# Retrieval-Augmented Language Generation (RAG)

Marco Gutierrez, Qianyi Shen, Ding Zhang

# Overview

- **Non-parametric knowledge** refers to information stored outside model parameters and retrieved at inference time, rather than memorized during training.
- Khandelwal et al. (2020) showed that language models learn contextual representations better than rare facts, and that k-nearest-neighbor retrieval can offload factual prediction to external memory.
- Karpukhin et al. (2020) strengthened this approach by introducing dense retrieval methods that enable scalable and accurate access to large knowledge sources.
- Lewis et al. (2021) proposed **Retrieval-Augmented Generation** (RAG), which integrates parametric language models with non-parametric retrieval to inject retrieved knowledge directly into generation.

# Generalization through Memorization: Nearest Neighbor Language Models

# Intuition: Representation Is Easier Than Prediction

- Learning contextual similarity is easier than predicting exact next tokens

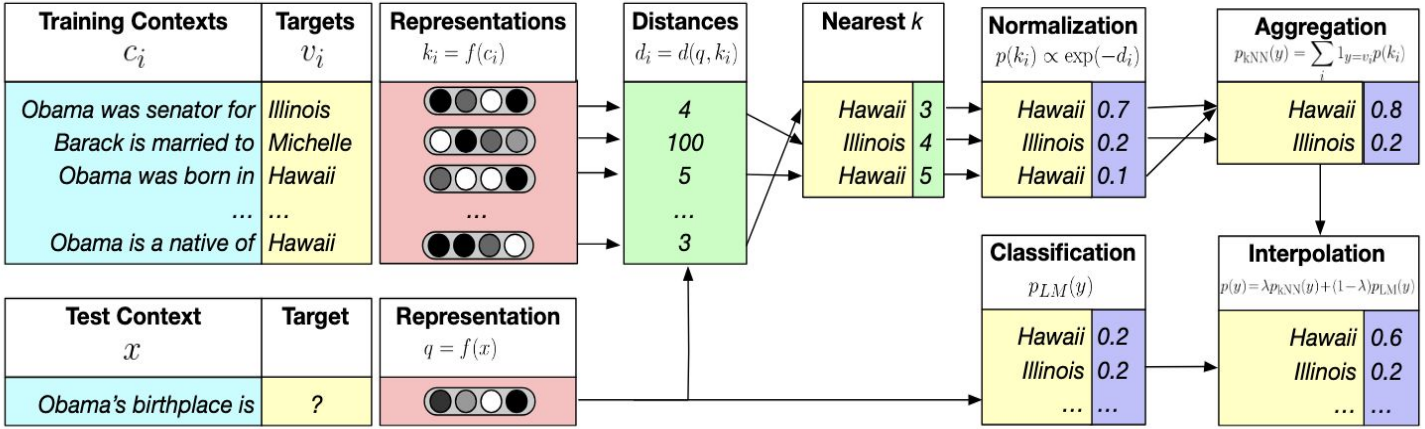
Any English speaker knows that:

“Dickens is the author of …” and “Dickens wrote …”

refer to the same underlying fact, even without knowing the precise next-word distribution

- Modern LMs excel at encoding semantic similarity between prefixes
- However, memorizing rare or factual continuations in parameters is difficult
- Key idea behind kNN-LM: learn prefix representations, retrieve similar past contexts, and let their observed continuations guide prediction rather than forcing the model to memorize facts.

# Mechanism




- The datastore is constructed offline from the training data, where each training context is encoded and stored together with its next-token target.
- At test time, the input context is encoded by the frozen language model into a query representation, which is used to retrieve the k most similar training contexts from the datastore and to form a neighbor-based probability distribution over next tokens.
- In parallel, the base language model produces its own next-token distribution via standard softmax classification, and the final prediction is obtained by interpolating this distribution with the neighbor-based distribution.

**Datastore construction**

For every training token, store a key-value pair: (billions of entries):

- Key: context embedding extracted from an intermediate Transformer layer

Best choice: input to the final feed-forward network after layer normalization

- Value: the next token to predict

**Nearest-neighbor inference (Test Time)**

- Encode the current context using the frozen LM
- Retrieve the top-k nearest neighbors in embedding space via FAISS (Johnson et al., 2017) for scalability
- Convert neighbor distances into a probability distribution over target tokens.
- Linearly interpolate this distribution with the LM's own softmax output using a tuned parameter λ

Jeff Johnson, Matthijs Douze, and Herv´e Je´gou. Billion-scale similarity search with gpus. arXiv preprint arXiv:1702.08734, 2017

# Most important technical details

**Key design motivations**

- L2 distance in representation space empirically better than inner product
- Large k yields monotonic performance gains
- Retrieval functions as a high-capacity, non-parametric classifier on top of a neural encoder

**Implementation pragmatics**

- FAISS enables billion-scale search via clustering and vector quantization.
- Datastore creation requires only a single forward pass and is CPU-parallelizable, making it far cheaper than retraining

# State-of-the-art performance with no additional training

| Model | Perplexity (↓) Dev | Test | # Trainable Params |
|---|---|---|---|
| Baevski & Auli (2019) | 17.96 | 18.65 | 247M |
| +Transformer-XL (Dai et al., 2019) | - | 18.30 | 257M |
| +Phrase Induction (Luo et al., 2019) | - | 17.40 | 257M |
| Base LM (Baevski & Auli, 2019) | 17.96 | 18.65 | 247M |
| +$k$NN-LM | **16.06** | **16.12** | 247M |
| +Continuous Cache (Grave et al., 2017c) | 17.67 | 18.27 | 247M |
| +$k$NN-LM + Continuous Cache | **15.81** | **15.79** | 247M |

Table 1: Performance on WIKITEXT-103. The $k$NN-LM substantially outperforms existing work. Gains are additive with the related but orthogonal continuous cache, allowing us to improve the base model by almost 3 perplexity points with no additional training. We report the median of three random seeds.

| Model | Perplexity (↓) Dev | Test | # Trainable Params |
|---|---|---|---|
| Base LM (Baevski & Auli, 2019) | 14.75 | 11.89 | 247M |
| +$k$NN-LM | **14.20** | **10.89** | 247M |

Table 2: Performance on BOOKS, showing that $k$NN-LM works well in multiple domains.

