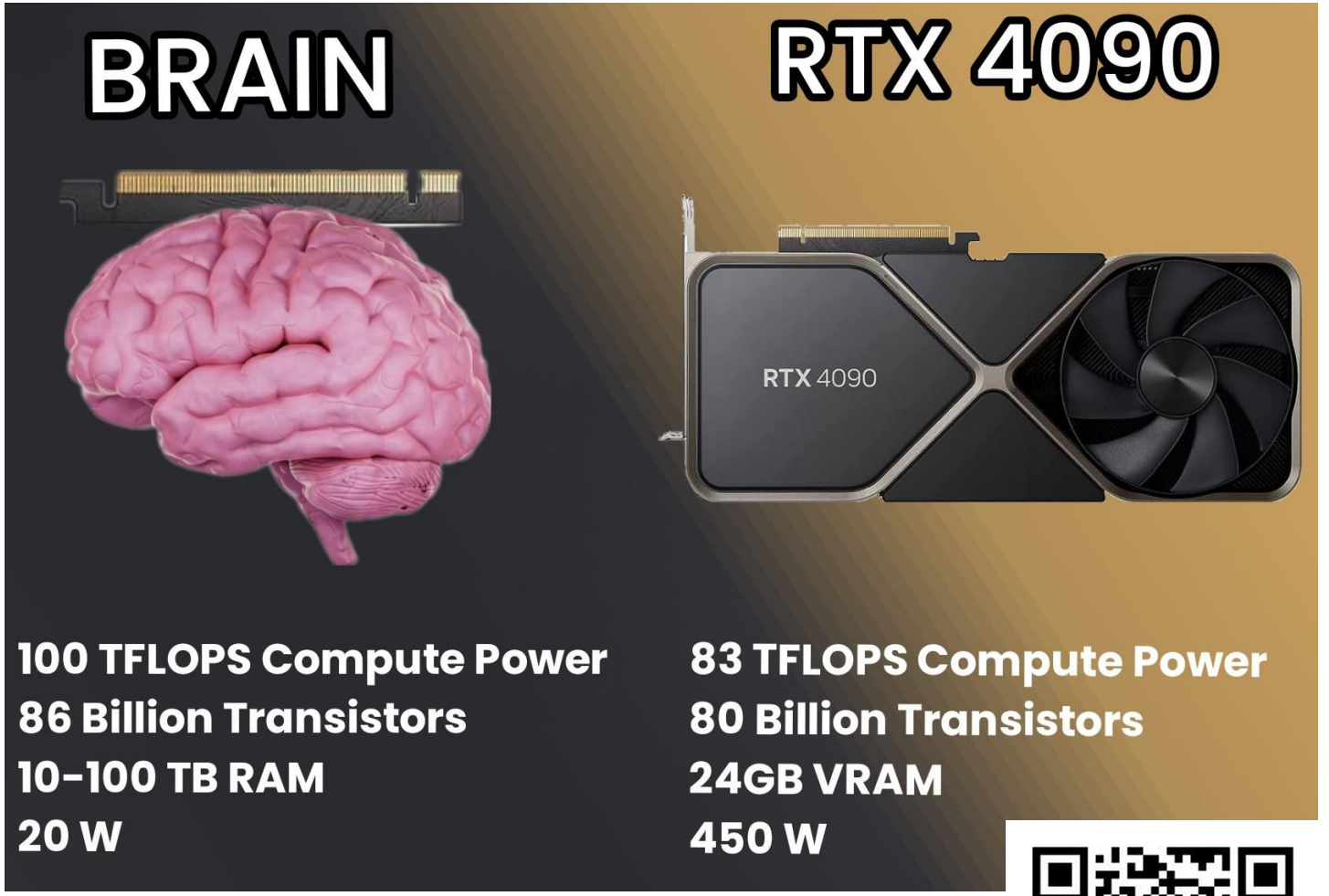
Dense Retrieval for Language Models

Soumen Chakrabarti

Burning 20 watts between our ears

- We cannot afford to…
- …read Wikipedia end-to-end
- …incorporate WikiData into our neurons
- …hallucinate (much)
- …model universe as only token sequences
- …mix instructions and data



Retrieval in the age of generative AI

Home 〉 News 〉 Large Language Models pose risk to science with false answers, says Oxford study

Large Language Models pose risk to science with false answers, says Oxford study

ARTIFICIAL INTELLIGENCE | RESEARCH | SOCIAL SCIENCES

Large Language Models (LLMs) pose a direct threat to science, because of so-called 'hallucinations' (untruthful responses), and should be restricted to protect scientific truth, says

TECH · AI

Google CEO Sundar Pichai says 'hallucination problems' still plague A.I. tech and he doesn't know why

BY WILL DANIEL

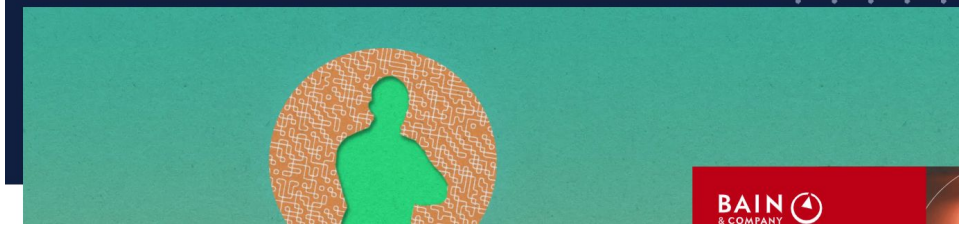
ARTIFICIAL INTELLIGENCE

Why Big Tech's bet on AI assistants is so risky

Tech companies have not solved some of the persistent problems with AI language models.

By Melissa Heikkilä

October 3, 2023



(h/t Rudra Murthy)

Retrieval from trusted sources remains a cornerstone of hallucination mitigation strategies

Brief history of retrieval

Discrete word-based ‘lexical’ indices based on inverted lists, super optimized for billions of documents, but suffering from “lexical gap”

‘Semantic’ indices built upon word (and audio, image, video, etc.) embeddings, based on locality sensitive hashing, vector databases, hierarchical search networks

What to retrieve

- Passages
  - Increase confidence in answer
  - Combine complementary info
  - Political inclination of Obama's grandparents
- (Knowledge) graphs
  - A step toward canonical entities and relations
  - Single node or edge has very limited info
  - Number of countries with more rivers than Brazil
- Tables
  - Textual tables embedded in documents
  - Relational tables with precise schema
  - Toward semantic interpretation
- Images, videos, brain impulses, …

Why dig up the ancient past?

- Historical perspective
  - Word embeddings were invented as early as 1993
  - First dense retrieval in 1995, then 1999
  - word2vec is a form of SVD
- Must understand sparse index issues to understand dense retrieval
- Back to the future!
  - sightseeing in Hamburg
  - High storage and processing overheads of dense retrieval
  - Use dense representation with sparse data structures?
- Return to sparse indices or hybrid solutions

- sightseeing in Hamburg
- places to visit around Hamburg
- tourist attractions near Hamburg
- tours not to miss when staying in Hamburg
- historic destinations in Hamburg
- tourist spots in Frankfurt

Sparse / lexical retrieval

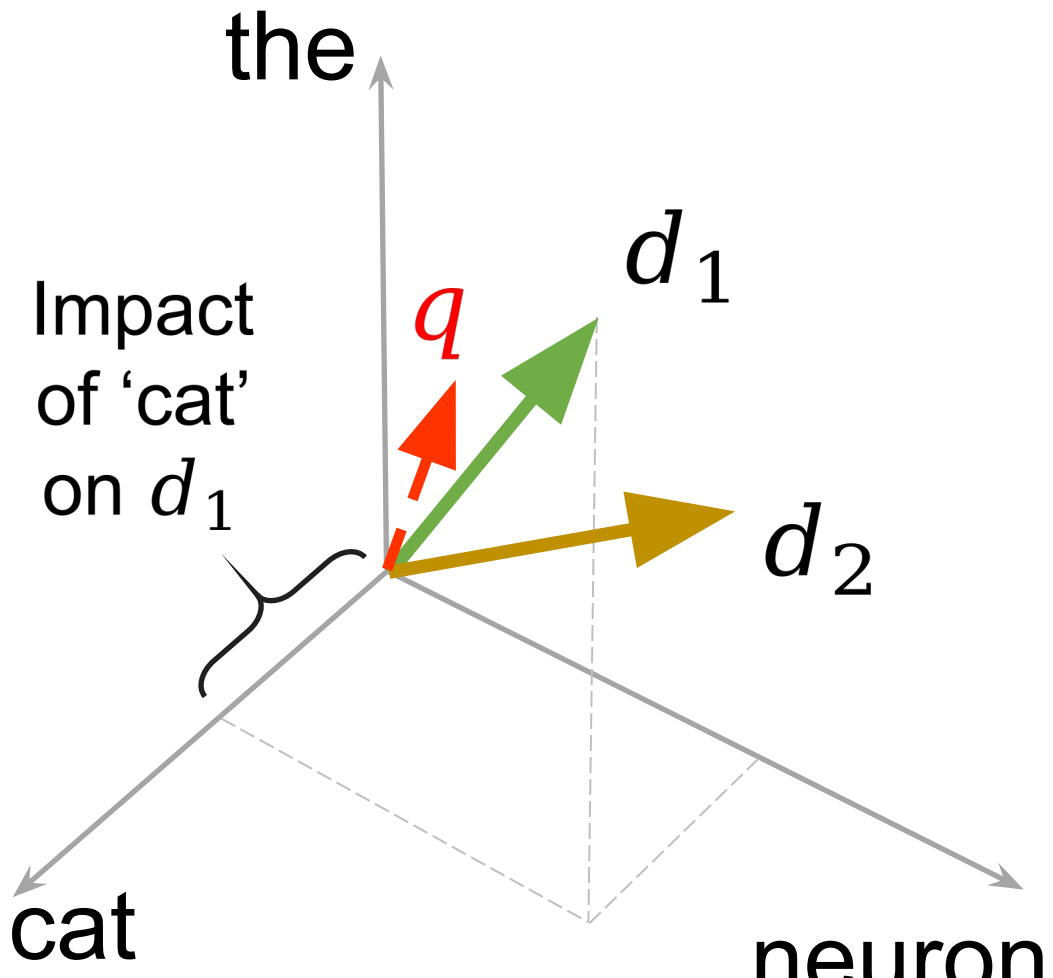
- Workhorse of search engines until ~2013
  - Can handle corpus with billions of documents (N)
- Document modeled as bag of words
  - Bag = multiset, take multiple occurrences into account
  - Corpus vocabulary of size $W \approx 100000$
  - But each document has fewer than 500 words
  - Hence, ‘sparse’
- Each word has some impact on the document
  - TF: how many times it occurs
  - IDF: how rare is the word in the whole corpus

