

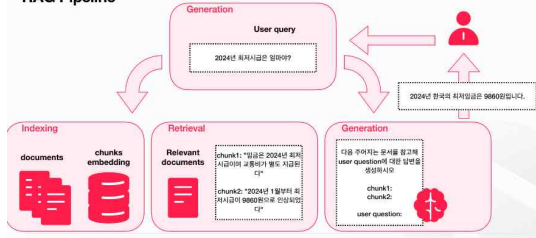
LLM의 한계

- 환각
- 오래된 데이터
- 부정확한 답변
- 알 수 없는 추론과정

RAG Pipeline



RAG Pipeline



Indexing : 문서들을 처리하는 단계. 나중에 쿼리에 답변할 때 사용할 데이터들. 이때 임베딩을 미리 만들어놓음.

Retrieval : 리트리버에서는 인덱싱 단계에서 잘 청킹한 것들 중에 필요한 걸 가져오는 것이 중요. 어느 것이 쿼리와 유사한지 잘 판단해야 함. 발전을 거침

Generation : 프롬프트 합성.

Plain LM : 파라미터에서 텍스트 패턴을 학습. 코퍼스가 크면 지식 학습. 새 지식이 들어오면 그것을 학습함. 요즘은 데이터가 커서 괜찮.

Search System : TF-IDF, BM25, Semantic search 등 유사도를 평가하여 검색 결과를 뱉어냄. 전달받은 결과를 유저가 직접 해석해야 함

RAG = Plain LM, Search Sys.

Static RAG



TF-IDF(Term Frequency - Inverse Document) :: 문서에서 단어의 상대적 중요도를 파악하는데 사용. 자주 등장하고 덜 자주 등장하는 정도에 따라 다른 가중치 부여. 특정 문서에만 자주 등장하는 단어에 반응.

BM25 문서의 길이 등을 고려하여 좀 더 정교하게 측정

Generation : Pretrained LM + No Fine-tuning. 별도의 학습이 이루어지지 않기 때문에 쿼리에 답변할 때 학습 발생

What

- Granularity : 청킹 사이즈는 어떻게?
- source : 표로 된 데이터(structured) | 텍스트 데이터 | LLM으로 생성 증감한 데이터 ?

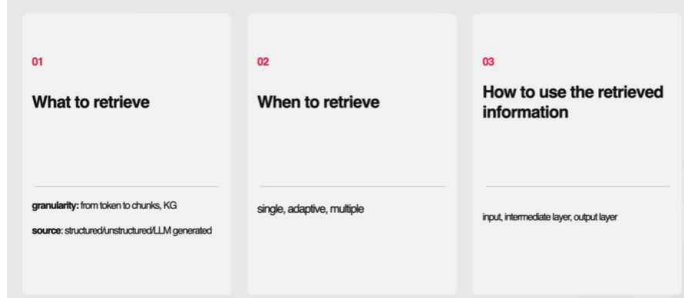
When :: 유저의 쿼리가 리트리버가 필요하지 않을 수도 있음

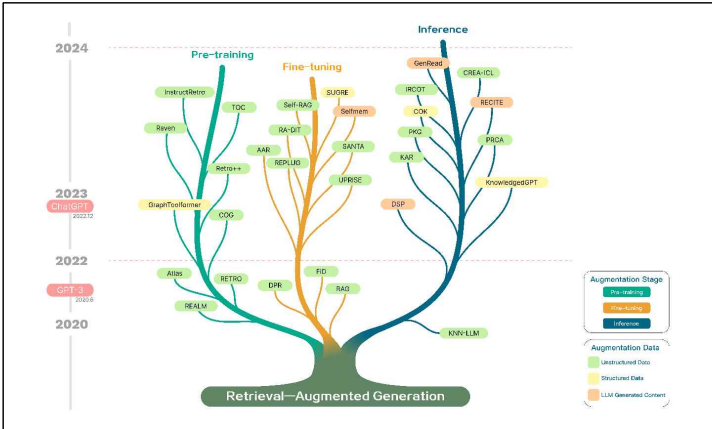
- adaptive : 쿼리를 보고 선택적으로 리트리버. 한 번에 가져온다는 보장은 없음 -> recursive, iterative 리트리버 사용

How to use

- 인풋에 retrieve + 유저쿼리 제공
- 인풋 대신 중간 레이어 or output 레이어와 합쳐서 제공하기도
- 어떻게 합칠지도 고민

Pivotal Questions in RAG





Fundamental papers in RAG

KNN-LM (2019)