- kNN-LM yields large gains without retraining

~2–3 perplexity improvement on WikiText-103

- Complementary to cache

Cache = recent test-time tokens
kNN-LM = similar training contexts
Together give the best performance

- No increase in parameters

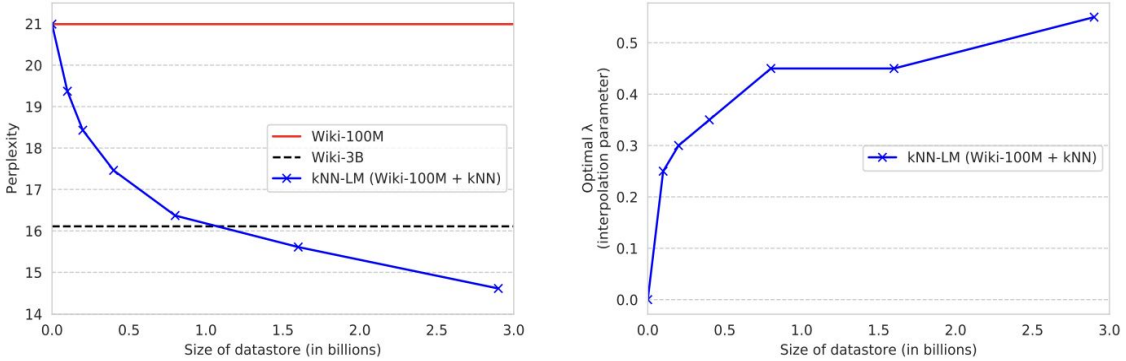
Same model size; gains come from retrieval

- General across domains

Improvements also hold on BOOKS

| Training Data | Datastore | Perplexity (↓) Dev | Test |
|---|---|---|---|
| WIKI-3B | - | 16.11 | 15.17 |
| WIKI-100M | - | 20.99 | 19.59 |
| WIKI-100M | WIKI-3B | 14.61 | 13.73 |

Table 3: Experimental results on WIKI-3B. The model trained on 100M tokens is augmented with a datastore that contains about 3B training examples, outperforming the vanilla LM trained on the entire WIKI-3B training set.




(a) Effect of datastore size on perplexities.

(b) Tuned values of $\lambda$ for different datastore sizes.

Figure 2: Varying the size of the datastore. (a) Increasing the datastore size monotonically improves performance, and has not saturated even at about 3B tokens. A $k$NN-LM trained on 100M tokens with a datastore of 1.6B tokens already outperforms the LM trained on all 3B tokens. (b) The optimal value of $\lambda$ increases with the size of the datastore.

- 100M + retrieval > 3B training
- Performance improves monotonically with datastore size and does not saturate
- The optimal interpolation weight lambda increases with datastore size
- Scaling performance can be achieved by scaling retrieval, not just training

# Domain Adaptation via Datastore Swapping

| Training Data | Datastore | Perplexity (↓) Dev | Test |
|---|---|---|---|
| WIKI-3B | - | 37.13 | 34.84 |
| BOOKS | - | 14.75 | 11.89 |
| WIKI-3B | BOOKS | 24.85 | 20.47 |

Table 4: Domain adaptation experiments, with results on BOOKS. Adding an in-domain datastore to a Wikipedia-trained model improves results by 23 points, approaching in-domain training.

- A Wikipedia-trained LM performs poorly on BOOKS
- Training directly on BOOKS gives the best performance
- Simply swapping the datastore to in-domain text (BOOKS) recovers most of the gain

# Effect of Interpolation Parameter

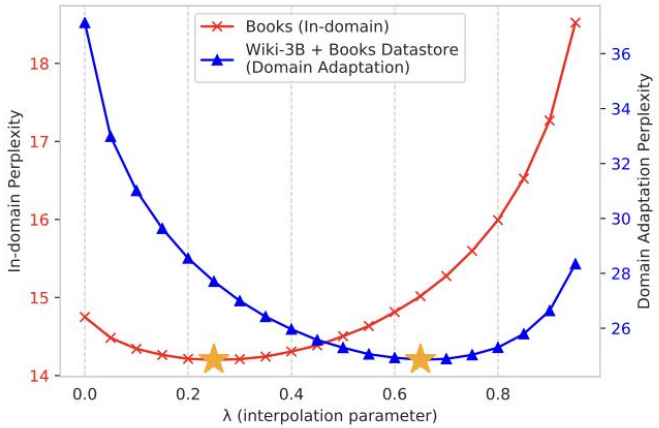



Figure 5: Effect of interpolation parameter $\lambda$ on in-domain (left y-axis) and out-of-domain (right y-axis) validation set performances. More weight on $p_{kNN}$ improves domain adaptation.

- Optimal lambda is larger for domain adaptation
- More weight on kNN retrieval improves out-of-domain performance

Interpretation:

In a new domain, retrieval (“looking up data”) matters more than the model’s prior experience.

# Why use this representation?

| Key Type | Dev ppl. (↓) |
|---|---|
| No datastore | 17.96 |
| Model output | 17.07 |
| Model output layer normalized | 17.01 |
| FFN input after layer norm | **16.06** |
| FFN input before layer norm | 17.06 |
| MHSA input after layer norm | 16.76 |
| MHSA input before layer norm | 17.14 |

Table 5: WIKITEXT-103 validation results using different states from the final layer of the LM as the representation function $f(\cdot)$ for keys and queries. We retrieve $k$=1024 neighbors and $\lambda$ is tuned for each.

- Different internal states of the final Transformer layer are tested as kNN keys
- Best performance comes from: FFN input after layer normalization

# Intuition behind the choice

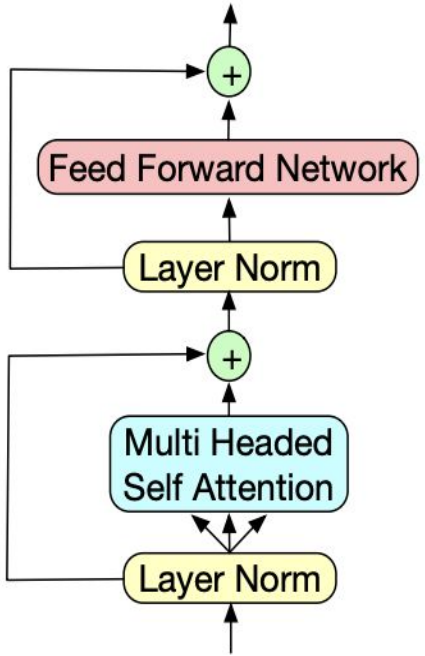



Figure 3: Transformer LM layer.

Self-attention: aggregates information and captures semantic similarity

FFN: specialized toward next-token prediction

Why before FFN?

- Representation remains general-purpose
- Preserves semantic similarity structure

Why after layer normalization?

- Normalization stabilizes scale across contexts
- Makes L2 distance more meaningful and comparable for retrieval

# Effect of k

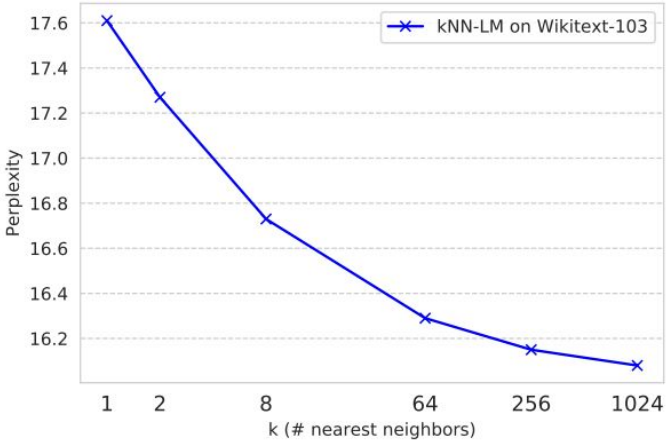



Figure 4: Effect of the number of nearest neighbors returned per word on WIKITEXT-103 (validation set). Returning more entries from the datastore monotonically improves performance.