TF, IDF, impact

- Documents numbered $n \in \{1, \cdots, N\} = [N]$
- Words numbered $w \in \{1, \cdots, W\} = [W]$
- TF represents the number of times $\#(n, w)$ a word w occurs in a document n
  - Occurring 20 times not 20 times more significant than occurring once
  - $\log(1 + \log(1 + \#(n, w)))$ or such squashing functions
- IDF represents the rareness of w in the whole corpus
  - 'leopard' has large IDF, 'the' has small IDF
  - Let w occur at least once in $\#(w)$ docs out of N
  - $\log\left(1 + {}^{N}/_{\#(w)}\right)$ commonly used
- Multiply TF and IDF to get impact of w on doc n

TF-IDF vector space document representation

- $W = 100000$ axes or dimensions
- One axis for each word
- Role of IDF
  - Rare word → magnify axis scale
  - Frequent word → shrink
- Query is another (short) doc
  - Overload q, d_n to mean both texts and their vectors $\vec{q}, \vec{d}_n$, each with W dims
- Similarity $s(q, d_n)$ = dot product (or cosine?)
  - $s(q, d_n) = q \bullet d_n = \sum_{w=1}^{W} q[w] d_n[w]$
  - Only shared words contribute

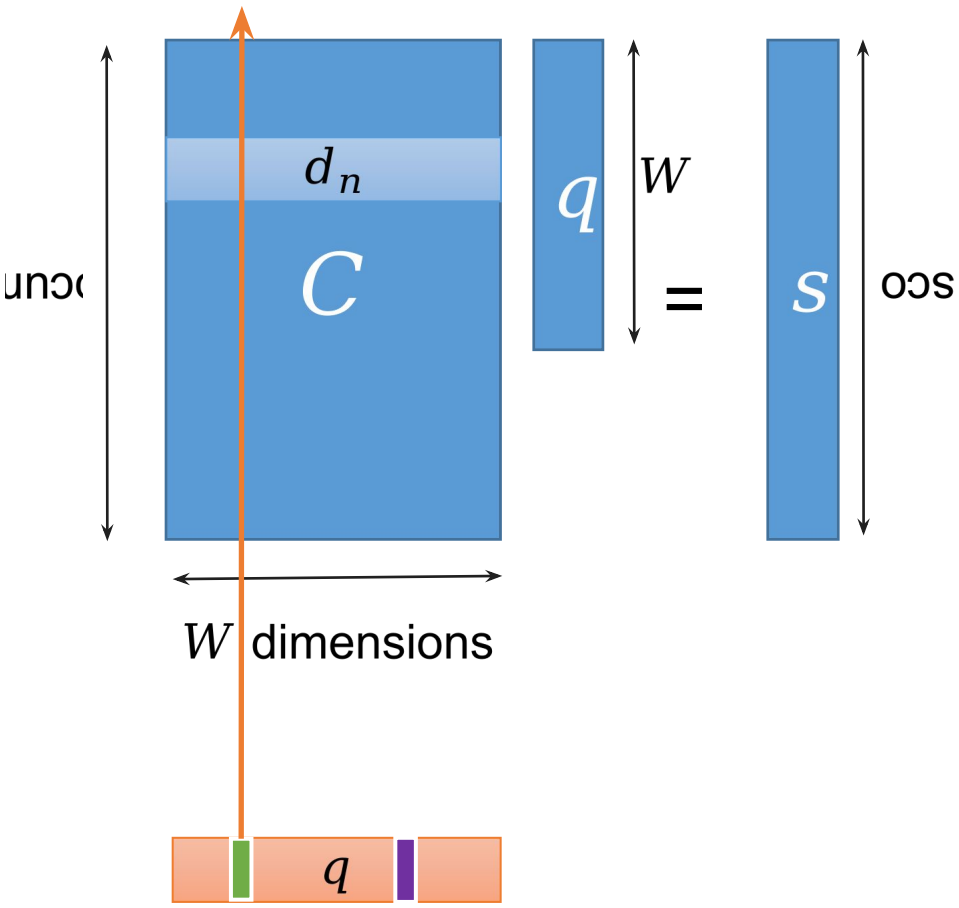


Dot product and cosine similarity

- Norm or length of a vector $\|q\| = \sqrt{\sum_w q[w]^2}$
- Cosine of the angle between vectors q, d_n
  - $\cos(q, d_n) = \frac{q \bullet d_n}{\|q\| \, \|d_n\|}$
- Alternatively, first normalize the lengths of q, d_n then compute $q \bullet d_n$
- Cosine similarity cares only about the directions of the vectors and not their absolute lengths
  - Replicating a doc 50 times over has no effect
- Collect these length-normalized TFIDF doc vectors into a $N \times W$ matrix
  - Losing word position info, can restore with some more storage cost

Document-word matrix C and scoring

- C is very sparse, so is q
  - If rows are sparse, so are columns
  - All matrix elements are non-negative
  - This will change with neural encoders
- Would never store them this way, this is just for illustration
- Scoring all d_n together amounts to matrix vector multiplication
  - Look up only those columns w of C where $q[w] > 0$
- Document are presented in row-major order
- During query, we want to access only the non-zero elements in a few columns of C



$$\sum_{w \in [W], q[w]>0} q[w] d_n[w]$$

From lexical to dense text representation

- Contextual text encoder (typ. transformer based)

[CLS] He swam near the bank because the current was swift.

‘Encoder’

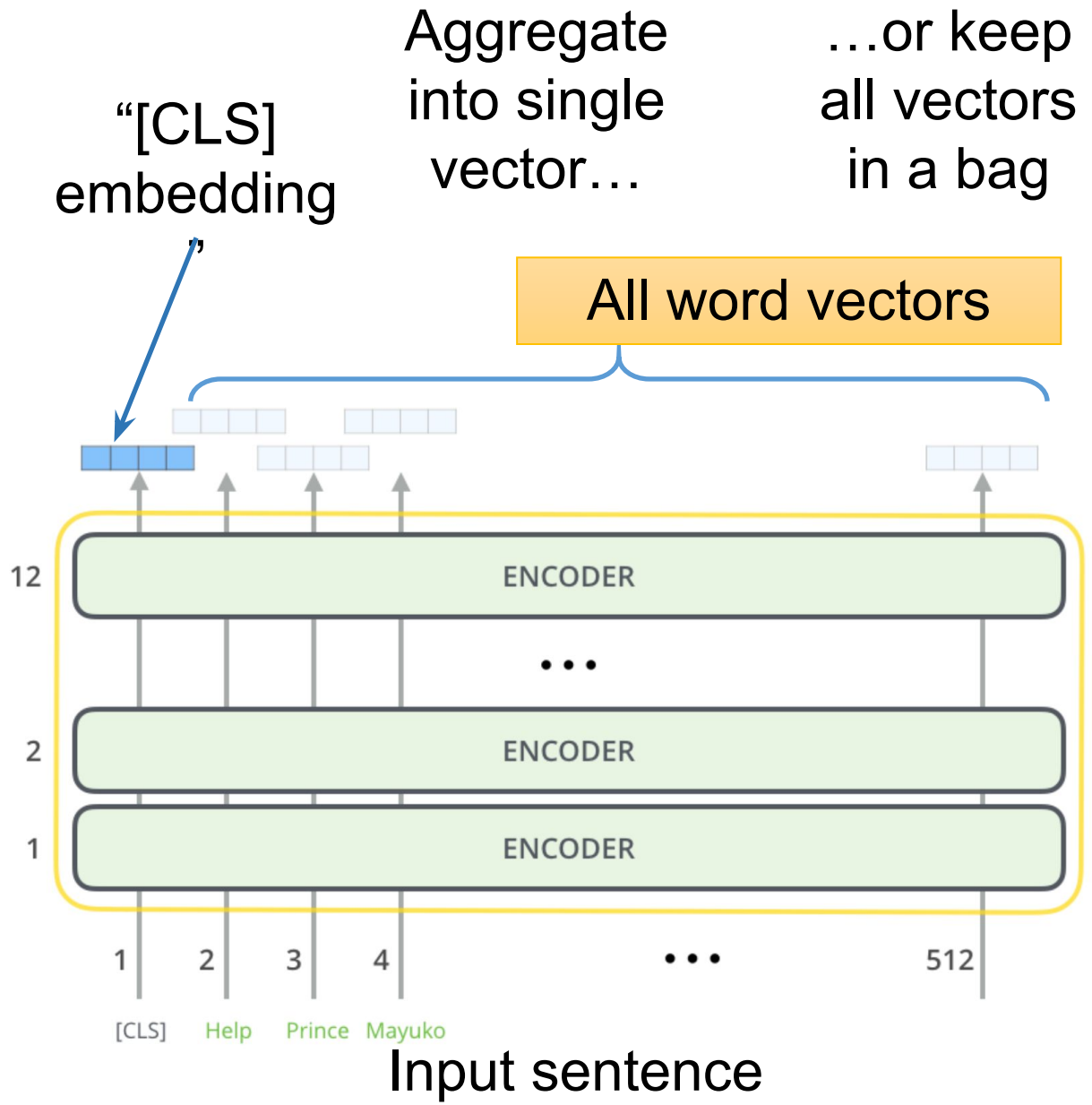
[CLS] His bank offered a low interest on current accounts.

