

SUMMARY

DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node

Suhas Jayaram Subramanya*

Carnegie Mellon University
suhas@cmu.edu

Devvrit*

University of Texas at Austin
devvrit.03@gmail.com

Rohan Kadekodi*

University of Texas at Austin
rak@cs.texas.edu

Ravishankar Krishaswamy

Microsoft Research India
rakri@microsoft.com

Harsha Vardhan Simhadri

Microsoft Research India
harshasi@microsoft.com

Summary presenter: Joohyung Yun

2024.10.7

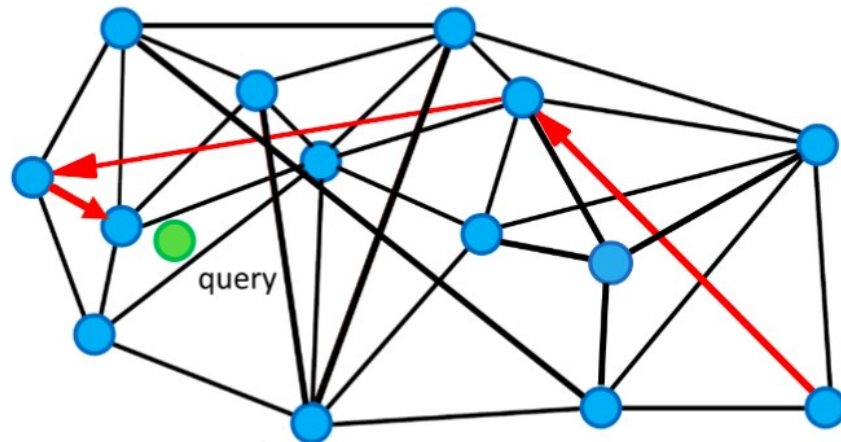
Approximate kNN with Index Structure

Motivation

- A naïve linear search over a billion-point vector database is infeasible.

Idea: Graph Index

- A graph structure that stores r nearest neighbors for each vector (expressed as a node) with edges.
- kNN search: Traversing the graph
 - Start node: fixed node within the graph



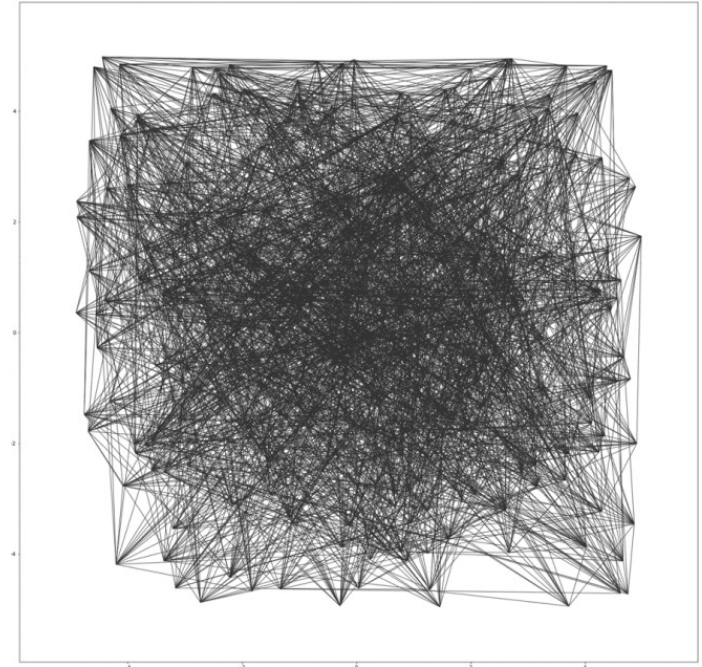
Motivation

Graph Index Generation

- Existing work struggles to generate graph index on a billion-point database
- Time complexity infeasibility: maximum n^2 calculations are needed (n = number of points).

Motivation:

- A *generation algorithm* to efficiently generate an approximate graph index
- A *search algorithm* to perform kNN search on such approximated graph index



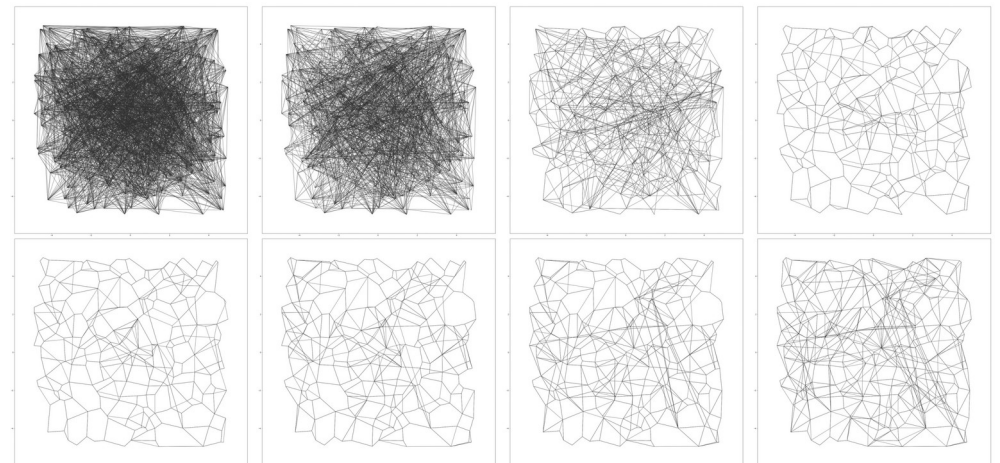
Graph Index Generation Algorithm

Vamana

Input: A set of nodes

Output: A graph index

1. Initialize a graph so that each node has r randomly chosen neighbors.
2. Compute the medoid of the graph.
3. For each node in the graph
 1. Perform a *greedySearch* to find that node.
 2. For each visited node, update the neighbors using *robustPrune*
4. Return the updated graph

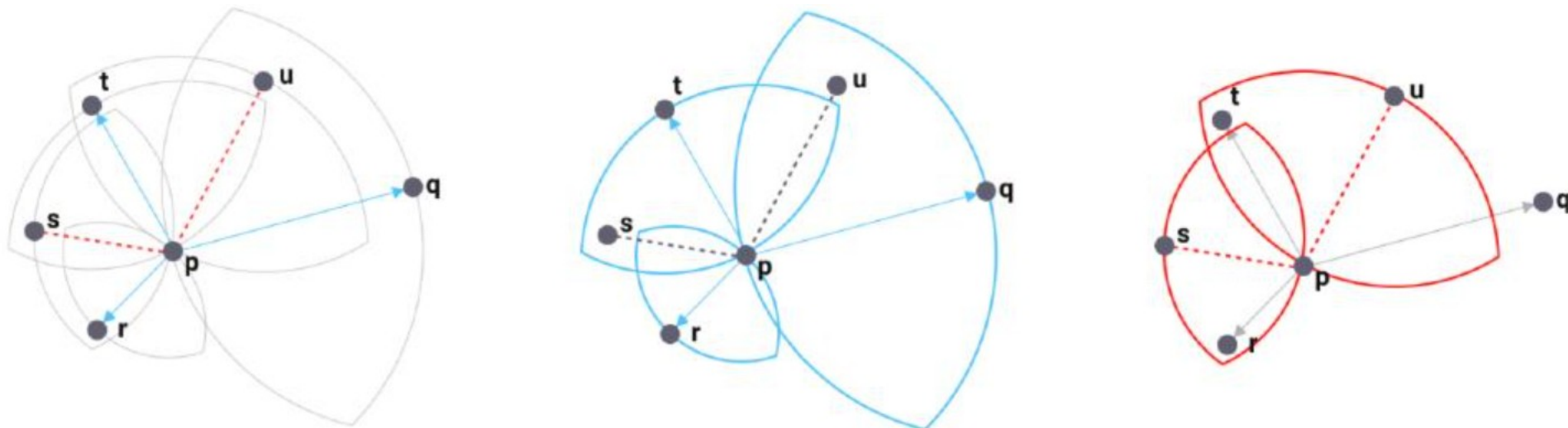


RobustPrune

Motivation: The edges are randomly generated, and they have to be updated

Idea: The nodes that were visited during the GreedySearch are ones that are relatively close to that point.

Update a node's edges using the visited nodes.



kNN Search Algorithm

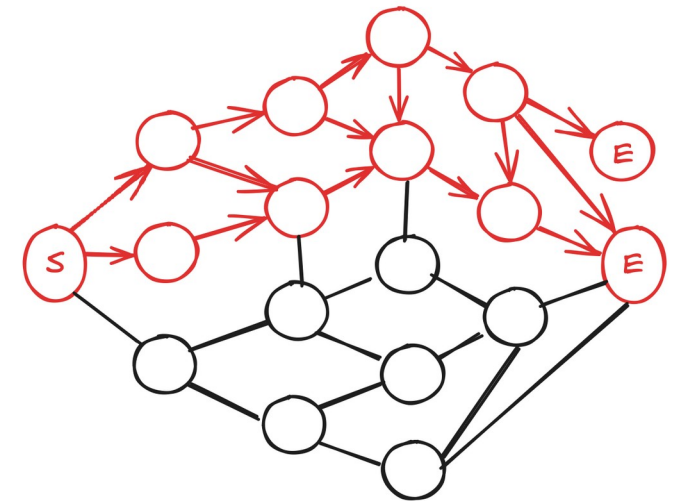
GreedySearch

It is more precisely a *beam search* with the beam width b .