- Performance improves monotonically as k increases
- Gains arise from aggregated evidence across similar contexts

# Most effective where LMs are prone to hallucination

| **Test Context** ($p_{\text{kNN}} = 0.998, p_{\text{LM}} = 0.124$) | **Test Target** |
|---|---|
| *it was organised by New Zealand international player Joseph Warbrick, promoted by civil servant Thomas Eyton, and managed by James Scott, a publican. The Natives were the first New Zealand team to perform a haka, and also the first to wear all black. They played 107 rugby matches during the tour, as well as a small number of Victorian Rules football and association football matches in Australia. Having made a significant impact on the...* | development |

| **Training Set Context** | **Training Set Target** | **Context Probability** |
|---|---|---|
| *As the captain and instigator of the 1888-89 Natives – the first New Zealand team to tour the British Isles – Warbrick had a lasting impact on the...* | development | 0.998 |
| *promoted to a new first grade competition which started in 1900. Glebe immediately made a big impact on the...* | district | 0.00012 |
| *centuries, few were as large as other players managed. However, others contend that his impact on the...* | game | 0.000034 |
| *Nearly every game in the main series has either an anime or manga adaptation, or both. The series has had a significant impact on the...* | development | 0.00000092 |

Figure 6: Example where the $k$NN model has much higher confidence in the correct target than the LM. Although there are other training set examples with similar local $n$-gram matches, the nearest neighbour search is highly confident of specific and very relevant context.

- Base LM assigns low probability to the correct token while kNN assigns near-certainty
- Retrieved examples are semantically similar, not exact n-gram matches

Common patterns: named entities; dates / historical facts; rare but near-duplicate contexts

# Why not n-gram?

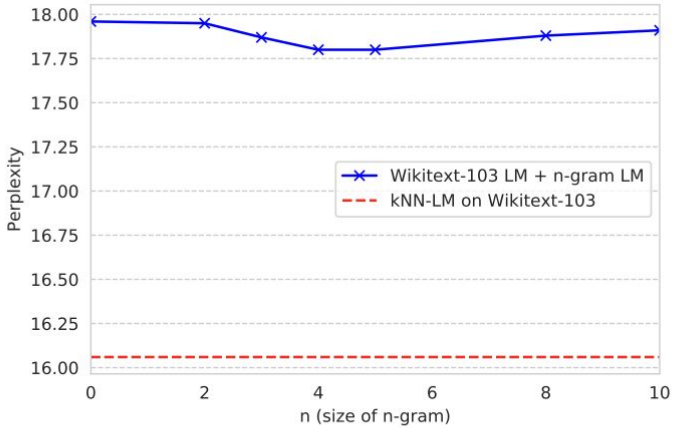



Figure 7: Interpolating the Transformer LM with $n$-gram LMs on WIKITEXT-103 (validation set). Using $k$NN-LM gives a much lower perplexity, suggesting that the representations are learning more than just matching local context.

- Interpolating with n-gram LMs yields little improvement
- kNN-LM achieves a substantial perplexity reduction

The gains come from semantic retrieval, not surface-level string repetition.

# Control Experiment: Memorization vs Retrieval

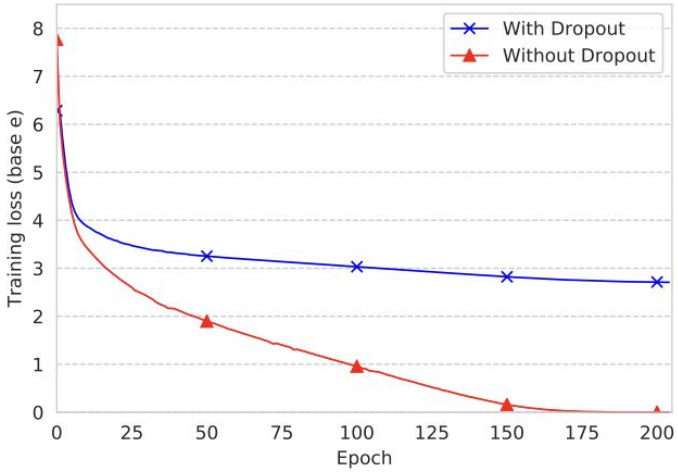



Figure 8: Training curves for the Transformer LM with and without dropout. Turning off dropout allows the training loss to go to 0, indicating that the model has sufficient capacity to memorize the training data.

- Removing dropout allows the Transformer to perfectly memorize the training set but generalization fails

Why kNN-LM works?

- kNN-LM offloads memorization to an external datastore
- Retrieval handles long-tail factual patterns efficiently

# Takeaway: Why these contributions are important?

**Separation of representation and prediction**

- LMs are better at learning contextual similarity than memorizing rare facts
- kNN-LM lets the model focus on representations, while explicit memory handles long-tail prediction

**Rethinking scaling and generalization**

- Retrieval over large corpora can substitute for additional training data
- Challenges the paradigm of ever-larger end-to-end training
- Motivates hybrid parametric + non-parametric systems

**A new view of factual knowledge in LMs**

- Rare facts (names, dates, historical details) are better stored in external memory
- Influenced later retrieval-augmented generation (RAG) approaches

# Limitations and Future Directions

**Inference-time overhead**

**Deployment and maintenance at scale**

**Static similarity function**

- Retrieval relies on representations not trained specifically for kNN
- Future work: learn embeddings optimized for nearest-neighbor retrieval

**Limited semantic abstraction**

- Most effective for rare facts and near-duplicate contexts
- Less helpful for novel, compositional generalization
- Retrieval complements but does not replace parametric reasoning

**Most evaluation limited to perplexity as metric**

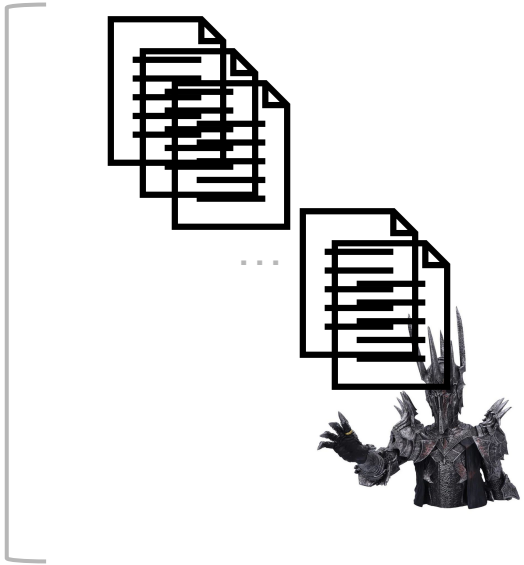
- Might not accurately reflect downstream task performance (e.g., factual question answering)