‘Encoder’

Want the two output vectors for ‘bank’ to be quite different because they mean different things

Pooled and bagged text representations

- A transformer encoder outputs a "[CLS] vector" representing the whole passage
- And one vector per token/word
- First we will discuss retrieval using single vector per passage
- Later we will discuss multi-vector representations

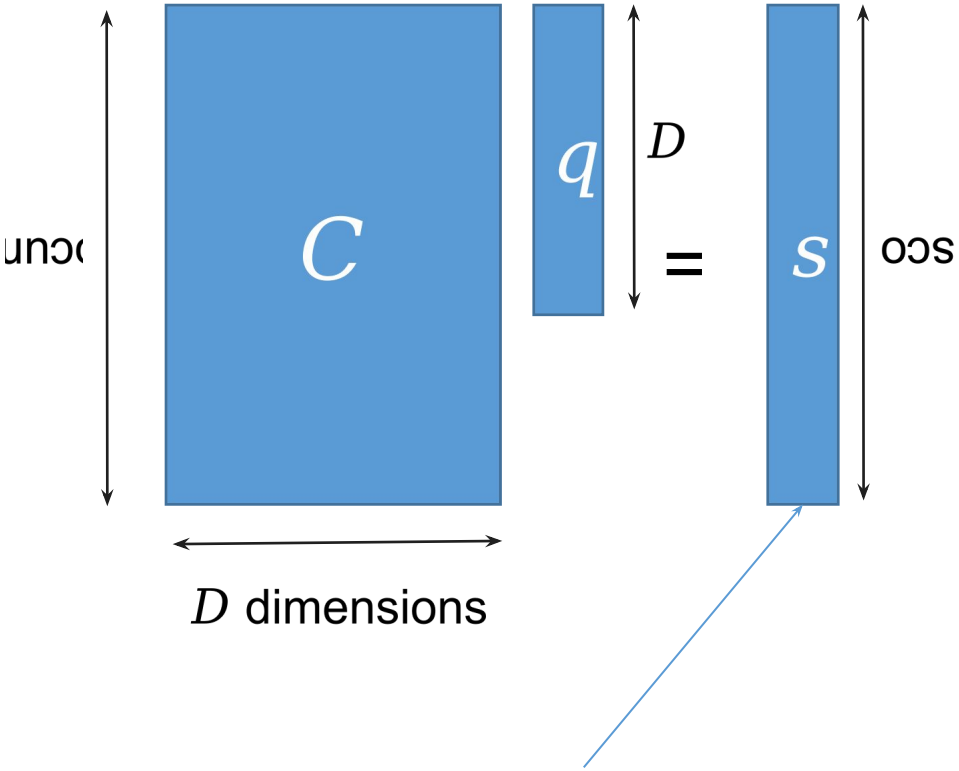


A first-cut setup for dense retrieval

- $\mathbb{R}^D$ is the Euclidean space with D dimensions
  - $q \in \mathbb{R}^D$ is a query vector; $x_n \in \mathbb{R}^D$ is a document vector
- Score is dot or inner product $s(q, x_n) = q \bullet x_n$
  - If we want length normalization, then use $s(q, x_n) = \frac{q \bullet x_n}{\|q\| \, \|x_n\|}$, where $\|x\| = \sqrt{x_1^2 + \cdots + x_{1D}^2}$
  - Equivalently, pre-scale vectors to have unit length
  - Cosine of the angle between the vector, written $\cos(q, x_n)$
- Goal: find K docs most similar to query: $\underset{n \in [N]}{\text{argmax}^{(K)}} \, s(q, x_n)$
- I.e., scoring same as before, vectors created differently

Brute force ranking

- Dense $N \times D$ corpus matrix
- Matrix-vector multiply with query vector to get scores
- Then pick top K elements
- Takes $O(N D)$ time
- Challenge: spend time that scales very slowly (sub-linearly) with N
- Scaling with D, K tolerable



Pick documents (row indexes) with top K scores

Idea: cluster and bucket

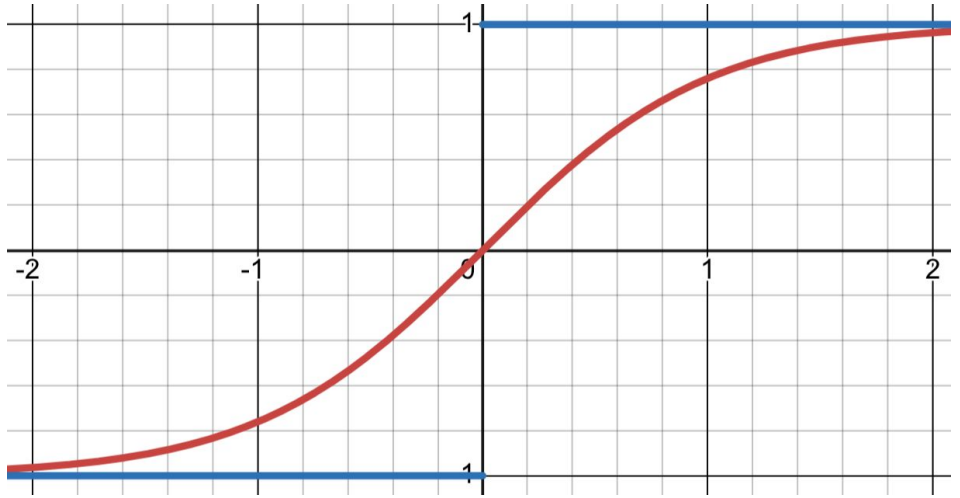
- Divide and conquer
- Initialize C cluster centers $\overline{x}^c$, where $c = 1, \cdots, C$
- Repeat (C-means clustering algorithm)
  - Move each x_n to cluster with largest $s(\overline{x}^c, x_n)$
  - Recompute cluster centers
- At query time, given query q
  - Find cluster c with largest $s(\overline{x}^c, q)$
  - Compute "true scores" $s(q, x_n)$ only for x_ns that belong to cluster c ("the bucket of c")
- If we pick $C \approx \sqrt{N}$
  - Find closest cluster in $\sqrt{N}$ time, then score all docs in it in $\sqrt{N}$ time

Query time reduction

- Assuming buckets are balanced, each bucket has N/C docs
- When query is assigned its bucket, those N/C docs need to be fully scored
- Usually, let $C = N^{\delta}$, so that typical query time is like $N/N^{\delta} = N^{1-\delta}$
- Should also pay attention to distribution of query workload over buckets
  - Unpopular buckets can afford to be bigger
  - Unbalanced bucket occupancy can reduce average query latency

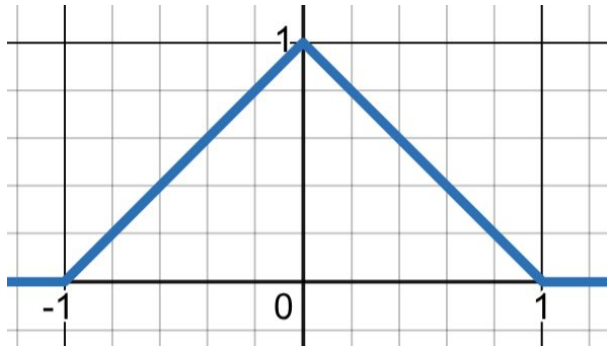
Neural ‘clustering’

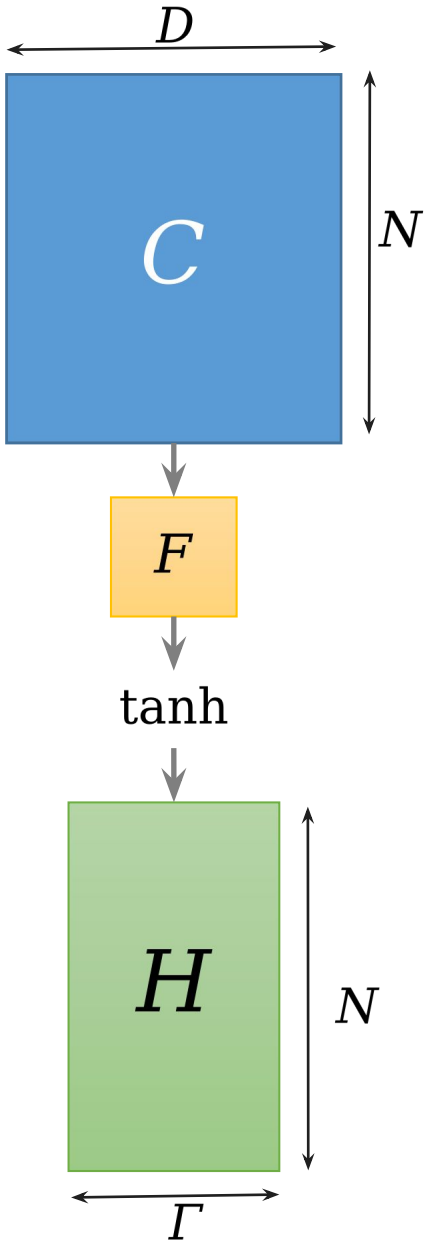
- C clusters means $\Gamma = \log C$ bits to address a cluster
- Design a feed forward network
  - Input can be x_n or query q
  - Output is intended to be D bits
- Approximate bits with $\tanh\left(F(x_n)\right)$
  - $F: \mathbb{R}^D \rightarrow \mathbb{R}^\Gamma$ is some simple network
  - $\tanh(\bullet) \in [-1,1]^\Gamma$
- F will adjust to corpus (and query) distributions
  - Once F is trained, discretize output as $\text{sign}\left(F(x_n)\right)$
  - (Note, F is not meant as a classification device)



Three loss objectives to train F

- Bit balance: each output element should be $+1$ for approximately half of the N instances and -1 for the other half
  - For each bit position γ, $\left|\sum_n \tanh\left(F(x_n)[\gamma]\right)\right|$
- Uncorrelated: two output bits should not be $+1$ for the same instances and -1 for their complement $\left|\sum_n F(x_n)[\gamma]F(x_n)[\gamma']\right| = 0$ for $\gamma \neq \gamma'$
- No sitting on the fence: push outputs toward ±1
  - $\sum_{n,\gamma}\left(1 - \left|\tanh\left(F(x_n)[\gamma]\right)\right|\right)$





Further informed by query workload

- Docs never touched by query workload can go anywhere
  - As long as they do not overpopulate frequently touched buckets
- For each q, get $N/2^{\gamma}$ docs $\{x_{\dagger}\}$ with best scores
  - Use scoring function (dot, cos, L2, etc.) on whole corpus
  - Or use relevant docs if known
- Want these best-true-score docs in query bucket, i.e. $\text{sign}(F(q)) = \text{sign}(F(x_{\dagger}))$
  - Suggests loss surrogate $\sum_{x_{\dagger}} \sum_{\gamma} (1 - F(q)[\gamma]\ F(x_{\dagger})[\gamma])$
  - (Other forms possible, may behave better)
- F is trained with these four loss components

