': '0.csv', 'sentence': 'Nival Online отправит российских геймеров в страну Пиратию'}, vector=None, shard_key=None),  
ScoredPoint(id=6434, version=804, score=0.819842, payload={'csv_name': '0.csv', 'sentence': 'В России появился хоккейный телеканал'}, vector=None, shard_key=None),  
ScoredPoint(id=14390, version=1798, score=0.8166189, payload={'csv_name': '0.csv', 'sentence': 'В России появится экспертный совет по зимним видам спорта'}, vector=None, shard_key=None),  
ScoredPoint(id=1119, version=139, score=0.8152795, payload={'csv_name': '0.csv', 'sentence': 'В российском баскетболе произошел раскол'}, vector=None, shard_key=None)]
```

# Векторные базы данных: как сравнивать?



- Скорость поиска.
- Точность поиска.
- Скорость создания индекса.
- Скорость загрузки БД.
- ...

Интересное начинается на количестве векторов от миллиона и далее. Все что до этого количества, обычно работает приемлемо в любой БД.

# Векторные базы данных: как сравнивать?



A comparison of leading vector databases

	Pinecone	Weaviate	Milvus	Qdrant	Chroma	Elasticsearch	PGvector
Is open source	✗	✓	✓	✓	✓	✗	✓
Self-host	✗	✓	✓	✓	✓	✓	✓
Cloud management	✓	✓	✓	✓	✗	✓	(✓)
Purpose-built for Vectors	✓	✓	✓	✓	✓	✗	✗
Developer experience	👍👍👍	👍👍	👍👍	👍👍	👍👍	👍	👍
Community	Community page & events	8k★ github, 4k slack	23k★ github, 4k slack	13k★ github, 3k discord	9k★ github, 6k discord	23k slack	6k★ github
Queries per second (using text nytimes-256-angular)	150 *for p2, but more pods can be added	791	2406	326	?	700-100 *from various reports	141
Latency, ms (Recall/Percentile 95 (millis), nytimes-256-angular)	1 *batched search, 0.99 recall, 200k SBERT	2	1	4	?	?	8
Supported index types	?	HNSW	Multiple (11 total)	HNSW	HNSW	HNSW	HNSW/IVFFlat
Hybrid Search (i.e. scalar filtering)	✓	✓	✓	✓	✓	✓	✓
Disk index support	✓	✓	✓	✓	✓	✗	✓
Role-based access control	✓	✗	✓	✗	✗	✓	✗

# Векторные базы данных: как сравнивать?

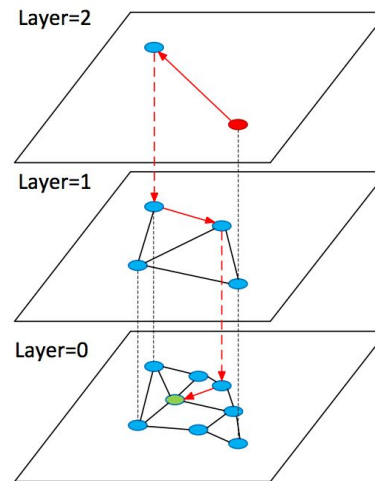
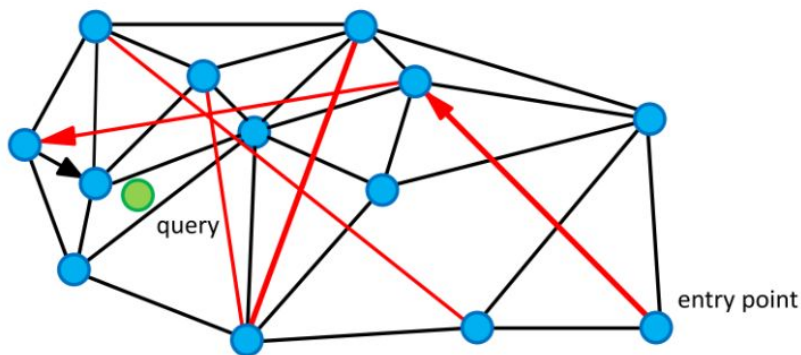