# Dense Passage Retrieval for Open-Domain Question Answering

# Open-Domain Question Answering (QA)

QA answers factoid questions using a large collection of documents

*“Who is the bad guy in the lord of the rings?”*



# Open-Domain Question Answering (QA)

Two-Stage Framework solution

*“Who is the bad guy in the lord of the rings?”*

# Retrieval

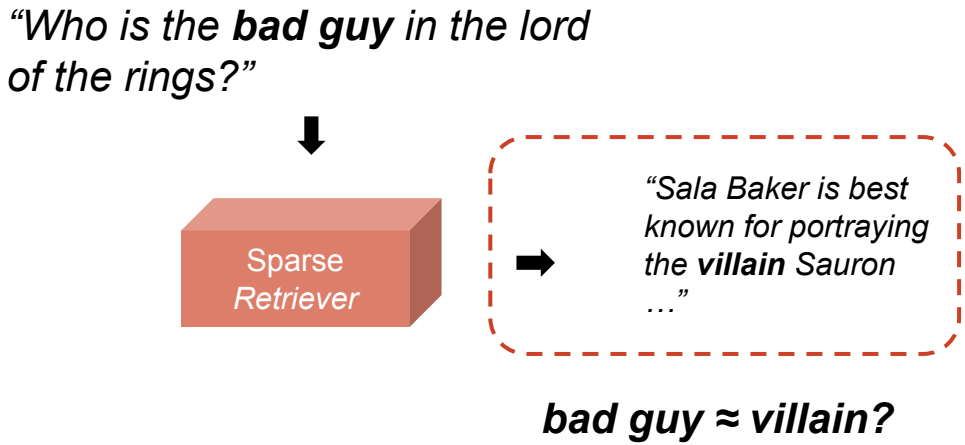
Usually implemented using TF-IDF or BM25

| Index → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| | **and** | **document** | **first** | **is** | **one** | **second** | **the** | **third** | **this** |
| *"This is the first document."* | 0 | 0.46979139 | 0.58028582 | 0.38408524 | 0 | 0 | 0.38408524 | 0 | 0.38408524 |
| *"This document is the second document."* | 0 | 0.6876236 | 0 | 0.28108867 | 0 | 0.53864762 | 0.28108867 | 0 | 0.28108867 |
| *"And this is the third one."* | 0.51184851 | 0 | 0 | 0.26710379 | 0.51184851 | 0 | 0.26710379 | 0.51184851 | 0.26710379 |
| *"Is this the first document?"* | 0 | 0.46979139 | 0.58028582 | 0.38408524 | 0 | 0 | 0.38408524 | 0 | 0.38408524 |

**Sparse** representations of words

# Retrieval

Usually implemented using TF-IDF or BM25





Difficulties understanding context


Learned task-specific representations

# Existing Methods: Retrieval with Dense Representations

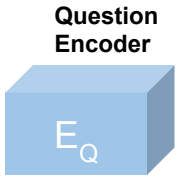
- **Computationally Intensive:** Require additional pretraining for the context encoder

- **Suboptimal representations:** Additional pretraining on regular sentences, not on question-answer pairs

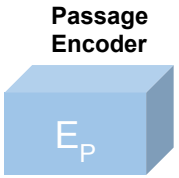
*“Can we train a better dense retriever using only pairs of questions and answers **without** additional pretraining?”*

# Dense Passage Retriever (DPR)

Two encoders




*"Who is the bad guy in the lord of the rings?"*




*"Sala Baker is best known for portraying the villain Sauron …"*

➡

- We need to max sim(q,p) = $E_Q(q)^T E_P(p)$
- Given a set D = {($q_i$, $p_i^+$, $p_{i1}^-$, $p_{i2}^-$, $p_{i3}^-$, …)}, we optimize L

$$L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) = -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\text{sim}(q_i, p_{i,j}^-)}}$$

*Negative log likelihood of positive passage*

# Dense Passage Retriever (DPR)

Picking positive passages

- From the Top-100 passages retrieved by BM25 for a question, they pick the highest ranked that contains the answer

Picking negative passages

- **Gold**: positive examples of other questions from same batch

If a batch has B pairs of questions and passages, use for each question the other B-1 passages as negatives

# Datasets

**Source** for documents/passages

- Wikipedia dump from Dec. 20, 2018
- Each article splitted into passages of 100 words
- 21 015 324 passages

| Dataset | Train | | Dev | Test |
|---|---|---|---|---|
| Natural Questions | 79,168 | 58,880 | 8,757 | 3,610 |
| TriviaQA | 78,785 | 60,413 | 8,837 | 11,313 |
| WebQuestions | 3,417 | 2,474 | 361 | 2,032 |
| CuratedTREC | 1,353 | 1,125 | 133 | 694 |
| SQuAD | 78,713 | 70,096 | 8,886 | 10,570 |

**Top-K Retrieval Accuracy:**
Correct if the answer is found within the Top-K retrieved documents

**Exact Match Accuracy:**
Correct if the exact answer is found

# Main Results

## Passage Retrieval

| Training | Retriever | Top-20 | | | | | Top-100 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | NQ | TriviaQA | WQ | TREC | SQuAD | NQ | TriviaQA | WQ | TREC | SQuAD |
| None | BM25 | 59.1 | 66.9 | 55.0 | 70.9 | 68.8 | 73.7 | 76.7 | 71.1 | 84.1 | 80.0 |
| Single | DPR | 78.4 | 79.4 | 73.2 | 79.8 | 63.2 | 85.4 | **85.0** | 81.4 | 89.1 | 77.2 |
| | BM25 + DPR | 76.6 | 79.8 | 71.0 | 85.2 | **71.5** | 83.8 | 84.5 | 80.5 | 92.7 | **81.3** |
| Multi | DPR | **79.4** | 78.8 | **75.0** | **89.1** | 51.6 | **86.0** | 84.7 | **82.9** | 93.9 | 67.6 |
| | BM25 + DPR | 78.0 | **79.9** | 74.7 | 88.5 | 66.2 | 83.9 | 84.4 | 82.3 | **94.1** | 78.6 |

Table 2: Top-20 & Top-100 retrieval accuracy on test sets, measured as the percentage of top 20/100 retrieved passages that contain the answer. *Single* and *Multi* denote that our Dense Passage Retriever (DPR) was trained using individial or combined training datasets (all the datasets excluding SQuAD). See text for more details.