Recap

- In ‘raw’ TFIDF space, corpus matrix C is $N \times W$
  - $W \approx 100000$
  - C is very sparse; each row has max ~ 500 nonzeros
  - Each row represents one document
- A neural encoder transforms C into $N \times D$
  - BERT: $D = 768$; Llama17B: $D = 7680$
  - These D dimensions are all non-zeros, can be negative
- Hashing network F further reduces to $N \times \Gamma$
  - $C = 2^{\Gamma}$ is the desired number of clusters
  - Should scale up gradually with corpus size N

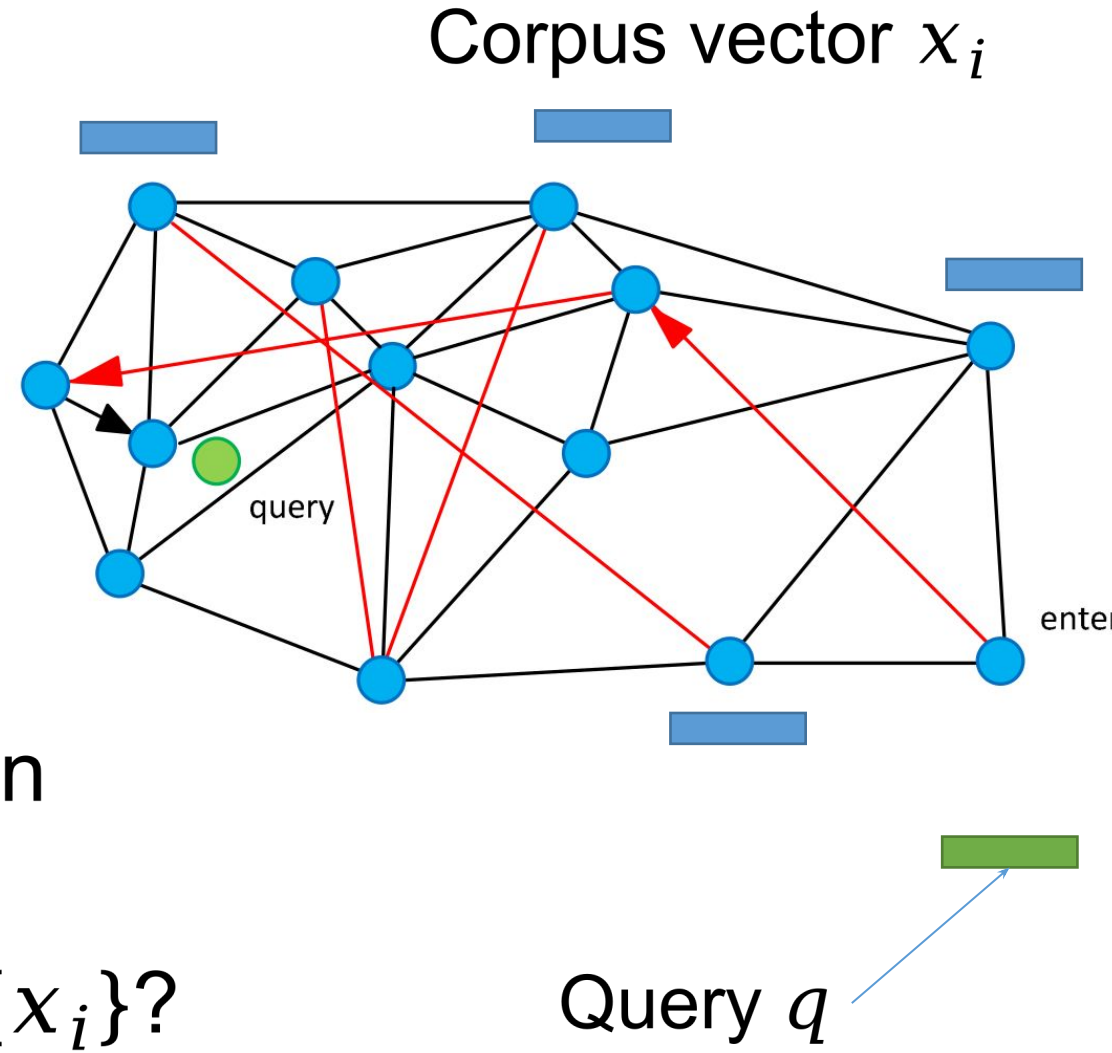
Navigating in a social network (Milgram)

- Source cities Omaha, Nebraska, and Wichita, Kansas
- Destination city Boston, Massachusetts
- Letters with instructions sent to random people in source cities
- Letters to be sent to some person in destination city
- Source person can send to someone in their social network if they don't know target directly
- "Six degrees of separation"

Name, age, gender, profession, …

Navigating a “graph of vectors”

- Each node i has associated vector x_i
- Start at node designated ‘enter’
  - Can use multiple entry nodes
- In each step, move from current node i to a ‘friend’ (neighbor) j if $d(q, x_j) < d(q, x_i)$
  - Move to best neighbor
- Stop when no neighbor is better than current node
- How is the graph built from corpus $\{x_i\}$?



Graph construction

- Start with empty graph
- For each vector x_i to be inserted
  - Consider x_i as a query
  - Navigate to some number of near neighbors $\{x_j\}$
    - May include neighbors of neighbors
  - Sort them in increasing $d(x_i, x_j)$ order
  - Insert a node corresponding to x_i
  - Link to nearest K nodes x_j

What we have achieved thus far

- Sparse index
  - Query, docs are vectors
  - As many dimensions as discrete, interpretable words in corpus vocabulary
  - Very sparse vectors
  - Relevance score = dot product
  - Lucene, Elastic Search, SOLR
- Dense index with single vector per query and doc
  - Fewer dimensions, but dense (all non-zeros)
  - Dimensions not interpretable
  - Relevance score still dot product (“late interaction”)
  - DiskANN, ScANN, FAISS, Milvus

Discontent with single vectors

"You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!" — Ray Mooney (2014)

"The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster." —Geoffrey Hinton (2014)

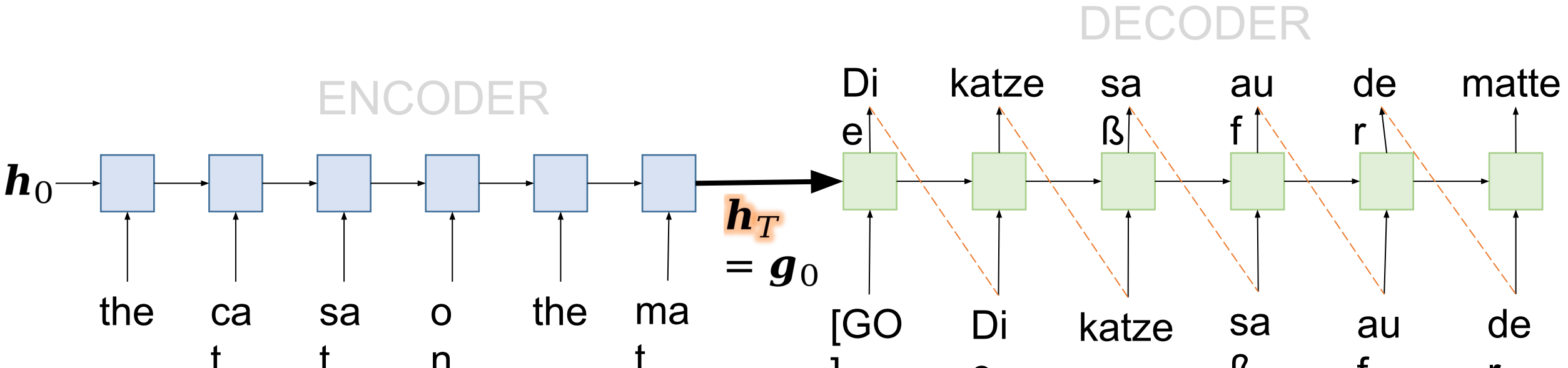
Google DeepMind

On the Theoretical Limitations of Embedding-Based Retrieval

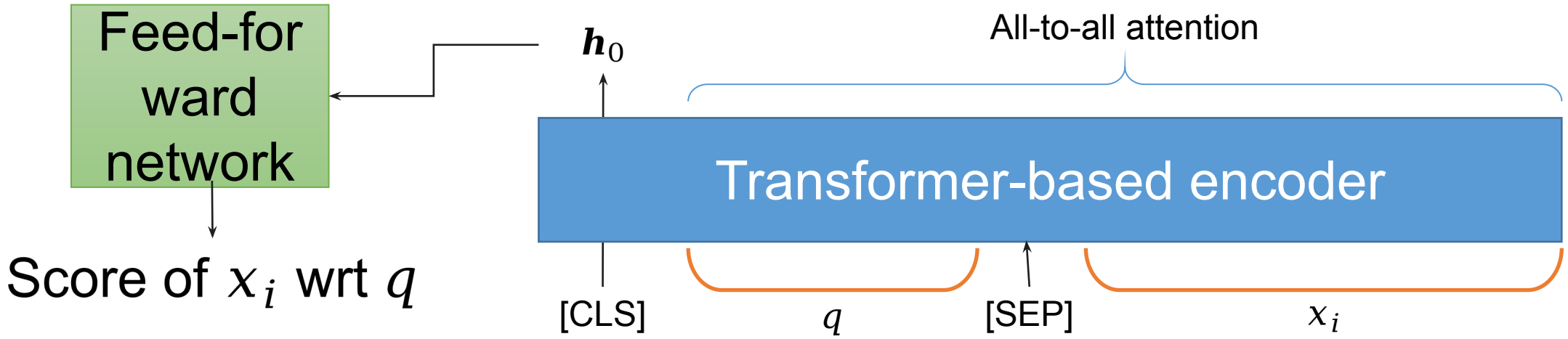
Orion Weller[*,1,2], **Michael Boratko**[1], **Iftekhar Naim**[1] **and Jinhyuk Lee**[1]
[1]Google DeepMind, [2]Johns Hopkins University

