AI 스터디 WEEK2

숙명여자대학교 인공지능공학부 이나연 / 2026.02

<Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2~2.5>

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

앞부분 복습

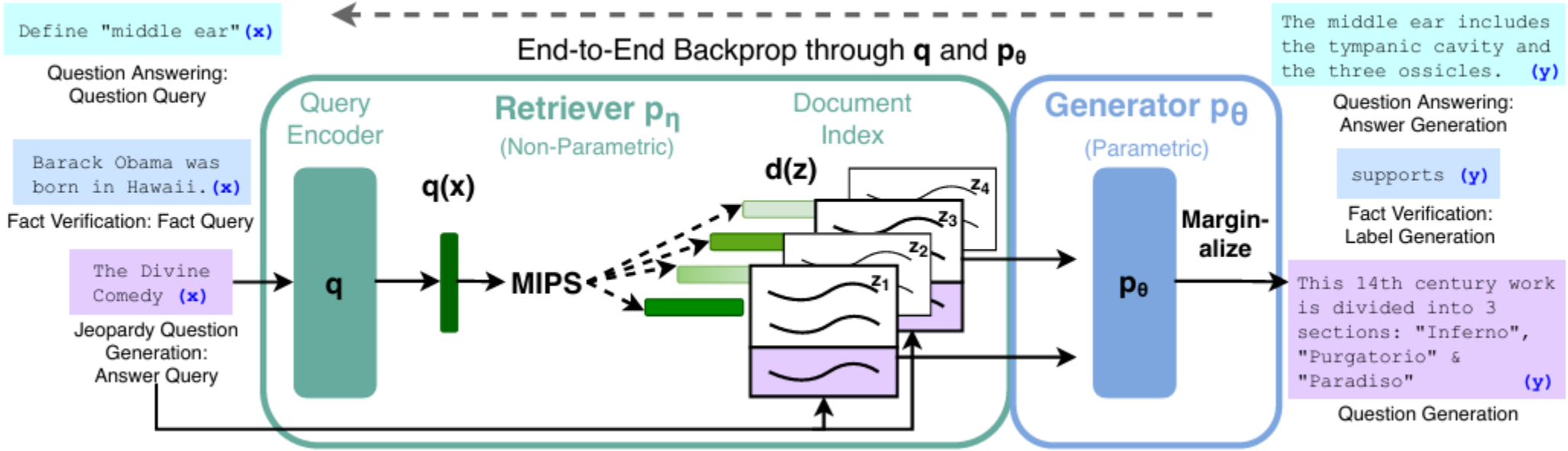
"**RAG**"
: Retrieval-Augmented Generation
정보 검색과 생성 모델을 결합한 자연어 처리 (NLP) 기술
➔ 사전 학습된 언어모델은 implicit한 Parametric memory를 사용 가능하지만, 여러 문제 O

➔ 외부메모리 활용을 위한 하이브리드 모델 등장, RAG는 사전 학습된 BART + neural retriever
➔ 정보검색의 역할이 Neural Retriever, 생성모델의 역할이 BART임.

* **Parametric memory**: (1) 모델 내부에 저장된 지식 (예: GPT의 가중치)
(2) implicit하게 저장되어 꺼내 쓰기 빠르지만 최신 정보는 X.

Non-parametric memory: (1) 외부에 저장된 지식 (예: 검색 데이터베이스, RAG 시스템)
(2) 필요할 때마다 explicit하게 참조. 최신 정보 반영에 효율적.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2



*** Overview of our approach**

input sequence x ⇒ 문서 z 검색 및 참조(additional context로 사용)

⇒ target sequence y 출력 (생성)

x : input sequence

z : text documents

y : target sequence

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

*** End-to-End Fine-tuning**
→ RAG의 End-to-End는 최종 정답이 틀렸다면,

그 책임이
1. 답변을 생성한 generator(BART)와
2. 문서를 가져온 Retriever에게 있다고 판단하여

두 모델을 한꺼번에 업데이트 (중간단계가 없음)
→ 정답을 잘 맞히기 위해 Retriever가 어떤 문서 가져오는 것이 유리한지 생성 모델의 피드백을 받아서 학습함.