# Main Results

## Passage Retrieval

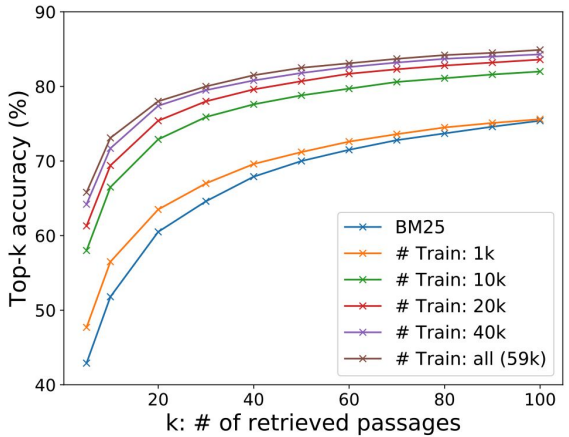



Figure 1: Retriever top-$k$ accuracy with different numbers of training examples used in our dense passage retriever vs BM25. The results are measured on the development set of Natural Questions. Our DPR trained using 1,000 examples already outperforms BM25.

| Type | #N | IB | Top-5 | Top-20 | Top-100 |
|---|---|---|---|---|---|
| Random | 7 | ✗ | 47.0 | 64.3 | 77.8 |
| BM25 | 7 | ✗ | 50.0 | 63.3 | 74.8 |
| Gold | 7 | ✗ | 42.6 | 63.1 | 78.3 |
| Gold | 7 | ✓ | 51.1 | 69.1 | 80.8 |
| Gold | 31 | ✓ | 52.1 | 70.8 | 82.1 |
| Gold | 127 | ✓ | 55.8 | 73.0 | 83.1 |
| G.+BM25$^{(1)}$ | 31+32 | ✓ | 65.0 | 77.3 | 84.4 |
| G.+BM25$^{(2)}$ | 31+64 | ✓ | 64.5 | 76.4 | 84.0 |
| G.+BM25$^{(1)}$ | 127+128 | ✓ | **65.8** | **78.0** | **84.9** |

Table 3: Comparison of different training schemes, measured as top-$k$ retrieval accuracy on Natural Questions (development set). #N: number of negative examples, IB: in-batch training. G.+BM25$^{(1)}$ and G.+BM25$^{(2)}$ denote in-batch training with 1 or 2 additional BM25 negatives, which serve as negative passages for all questions in the batch.

# Main Results

## Question Answering

| Training | Model | NQ | TriviaQA | WQ | TREC | SQuAD |
|---|---|---|---|---|---|---|
| Single | BM25+BERT (Lee et al., 2019) | 26.5 | 47.1 | 17.7 | 21.3 | 33.2 |
| Single | ORQA (Lee et al., 2019) | 33.3 | 45.0 | 36.4 | 30.1 | 20.2 |
| Single | HardEM (Min et al., 2019a) | 28.1 | 50.9 | - | - | - |
| Single | GraphRetriever (Min et al., 2019b) | 34.5 | 56.0 | 36.4 | - | - |
| Single | PathRetriever (Asai et al., 2020) | 32.6 | - | - | - | **56.5** |
| Single | $\text{REALM}_{\text{Wiki}}$ (Guu et al., 2020) | 39.2 | - | 40.2 | 46.8 | - |
| Single | $\text{REALM}_{\text{News}}$ (Guu et al., 2020) | 40.4 | - | 40.7 | 42.9 | - |
| Single | BM25 | 32.6 | 52.4 | 29.9 | 24.9 | 38.1 |
| | DPR | **41.5** | 56.8 | 34.6 | 25.9 | 29.8 |
| | BM25+DPR | 39.0 | 57.0 | 35.2 | 28.0 | 36.7 |
| Multi | DPR | **41.5** | 56.8 | **42.4** | 49.4 | 24.1 |
| | BM25+DPR | 38.8 | **57.9** | 41.1 | **50.6** | 35.8 |

Table 4: End-to-end QA (Exact Match) Accuracy. The first block of results are copied from their cited papers. $\text{REALM}_{\text{Wiki}}$ and $\text{REALM}_{\text{News}}$ are the same model but pretrained on Wikipedia and CC-News, respectively. *Single* and *Multi* denote that our Dense Passage Retriever (DPR) is trained using individual or combined training datasets (all except SQuAD). For WQ and TREC in the *Multi* setting, we fine-tune the reader trained on NQ.

# Limitations

“[...] DPR excels at semantic representation, but might lack sufficient capacity to represent salient phrases which appear rarely.

- Assumes questions are well formed
- Answers exist word-for-word in passages
- Relies on BM25 for training
- DPR is more expensive than BM25 in terms of training and inference due to bi-encoder model

How can we overcome these issues?

# Retrieve-Augmented Generation for Knowledge-Intensive NLP Tasks

# Problem of Interest

Pretrained LMs:

- Large pre-trained LMs are powerful
  - Store factual knowledge in their parameters

- Downsides:
  - Can’t expand or revise their memory
  - Hallucinations

# Problem of Interest

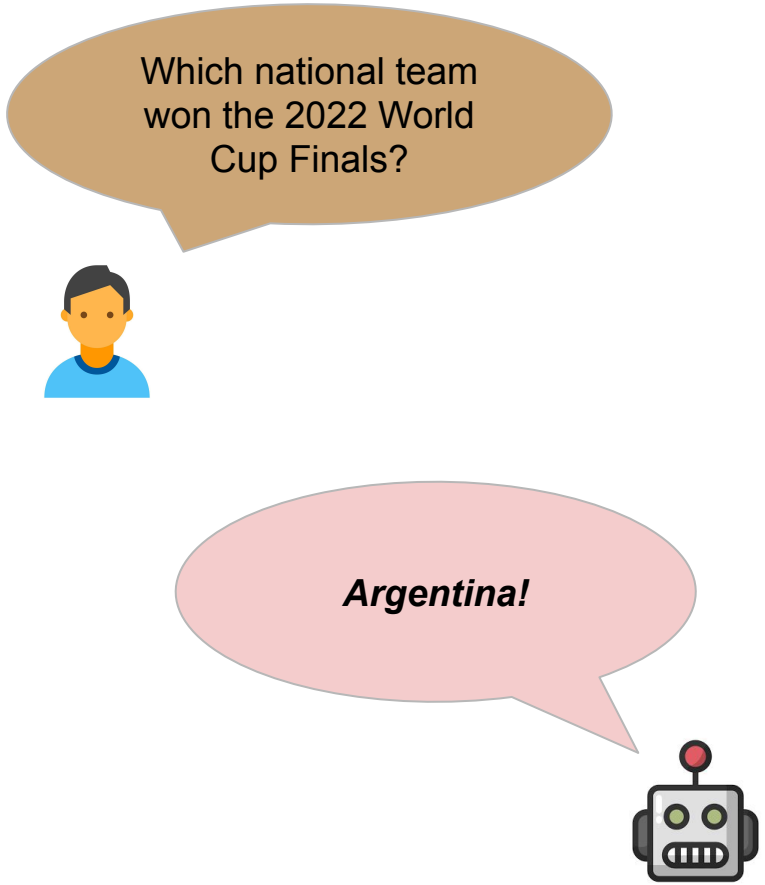
Retrieval

- Externally-retrieved knowledge is useful for NLP tasks
  - Accurate knowledge access
  - Trivial to update during inference time

- Downsides:
  - Requires labels for retrieval
  - If no labels, require “heuristics” retrieval (e.g., TF-IDF)
  - Modification required for downstream tasks

# Retrieval-Augmented Generation (RAG)

- End2end architecture that:
  - Learns to ***retrieve + generate***
- Latent Retrieval:
  - No labels needed for retrieved documents
- Applicable to any seq2seq task