Input: Query point q

Input: A graph index

1. Start from the medoid.
2. Generate a beam of medoid's neighbors.
3. Until there is no change in the beam
 1. Visit the neighbors of nodes within the beam
 2. Generate a new beam of b closest nodes to the query point q .
4. Return beam



Experiment

Latency vs Recall

- Vamana matches and exceeds the then-current best ANN methods
- **Competitors**
 - NSG
 - HNSW

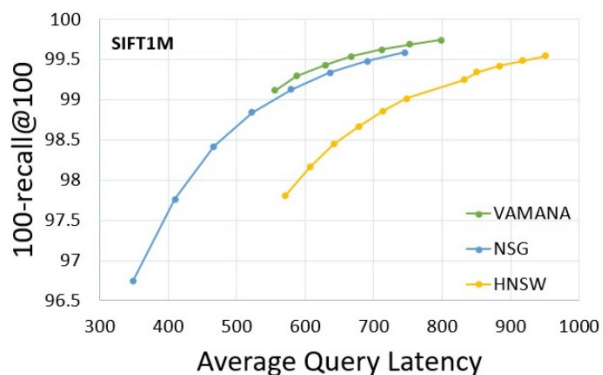
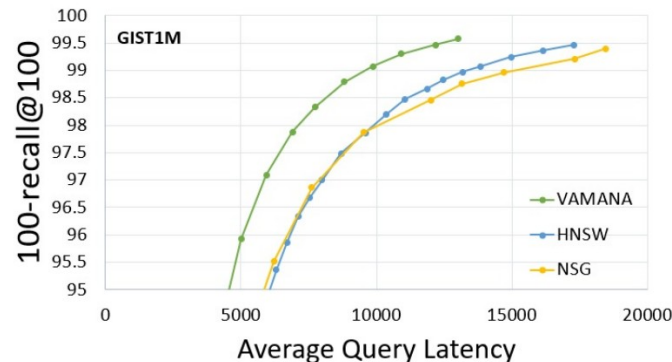
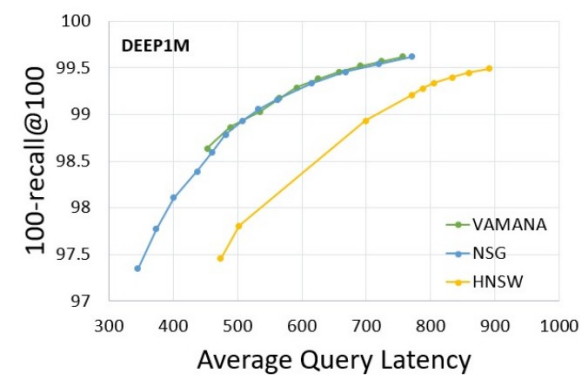


Figure 3: Latency (microseconds) vs recall plots comparing HNSW, NSG and Vamana.

Improving Language Models by Retrieving from Trillions of Tokens

Sebastian Borgeaud^{*} Arthur Mensch^{*} Jordan Hoffmann^{*} Trevor Cai Eliza Rutherford Katie Millican
George van den Driessche Jean-Baptiste Lespiau Bogdan Damoc Aidan Clark Diego de Las Casas
Aurelia Guy Jacob Menick Roman Ring Tom Hennigan Saffron Huang Loren Maggiore Chris Jones
Albin Cassirer Andy Brock Michela Paganini Geoffrey Irving Oriol Vinyals Simon Osindero
Karen Simonyan Jack W. Rae[†] Eric Elsen[†] Laurent Sifre^{*†}

DeepMind

Presenter: Joohyung Yun

2024.10.7

Recap

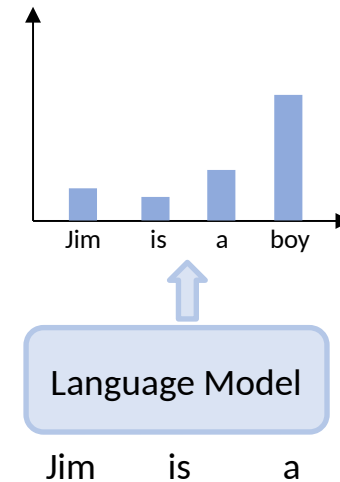
Language Modeling

• The task of assigning a probability to a sequence of tokens.

- Most language models solve this with an assumption:

To calculate $p(w_1 \dots w_l)$, iteratively calculate $p(w_t | c_t)$ for $t \in \{1, \dots, l\}$ and then calculate their joint probability.

c_t is a 'context' sequence of tokens $c_t = \{w_1, w_2, \dots, w_{t-1}\}$



$$p(\text{Jim is a boy}) = p(\text{Jim is a}) \times p(\text{boy} | \text{Jim is a})$$

Recap

Retrieval Augmented Generation

Motivation: Traditional language models rely only on *parametric knowledges* which are

- hard to fine-tune
- hard to explain (e.g., why is this output generated?)

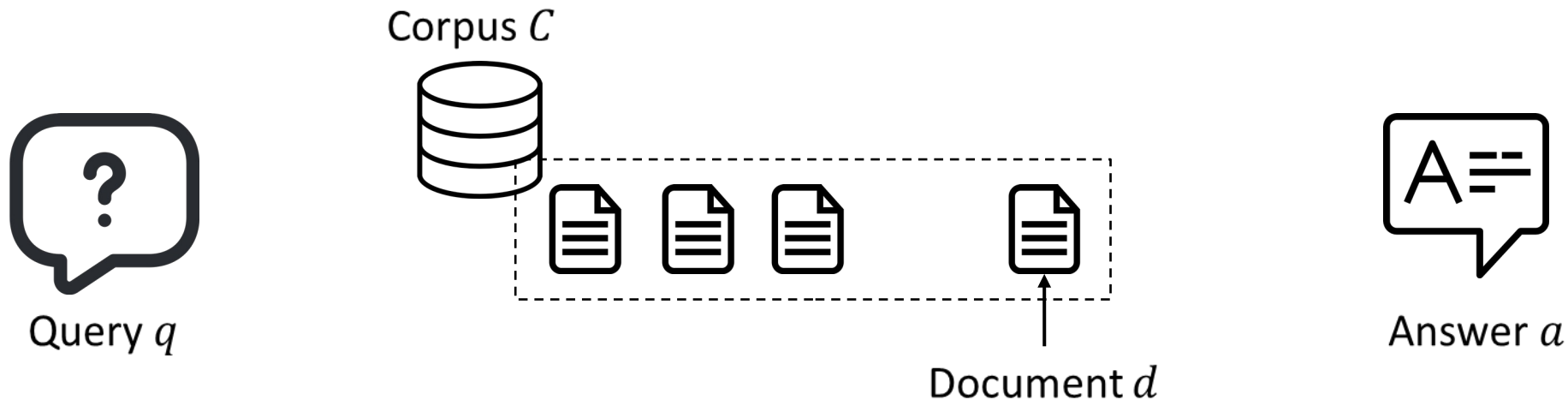
Idea: *Retrieve, then generate*

- A *retriever* retrieves query-relevant documents from a corpus (i.e., database containing text)
- A *reader* generates the output based on the query and the retrieved results

Problem Description

Given a query q and a corpus C , generate the correct answer a .

- Query $q = [q_1, \dots, q_t]$: A sequence of tokens
- Corpus C : A set of documents (i.e., sequence of tokens)
- Answer $a = [a_1, \dots, a_r]$: A sequence of tokens



Preliminaries

- Transformer
 - Self-Attention
 - Cross-Attention
 - BERT
- k-Nearest Neighbor Search (kNN Search)

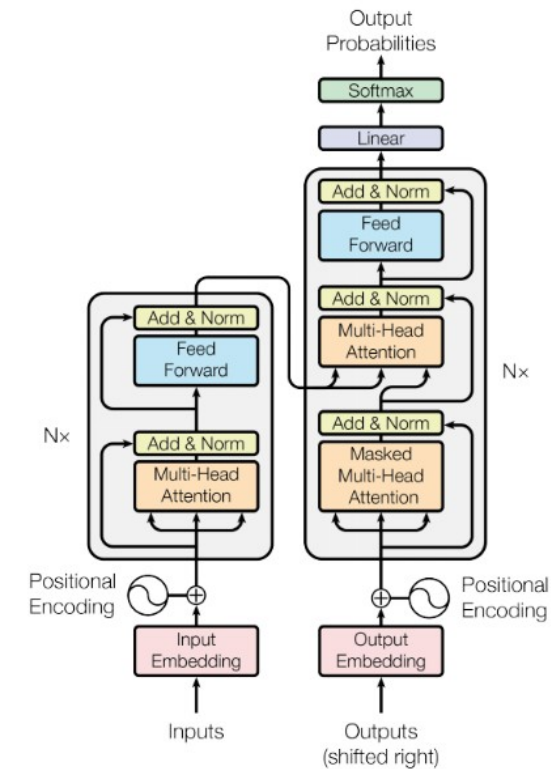


Figure 1: The Transformer - model architecture.

Motivation for RETRO

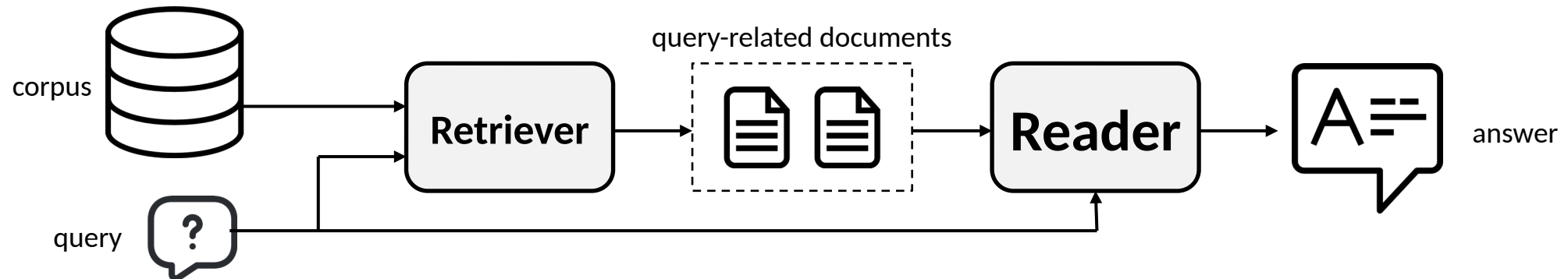
Larger Model & Larger Corpus

Larger model is needed for

- advanced inference (more calculation during inference).
- higher generalization ability (model capacity).

Larger corpus is needed to

- incrementally update the information we have.
- save more fact (probability of containing correct information $\uparrow$).



Problems of Existing Works (as of 2022)

Existing works show poor scalability regarding model size and corpus size due to

- **Inference inefficiency**

- Too much time spent for retrieval: kNN search is called for every decoding token.
- Too much time spent for decoding: The cross attention cost is linear to 'number of retrieved documents $\times$ the number of decoded tokens'.