Engine	Setup	Dataset	Upload Time(m)	Upload + Index Time(m)	P95(ms)	RPS	P99(ms)	Latency(ms)	Precision
qdrant	<a href="#">qdrant-sq-rps-m-64-ef-512</a>	dbpedia-openai-1M-1536-angular	3.51	24.43	4.95	<b>1238.0016</b>	8.62	3.54	0.99
elasticsearch	<a href="#">elasticsearch-m-32-ef-128</a>	dbpedia-openai-1M-1536-angular	19.18	83.72	72.53	<b>716.80</b>	135.68	22.10	0.98
redis	<a href="#">redis-m-32-ef-256</a>	dbpedia-openai-1M-1536-angular	92.49	92.49	160.85	<b>625.27</b>	167.35	140.65	0.97
weaviate	<a href="#">weaviate-m-32-ef-256</a>	dbpedia-openai-1M-1536-angular	25.98	25.98	465.42	<b>352.50</b>	782.39	279.44	0.97
milvus	<a href="#">milvus-m-16-ef-128</a>	dbpedia-openai-1M-1536-angular	0.27	1.16	441.32	<b>219.11</b>	576.65	393.31	0.99

# Hierarchical Navigable Small World (HNSW)



- Известный ANN алгоритм,
- Идея в использовании графа, в котором путь от любой вершины к любой другой будет занимать небольшое количество операций,
  - Сложность поиска в таком графе оценивается как  $O(\log^k n)$ ,
- Иерархический алгоритм - графов несколько, разного уровня подробности.



# HNSW: индексация



- Вершины добавляются в граф **последовательно одна за другой** в случайном порядке,
- На каждом шаге новая вершина связывается ребром с **M наиболее близкими вершинами** в полученном на данном этапе графе,
- При добавлении новой вершины случайно выбирается число  $l$  от 0 до  $L$  ( $L$  - максимальный индекс слоя), означающее **максимальный номер слоя**, на котором будет присутствовать данная вершина.



- От верхних слоев к нижним,
- Переход ниже при нахождении наиболее близкой вершины на текущем уровне,
- Поиск вершин происходит следующим образом:
  - Выбирается точка входа, это либо случайная вершина вначале, либо же вершина, на которой алгоритм остановился на прошлом слое,
  - На каждой вершине рассчитывается близость ко всем соседям, таким образом двигаясь в сторону повышения близости к вектору-запросу.

# Зачем нам нужны векторные БД



- Основная цель: хороший поиск релевантных сегментов текста / предложений / картинок ...