* Memory인데 학습된다고?
→ 여기서의 memory는 단순히 저장장치 X, 지식을 담은 주체로 간주하기

* **marginalize** : 수학적으로 합산 또는 적분의 의미.
'marginlize(주변화)한다' → 모든 경우의 수를 고려하여 하나로 합친다
→ 특정 문서 하나에만 의존 X, 여러 문서가 정답에 기여하는 모든 확률을 더한다.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

*** 2.1 Models (간단하게만)**

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

1. RAG-Sequence Model : 검색된 동일한 하나의 문서로 sequence 완성 (sequence → token들이 특정한 순서대로 나열된 묶음으로 생각하기)

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

2. RAG-Token Model : 각 token별로 다른 latent 문서를 참조하도록 하고 이에 따라 marginalize함.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

2.2 Retriever: DPR

The retrieval component $p_\eta(z|x)$ is based on DPR [26]. DPR follows a bi-encoder architecture:

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

where $\mathbf{d}(z)$ is a dense representation of a document produced by a $\mathrm{BERT}_{\mathrm{BASE}}$ *document encoder* [8], and $\mathbf{q}(x)$ a query representation produced by a *query encoder*, also based on $\mathrm{BERT}_{\mathrm{BASE}}$. Calculating top-k$(p_\eta(\cdot|x))$, the list of k documents z with highest prior probability $p_\eta(z|x)$, is a Maximum Inner Product Search (MIPS) problem, which can be approximately solved in sub-linear time [23]. We use a pre-trained bi-encoder from DPR to initialize our retriever and to build the document index. This retriever was trained to retrieve documents which contain answers to TriviaQA [24] questions and Natural Questions [29]. We refer to the document index as the *non-parametric memory*.

⇒ 2.2 Retriever: DPR

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

2.2 Retriever: DPR
Retriever pη(z|x)는 **DPR**을 기반으로 하며, DPR은 **bi-encoder** 아키텍처를 따름.

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

*** bi-encoder가 무엇일까?**
*context encoder*와 *candidate encoder*로 구성되었으며,
각각 **context query**와, 이와 연관된 **후보 문서들을** 인코딩함(벡터화).
➔ context, candidate embedding을 각각 얻음
➔ 두 벡터의 내적으로 적절한지(유사도)를 계산함. (관계는 고려 X)

d(z) = BERTd(z)는 BERT_base 모델 기반 document encoder에서 만든 문서의 dense(밀집)한 표현
q(x) = BERTq(x)도 BERT_base 모델 기반 query encoder에서 만든 query 표현임.
(query → 질문 text를 백터화. Q,K,V 생각)

context query → q(x) / candidate document → d(z)

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

2.2 Retriever: DPR

top-k($p_\eta(\cdot|x)$)을 계산하기 (가장 높은 확률의 $p_\eta(z|x)$를 갖는 k개 문서 z를 구하기)
=> Maximum Inner Product Search(MIPS) 문제이고, 이는 대략 sub-linear 시간에 풀 수 있음

DPR에서 미리 훈련된 bi-encoder를 사용함으로써 retriever를 초기화하고, 문서 index를 만듦.
이 retriever는 TriviaQA질문과 Natural Questions에 대한 답변이 포함된 문서를 검색하도록 학습됨.

이때 Retriever는 외부 문서를 활용하기 때문에,
document index를 non-parametric memory라고 함.
(파라미터 η는 이를 검색하기 위한 encoder의 가중치)

*d(z)는 **dense representation**(밀집한 표현)인데, 이는 결국 벡터를 의미.
MIPS 연산을 빠르게 수행하기 위해 만든 문서 벡터(d(z))를 index에 모아놓은 것.
외부에 저장되었기 때문에 explicit한 non-parametric이다.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.3

2.3 Generator: BART

The generator component $p_\theta(y_i|x, z, y_{1:i-1})$ could be modelled using any encoder-decoder. We use BART-large [32], a pre-trained seq2seq transformer [58] with 400M parameters. To combine the input x with the retrieved content z when generating from BART, we simply concatenate them. BART was pre-trained using a denoising objective and a variety of different noising functions. It has obtained state-of-the-art results on a diverse set of generation tasks and outperforms comparably-sized T5 models [32]. We refer to the BART generator parameters θ as the *parametric memory* henceforth.