# RAG-Architecture

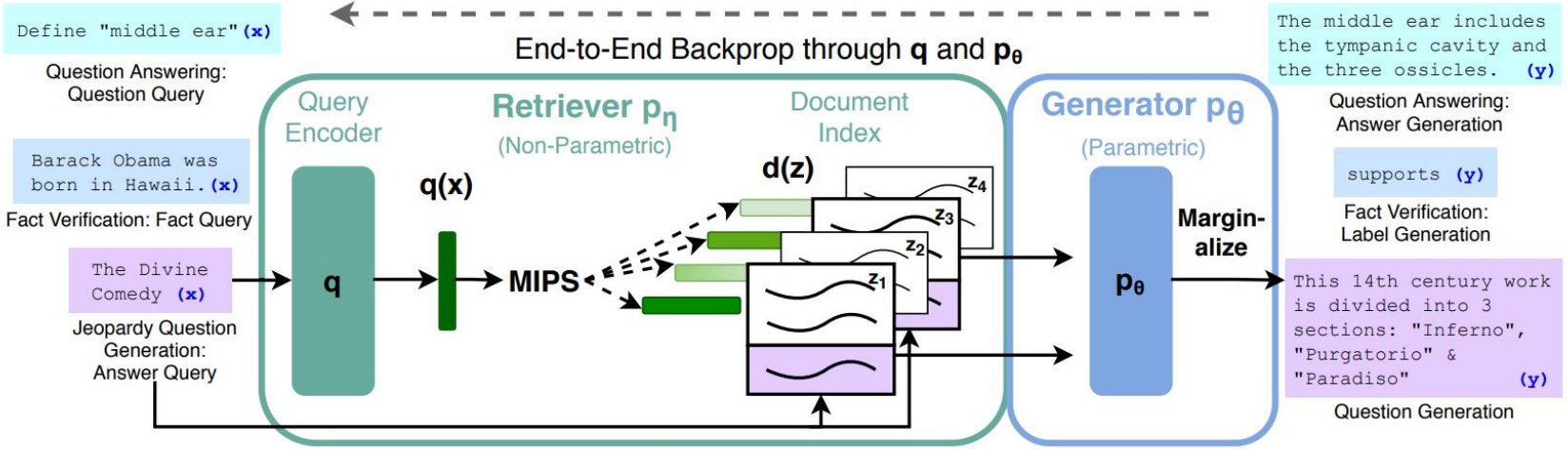



Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder* + *Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query $x$, we use Maximum Inner Product Search (MIPS) to find the top-K documents $z_i$. For final prediction $y$, we treat $z$ as a latent variable and marginalize over seq2seq predictions given different documents.

# Models

RAG - Sequence Model

- Same retrieved document to generate the ***complete sentence***

$$p_{\text{RAG-Sequence}}(y|x) \approx \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y|x,z) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) \prod_i^N p_\theta(y_i|x,z,y_{1:i-1})$$

Retriever

Generator

# Models

RAG - Token Model

- Draw a different latent document for ***each target token***

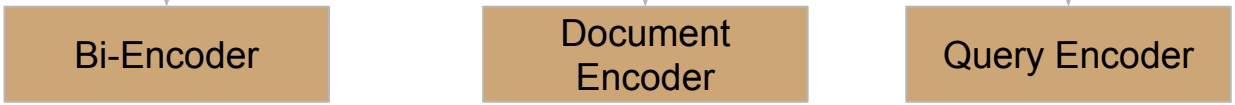
$$p_{\text{RAG-Token}}(y|x) \approx \prod_{i}^{N} \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y_i|x, z, y_{1:i-1})$$

Retriever

Generator

# Retriever: Dense Passage Retriever

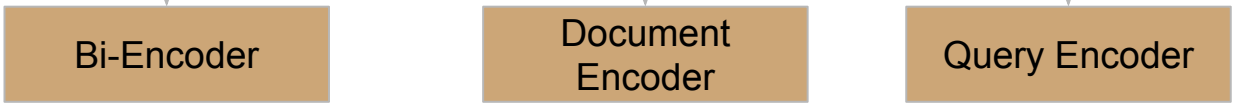
$$p_\eta(z|x) \propto \exp\left(\mathbf{d}(z)^\top \mathbf{q}(x)\right) \qquad \mathbf{d}(z) = \mathrm{BERT}_d(z), \quad \mathbf{q}(x) = \mathrm{BERT}_q(x)$$




- A Pretrained Bi-Encoder
  - Initialize retriever + build document index
  - ***Non-parametric memory***
- Document Encoder: encode Wikipedia Document only **once**
- Fine-tune Query Encoder

# Retriever: Dense Passage Retriever

$$p_\eta(z|x) \propto \exp\left(\mathbf{d}(z)^\top \mathbf{q}(x)\right) \qquad \mathbf{d}(z) = \mathrm{BERT}_d(z), \quad \mathbf{q}(x) = \mathrm{BERT}_q(x)$$




- Goal: calculate the top-k documents z with highest probability: $p_\eta(z|x)$
- Maximum Inner Product Search Problem (MIPS)
  - Solved in sub-linear time

# Generator: BART

- BART-Large, a pre-trained seq2seq transformer with 400M parameters
  - Pretrained using: denoising objective + different noising functions
- ***Parametric Memory***
- Can use other architectures for Generator: e.g., GPT

# Decoding from RAG Models

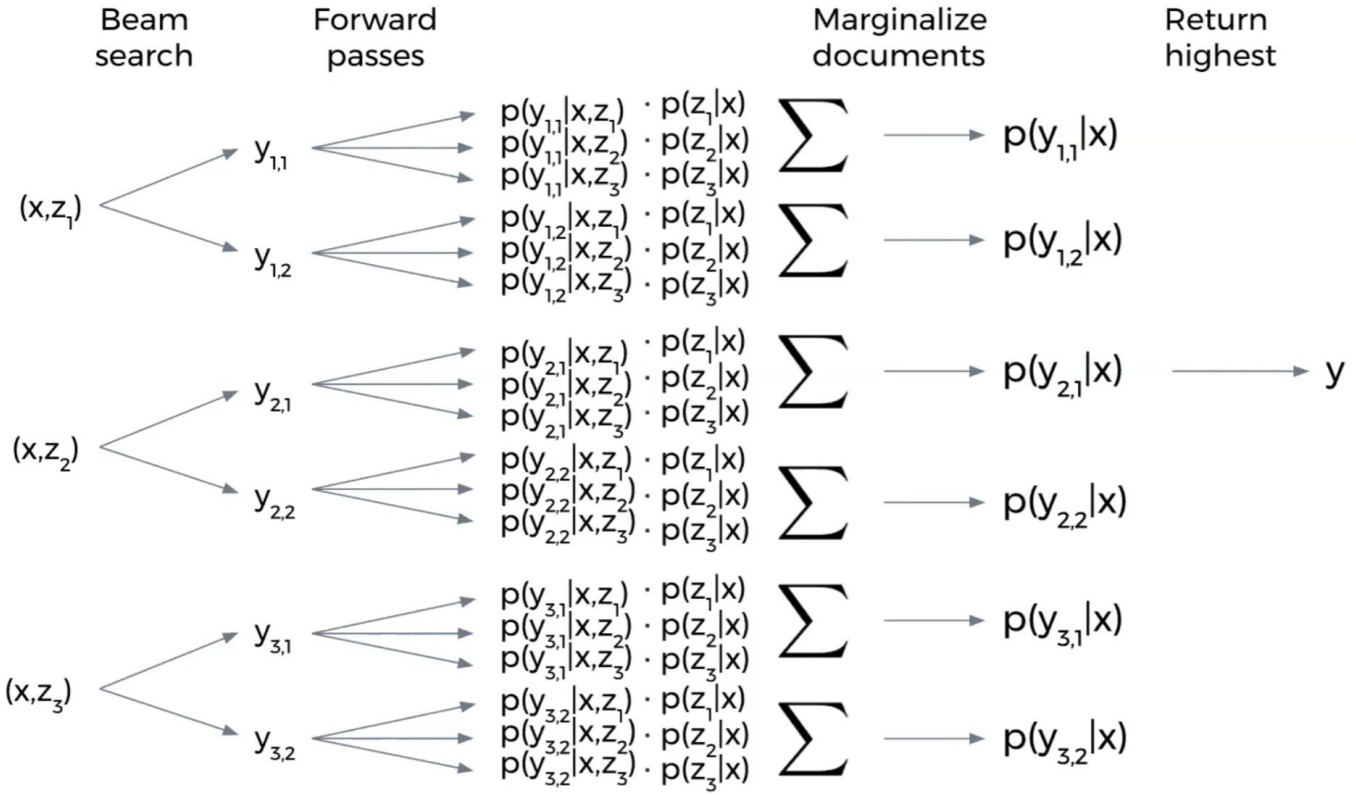
- At inference time: RAG - Sequence & RAG - Token require different ways to approximate output token