  - Сильно зависит от используемой модели векторизации.
  - Размер и содержимое сегментов играет важную роль.
- Как можно повысить качество поиска:
  - Гибридный поиск: BM25 + векторный поиск.
  - Переранжирование (re-ranking) с использованием cross-encoder модели.
  - MMR для борьбы с дубликатами в выдаче.

**RAG**

# Проблема актуальных знаний языковых моделей



- Языковые модели являются по сути неявным представлением знаний, которые выучиваются по корпусу текстов.
- Как следствие, обученная модель имеет статичный набор знаний.
- Но наш мир меняется каждую секунду!

Model
openchat-3.5-0106/
Task
Какой сейчас год?
Output
Сейчас 2022 год.< end_of_turn >

# Проблемы контекстов



- Допустим мы строим чат-бота помощника для скалолазов.
- Где купить хорошую кошку?

Model

openchat-3.5-0106/

Task

Где купить хорошую кошку?

Output

Для покупки хорошей кошки вам следует обратить внимание на несколько важных факторов:

1. **Супермаркеты и магазины питомцев**: Обычно в таких местах продаются кошки, которые были спасены из условий, не подходящих для них. Это может быть хороший способ найти себе друга, если вы готовы предоставить ему дом и любовь.
2. **Собачьи и кошачьи клубы**: В таких клубах часто проводятся выставки и соревнования, где можно найти различных пород кошек. Обратите внимание на их характеристики и поведение, чтобы выбрать подходящую породу.
3. **Кошачьи приюты и спасательные организации**: Этот вариант позволяет спасти кошку, которая оказалась в неприятной ситуации. В таких приютах кошки часто получают медицинскую помощь и ухода, чтобы быть готовыми к новой жизни.

# Проблемы контекстов



Model

openchat-3.5-0106/

Task

Где купить хорошую кошку?

Context

Кошки[1] (кошки альпинистские[2]) (англ. Crampons, нем. Steigeisen, итал. Ramponi) — металлические приспособления для увеличения противоскольжения, обычно, при передвижении по снежно-ледовому рельефу. В современном виде представляют собой металлическую пластину из высокотехнологичных металлических сплавов с 10/12 (реже 14/15) клиновидными зубьями, имеющую определённую систему крепления к обуви. Как элемент снаряжения используются, преимущественно, в таких видах спорта, как альпинизм (и его производных дисциплинах, например, ледолазании), горном туризме, а также в ряде профессий[3].