$\Rightarrow$ 2.3 Generator: BART

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.

2.3 Generator:BART

$$p_\theta(y_i|x, z, y_{1:i-1})$$

generator를 구성하는 **pθ(yi|x,z,y1:i−1)**는 임의의 encoder-decoder를 써서 모델링할 수 있음.

본 논문에서는 400M개의 파라미터로 구성된, 사전 학습된 seq2seq transformer 구조인 BART-large 모델을 사용함.

BART로부터 생성할 때, input x를 검색한 문서 z와 합치기 위해서
본 논문에서는 간단하게 concat하여 결합하여 generator에 입력함.

BART는 denoising과 다양한 noising함수를 활용하여 사전 학습되었으며, 준수한 언어생성능력을 가짐. 비슷한 크기의 T5모델보다 성능이 뛰어났음.

BART generator의 파라미터 θ를 **paramentric memory**라고 한다.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.4

2.4 Training

We jointly train the retriever and generator components without any direct supervision on what document should be retrieved. Given a fine-tuning training corpus of input/output pairs (x_j, y_j), we minimize the negative marginal log-likelihood of each target, $\sum_j -\log p(y_j|x_j)$ using stochastic gradient descent with Adam [28]. Updating the document encoder BERT_d during training is costly as it requires the document index to be periodically updated as REALM does during pre-training [20]. We do not find this step necessary for strong performance, and keep the document encoder (and index) fixed, only fine-tuning the query encoder BERT_q and the BART generator.

⇒ 2.4 Training

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

2.4 Training

본 논문에서는 retriever와 generator를,
어떤 문서를 참조할지에 대한 직접적인 감독 없이 동시에 훈련시킴.

입출력 쌍 (xj,yj)로 구성된 fine-tuning 데이터가 주어지면,
→ Adam을 활용한 SGD(stochastic gradient descent)을 사용하여
각 타겟의 the negative marginal log-likelihood를 최소화하는 것을 목표로 함.

$$\sum_j -\log p(y_j|x_j)$$

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

* 입출력 쌍 (xj,yj)로 구성된 fine-tuning 데이터가 주어지면,
→ Adam을 활용한 SGD(stochastic gradient descent)을 사용하여
각 타겟의 the negative marginal log-likelihood를 최소화하는 것을 목표로 함.

(SGD) stochastic gradient descent : 정답과 답변 사이 오차를 계산하여 오차의 원인을 찾아가며 가중치를 조금씩 수정하는 방식 (확률적 경사 하강법)

Adam : 이전의 수정 방향과 속도를 기억해서 더 효율적으로 정답을 찾아가도록 함.

log-likelihood : 모델이 정답 y를 맞힐 확률에 log를 취한 값.
확률이 1(100%)에 가까울수록 이 값은 0에 가까워짐.
(likelihood : 우도/가능도, 특정 사건들이 일어날 확률을 의미함)

negative marginal
→ 딥러닝의 손실함수는 확률을 최소화 하는 방식으로 목표를 설계
(확률이 1에 멀수록 0에서 멀어지므로 마이너스를 붙이는 것)

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

과정을 하나씩 살펴보면,
1. x(질문) → y(정답)

$$\sum_j -\log p(y_j|x_j)$$

2. 수식 - log p(y|x) 계산 (-> 손실함수)

3. 수식 안에 Retriever와 Generator의 파라미터가 모두 들어있음 (Retriever pη(z|x), Generator pθ(yi|x,z,y1:i−1)에서 각 파라미터 η, θ)

4. 역전파 과정을 통해, Retriever와 Generator에게 오차의 원인이 있는지 계산하여 두 모델의 가중치를 동시에 업데이트 함.