Single vector limitation

- Already seen RNN encoder-decoders
- In principle, $\boldsymbol{h}_T$ is a complete digest of input
- In practice, narrow information bottleneck
- Decoder attention on all encoder states helps a lot
- Motivates bag of vectors for retrieval as well

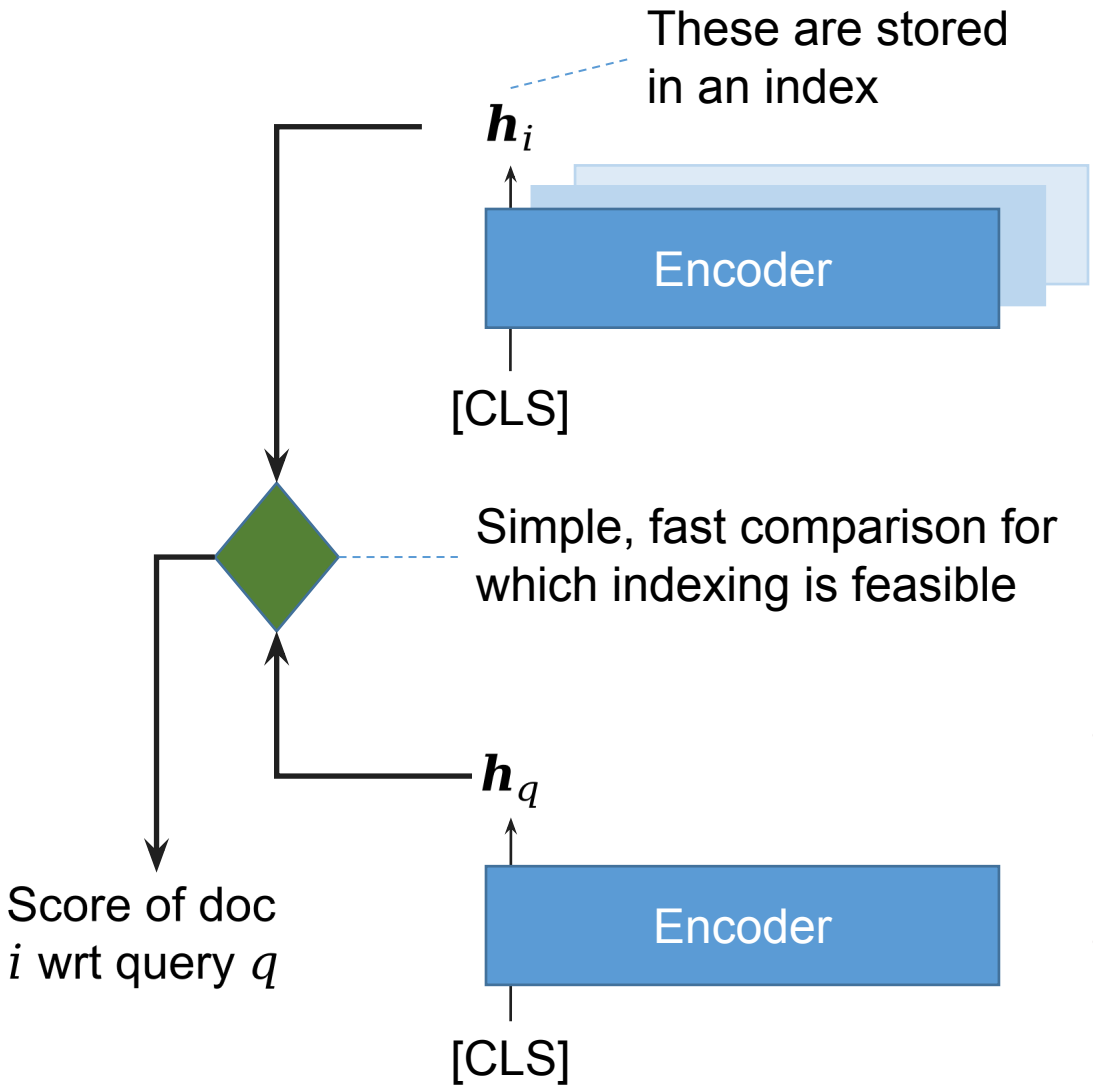


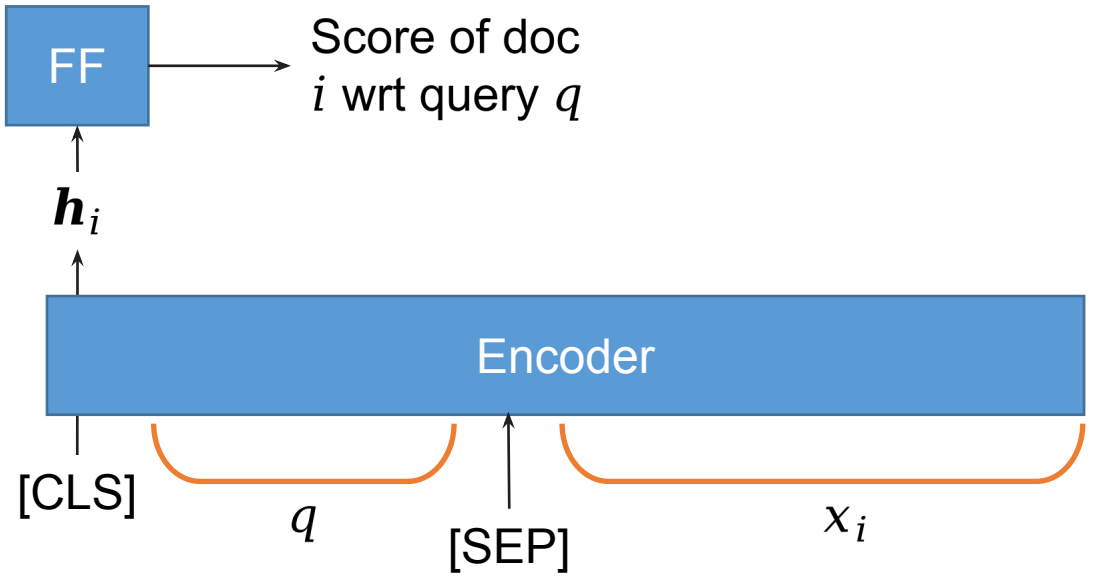
All-to-all cross (early) interaction



- For each candidate document x_i
  - Concat query q with x_i, encode using transformer
  - Allows all-to-all attention between q and x_i
  - Use [CLS] embedding $\boldsymbol{h}_0$ to regress to a score for x_i
- Every candidate must be scored, takes $\Omega(N)$ time
- However, generally gives best retrieval quality

Early vs late interaction: high level sketch





- Compromise to get part of the benefit of early interaction?
- Need embedding of q, x_i to be independent of x_i, q
- But allow granular interaction

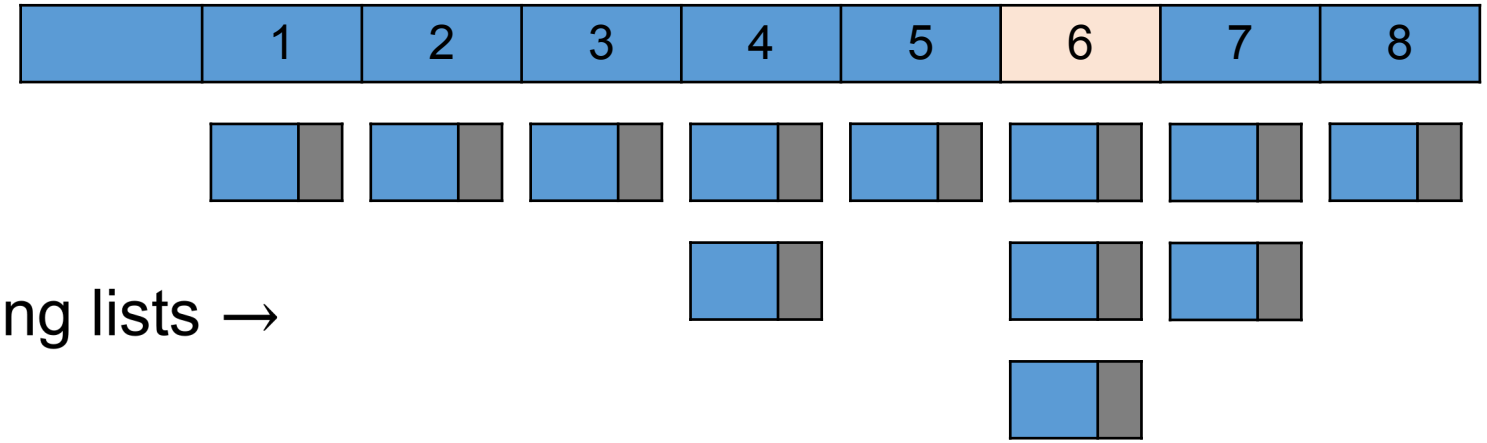
Compromise between early and late (ColBERT)

- Query word vectors $q_1, q_2, \cdots, q_M$ (M words)
- Document has T word vectors $x_{i,1}, x_{i,2}, \cdots, x_{i,T}$
- For query word q_m, best 'partner' in the doc gives 'coverage' $\max_{t \in [T]} s(q_m, x_{i,t})$
  - Here $s(q_m, x_{i,t})$ can be dot, or cosine similarity
- Want all query words to be 'covered' by document
- $s(q, x_i) = \frac{1}{M} \sum_{m \in [M]} \max_{t \in [T]} s(q_m, x_{i,t})$
  - Called "max-sum" or "chamfer" score
- Can top documents under $s(q, x_i)$ score be quickly retrieved via suitable index?