- RAG - Token:

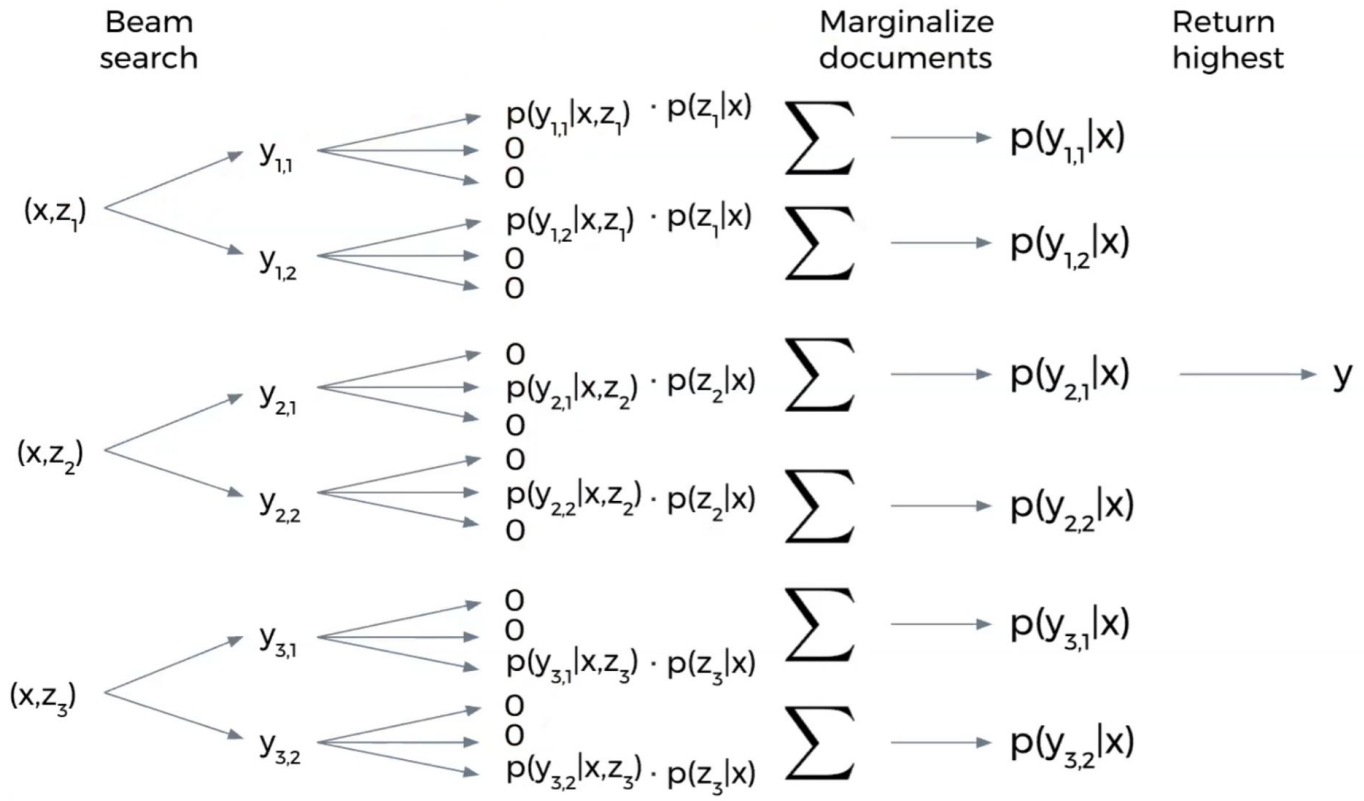
$$p'_\theta(y_i|x, y_{1:i-1}) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z_i|x) p_\theta(y_i|x, z_i, y_{1:i-1})$$

- RAG - Sequence:
  - p (y | x) does not break into a conventional per-token likelihood
  - Cannot solve it with **a single beam search**

# Decoding from RAG Models: RAG - Sequence



# Decoding from RAG Models: RAG - Sequence



# Experiments

Settings:

- All experiments ⇒ use a single Wikipedia dump (December 2018 Dump)
- Each Wikipedia article ⇒ split into disjoint 100-word chunks
  - A total of 21M documents
- Document Encoder ⇒ an embedding for each document ⇒ build a single MIPS index
- Training ⇒ retrieve top k documents for each query ($k \in \{5, 10\}$)

# Experiments

Dataset Types

- Open-domain Question Answering
  - Natural Questions, TriviaQA, WebQuestions, CuratedTREC
- Abstractive Open-domain QA
  - MS MARCO
- Question Generation
  - Jeopardy Questions
- Fact Verification (classification task)
  - FEVER

# Results: Main

- OpenDomain QA

Table 1: Open-Domain QA Test Scores. For TQA, left column uses the standard test set for Open-Domain QA, right column uses the TQA-Wiki test set. See Appendix D for further details.

| | Model | NQ | TQA | WQ | CT |
|---|---|---|---|---|---|
| Closed Book | T5-11B [52] | 34.5 | - /50.1 | 37.4 | - |
| | T5-11B+SSM[52] | 36.6 | - /60.5 | 44.7 | - |
| Open Book | REALM [20] | 40.4 | - / - | 40.7 | 46.8 |
| | DPR [26] | 41.5 | **57.9**/ - | 41.1 | 50.6 |
| | RAG-Token | 44.1 | 55.2/66.1 | **45.5** | 50.0 |
| | RAG-Seq. | **44.5** | 56.8/**68.0** | 45.2 | **52.2** |