- inference시에 explicit memory 사용
- decoder-only Transformers

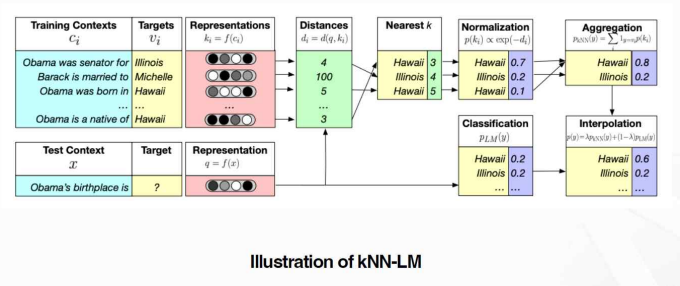
REALM (2020)

- Retrieval-Augmented Pre-training
- BERT Based

RAG (2020)

- Retrieval-Augmented Fine-tuning
- fine-tuning only
- BERT encoder, BART generator

kNN-LM



- 다음 토큰 예측을 위해 이전 임베딩 기억 활용
- 학습 과정에서 등장한 토큰과 그에 해당하는 임베딩을 DB에 저장하고 다음 토큰 예측할 때 비슷한 문맥에서 사용된 단어를 찾아서 확률에 참고
- 드물게 등장하는 토큰에 효과적
- 왼쪽 위가 지난 임베딩의 데이터 스토어
- 보통의 key-value처럼 key인 임베딩을 보고 value인 토큰을 찾아가
- inference 타임에 질문이 들어오면 두 가지 방법 결합
 1. 문장을 디코더에 넣고 리니어 소프트맥스를 거침
 2. inference 임베딩과 기존 DB 임베딩 비교
 - 1) k개의 value 뽑아냄. 거리 기반해서 뽑아낸 것
 - 2) 정규화 후 통합

kNN-LM Datastore



입력 문장이 Decoder > linear > softmax를 거치면서 중간의 값들을 이용
 실제 학습데이터를 가지고 임베딩 데이터 스토어를 구성

- 다음 단어 예측을 위해 Data store를 참고
- 모든 데이터와 타겟 스토어를 가지고 있음
- 현재 문맥의 임베딩과 데이터 저장소의 임베딩 비교 > 유클리디안 거리

kNN-LM Inference

Context: 위대한 개츠비의 작가인, 개츠비는 고향을 떠나, 낙원의 저편을 집필한, 피츠제럴드가 졸업한

Key: 피츠제럴드, 뉴욕, 피츠제럴드, 프린스턴

Distance: 3, 5, 6, 10

Decoder: 개츠비는, 미국의, 소설가인

kNN-LM Inference: p_{kNN}

Context:위대한 개츠비의 작가인, 개츠비는 고향을 떠나, 낙원의 저편을 집필한, 피츠제럴드가 졸업한

Key: 피츠제럴드, 뉴욕, 피츠제럴드, 프린스턴

Dist: 3, 4, 5, 10

Softmax(-d): 0.7, 0.2, 0.1, 10

Top-k Nearest: 3, 4, 5, 10

Agg: 피츠제럴드 0.8, 뉴욕 0.2

Decoder: 개츠비는, 미국의, 소설가인

- 샘플이 뽑힐 가능성을 계산
- 피츠제럴드가 두 번 등장 > 중복
- 이때 확률을 합쳐줌

kNN-LM Inference: interpolation

Context:위대한 개츠비의 작가인, 개츠비는 고향을 떠나, 낙원의 저편을 집필한, 피츠제럴드가 졸업한

Key: 피츠제럴드, 뉴욕, 피츠제럴드, 프린스턴

Dist: 3, 4, 5, 10

Softmax(-d): 0.7, 0.2, 0.1, 10

Top-k Nearest: 3, 4, 5, 10

Agg: 피츠제럴드 0.8, 뉴욕 0.2

Decoder: 개츠비는, 미국의, 소설가인

Interpolation: $p(y|x) = \lambda p_{kNN}(y|x) + (1 - \lambda) p_{LM}(y|x)$

- 단순히 Data store로만하는건 아님
- 기존 언어모델식 접근법 + kNN 방법론 모두 사용
- 유사 컨텍스트로 얻게 되는 확률 : P_{kNN}
- 전통방식 : P_{LM}

=> 파라미터 람다로 최종 확률 (선형보간)

kNN-LM: evaluation

Perplexity: $PPL(W) = (\prod_{i=1}^N p(w_i | w_1, w_2, \dots, w_{i-1}))^{-1/N}$

Model	Perplexity (↓)		# Trainable Params
	Dev	Test	
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
+kNN-LM	16.06	16.12	247M
+Continuous Cache (Grave et al., 2017c)	17.67	18.27	247M
+kNN-LM + Continuous Cache	15.81	15.79	247M

Table 1: Performance on WIKITEXT-103. The kNN-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.