Подобные кошкам приспособления для увеличения противоскольжения при передвижении по рельефу, согласно археологическим находкам, появились более 3000 лет назад. Как элемент спортивного снаряжения для альпинизма первые модели появились уже в XIX веке, но популярность приобрели только во второй половине XX века[4].

System hint

Здесь можно ввести подсказку для генерации

Output

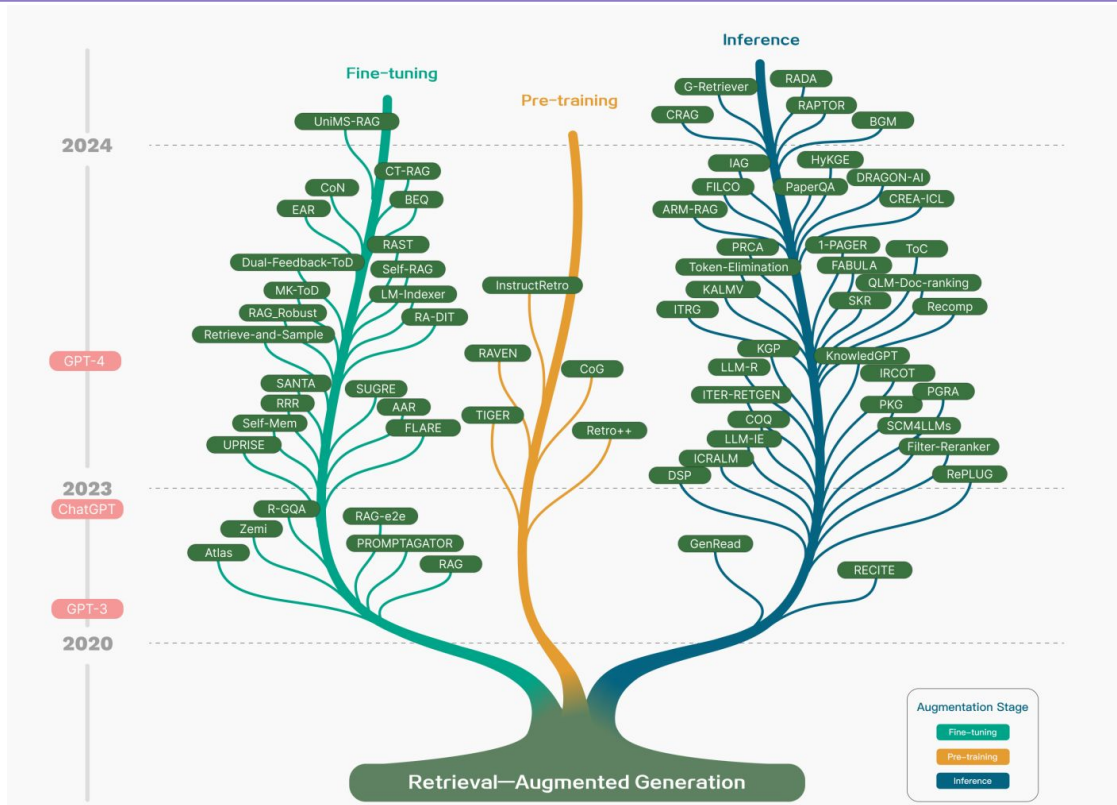
Для покупки хорошей кошки (альпинистской) можно обратиться к специализированным магазинам спортивного снаряжения, таким как "Спортмастер", "Спорт-Экспресс" или "Спорт-Контакт". Кроме того, вы можете покупать кошки в интернет-магазинах, таких как Wildberries, Ozon или других онлайн-платформах, где предлагается широкий ассортимент спортивного снаряжения.

При поиске хорошей кошки важно обратить внимание на следующие параметры:

1. **Материал:** кошки должны быть изготовлены из высокотехнологичных металлических сплавов, чтобы обеспечить максимальную прочность и долговечность.
2. **Количество и размер зубьев:** кошки с 10/12 (или 14/15) клиновидными зубьями обеспечат лучшую противоскольжение на снежно-ледовом рельефе.
3. **Система крепления:** кошки должны иметь удобную систему крепления к обуви, чтобы не создавать дискомфорта при использовании.
4. **Марка и репутация производителя:** покупайте кошки у известных и надежных производителей, таких как La Sportiva, Scarpa, Black Diamond и других.
5. **Цена:** стоимость хорошей кошки может варьироваться в зависимости от бренда и качества, поэтому сравните цены в разных магазинах и онлайн-платформах, чтобы найти оптимальное соотношение цены и качества.