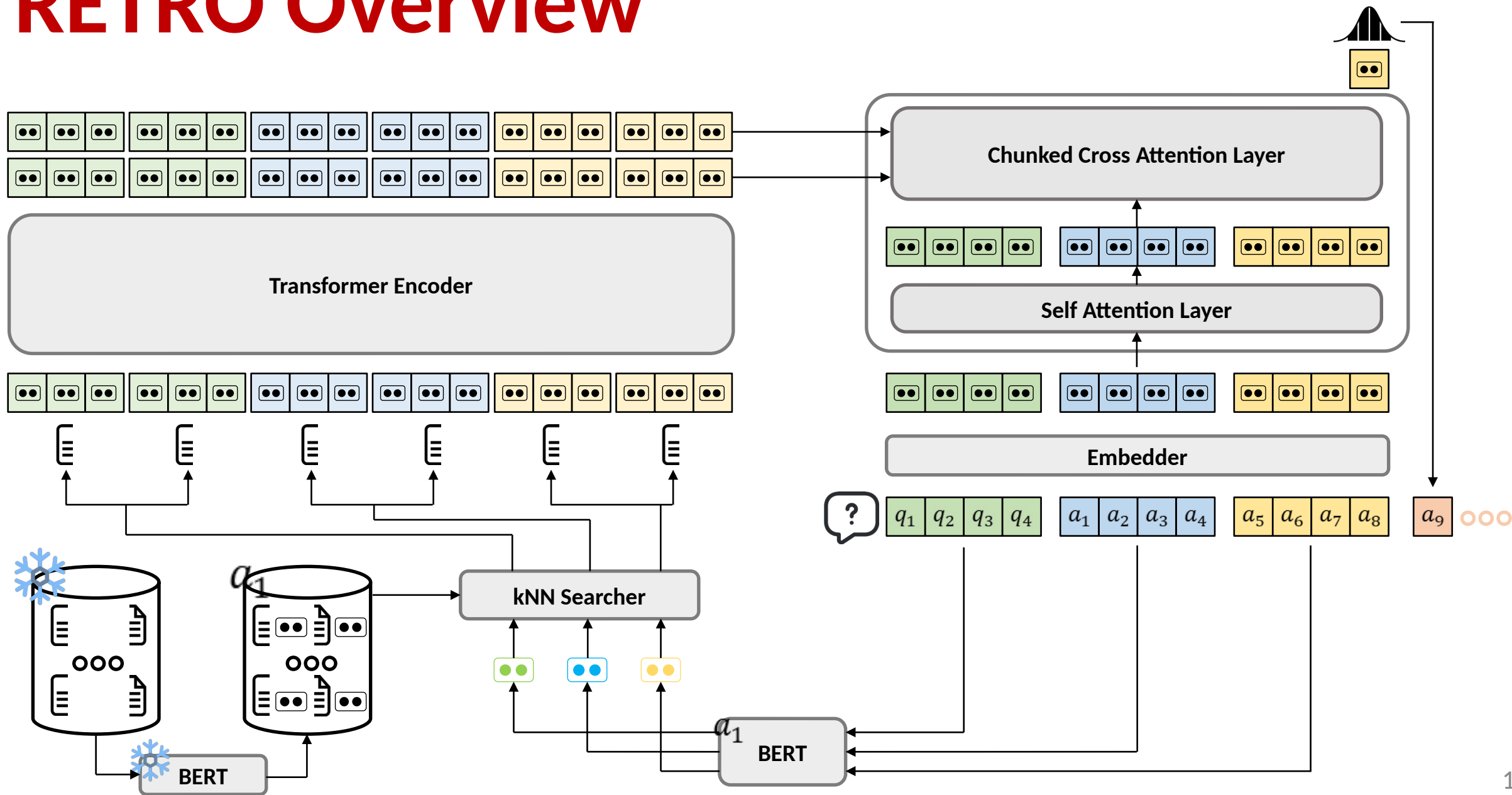
- **Training inefficiency**

- Corpus re-embedding cost: the corpus has to be reembedded if the corpus embedder's parameters updated.

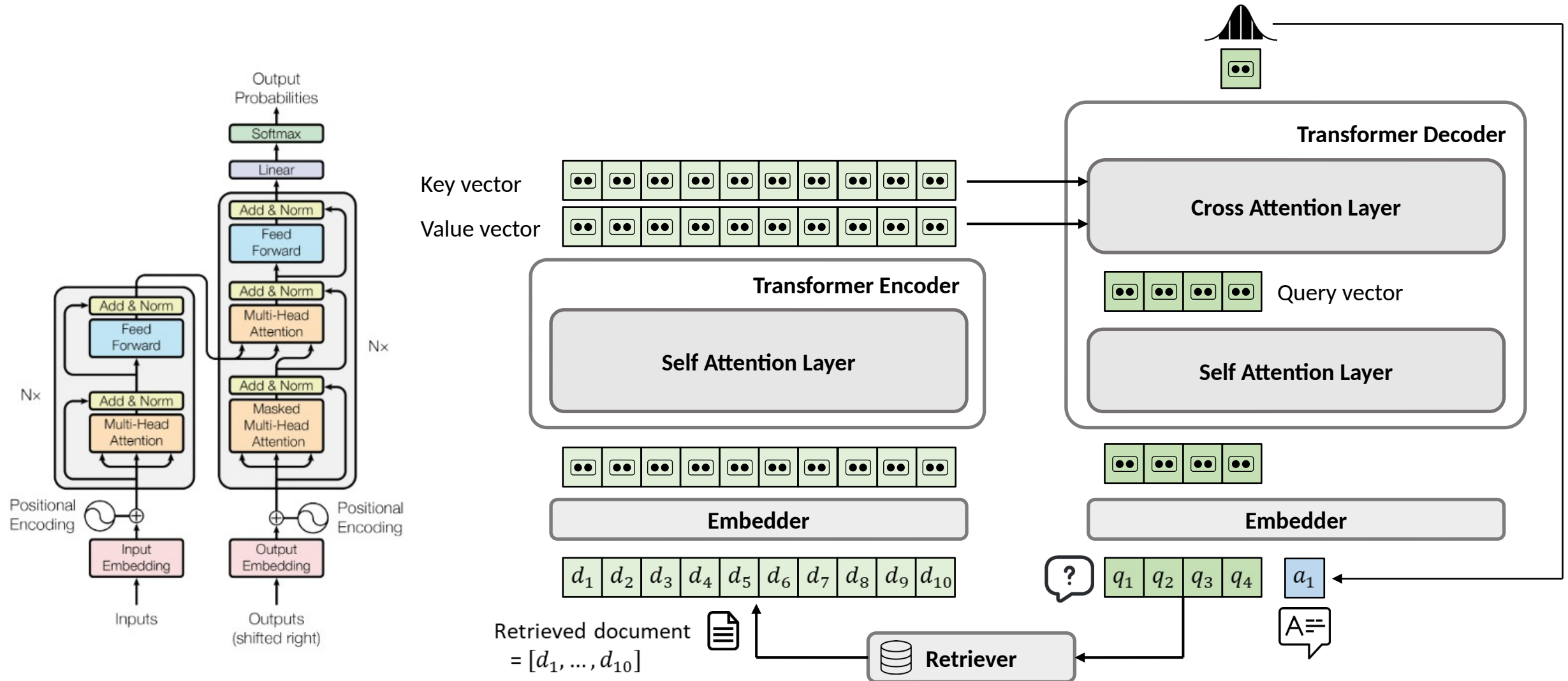
-> Sequentially modify the naïve Token-level RAG model to solve each of the three problems.

-> End up with the RETRO model.