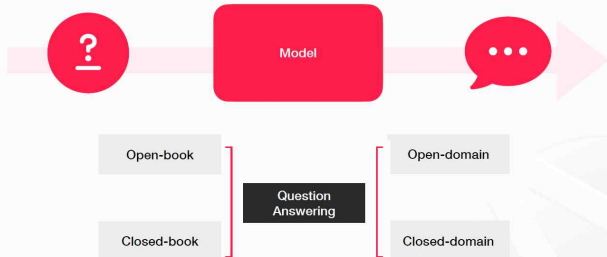
Model	Perplexity (↓)		# Trainable Params
	Dev	Test	
Base LM (Baevski & Auli, 2019)	14.75	11.89	247M
+kNN-LM	14.20	10.89	247M

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

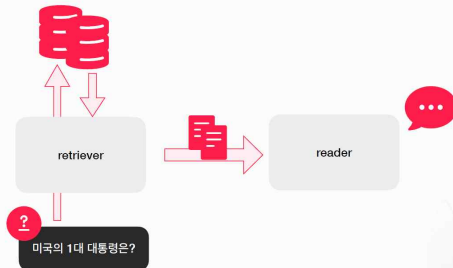
- 목표는 perplexity를 낮추는 것. 평균적으로 후보를 줄이는 것. 이게 작을수록 모델이 더 신뢰성 있는 결과를 도출한다고 볼 수 있음

<p>REALM</p>	<ul style="list-style-type: none"> - 리트리버 Augmented Pre training - BERT Based - 목적 : QA문제 해결 <ul style="list-style-type: none"> - open book(domain) : 정답 포함 문맥에 접근 가능 상태. - open domain : 도메인 미지정 (ex 퀴즈) - closed book : 로직만으로만 답 도출 - close domain : 분야 한정
---------------------	---

Question Answering



Retrieve-then-predict



- 리트리버 reader 구조라고도 함
- RAG와 거의 비슷
- 리트리버는 쿼리와 관련한 문서들 탐색
- 리더에게 전달
- 전달 받은 문서들로부터 답변 추출
- 여기서 생성이 아니라 추출이라는 건 인코더 모델 BERT를 사용하기 때문. 대답을 지목하는 것
- 결과로서 나오는 것은 문장이 아닌 span

- 중간 retrieval 문서 종류와 무관
- 모든 z 에 대해서 쿼리가 주어졌을 때 z 가 찾아질 확률을 구하고 이 확률과 z, x 가 주어졌을 때 y 가 출력될 확률을 구함
- 즉 x, y, z 가 다 나올 확률
- 즉 모델에서 순서대로 나올 확률을 구하는 것
- x 가 쿼리일 때 y 출력 확률과 같음

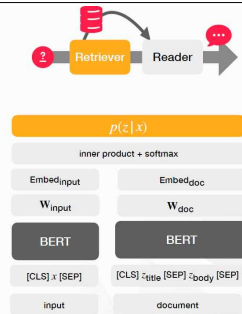
Neural Knowledge Retriever

* \mathcal{X} : knowledge corpus

$$p(y|x) = \sum_{z \in \mathcal{X}} p(y|z, x) p(z|x)$$

$$p(z|x) = \frac{\exp f(x, z)}{\sum_{z'} \exp f(x, z')}$$

* relevance score: $f(x, z) = \text{Embed}_{\text{input}}(x)^T \text{Embed}_{\text{doc}}(z)$



1. 입력과 문서를 BERT로 임베딩 벡터로 변화
 - 1) 이때 문서는 문장을 분리하는 토큰을 포함
2. 가중치를 곱한 후 유사도를 가짐
3. 전체 문서에 대해 soft max => 문서가 선택될 확률