An indexing hurdle

	text
1	ape
2	bat
3	bee
4	cat
5	cow
6	dog
7	elk
8	fox

← Word dictionary



- Docs d_1, d_2, d_4 all contain exact word $w = 6$ (dog)
- Now consider these two docs
  - I swam along the river bank staying close to the boat.
  - The fisherman sat at the water's bank, patiently waiting for a bite.
- "bank" has the exact same meaning in both docs
- But they have different contextual embeddings output by transformer

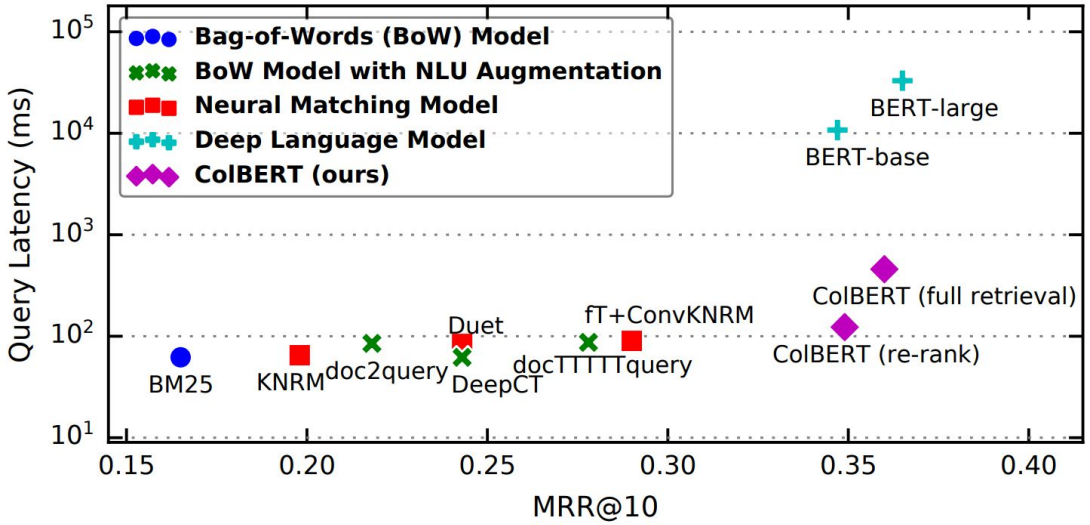
Contents of clustered buckets

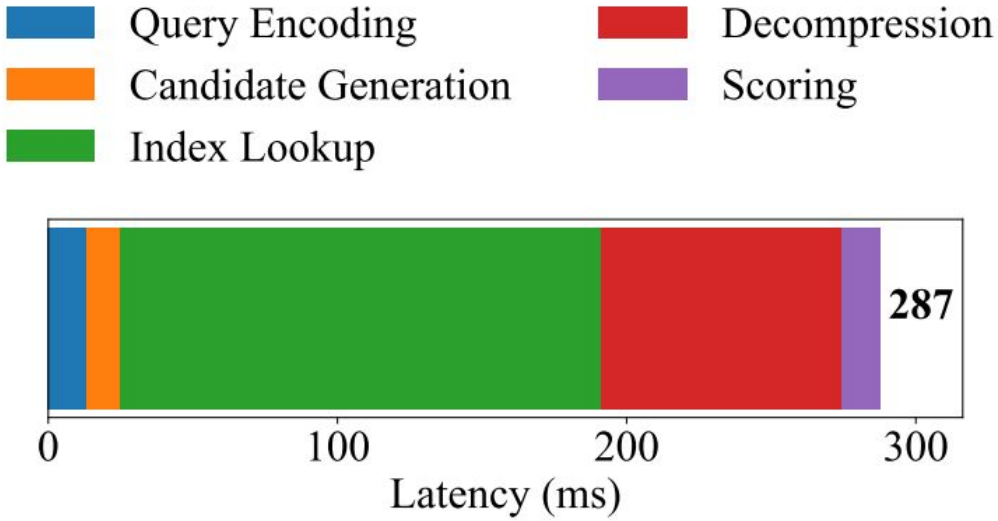
- In a sparse index, a posting list
  - Was accessed using a discrete word
  - Contained (doc ID, word impact) pairs
- In a dense clustered vector index
  - Query key will be one vector q_m
  - Closest to one or few cluster centers $\overline{x}^c$
  - Each cluster bucket holds (doc ID i, contextual word vector $x_{i,t}$
  - $x_{i,t}$ may be close to $\overline{x}^c$ but not identical
  - Store (doc ID, low resolution $x_{i,t} - \overline{x}^c$) in bucket

	HSBC	ATM	is	located	on	the	bank	of	river	Yamuna
where										
can										
I										
get										
cash										
near										
Yamuna										

Query execution at a high level

- Candidate docs = ∅
- For each query word vector
  - Probe ANN index
  - Augment candidates
- Initialize score accumulators
- For each candidate
  - Decompress word vectors
  - Compute true score
  - Accumulate scores
- Report top docs





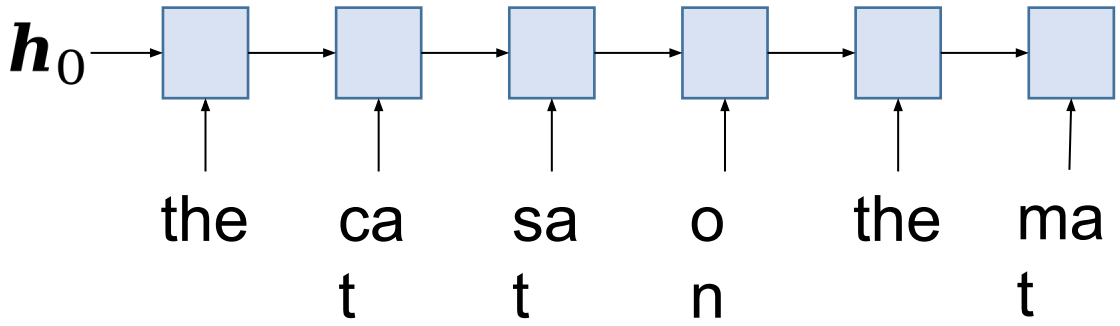
(a) Vanilla ColBERTv2 (nprobe=4, ncandidates=2^{16}).

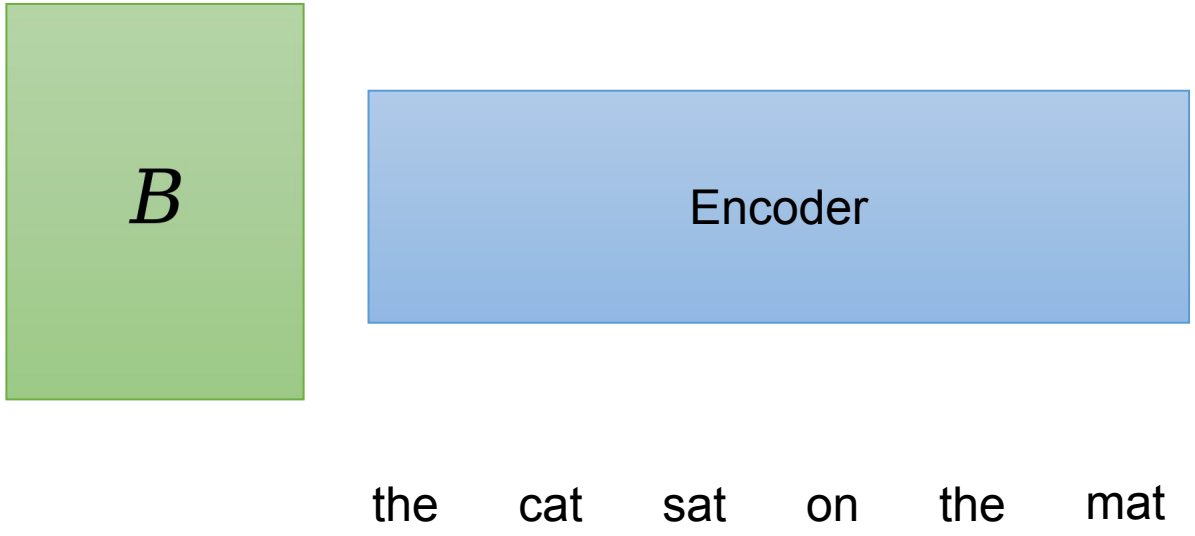
Back to the future (dense → sparse)

- ColBERT: implementation is nontrivial, considerable execution overhead
  - Contrast with super-optimized sparse inverted lists
- Idea: Use dense representation to expand doc in discrete word space
  - Doc already had ‘panther’ ⇒ add ‘wildlife’
  - Don’t assign word impact blindly as in classic indices
  - Learn word impact using relevance judgments
- For each training query q, collect some relevant ‘good’ docs $D_{q\oplus}$ and irrelevant ‘bad’ docs $D_{q\ominus}$

Base embedding matrix B

- Word strings not directly input to encoder
- Each word indexes a row in B
- This vector injected into encoder as $\boldsymbol{h}_t^0$
- Output vector $\boldsymbol{h}_t^L$ compared with each row of B to sample next word
- We will use B to expand docs with padded words





SPLADE setup

- Text $x = (x_t : t \in [T])$ can be query or passage
- Pass x through encoder to get output contextual embeddings $(h_t : t \in [T])$
- Compare h_t with every row of base embeddings B
  - If h_t is similar to row $B[w, :]$, consider expanding doc with word indexed w
  - E.g. text: the jaguar snarled before pouncing
  - Possible ws correspond to wildlife, predator
- Next we will design the impact of expanded words

Word impacts in SPLADE

- Set $z_{t,w} = \text{transform}(h_t) \bullet B[w,:] + o_w$
  - transform = linear layer, GELU activation, layer norm
  - o_w is a bias term specific to word w
  - Vote of text at position t on vocabulary word w
- Aggregate $z_{t,w}$ over all text tokens (indexed t) to get overall vote of text on vocabulary word w (impact)

$$z_w = \sum_{t \in [T]} \log\left(1 + \text{ReLU}(z_{t,w})\right)$$

- In v2, modified to

$$z_w = \max_{t \in [T]} \log\left(1 + \text{ReLU}(z_{t,w})\right) \geq 0$$

- Bias o_j chosen/trained so most $z_w = 0$ (encourage sparsity)
- Text x represented as vector $(z_w : w \in [W])$ — back in vector space
  - Recall W is the interpretable word vocabulary
- Query and doc texts compared via dot product

Training retrievers (ranking loss)