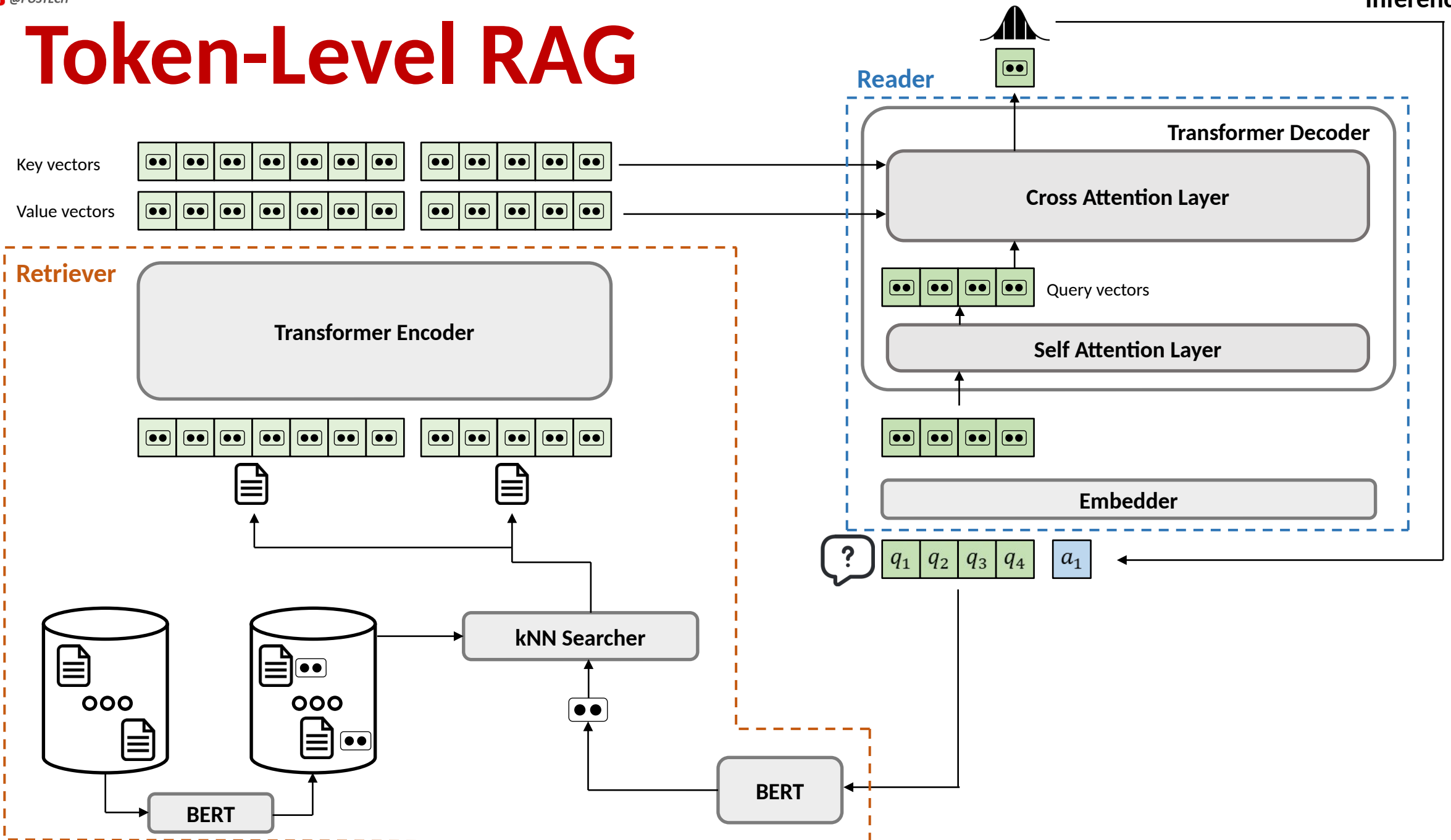
RETRO Overview



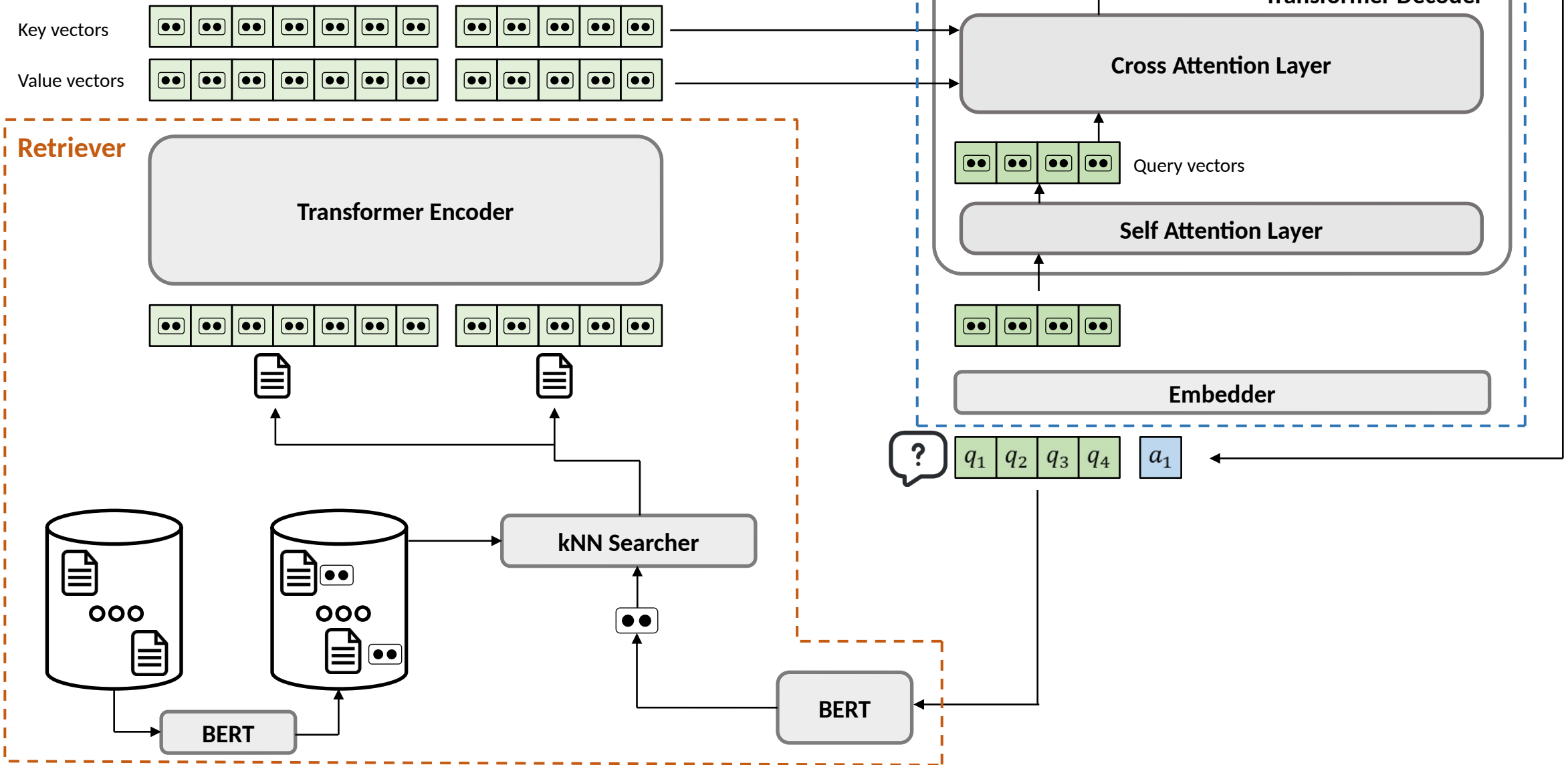
Recap: Transformer Architecture



Token-Level RAG



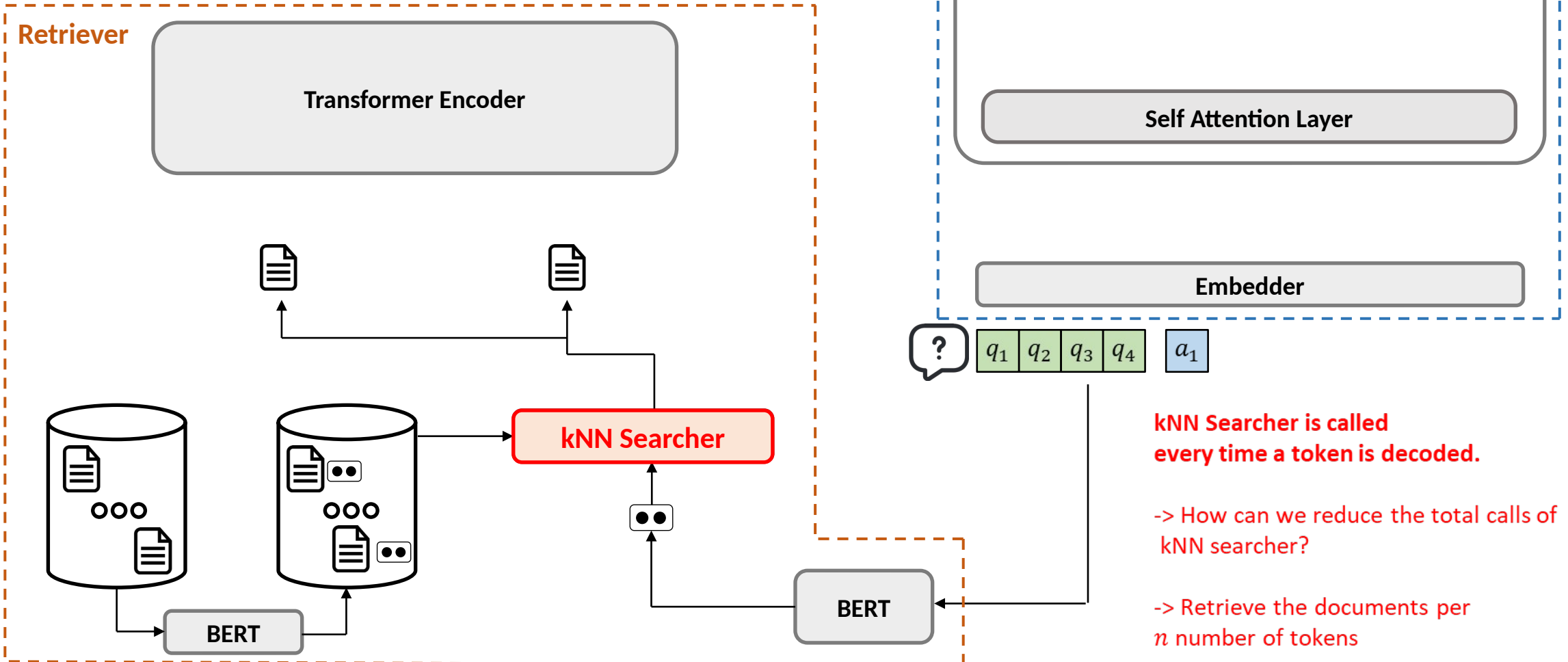
Token Generation



Problem 1:

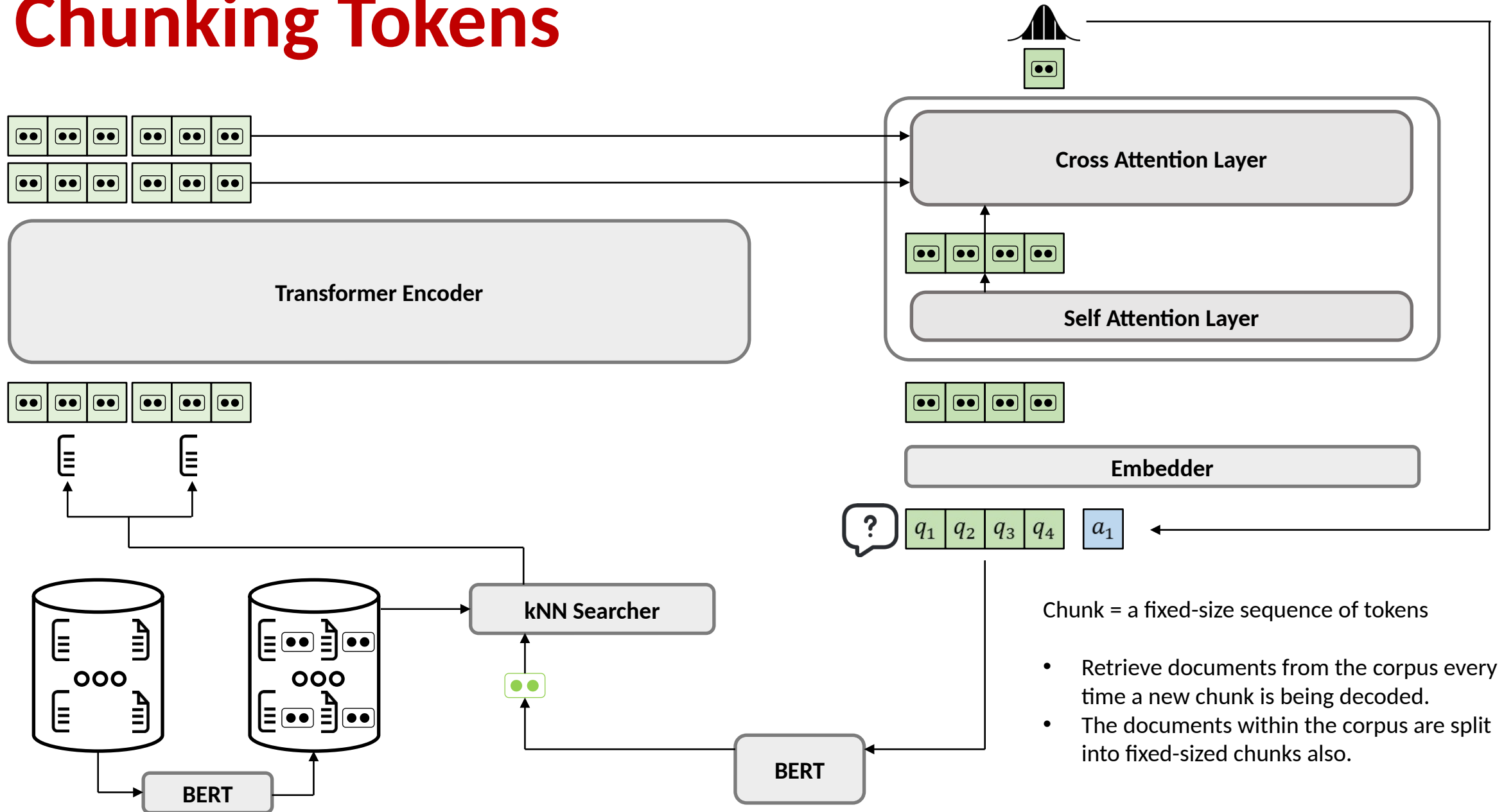
Recurrent Retrieval

Let's decode the second token



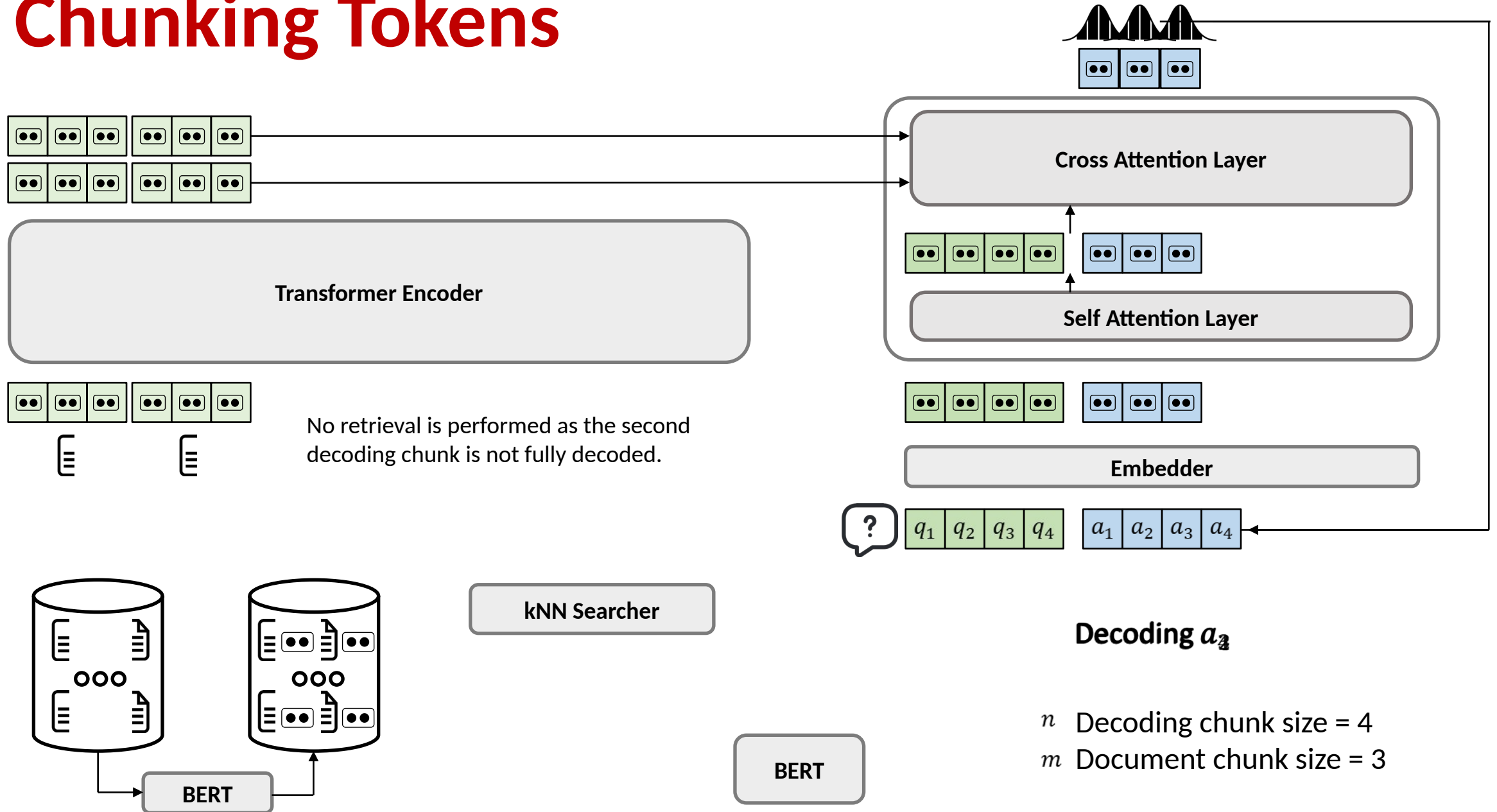
Key Idea 1:

Chunking Tokens



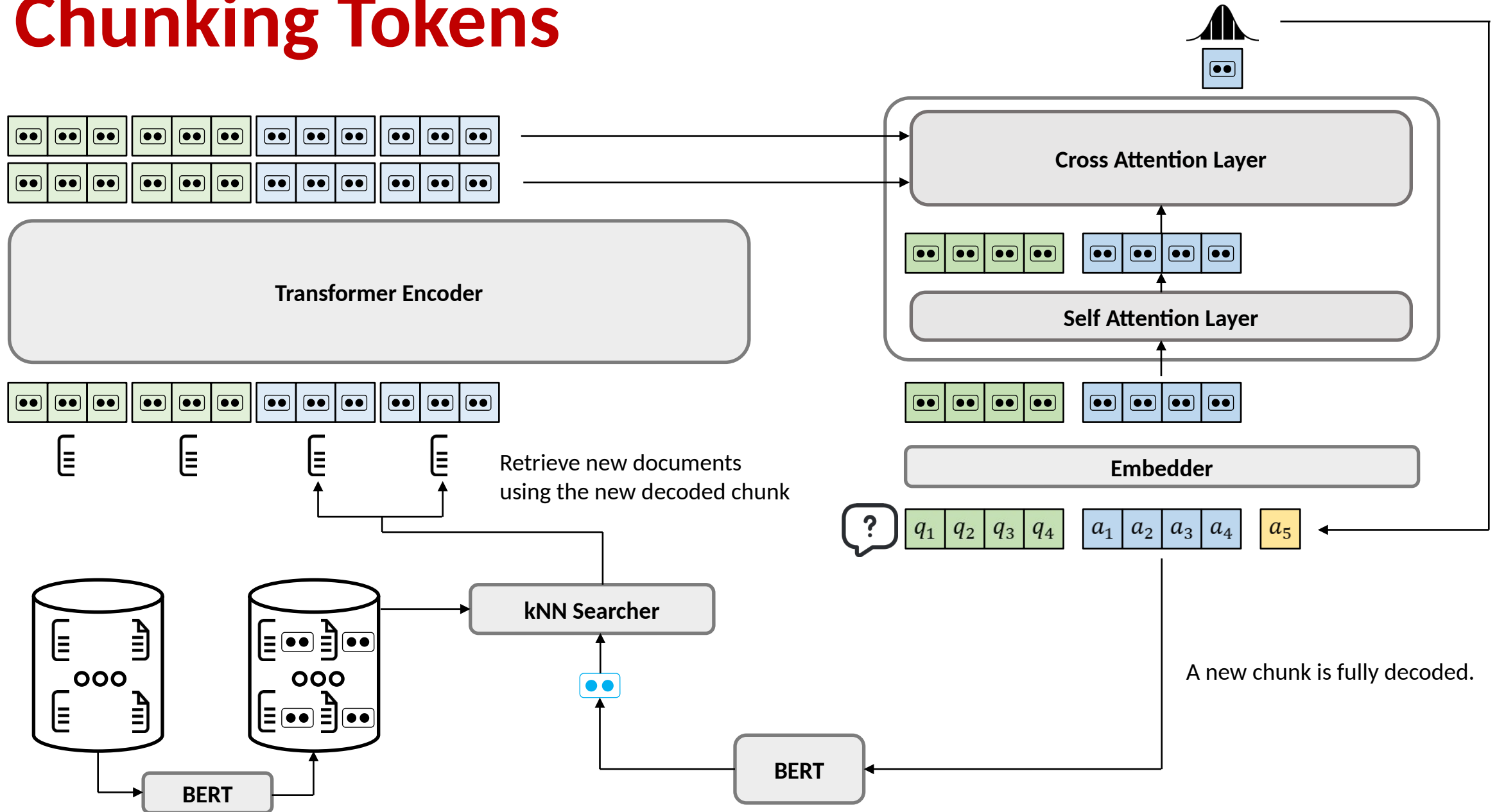
Key Idea 1:

Chunking Tokens



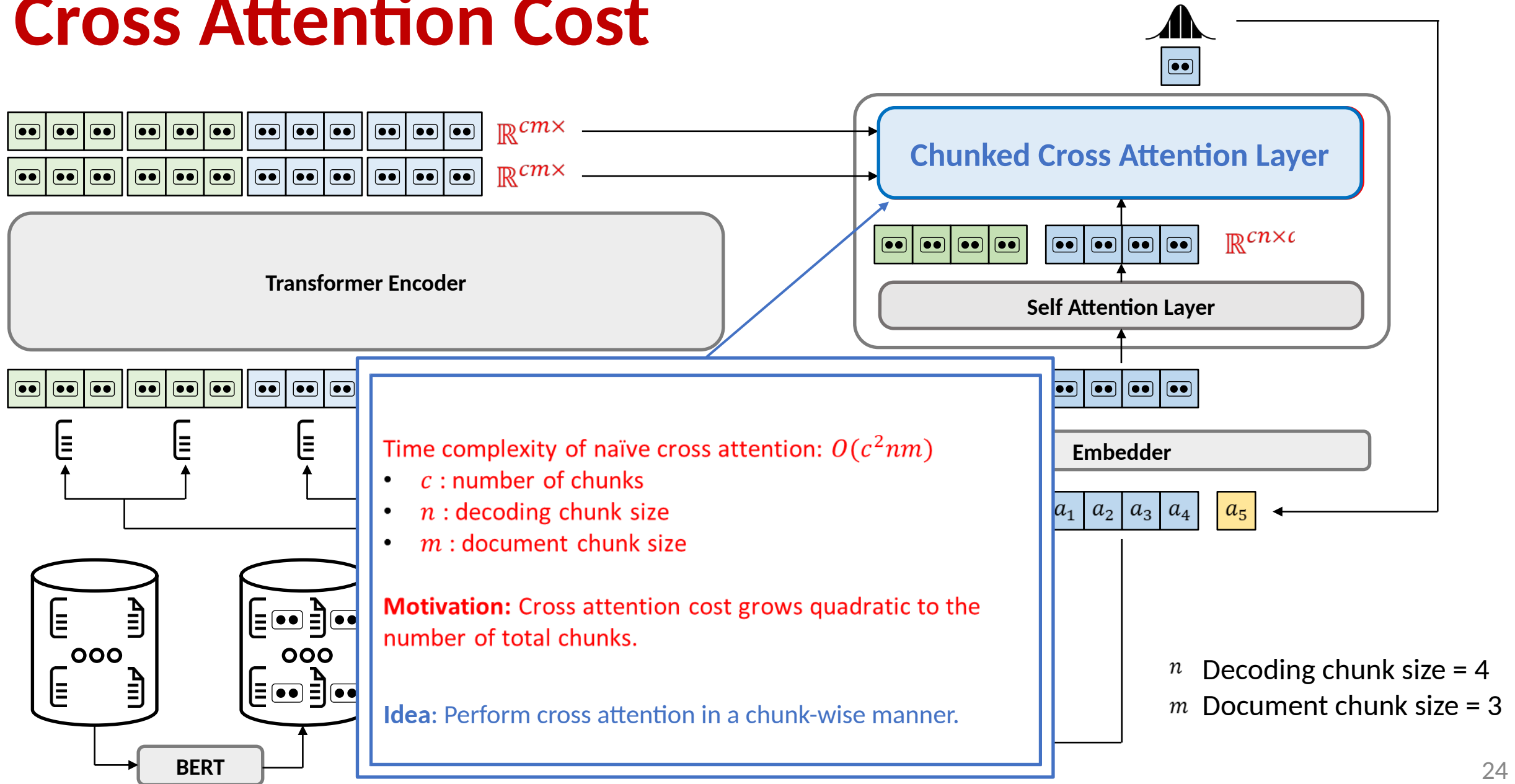
Key Idea 1:

Chunking Tokens



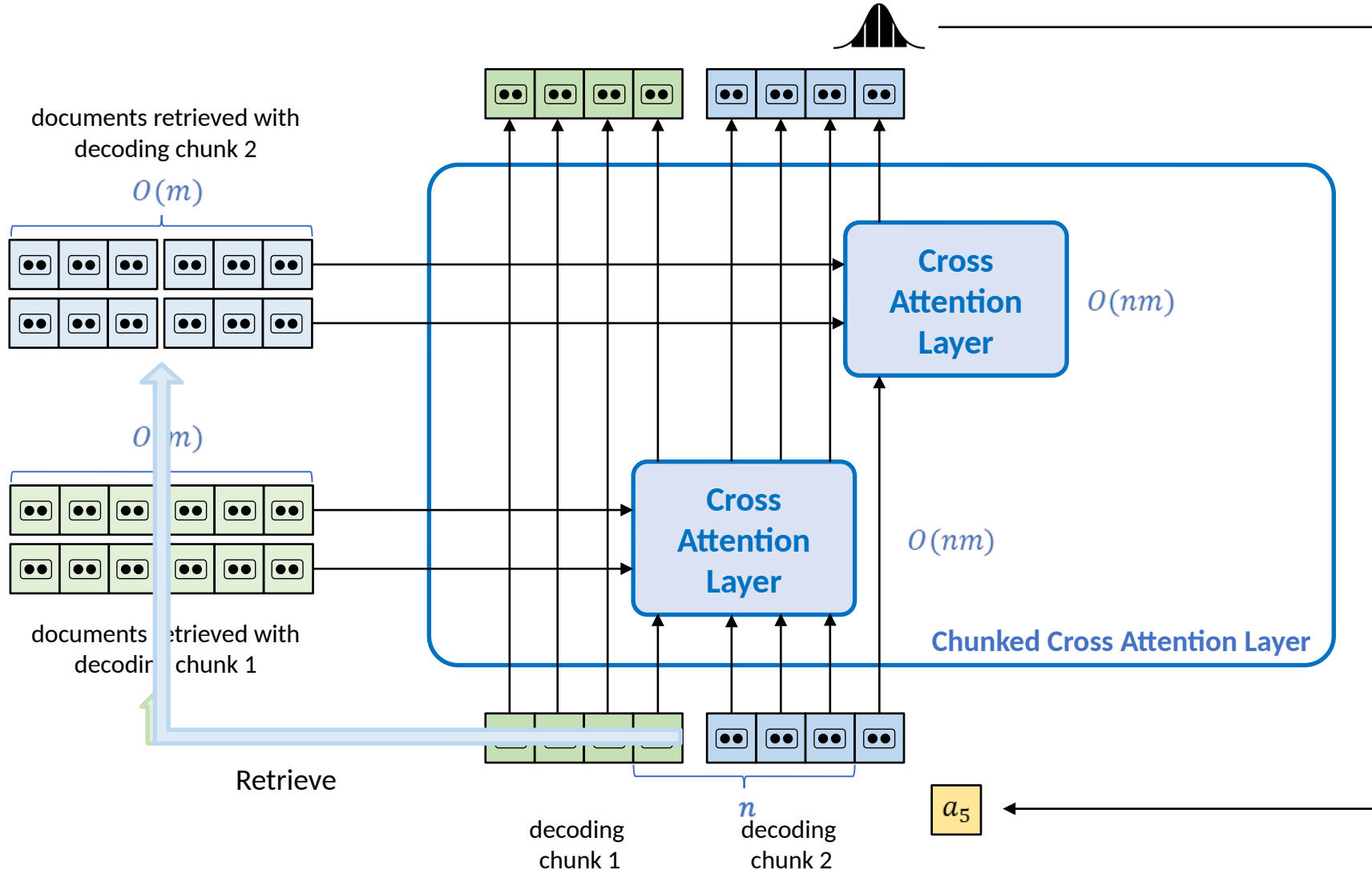
Problem 2:

Cross Attention Cost



Key Idea 2:

Chunked Cross Attention



Idea: Perform cross attention in a chunk-wise manner.

For newly decoding chunk, perform across attention only with the latest retrieved documents.

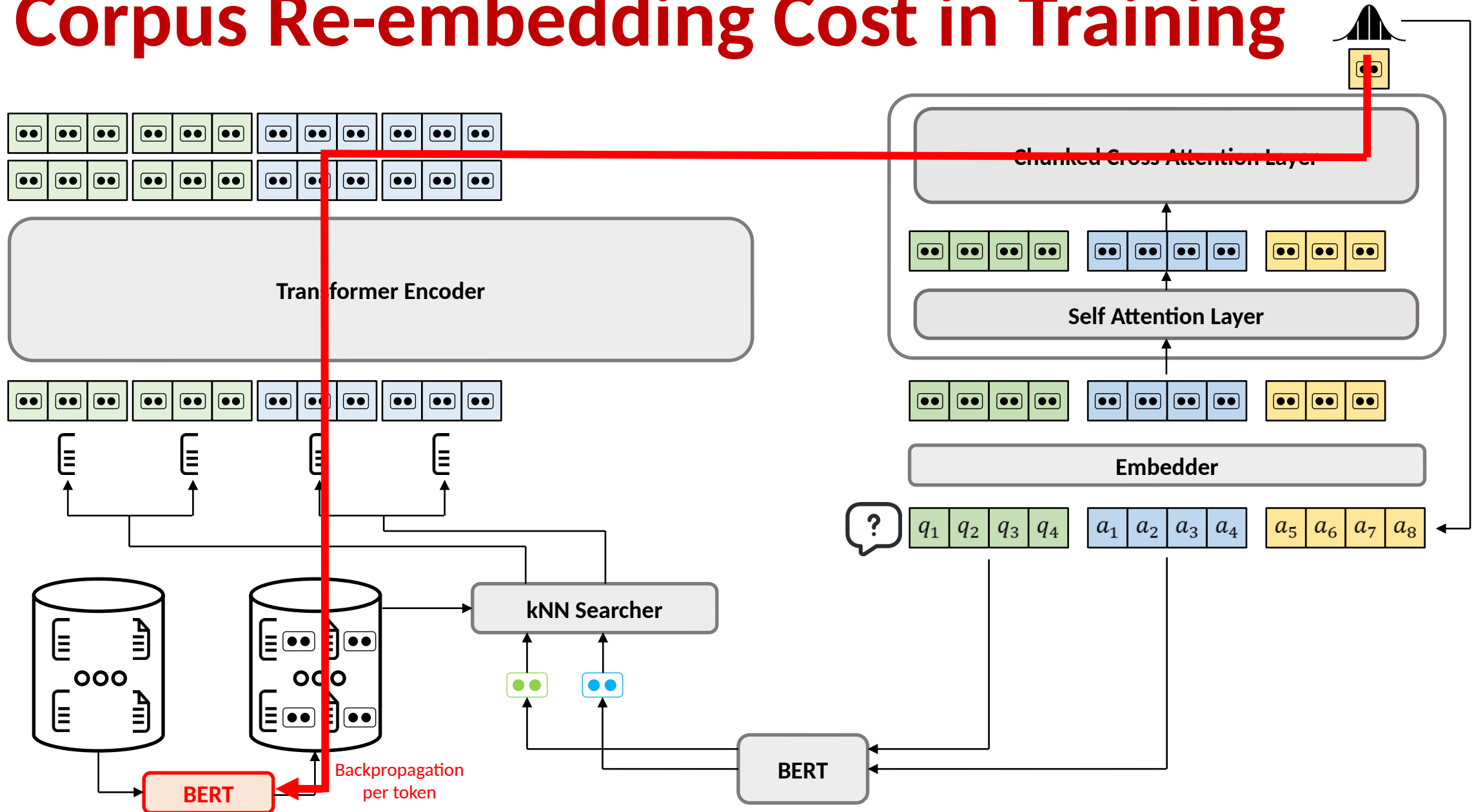
Assumption: The context for the newly decoding tokens will depend highly on the recently retrieved documents.