Neural Knowledge Retriever

* \mathcal{X} : knowledge corpus

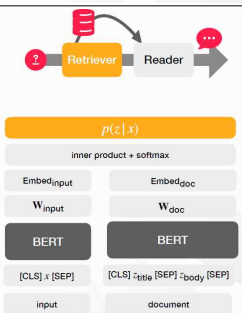
$$p(y|x) = \sum_{z \in \mathcal{X}} p(y|z, x) p(z|x) \approx \sum_{z \in \text{topk}(x)} p(y|z, x) p(z|x)$$

$$p(z|x) = \frac{\exp f(x, z)}{\sum_{z'} \exp f(x, z')}$$

* relevance score: $f(x, z) = \text{Embed}_{\text{input}}(x)^T \text{Embed}_{\text{doc}}(z)$

* $\text{Embed}_{\text{input}}(x) = \mathbf{W}_{\text{input}} \text{BERT}_{\text{CLS}}(\text{joinBERT}(x))$

* $\text{Embed}_{\text{doc}}(z) = \mathbf{W}_{\text{doc}} \text{BERT}_{\text{CLS}}(\text{joinBERT}(z_{\text{title}}, z_{\text{body}}))$



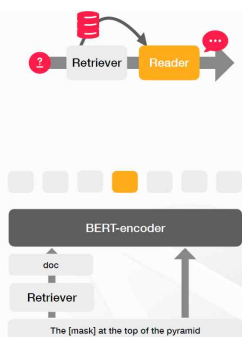
- 이후 reader 과정에서 문서가 많아질수록 계산량 많아짐
- 그래서 나중에는 연관 점수가 높은 k개에 대해서만 수행

Knowledge-Augmented Encoder: Pre-train

Masked Language Modeling

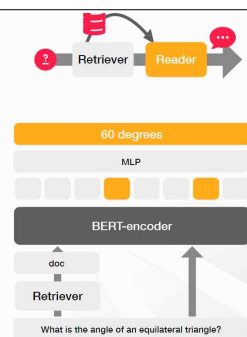
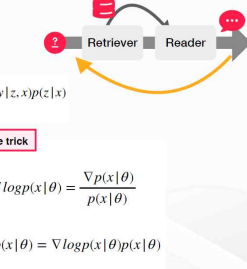
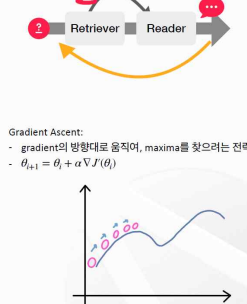
$$p(y_j|z, x) = \prod_{j=1}^J p(y_j|z, x)$$