- Fine-tune encoder for each retrieval task
  - 'Potential' in general corpus similar to 'promise' of a person's capability; 'charge' = 'accusation'
  - How about https://physics.stackexchange.com/
- Collect relevance judgments
  - Each query q comes with one/few good doc/s $d_{q\oplus}$
  - Sample in-batch 'silver' / 'easy' negatives $D_{q\boxminus}$
  - Mine 'hard' negative/s $d_{q\ominus}$
- $$\mathcal{L}_{\text{rank}} = \frac{\exp\left(s(q, d_{q\oplus})\right)}{\exp\left(s(q, d_{q\oplus})\right) + \exp\left(s(q, d_{q\ominus})\right) + \sum_{d_{q\boxminus} \in D_{q\boxminus}} \exp\left(s(q, d_{q\boxminus})\right)}$$

Training retrievers (non-sparsity loss)

- Recall doc x_n encoded to $z_n = (z_{n,w} : w \in W)$
  - Corpus has N documents
  - Non-negative real matrix of size $N \times W$
- We want $z_{n,w} = 0$ for most words w
- $\mathcal{L}_{\text{FLOPS}} = \sum_{w \in [W]} \left(\frac{1}{N} \sum_{n \in [N]} z_{n,w} \right)^2$
- Overall loss is a combination $\mathcal{L}_{\text{rank}} + \clubsuit \mathcal{L}_{\text{FLOPS}}$
- Ranking quality somewhere between sparse and ColBERT, but way faster

Summary and learning outcomes

- Passage retrieval is a core capability needed by LLMs
- For decades, sparse inverted indices were the solution
- Now passages and queries are represented by dense vectors, or bags of vectors
  - Generally superior to sparse representation
- Can benefit from both dense representation and sparse indices
- We have studied the latest techniques of indexing and searching dense corpora
- Such indices are used by LLMs to access passages in RAG setups

“Closed-book” LMs and their discontent

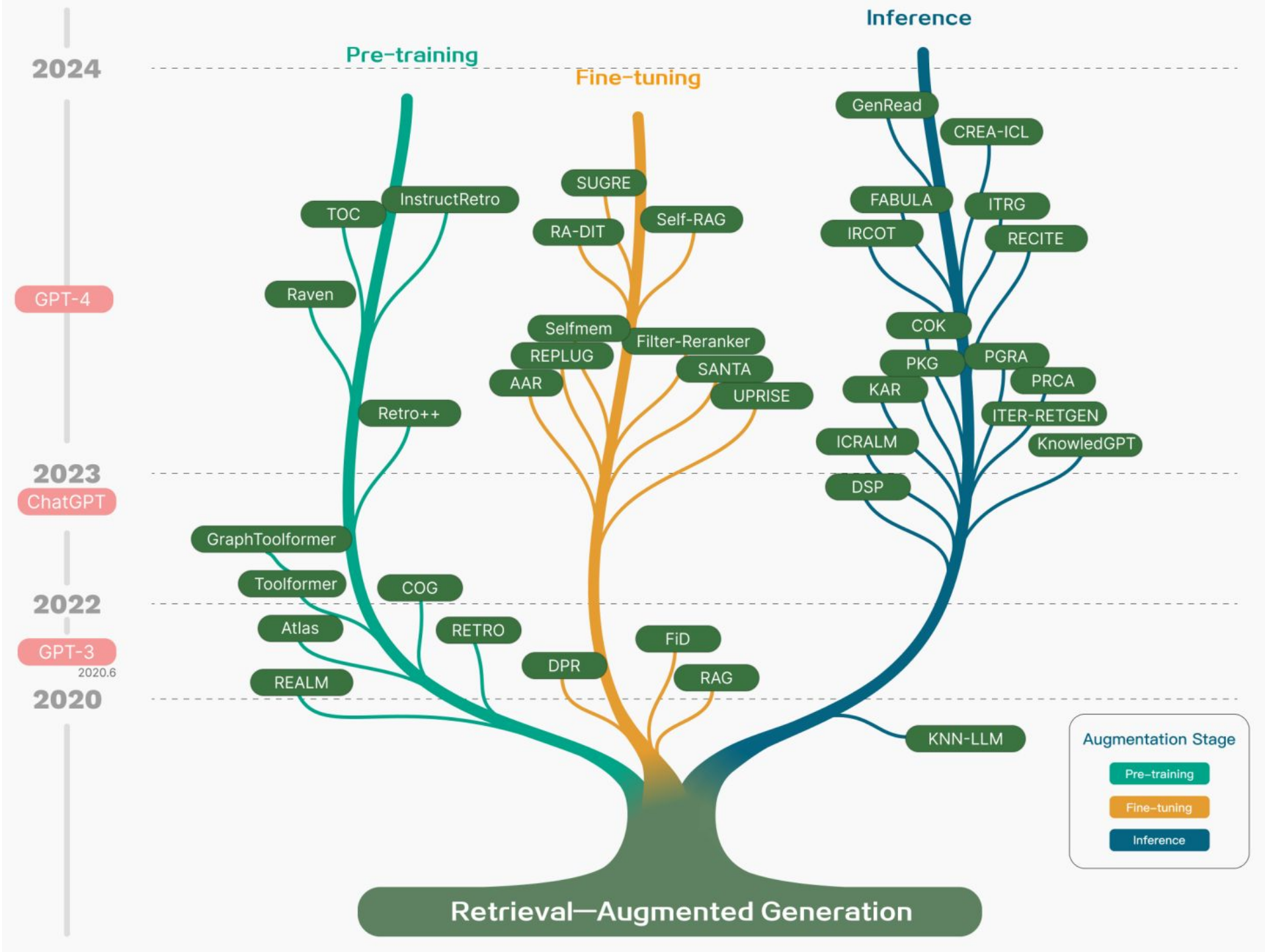
- Early LM paradigm
  - Collect a large corpus, prepare training data
  - Train LM with MLM, next-sentence, instruction, etc. loss
  - “Throw away” the corpus
  - Input = token sequence, output = continuation
- Parameter size of large LMs comparable to size of Wikipedia (LLMs, much larger)
- Should be able to rote-learn the corpus
  - No idea how corpus is stored in LM parameters
  - Interference and hallucination
  - Cannot trace generated output to corpus sources
  - Cannot keep up as corpus changes

“Open-book” LMs

- Keep corpus around even after training LM
  - Say, flat set of passages/sentences = ‘item’
- Input = question = query = ‘prompt’ arrives
  - Encode it using the LM
- In decoder mode
  - Consult the corpus one or more times
    - How often? at what granularity?
  - Incorporate info from top-responding corpus items
  - Make generation sensitive to this info
- Potential benefits
  - Reduce hallucination by biasing output distribution
  - Provide ‘sources’ to ‘explain’ generation
  - Offload rote learning to corpus/index and make LM smaller

Historical perspective wrt search

- (Dense) passage retrieval
  - Preprocessing input: corpus of passages
  - Query-time input: query
  - Output: ranked list of passages
- "Reading comprehension" (SQuAD)
  - System inputs: passage(s) and question; output = answer span in passage(s)
  - "Which scientists played musical instruments?" → Edison, Einstein, Feynman
- Combine the above with a generator:
  - Which scientists played musical instruments?
  - There are many scientists who are also talented musicians or have played musical instruments. Einstein was a skilled violinist. Feynman played the bongo. Brian May, guitarist for the band Queen, is an astrophysicist. Oliver Sacks, a neurologist, played the piano.
  - (Usually collected and paraphrased from diverse documents)



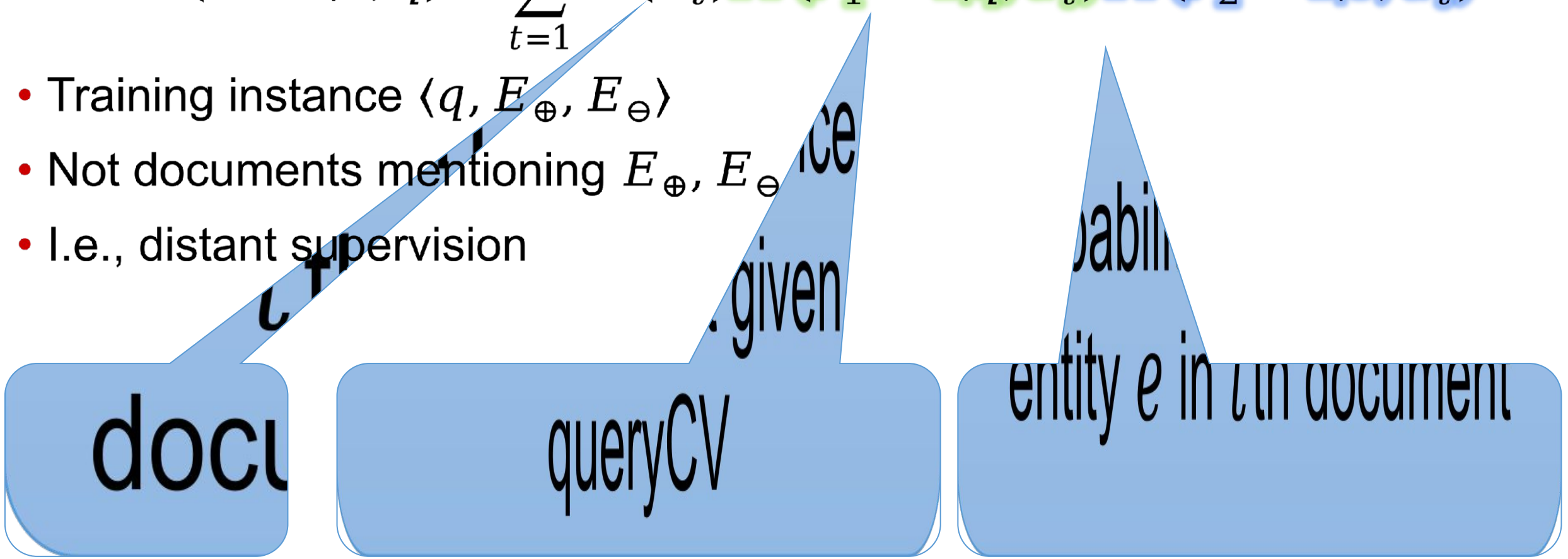
2312.10997.pdf

Entity search paper from 2010

- Entity e, document d, query q, relevance label r
- Equation (6):