После покупки кошки не забывайте проверять их на наличие дефектов и соблюдайте правильный способ их использования и хранения, чтобы продлить срок их службы и обеспечить безопасность при альпинизме и других видах спорта на снежно-ледовом рельефе.<|end\_of\_turn|>

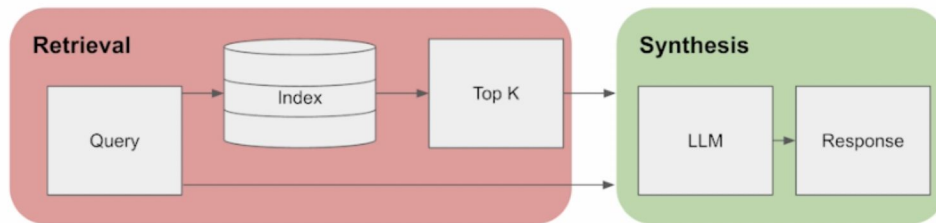
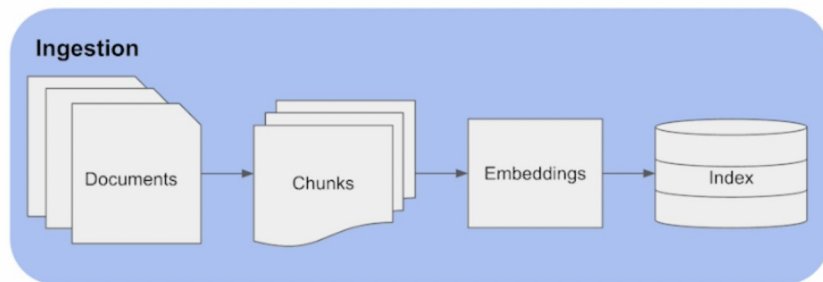
# RAG



# Базовая схема RAG



## Basic RAG Pipeline



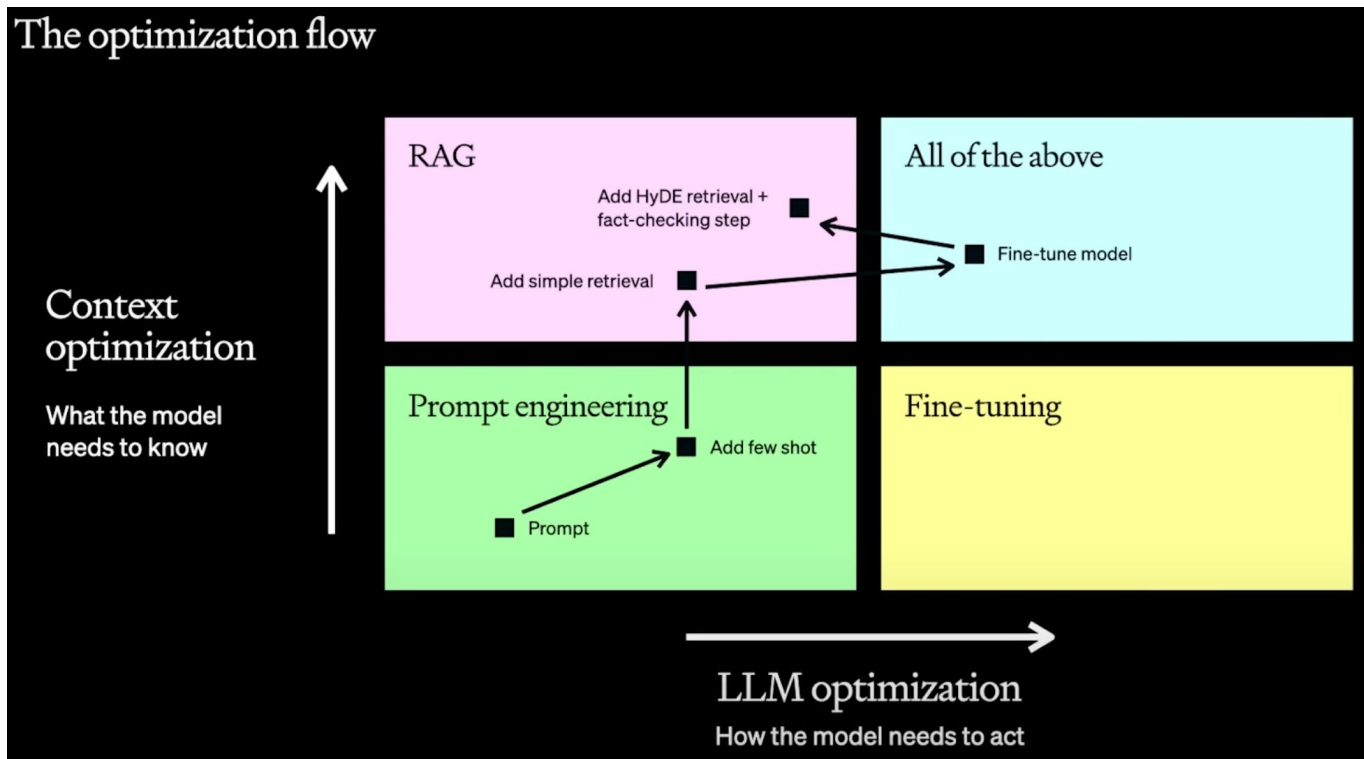


- Хороший сегмент - тот сегмент, который помещается в контекст модели и содержит релевантную информацию и минимум шума.
  - Существуют ограничения с точки зрения модели векторизации, обычно 512 токенов.
- Слишком длинные сегменты хуже векторизуются и с точки зрения качества поиска.
- Слишком короткие сегменты приводят к большому размеру базы данных.
  - На каждый сегмент по hidden size вектору
- Слишком короткие сегменты не содержат достаточно релевантной информации и часто оторваны от контекста

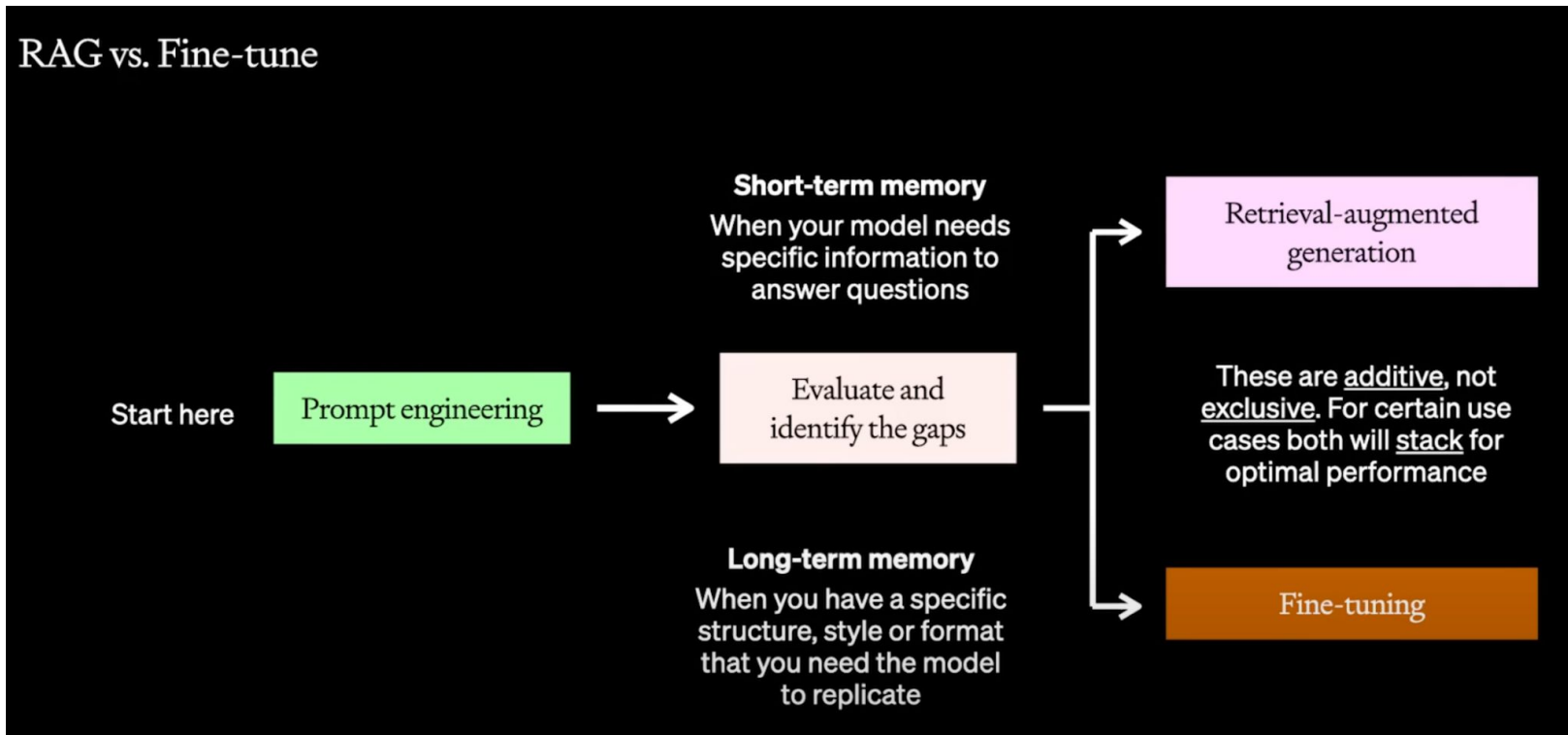


- По предложениям:
  - Просто, но скорее всего плохо.
- По параграфам ( $\backslash n$ ):
  - Параграфы все еще могут быть также слишком длинными и слишком короткими.
  - Дополнительные эвристики обычно улучшают результат: объединение коротких, разбиение длинных на несколько сегментов.
- Small to Big Retrieval / Sentence Window Retrieval.

# Как заставить LLM работать лучше?



# RAG vs Fine-tune



# Когда нужно использовать RAG



Для чего RAG подходит:

- Предоставление модели дополнительной информации / знаний.
- Борьба с галлюцинированием моделей.

Для чего RAG не подходит:

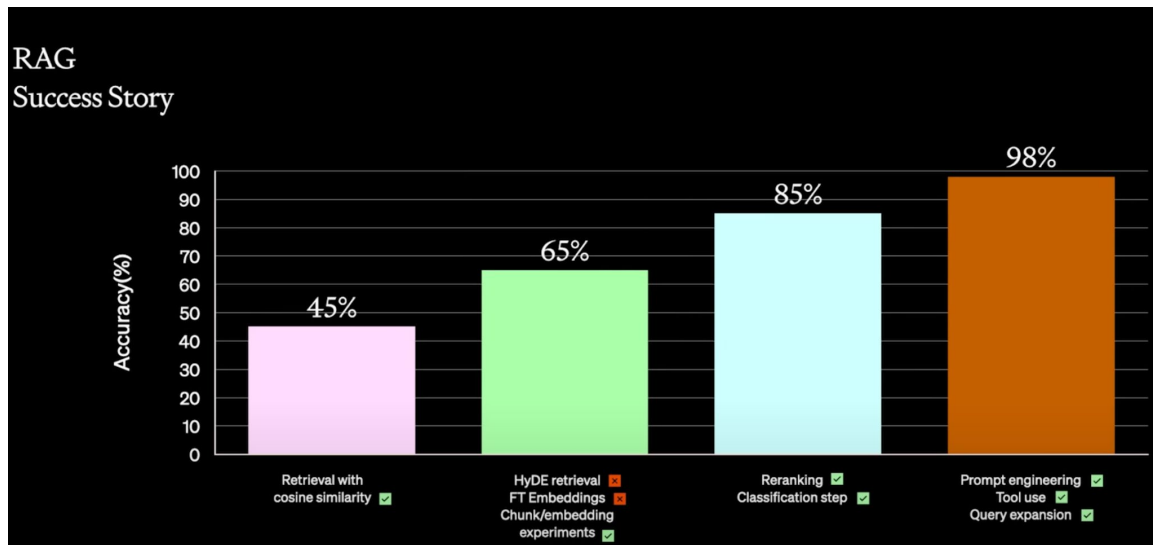
- Адаптация модели на целую предметную область
- Новый стиль, язык, манера поведения
- Снижение затрат токенов / времени ответа.

# RAG Success Story от OpenAI



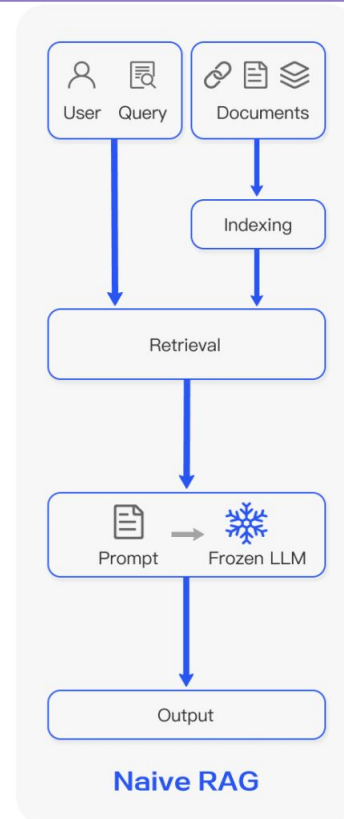
## Помогло:

- Подбор подходящих размеров чанков и способов их кодирования.
- Переранжирование
  - Кросс-энкодер
  - Rule-based признаки: дата и тп.
- Классификация (в данных было два домена)
- Тщательный подбор промптов и расширение запроса

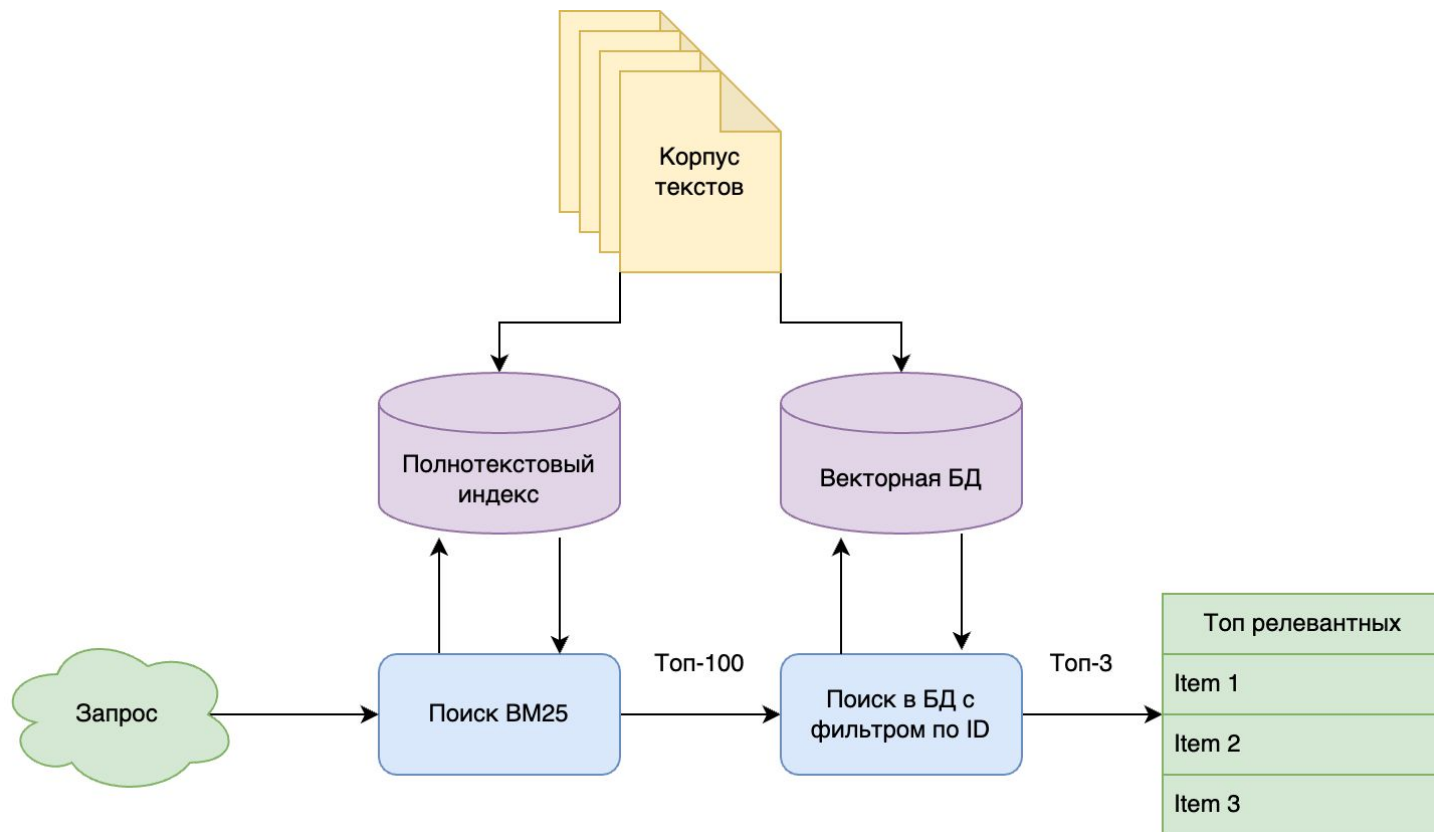