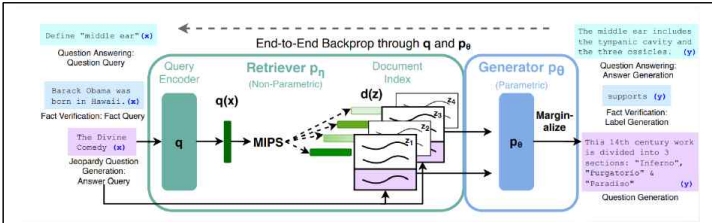
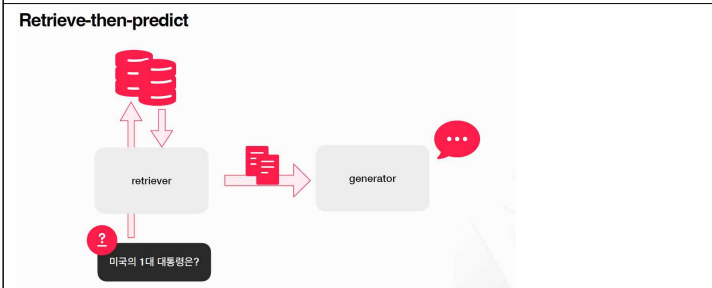
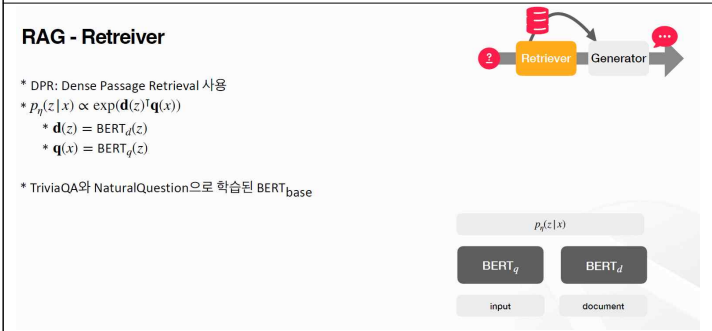
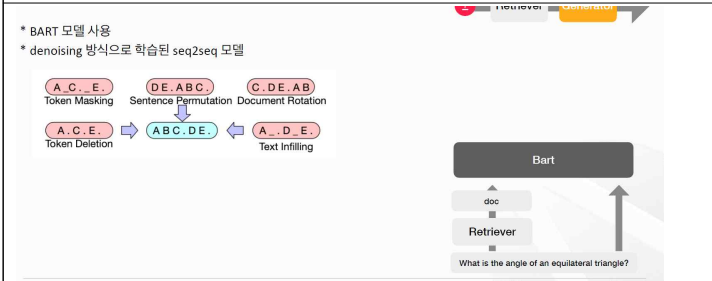
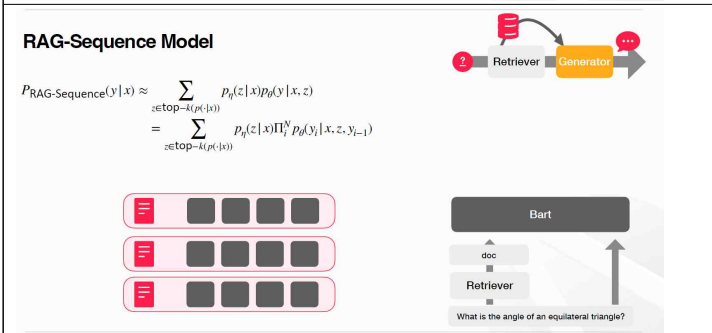
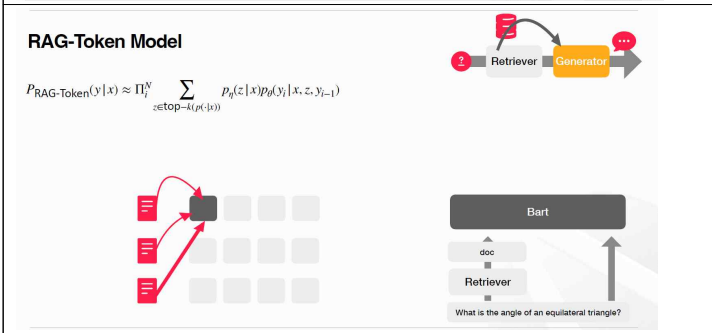
$$p(y_j|z, x) \propto \exp(w_j^T \text{BERT}_{\text{MASK}}(f_j(\text{joinBERT}(x, z_{\text{body}}))))$$



Reader

- 인코더는 쿼리와 함께 docs를 가짐
- retriever된 docs와 입력 쿼리 x 를 결합해 BERT로 넣음 *아까 BERT랑은 다름
- Pre / fine tuning 다 함
- Pre training :: 마스킹 => 랜덤 단어 마스킹 후 복원
 - 각각 masked는 독립적이라 곱
- fine tuning :: QA
 - 인코더의 거친 결과가 y_j 와 유사해질수록 선택 확