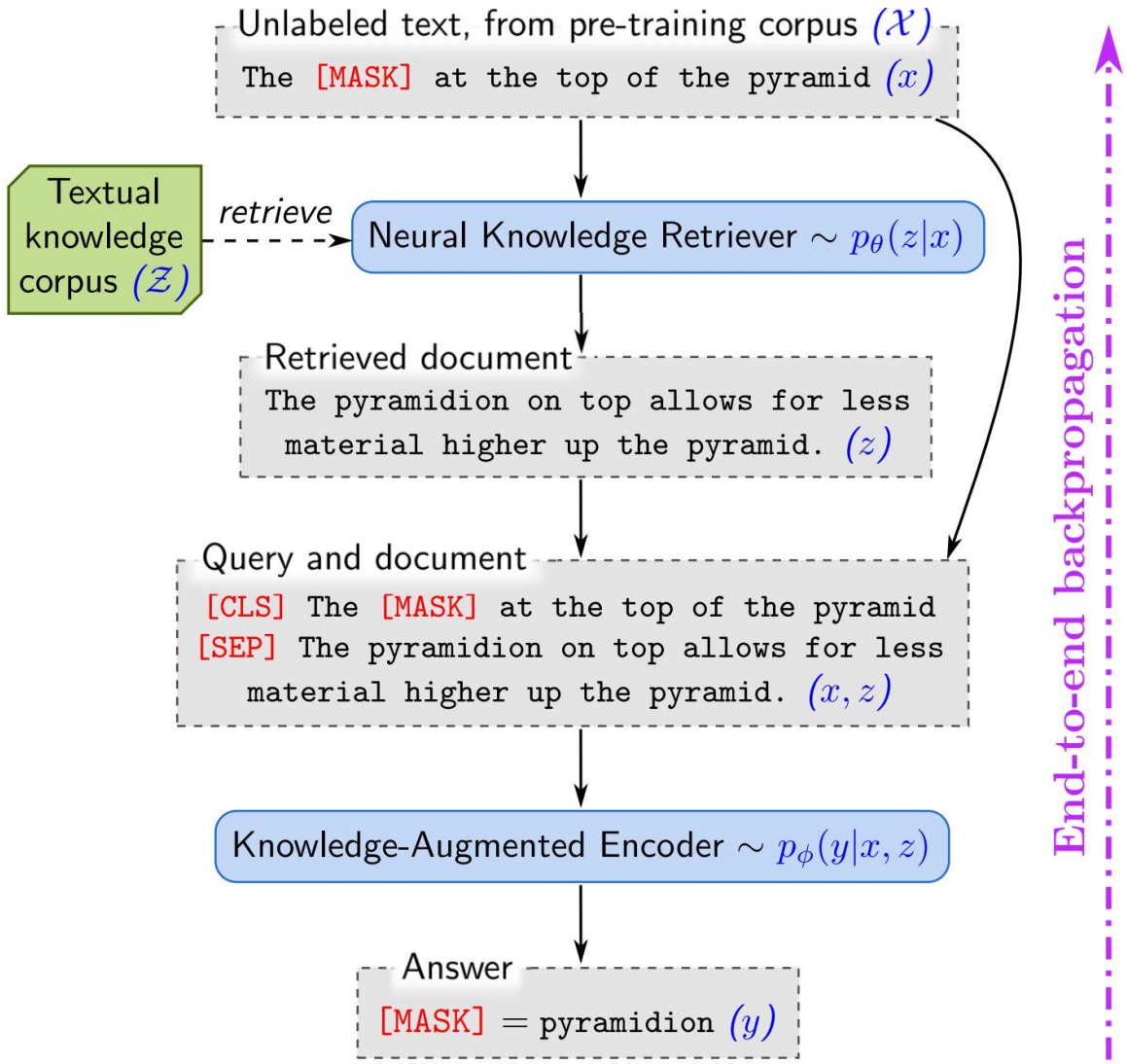
$$\Pr(r = 1|e, q) = \sum_{t=1}^{n} \Pr(d_t)\Pr(r_1 = 1|q, d_t)\Pr(r_2 = 1|e, d_t)$$

- Training instance $\langle q, E_{\oplus}, E_{\ominus} \rangle$
- Not documents mentioning $E_{\oplus}, E_{\ominus}$
- I.e., distant supervision



REALM (2020)

- One document is enough
- $x = \text{query}, \quad y = \text{answer}, \quad z = \text{doc}$
- $\text{join}(x) = [\text{CLS}]x[\text{SEP}]$
- $\text{Embed}_{\text{in}}(x) = \boldsymbol{W}_{\text{in}}\ \text{BERT}_{\text{CLS}}(\text{join}(x))$
- $\text{Embed}_{\text{doc}}(z) = \boldsymbol{W}_{\text{doc}}\ \text{BERT}_{\text{CLS}}(\text{join}(z))$
- $f(x, z) = \text{Embed}_{\text{in}}(x) \cdot \text{Embed}_{\text{doc}}(z)$
  - (Asymmetric late interaction)
- $p(z|x) = \frac{\exp(f(x,z))}{\sum_{z'} \exp(f(x,z'))}$ (softmax)
  - Conceptually, documents compete; they don't "team up" to answer the question
- $p(y|x) = \sum_z p(y|z,x)\ p(z|x)$



Pre-training and fine-tuning

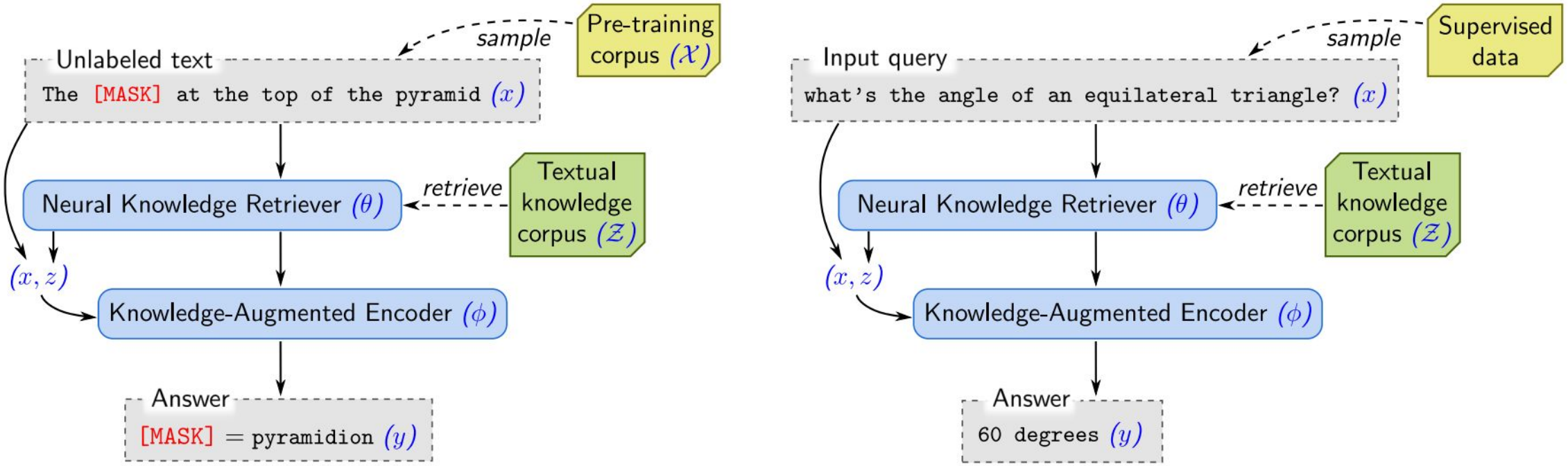


Figure 2. The overall framework of REALM. **Left:** *Unsupervised pre-training.* The knowledge retriever and knowledge-augmented encoder are jointly pre-trained on the unsupervised language modeling task. **Right:** *Supervised fine-tuning.* After the parameters of the retriever (θ) and encoder (ϕ) have been pre-trained, they are then fine-tuned on a task of primary interest, using supervised examples.

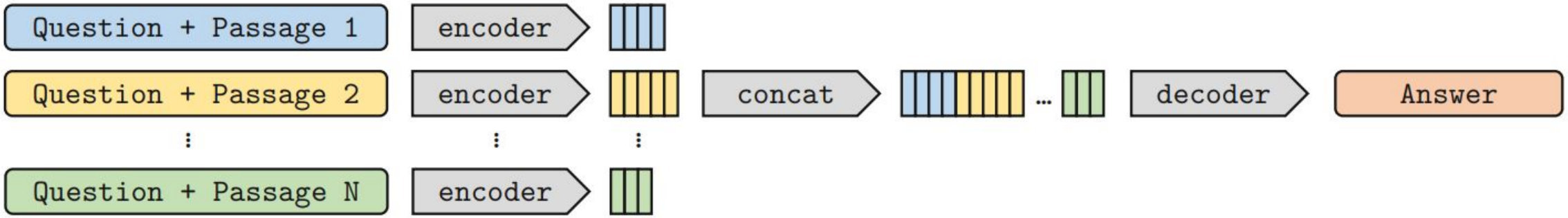
Joint decoding of y

- Independent mask-filling in query $p(y|z,x) = \prod_{j=1}^{J_x} p(y_j|z,x)$
  - J_x is the number of masked positions in the query
- During training…
- $p(y_j|z,x) \propto \exp(\boldsymbol{w}_j \cdot \mathrm{BERT}_{\mathrm{MASK}(j)}(\mathrm{join}(x,z)))$
  - Concat question x and document /passage z
  - Keep position j of question masked
  - Pass through BERT and read off embedding of masked position
  - $\boldsymbol{w}_j$ is a trained (base?) embedding of masked question word
- Implementation details
  - How do they prune the space of docs z?
  - As encoders change, dense index is lazily updated

Design choices

- REALM retrieves exactly once
  - In general, want to retrieve multiple times or multiple pieces of evidence
- How early do evidence items interact with each other (and possibly with input)
- Retrieval granularity
  - While decoding every sentence, word, …?
- What is practical to train and fine-tune
  - Retriever vs. planner / generator

Fusion-in-Decoder (FiD)



- Multiple passages retrieved
  - All in response to question? May not have enough info about later hops
- Passages do not interact with each other until encoding is completed
  - All inter-passage interaction ('fusion') is limited to decoder
- Simple, yet effective for some tasks (NQ, TriviaQA, SQuAD)
  - Better than GPT3, T5, DPR, not surprising
  - Also better than REALM and RAG (coming up next)