## Текущий “наивный” подход к RAG:

- Retrieval и Generation шаги разделены.
  - Retrieval - поиск на основе **энкодеров** текстов (e5, contriever ... )
  - Generation - в основе мощная **инструктивная LLM**.
- Retrieval и Generation модули **не дообучаются** (в простом, наивном подходе)
- Результат поиска добавляется в промпт LLM



# Гибридный поиск: BM25 + vector sim



# MMR

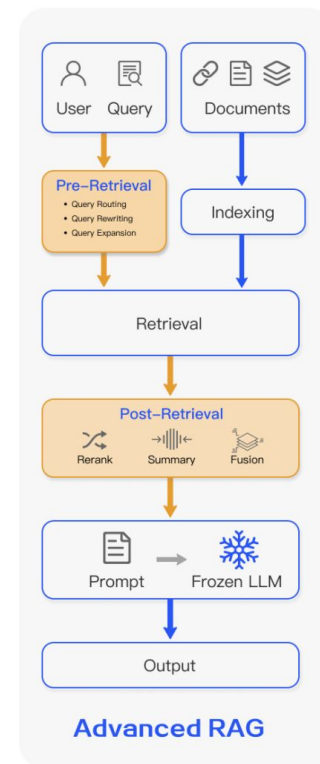
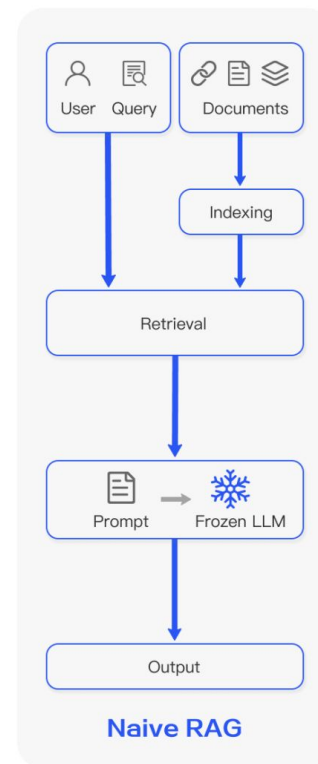


- Цель в повышении разнообразия в выдаче.
- Вес документа штрафуетя на основании его близости к уже извлеченным.
- Параметр позволяет регулировать силу штрафа за похожесть.

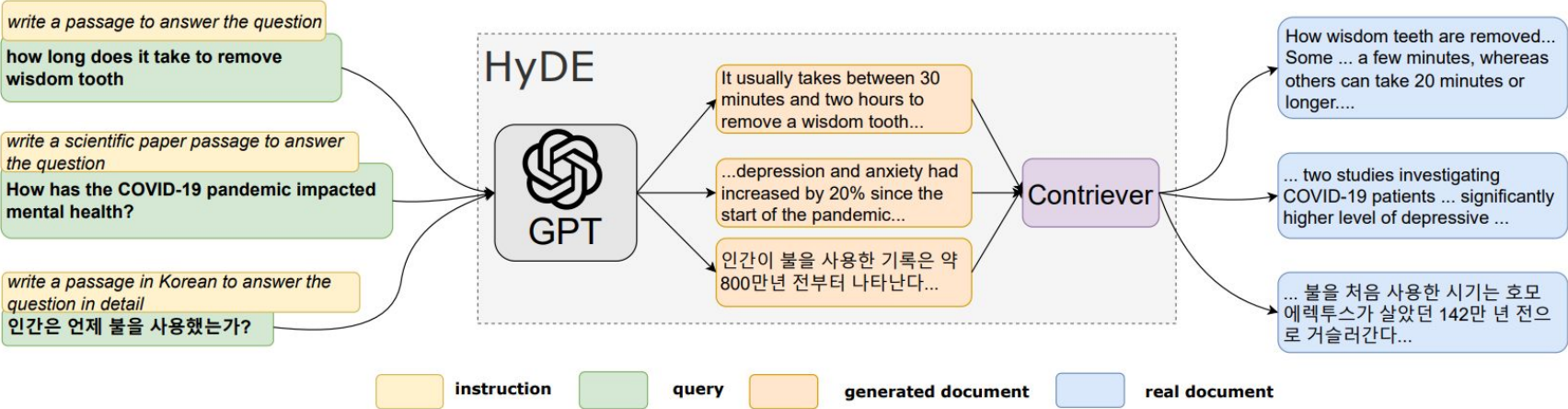
$$MMR \stackrel{\text{def}}{=} \text{Arg} \max_{D_i \in R \setminus S} \left[ \lambda (\text{Sim}_1(D_i, Q)) - (1 - \lambda) \max_{D_j \in S} \text{Sim}_2(D_i, D_j) \right]$$

“Advanced” RAG -> по сути сосредоточен на том, чтобы:

- Улучшить качество поиска.
- Улучшить качество представления извлеченных данных для LLM.



# Hypothetical Document Embeddings





- Не относитесь к LLM как к экспертной системе (особенно в случаях конкретных областей / задач).
- **Качество поиска играет ключевую роль**
  - Подбор подходящего эмбеддера.
  - Гибридный поиск.
  - Переранжирование (например, кросс-энкодером).
  - MMR
- Подход к разбиению на сегменты прямо связан с поиском
- Промптинг LLM практически всегда может улучшить итоговое качество.
- Может быть полезен краткий курс:  
<https://www.deeplearning.ai/short-courses/building-evaluating-advanced-rag/>



- Удобный фреймворк для объединения в одну систему различных моделей, систем поиска и тп.
- Большое комьюнити, открытый код.
- Интегрированы различные инструменты, базы данных и большинство из того, что может понадобится при создании своей вопросно-ответной системы.



# langchain: основные абстракции

- Модели
  - Как локальные, так и по api.
- Индексы
  - Базы данных как раз тут.
- Промпты
- Цепочки
  - Последовательность модулей/действий.
- Инструменты

# langchain: векторизация