<p>Knowledge-Augmented Encoder: Finetuning</p> <p>Open-QA Fine-tuning</p> <ul style="list-style-type: none"> * $p(y z, x) \propto \sum_{a \in S(z, y)} \exp(\text{MLP}([h_{\text{start}(a)}; h_{\text{end}(a)}]))$ * $h_{\text{start}(a)} = \text{BERT}_{\text{start}(a)}(\text{join}(\text{BERT}(x, z_{\text{body}})))$ * $h_{\text{end}(a)} = \text{BERT}_{\text{end}(a)}(\text{join}(\text{BERT}(x, z_{\text{body}})))$ * $S(z, y)$: set of spans matching y in z 	<p>를 증가</p> <p>Open QA Fine tuning</p> <ul style="list-style-type: none"> - 주어진 부분에서 정답 어딴지 찾음 - 각각의 span을 찾고 정답이 될 확률을 구함 - 시작점과 끝점에 BERT를 취함 - span이 결과일 확률은 feed forward neural network에 거침
<p>Joint training What does the retriever learn?</p> $\nabla \log p(y x) = p(y x)^{-1} \nabla p(y x)$ $= p(y x)^{-1} \sum_z p(y x, z) \nabla p(z x)$ $= p(y x)^{-1} \sum_z p(y x, z) p(z x) \nabla \log p(z x)$ <p>recall $p(z x) = \frac{\exp f(x, z)}{\sum_z \exp f(x, z)}$</p> <p>log derivative trick $\nabla \log p(x \theta) = \frac{\nabla p(x \theta)}{p(x \theta)}$</p> <p>$\nabla p(x \theta) = \nabla \log p(x \theta) p(x \theta)$</p> 	<p>동시학습 :: 리트리버는 무엇을 학습하는가?</p> <ul style="list-style-type: none"> - 리트리버의 학습이 유효함을 어떻게 아는가? - 리트리버가 어떻게 학습되는가? <p><수학적 유도></p>
<p>Joint training What does the retriever learn?</p> $\nabla \log p(y x) = \sum_{z \in \mathcal{Z}} r(z) \nabla f(x, z)$ $r(z) = \left(\frac{p(y z, x)}{p(y x)} - 1 \right) p(z x)$ <p>Gradient Ascent:</p> <ul style="list-style-type: none"> - gradient의 방향으로 움직여, maxima를 찾으려는 전략 - $\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$ 	<ul style="list-style-type: none"> - f(x,z)가 학습해야할 부분 - 얼마나 관련있는가? - r(z) : 문서 전달 효과가 긍정적인지를 판단. 이게 최대가 되는 방향에서 최적화를 해야함
<p>Injecting inductive biases</p> <div> <div> <p>01</p> <p>Salient span masking</p> <ul style="list-style-type: none"> * 문장내 중요한 부분 masking * BERT-based Tagger 사용 <p>미국이 1차(대통령은 조지 워싱턴이다) => 미국의 1차(대통령은 [MASK])</p> </div> <div> <p>02</p> <p>Null document</p> <ul style="list-style-type: none"> * 정답 추론에 문서가 필요하지 않을 때가 있음 * top-k doc에 null doc 포함 </div> <div> <p>03</p> <p>Prohibiting trivial retrievals</p> <ul style="list-style-type: none"> * Pre-training 기간 동안 trivial candidate 제외 </div> <div> <p>04</p> <p>Initialization</p> <ul style="list-style-type: none"> * retriever: ICT trained BERT * encoder: uncased BERT-base </div> </div>	<p>성능을 높이기 위해 쓴 네 가지 방식</p> <ol style="list-style-type: none"> 1. Salient span masking <ul style="list-style-type: none"> - 문장 내 중요한 부분 마스킹 2. Null document <ul style="list-style-type: none"> - 정답추론에 문서가 필요하지 않을 때. - top k -doc에 빈 문서를 포함 3. Prohibiting trivial retrievals <ul style="list-style-type: none"> - 사전 훈련 기간동안 사소한 후보는 제외하고 진행 4. 초기화 <ul style="list-style-type: none"> - 리트리버와 리버가 처음부터 training 되면 리트리버는 랜덤. 리더 입장에서는 항상 참조하면 마이너스 > 리트리버 문서 무시 - 이를 방지하기 위해 간단한 걸로 warm starting. 리트리버는 ICT 트레이닝