The total time complexity is reduced to $O(cnm)$.

- Linear to the number of chunks decoded.

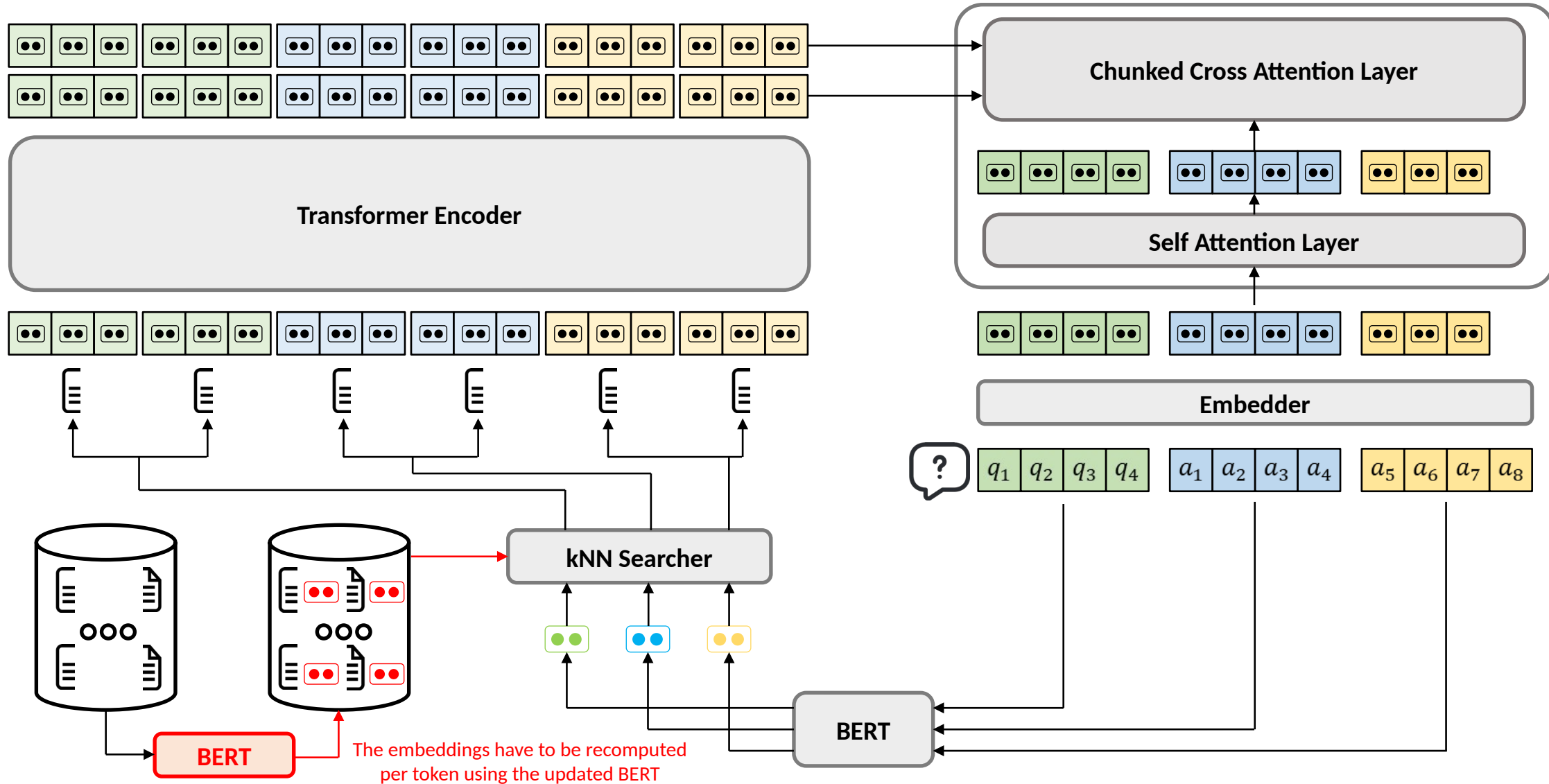
Problem 3:

Corpus Re-embedding Cost in Training



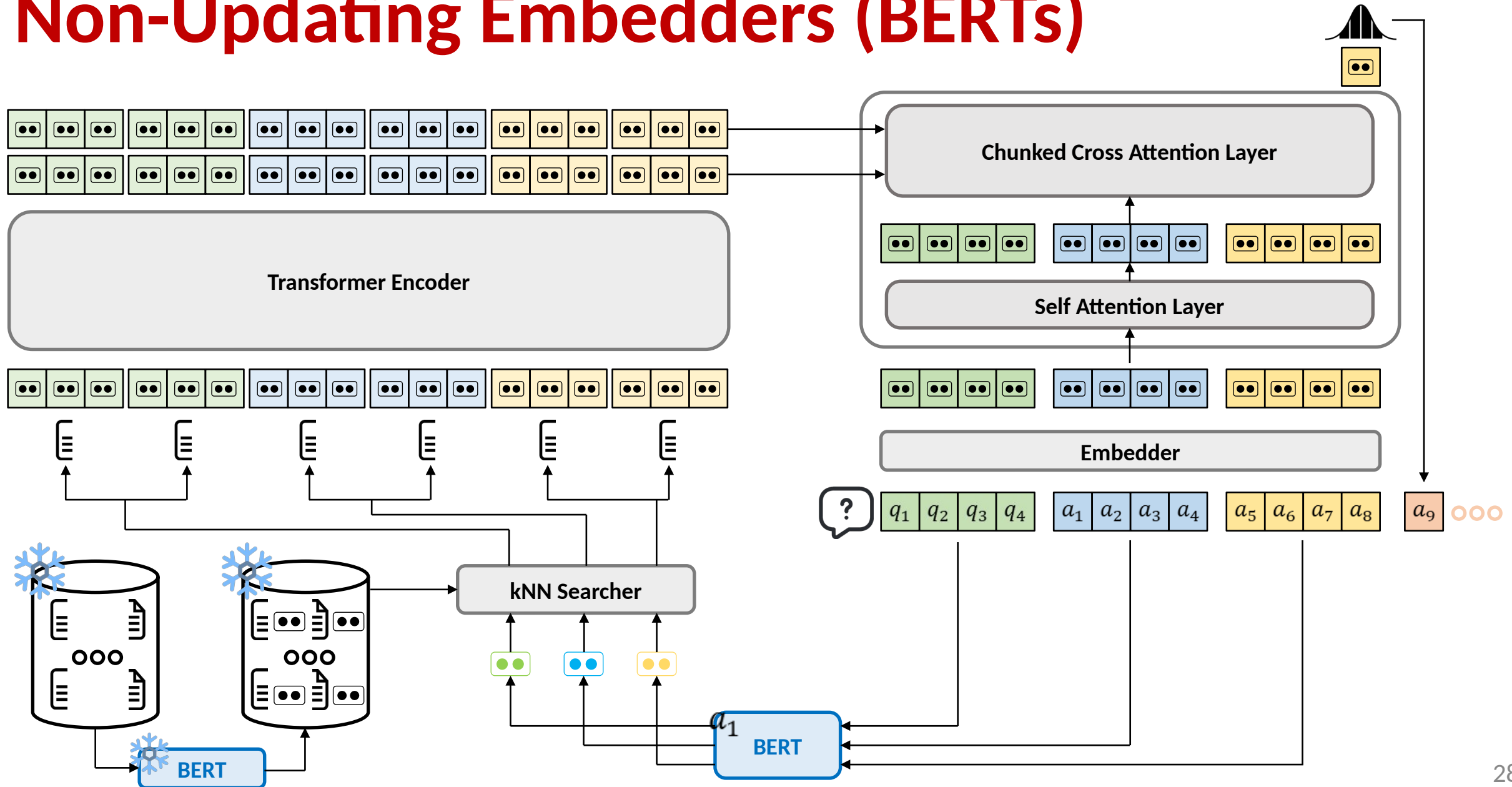
Problem 3:

Corpus Re-embedding Cost in Training

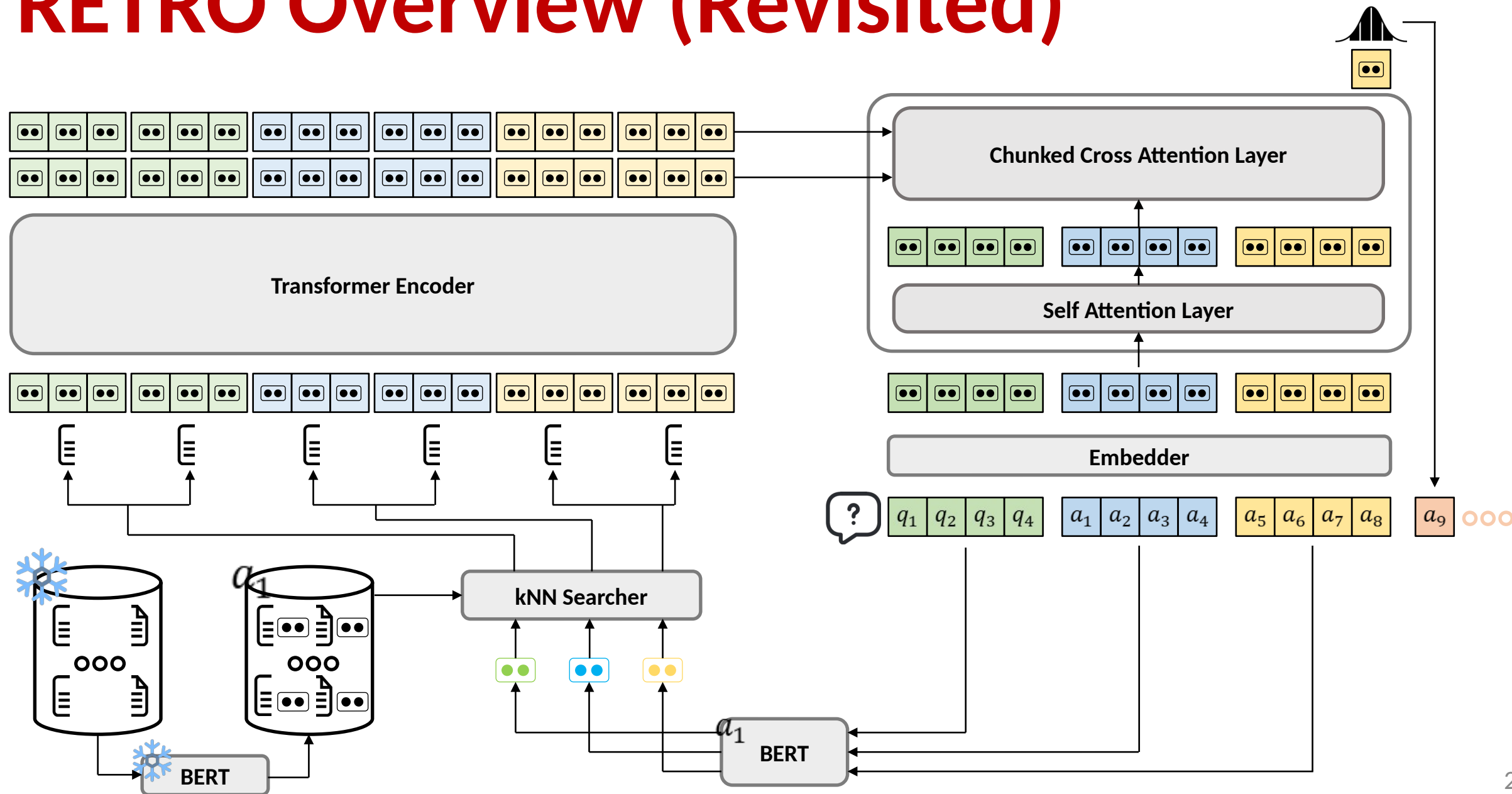


Key Idea 3:

Non-Updating Embedders (BERTs)



RETRO Overview (Revisited)



Experiments

Experiment Objectives

- **Language modeling accuracy comparison:** does RETRO outperform the SOTA models in the language modeling task?
- **Question-answering accuracy comparison:** does RETRO outperform the SOTA models in the question-answering task?
- **Parameter sensitivity (model size):** does RETRO's language modeling accuracy increase as the size of its model increases?
- **Parameter sensitivity (corpus size):** does RETRO's language modeling accuracy increase as the size of the corpus increases?

Experiment Settings

Competitors

- **Retriever-reader models**
 - SPALM
 - kNN-LM
 - REALM
 - DPR
 - RAG
- **Reader-only models**
 - Jurassic-1
 - Gopher
 - Adapt. inputs
 - Megatron

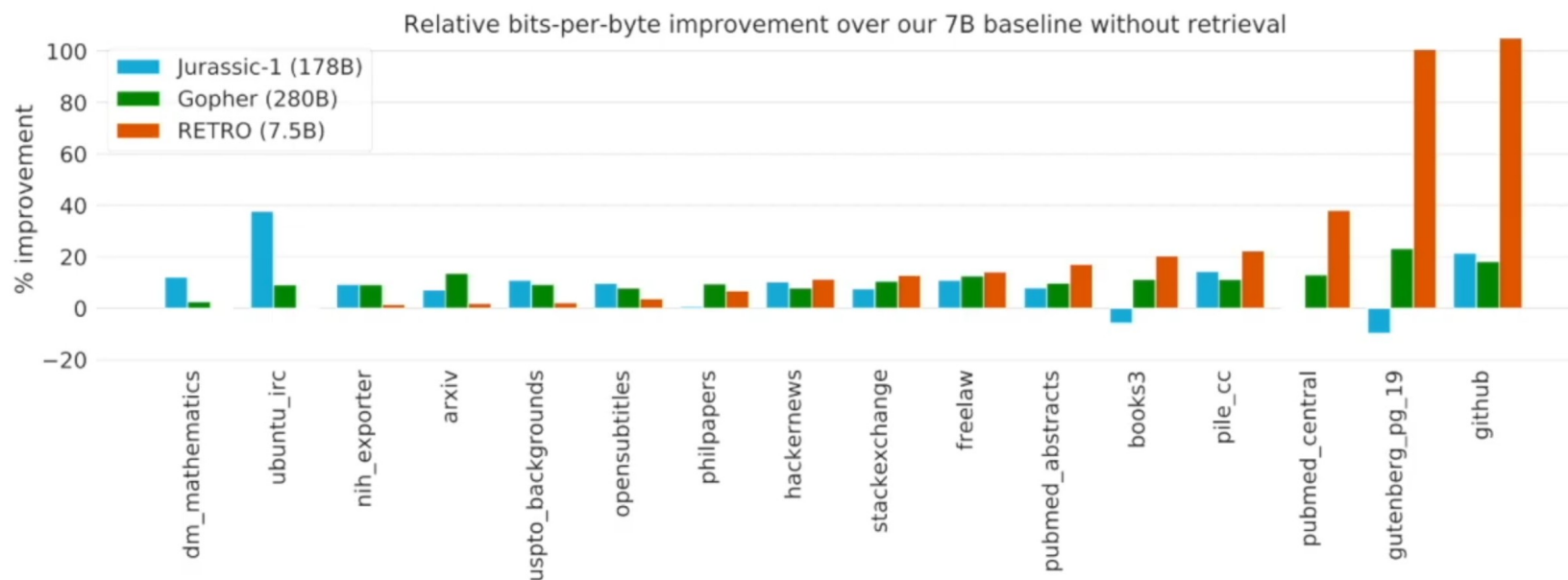
Metrics

- **Language modeling task**
 - Bits-per-byte (bpb)
 - Perplexity $perplexity = 2^{\frac{|bytes|}{|words|} \times bpb}$
 - Accuracy
- **Question-answering task**
 - accuracy

Datasets

- **Language modeling accuracy**
 - The Pile (Gao et al., 2020)
- **Question-answering accuracy**
 - Natural Questions
- **Parameter sensitivity**
 - Lambada (Paperno et al., 2016)
 - Curation Corpus (Curation, 2020)
 - Wikitext103 (Merity et al., 2017)
 - Wikipedia
 - C4 (Raffel et al., 2020)

Language Modeling Accuracy



- RETRO 7.5B outperforms Jurassic-1 and Gopher on a majority of the test sets.
- On the `dm_mathematics` and `ubuntu_irc`, RETRO model underperforms both the 7B baseline and Jurassic-1.
 - The authors hypothesized that the retrieved neighbors on these sub-datasets are not helpful

Question-Answering Accuracy

Natural Questions:

Answer span detection
task benchmark

Table 3. Question answering results on Natural Questions.

Model	Test Accuracy
REALM (Gua et al., 2020)	40.4
DPR (Karpukhin et al., 2020)	41.5
RAG (Lewis et al., 2020)	44.5
EMDR ² (Sachan et al., 2021)	52.5
FID (Izacard & Grave, 2021)	51.4
FID + Distill. (Izacard et al., 2020)	54.7
Baseline 7B (closed book)	30.4
RETRO 7.5B (DPR retrieval)	45.5

Dataset: Natural Questions

Competitors:

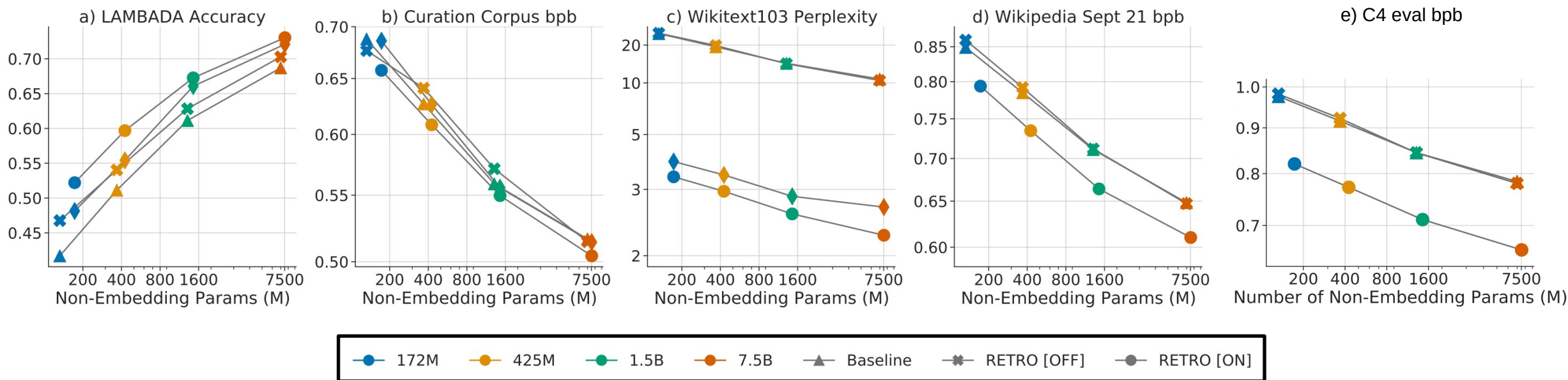
- REALM
- DPR
- RAG
- EMDR²
- FID
- FID + Distill

- RETRO is competitive with previous approaches such as REALM, RAG and DPR.
- Underperforms FID
 - FID uses T5's encoder-decoder structure and pre-training objective, which leads to a model that relies more on the encoder output.

Parameter Sensitivity

Model Size

- Compares language modeling quality for RETRO models with varying model size.
 - Performed on 5 datasets with differing metrics



Parameter Sensitivity

Corpus Size

- Compares language modeling quality for RETRO models with varying corpus size.
 - Performed on C4 only with bpb