```
from langchain_community.embeddings import HuggingFaceEmbeddings

embeddings = HuggingFaceEmbeddings(
    model_name='intfloat/multilingual-e5-large',
    model_kwargs={"device": "cpu"}
)

[2] ✓ 6.1s

... modules.json: 100% ██████████ 387/387 [00:00<00:00, 53.3kB/s]
... README.md: 100% ██████████ 160k/160k [00:00<00:00, 2.21MB/s]
...
... sentence_bert_config.json: 100% ██████████ 57.0/57.0 [00:00<00:00, 10.7kB/s]
...
... 1_Pooling/config.json: 100% ██████████ 201/201 [00:00<00:00, 35.7kB/s]

import numpy as np
query = 'Киберспорт в России'
vec = np.array(embeddings.embed_query(f'query: {query}'))
vec

[6] ✓ 0.1s

... array([-0.00479927, -0.0066398 , -0.0316752 , ...,  0.00573863,
        -0.01711251,  0.00577613])
```

# langchain: поиск в Qdrant



```
from langchain_community.vectorstores import Qdrant
from qdrant_client import QdrantClient

qdrant_client = QdrantClient("localhost", port=6333, timeout=100)
doc_store = Qdrant(qdrant_client, collection_name='qdrant_news_lenta',
                  embeddings=embeddings,
                  content_payload_key="sentence")
```

✓ 0.0s

# langchain: поиск в Qdrant



```
query = 'Цены на нефть'|
7] ✓ 0.0s

retriever = doc_store.as_retriever(search_type='similarity', search_kwargs={"k": 10})
retriever.get_relevant_documents('query: ' + query)
8] ✓ 0.1s

[Document(page_content='Цены на нефть снижаются', metadata={'_id': 18363, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='Цены на нефть падают', metadata={'_id': 15889, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='Цены на нефть продолжают падать', metadata={'_id': 2597, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='Цены на нефть продолжают падать', metadata={'_id': 4370, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='На биржах заметно подорожала нефть', metadata={'_id': 14378, '_collection_name': 'qdrant_news_lenta'})
 Document(page_content='Обзор рынков: нефть подорожала на два доллара', metadata={'_id': 16732, '_collection_name': 'qdrant_news_lenta'})
 Document(page_content='Минэкономразвития повысил базовую цену на нефть', metadata={'_id': 9025, '_collection_name': 'qdrant_news_lenta'})
 Document(page_content='Мировые цены на нефть значительно снизились', metadata={'_id': 19947, '_collection_name': 'qdrant_news_lenta'})
 Document(page_content='Цена на нефть не удержалась выше 70 долларов за баррель', metadata={'_id': 2197, '_collection_name': 'qdrant_news_lenta'})
 Document(page_content='Обзор рынков: нефть подорожала почти на два доллара за баррель', metadata={'_id': 1603, '_collection_name': 'qdrant_news_lenta'})]
```

# langchain: поиск в Qdrant + mmr



```
retriever = doc_store.as_retriever(search_type='mmr', search_kwargs={"k": 10})  
retriever.get_relevant_documents('query: ' + query)
```

✓ 0.1s

```
[Document(page_content='Цены на нефть снижаются', metadata={'_id': 18363, '_collection_name': 'qdrant_news_lenta'}),  
Document(page_content='Цены на нефть побили новый рекорд; ОПЕК называет это безумием, но ничего не может сделать', metadata={'_id': 16732, '_collection_name': 'qdrant_news_lenta'}),  
Document(page_content='Обзор рынков: нефть подорожала на два доллара', metadata={'_id': 8912, '_collection_name': 'qdrant_news_lenta'}),  
Document(page_content='Иранский кризис поднял цены на нефть', metadata={'_id': 10463, '_collection_name': 'qdrant_news_lenta'}),  
Document(page_content='Утренние цены на нефть приближаются к рекордной отметке', metadata={'_id': 15742, '_collection_name': 'qdrant_news_lenta'}),  
Document(page_content='ОПЕК назвала оптимальный уровень цен на нефть', metadata={'_id': 2197, '_collection_name': 'qdrant_news_lenta'}),  
Document(page_content='Цена на нефть не удержалась выше 70 долларов за баррель', metadata={'_id': 14378, '_collection_name': 'qdrant_news_lenta'}),  
Document(page_content='На биржах заметно подорожала нефть', metadata={'_id': 19085, '_collection_name': 'qdrant_news_lenta'}),  
Document(page_content='Обзор рынков: рекордные цены на нефть опустили мировые фондовые индексы', metadata={'_id': 6639, '_collection_name': 'qdrant_news_lenta'}),  
Document(page_content='Обзор рынков: баррель нефти стоит 78 долларов', metadata={'_id': 6639, '_collection_name': 'qdrant_news_lenta'})]
```

# langchain: BM25