 <p>Define "middle ear" (x) Question Answering: Question Query Barack Obama was born in Hawaii. (x) Fact Verification: Fact Query The Divine Comedy (x) Jeopardy Question Generation: Answer Query</p> <p>End-to-End Backprop through q and p_g</p> <p>Retriever p_q (Non-Parametric) Document Index Generator p_g (Parametric) Marginalize</p> <p>The middle ear includes the tympanic cavity and the three ossicles. (y) Question Answering: Answer Generation supports (y) Fact Verification: Label Generation This 14th century work is divided into 3 sections: "Inferno", "Purgatorio" & "Paradiso". (y) Question Generation</p>	<ul style="list-style-type: none"> - pre train 없이 fine tuning만 존재 - REALM과 거의 유사
<p>Retrieve-then-predict</p>  <p>retriever → generator</p> <p>미국의 1대 대통령은?</p>	<ul style="list-style-type: none"> - 넘겨주는 대상이 reader가 아니라 generator라는 점에서 차이가 있음
<p>main goal</p> <p>$p(y x)$</p> <p>retrieve</p> <p>generate</p> <p>$p(z x)$: 예측에 도움이 될 $z \in \mathcal{Z}$ 리트리벌</p> <p>$p(y_i z, x, y_{1:i-1})$: z와 x, 이전 시점의 토큰들이 주어졌을 때 출력값 y 예측</p>	<ul style="list-style-type: none"> - REALM과 거의 흡사 - generate 공식이 약간 다름. 직전에 예측한 토큰도 포함
<p>RAG - Retriever</p> <p>* DPR: Dense Passage Retrieval 사용</p> <p>* $p_q(z x) \propto \exp(\mathbf{d}(z)^T \mathbf{q}(x))$</p> <p>* $\mathbf{d}(z) = \text{BERT}_d(z)$</p> <p>* $\mathbf{q}(x) = \text{BERT}_q(x)$</p> <p>* TriviaQA와 NaturalQuestion으로 학습된 BERT base</p>  <p>Retriever → Generator</p> <p>What is the angle of an equilateral triangle?</p>	<ul style="list-style-type: none"> - 위와 같은 Retriever 구조를 가짐 - 인코더로 BERT를 쓰고 내적으로 활용
<p>* BART 모델 사용</p> <p>* denoising 방식으로 학습된 seq2seq 모델</p> <p>Token Masking: A . C . . E . → A B C . D E .</p> <p>Sentence Permutation: D E . A B C . → A B C . D E .</p> <p>Document Rotation: C . D E . A B → A B C . D E .</p> <p>Token Deletion: A . C . E . → A B C . D E .</p> <p>Text Infilling: A . C . E . → A B C . D E .</p>  <p>Bart</p> <p>doc</p> <p>Retriever</p> <p>What is the angle of an equilateral triangle?</p>	<ul style="list-style-type: none"> - Generator로 BART 사용 - denoising 방식으로 학습된 생성모델
<p>RAG-Sequence Model</p> $P_{\text{RAG-Sequence}}(y x) \approx \sum_{z \in \text{top-}k(p(z x))} p_q(z x) p_g(y x, z)$ $= \sum_{z \in \text{top-}k(p(z x))} p_q(z x) \prod_{i=1}^N p_g(y_i x, z, y_{1:i-1})$  <p>Bart</p> <p>doc</p> <p>Retriever</p> <p>What is the angle of an equilateral triangle?</p>	<ul style="list-style-type: none"> - 옵션 두 개 1. 시퀀스 모델 : <ul style="list-style-type: none"> - 하나의 시퀀스를 보고 문서 하나씩 보고 출력. top-k 방식 활용 - 상위 k개의 문서 중 출력 확률이 제일 높은거 선택
<p>RAG-Token Model</p> $P_{\text{RAG-Token}}(y x) \approx \prod_{i=1}^N \sum_{z \in \text{top-}k(p(z x))} p_q(z x) p_g(y_i x, z, y_{1:i-1})$  <p>Bart</p> <p>doc</p> <p>Retriever</p> <p>What is the angle of an equilateral triangle?</p>	<ul style="list-style-type: none"> 2. Token Model <ul style="list-style-type: none"> - 반대로 개별적인 토큰을 생성할 때 모든 문서 고려 - 다음 토큰에서도 또 모든 문서 고려 - 위와는 시그마와 파이 순서가 반대. - 생성시마다 모두 고려