# Results: Main

- Abstractive Open-domain QA
- Jeopardy Question Generation
- Fact Verification

Table 2: Generation and classification Test Scores. MS-MARCO SotA is [4], FEVER-3 is [68] and FEVER-2 is [57] *Uses gold context/evidence. Best model without gold access underlined.

| Model | Jeopardy B-1 | Jeopardy QB-1 | MSMARCO R-L | MSMARCO B-1 | FVR3 Label Acc. | FVR2 Label Acc. |
|---|---|---|---|---|---|---|
| SotA | - | - | **49.8*** | **49.9*** | **76.8** | **92.2*** |
| BART | 15.1 | 19.7 | 38.2 | 41.6 | 64.0 | 81.1 |
| RAG-Tok. | **17.3** | **22.2** | 40.1 | 41.5 | 72.5 | <u>89.5</u> |
| RAG-Seq. | 14.7 | 21.4 | <u>40.8</u> | <u>44.2</u> | | |

# Results: Jeopardy Question Generation Example

When generating “sun” ⇒ Document 2 (..... “The Sun Also Rises”, .....)

When generating “A Farewell to Arms” ⇒ Document 1 (..... his novel “A Farewell to Arsms”.....)

**Document 1**: his works are considered classics of American literature ... His wartime experiences formed the basis for his novel **"A Farewell to Arms"** (1929) ...

**Document 2**: ... artists of the 1920s "Lost Generation" expatriate community. His debut novel, **"The Sun Also Rises"**, was published in 1926.

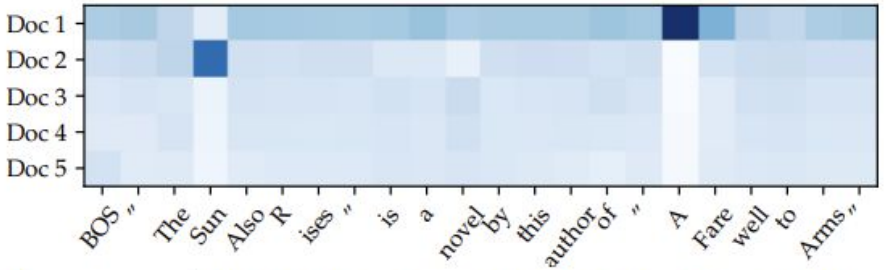



Figure 2: RAG-Token document posterior $p(z_i|x, y_i, y_{-i})$ for each generated token for input “Hemingway" for Jeopardy generation with 5 retrieved documents. The posterior for document 1 is high when generating “A Farewell to Arms" and for document 2 when generating “The Sun Also Rises".

# Results: Ablations

Table 6: Ablations on the dev set. As FEVER is a classification task, both RAG models are equivalent.

| Model | NQ | TQA | WQ | CT | Jeopardy-QGen | | MSMarco | | FVR-3 | FVR-2 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Exact Match | | | B-1 | QB-1 | R-L | B-1 | Label Accuracy | |
| RAG-Token-BM25 | 29.7 | 41.5 | 32.1 | 33.1 | 17.5 | 22.3 | 55.5 | 48.4 | **75.1** | **91.6** |
| RAG-Sequence-BM25 | 31.8 | 44.1 | 36.6 | 33.8 | 11.1 | 19.5 | 56.5 | 46.9 | | |
| RAG-Token-Frozen | 37.8 | 50.1 | 37.1 | 51.1 | 16.7 | 21.7 | 55.9 | 49.4 | 72.9 | 89.4 |
| RAG-Sequence-Frozen | 41.2 | 52.1 | 41.8 | 52.6 | 11.8 | 19.6 | 56.7 | 47.3 | | |
| RAG-Token | 43.5 | 54.8 | **46.5** | 51.9 | **17.9** | **22.6** | 56.2 | **49.4** | 74.5 | 90.6 |
| RAG-Sequence | **44.0** | **55.8** | 44.9 | **53.4** | 15.3 | 21.5 | **57.2** | 47.5 | | |

# Results: Parameter Sensitivity for K

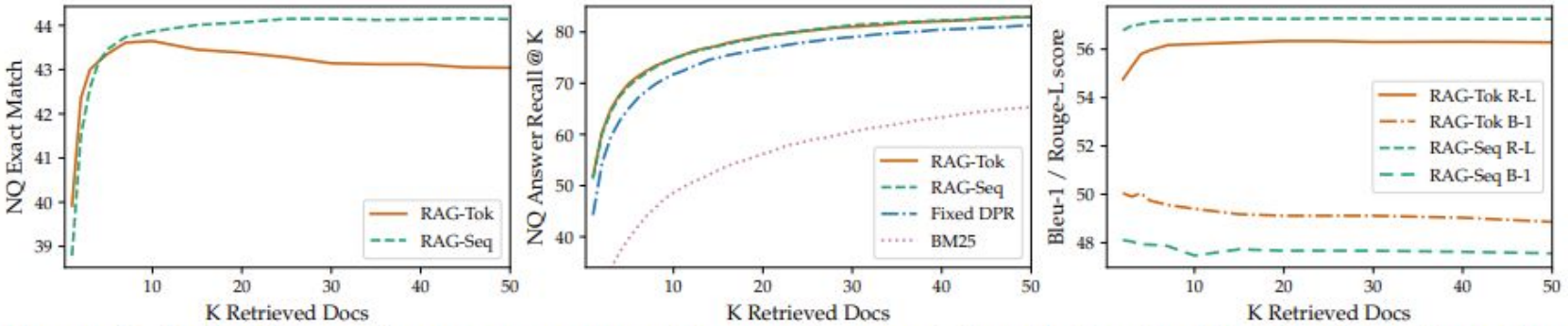



Figure 3: Left: NQ performance as more documents are retrieved. Center: Retrieval recall performance in NQ. Right: MS-MARCO Bleu-1 and Rouge-L as more documents are retrieved.

# Limitations

- Retriever needs to be trained jointly with the generator ⇒ computationally expensive
- Retrieval here is performed only once for each query: some multi-hop questions might require multiple retrievals with potentially different queries
- Low retrieval quality: not all blocks within the retrieval set correlate with the query, leading potential hallucinations.

# References


- Original Paper: https://arxiv.org/pdf/2005.11401
- Paper Explanation Video: https://www.youtube.com/watch?v=JGpmQvlYRdU&t=1834s
- RAG Explanation Overall: https://www.youtube.com/watch?v=rhZgXNdhWDY
- RAG Survey paper: https://simg.baai.ac.cn/paperfile/25a43194-c74c-4cd3-b60f-0a1f27f8b8af.pdf

# Summary

Large language models store factual knowledge in their parameters, but accessing and manipulating it precisely is still challenging, struggling with knowledge intensive tasks

- *Khandelwal et al. (2020)* showed how this prediction problem can be tackled from the representation learning side by augmenting the model knowledge through KNN
- *Karpukhin et al. (2020)* expanded upon this idea by improving the retrieval process relying on dense representations aline
- Lewis et al. (2021) introduced RAG consolidating previous ideas of parametric memory into a more efficient technique

# Thank you!

Questions?