```
from langchain_core.documents import Document
from langchain_community.retrievers import BM25Retriever
import pymorphy2

parser = pymorphy2.MorphAnalyzer()
# cheap preprocessing
def preprocess(text):
    words = text.lower().split()
    words = [parser.parse(w)[0].normal_form for w in words]
    return words

print(preprocess('Цены на нефть снижаются'))
docs = [Document(page_content=point.payload['sentence'], metadata={'id': point.id})
        for point in qdrant_client.scroll(
            collection_name="qdrant_news_lenta", limit=qdrant_client.get_collection('qdrant_news_lenta').vectors_count
        )[0]]

bm25_retriever = BM25Retriever.from_documents(docs, preprocess_func=preprocess, k=10)
```

✓ 15.8s

['цена', 'на', 'нефть', 'снижаться']

```
bm25_retriever.get_relevant_documents(query)
```

✓ 0.0s

```
[Document(page_content='Цены на нефть снижаются', metadata={'id': 18363}),
 Document(page_content='Цены на нефть падают', metadata={'id': 15889}),
 Document(page_content='Цены на нефть продолжают падать', metadata={'id': 4370}),
 Document(page_content='Цены на нефть продолжают падать', metadata={'id': 2597}),
 Document(page_content='Минэкономразвития повысил базовую цену на нефть', metadata={'id': 9025}),
 Document(page_content='Иранский кризис поднял цены на нефть', metadata={'id': 8912}),
 Document(page_content='Цена на нефть превысила 50 долларов', metadata={'id': 9644}),
 Document(page_content='Мировые цены на нефть значительно снизились', metadata={'id': 19947}),
 Document(page_content='Кудрин предсказал снижение цен на нефть', metadata={'id': 15143}),
 Document(page_content='Цена на нефть пробила психологическую отметку', metadata={'id': 10773})]
```

# langchain: поиск + LLM



```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch
model_name = 'openchat/openchat-3.5-1210'
model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype=torch.float16, attn_implementation='flash_attention_2', device_map='auto')
model.eval()
```

```
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

✓ 8.7s

Loading checkpoint shards: 100%  3/3 [00:06<00:00, 2.16s/it]

```
/home/mtikhomi/.virtualenvs/pytorch-dev1/lib/python3.10/site-packages/transformers/generation/configuration_utils.py:410: UserWarning: `do_sample`  
warnings.warn(
```

```
/home/mtikhomi/.virtualenvs/pytorch-dev1/lib/python3.10/site-packages/transformers/generation/configuration_utils.py:410: UserWarning: `do_sample`  
warnings.warn(
```

Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.

```
from transformers import pipeline
from langchain_community.llms.huggingface_pipeline import HuggingFacePipeline

pipeline_kwargs={"max_new_tokens": 128, "do_sample": True, "temperature": 0.2}
pipe = pipeline("text-generation", model=model, tokenizer=tokenizer, **pipeline_kwargs)
hugging_face_pipeline = HuggingFacePipeline(pipeline=pipe)
```

✓ 0.0s

# langchain: поиск + LLM



```
query = 'Как дела 📺 киберспортом в РФ?'
relevant_docs = retriever.get_relevant_documents('query: ' + query)
docs_text = '\n\n'.join([doc.page_content for doc in relevant_docs])
chat = [
    {"role": "user", "content": f"Релевантная информация: {docs_text}\n\nВопрос: {query}"},
]
relevant_docs
```

✓ 0.1s

Python

```
[Document(page_content='Российского солдата будущего срисовали с компьютерной игры', metadata={'_id': 133, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='В фильм CBS о допинге в российском спорте включены заявления без доказательств', metadata={'_id': 12229, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='Создатель World of Warcraft предложил включить кибертурниры в программу Олимпиады', metadata={'_id': 10598, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='Nival Online потратит на создание российской MMORPG 20 миллионов долларов', metadata={'_id': 18037, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='Российским букмекерам не продлили лицензии', metadata={'_id': 1191, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='Нурмагомедов усомнился в существовании российского футбола', metadata={'_id': 10636, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='В России появился хоккейный телеканал', metadata={'_id': 6434, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='ФИФА выяснила проблемы России при подготовке к ЧМ-2018', metadata={'_id': 10761, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='В России отреагировали на информацию о допинге среди футболистов', metadata={'_id': 4623, '_collection_name': 'qdrant_news_lenta'}),
 Document(page_content='Российская сеть игровых автоматов выходит на биржу', metadata={'_id': 812, '_collection_name': 'qdrant_news_lenta'})]
```

```
def apply_chat(messages):
    prompt = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
    #print(prompt)
    return prompt
chain = apply_chat | hugging_face_pipeline
res = chain.invoke(chat)
print(res[res.rfind('GPT4 Correct Assistant:'):])
```

✓ 3.7s

Python

GPT4 Correct Assistant: Киберспорт в России развивается активно. Недавно компания Nival Online объявила о своем решении вложить в создание российской MMORPG 20

# Оценка качества RAG: Ragas



Ragas score	
Generation	Retrieval
<p>How well the LLM answered the question</p>	<p>How relevant the content retrieved is to the question</p>
<p><b>Faithfulness</b> How factually accurate is the generated answer</p>	<p><b>Context precision</b> The signal to noise ratio of retrieved context</p>
<p><b>Answer relevancy</b> How relevant is the generated answer to the question</p>	<p><b>Context recall</b> Can it retrieve all the relevant information required to answer the question</p>



- Требуется тестовый набор триплетов (вопрос, контексты, ответ).
- Все метрики рассчитываются за счет использования мощной LLM, обычно GPT-4, что может быть затратно / не применимо в ряде случаев.
- Встроенные промпты для работы на английском языке.
- Очень ограниченная применимость в специальных предметных областях.

Однако

- Возможна кастомизация: выбор LLM для оценки качества.
- Интегрирован langchain.
- Открытый код.