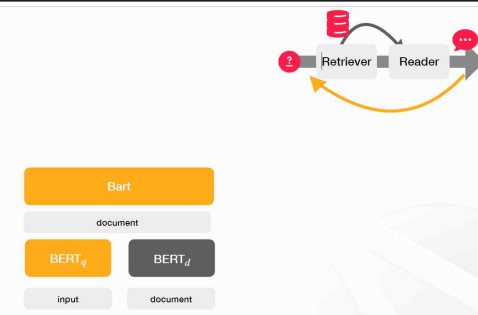
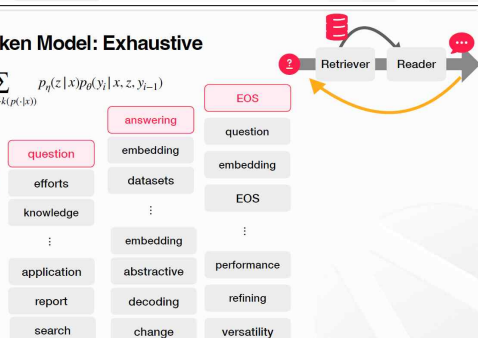
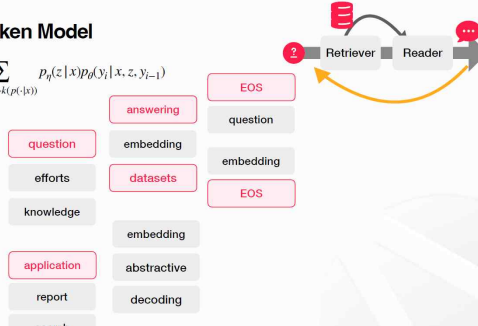
<div>Joint training</div> <div>Training loss: $\sum_j -\log p(y_j x_j)$</div> <div></div>	<div><div>- Document 인코더는 학습 X</div><div>- 파라미터 변하면 임베딩 바뀌고 이러면 계산량 불필요하게 증가</div></div>
<div>Decoding: RAG-Token Model: Greedy</div> <div>$P_{\text{RAG-Token}}(y x) \approx \prod_i^N \sum_{z \in \text{top-}k(p(\cdot x))} p_{\theta}(z x)p_{\theta}(y_i x, z, y_{i-1})$</div> <div></div>	<div><div>- 그리디 서치라면</div><div>- 한 번이라도 잘못된 선택을 하면 엄청 틀어진다는 위험성</div></div>
<div>Decoding: RAG-Token Model: Exhaustive</div> <div>$P_{\text{RAG-Token}}(y x) \approx \prod_i^N \sum_{z \in \text{top-}k(p(\cdot x))} p_{\theta}(z x)p_{\theta}(y_i x, z, y_{i-1})$</div> <div></div>	<div><div>- 매번 최고확률 단어 선택</div><div>- 중간중간에 잘못된게 있을 수는 있는데 결과적으로는 괜찮을 것</div><div>- 다만 계산량 많음</div></div>
<div>Decoding: RAG-Token Model</div> <div>$P_{\text{RAG-Token}}(y x) \approx \prod_i^N \sum_{z \in \text{top-}k(p(\cdot x))} p_{\theta}(z x)p_{\theta}(y_i x, z, y_{i-1})$</div> <div></div>	<div><div>- 절충안이 빔서치</div><div>- 그리디와 유사하지만 한번에 n 개를 선택한다는 점에서 다름</div></div>
<div>Decoding: RAG-Sequence Model</div> <div>$P_{\text{RAG-Sequence}}(y x) = \sum_{z \in \text{top-}k(p(\cdot x))} p_{\theta}(z x)\prod_i^N p_{\theta}(y_i x, z, y_{i-1})$</div> <div><div><div><div><div>Open-domain QA is common task</div><div>hypothesis y_1</div><div>$p(y_1 z_1, x)$</div></div><div><div>z_1</div><div><div>BART is seq2seq model</div><div>hypothesis y_2</div><div>$p(y_2 z_1, x)$</div></div><div><div>RAG approaches SOTA performance</div><div>hypothesis y_3</div><div>$p(y_3 z_1, x)$</div></div><div><div>BART is seq2seq model</div><div>hypothesis y_2</div><div>$p(y_3 z_2, x)$</div></div><div><div>z_2</div><div><div>Attention is all you need</div><div>hypothesis y_4</div><div>$p(y_4 z_2, x)$</div></div><div><div>Open-domain QA is common task</div><div>hypothesis y_1</div><div>$p(y_1 z_2, x)$</div></div></div><div>$p(y_1 x) = p(y_1 z_1, x)p(z_2 x) + p(y_1 z_2, x)p(z_2 x)$$p(y_1 x) = p(y_1 z_1, x)p(z_1 x) + p(y_1 z_2, x)p(z_2 x)$$p(y_2 x) = p(y_2 z_1, x)p(z_1 x) + p(y_2 z_2, x)p(z_2 x)$$p(y_3 x) = p(y_3 z_1, x)p(z_1 x) + p(y_3 z_2, x)p(z_2 x)$$p(y_4 x) = p(y_4 z_1, x)p(z_1 x) + p(y_4 z_2, x)p(z_2 x)$</div></div></div></div></div>	<div><div>시퀀스 모델 디코딩</div><div><div>- docs z1과 z2에서 문장 생성</div><div>- 같은 시퀀스도 있고 다른 것도 있음</div><div>- 각 문장별 확률 계산</div><div>- 후보에 없으면 알지 못함</div><div>- 따라서 모든 시퀀스를 따지는 건 비효율적</div></div><div>sol. Fast Decoding</div><div><div>- 등장하지 않는건 0이라고 추정</div></div></div>
	<div><div>-</div></div>

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

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	MSMARCO	FVR3	FVR2		
	B-1	QB-1	R-L	B-1	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>		

	KNN-LM	REALM	RAG
retrieval	inference	Pre-train + Fine-tuning	Fine-tuning
retrieval object	token	document	document
type	generative	extractive	generative
evaluation	perplexity	open-domain QA	open-domain QA

-