이때 질문과 정답 쌍을 모델에게 주고, 전체 문서 확률을 고려했을 때 정답률을 높이기 위해 Adam이라는 도구를 사용하여 End-to-end로 두 모델의 가중치를 수정하려는 것.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

2.4 Training

학습 중에 document encoder BERTd를 업데이트 하는 것은 (REALM이 사전 학습에 한 것처럼)
document index를 주기적으로 업데이트 해야하므로 비용이 많이 든다.

본 논문에서는 이 과정이 강력한 성능에 필수적이지 않다고 판단하여,

document encoder와 index를 고정시키고,
query encoder BERTq와 BART generator만 파인튜닝함.

*헷갈릴 수 있는 지점. BERT vs BART
두 모델 모두 transformer 구조에서 만들어졌지만, 설계 목적이 완전이 다름
BERT (encoder-only) : 문장을 읽고 그 의미를 벡터로 변환. RAG에서 Retriever에 쓰임.
BART (encoder-decoder) : 문장을 읽고 새로운 문장 쓰는 것까지 가능. RAG에서 Generator로 쓰임.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.4

⇒ 2.5 Decoding

2.5 Decoding

At test time, RAG-Sequence and RAG-Token require different ways to approximate $\arg\max_y p(y|x)$.

RAG-Token The RAG-Token model can be seen as a standard, autoregressive seq2seq generator with transition probability: $p'_\theta(y_i|x, y_{1:i-1}) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z_i|x) p_\theta(y_i|x, z_i, y_{1:i-1})$ To decode, we can plug $p'_\theta(y_i|x, y_{1:i-1})$ into a standard beam decoder.

RAG-Sequence For RAG-Sequence, the likelihood $p(y|x)$ does not break into a conventional per-token likelihood, hence we cannot solve it with a single beam search. Instead, we run beam search for each document z, scoring each hypothesis using $p_\theta(y_i|x, z, y_{1:i-1})$. This yields a set of hypotheses Y, some of which may not have appeared in the beams of all documents. To estimate the probability of an hypothesis y we run an additional forward pass for each document z for which y does not appear in the beam, multiply generator probability with $p_\eta(z|x)$ and then sum the probabilities across beams for the marginals. We refer to this decoding procedure as "Thorough Decoding." For longer output sequences, $|Y|$ can become large, requiring many forward passes. For more efficient decoding, we can make a further approximation that $p_\theta(y|x, z_i) \approx 0$ where y was not generated during beam search from x, z_i. This avoids the need to run additional forward passes once the candidate set Y has been generated. We refer to this decoding procedure as "Fast Decoding."

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

2.5 Decoding

test시에는, RAG-Sequence와 RAG-Token이 서로 다른 방법으로 argmax_y p(y|x)를 근사함.
*argmax → 함수 f(x)를 최댓값으로 만들기 위한 x를 구함.

1. RAG-token

RAG-token 모델은 transition probability(전이확률)가 표준적인 autoregressive(자기 회귀) seq2seq generator로 간주됨

디코딩하려면 pθ(yi|x,y1:i−1)를 표준 beam search decoder에 넣는다.

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

*beam search → heuristic한 탐색 방법. 각 step에서 탐색의 영역을 k개의 가장 가능도가 높은 토큰들로 유지하며 다음 단계를 탐색.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

2.5 Decoding

2. RAG-Sequence

RAG-sequence의 경우, p(y|x)는 per-token likelihood로 분리 X (각 토큰별로 분해 X),
따라서 한 번의 beam search로 해결할 수 없음

대신, 각 문서 z에 대해 beam search를 수행하여, pθ(yi|x,z,y1:i−1)를 활용하고 각 가설에 점수를 매김.
⇒ 일련의 가설들의 집합 Y가 생성되는데, 몇몇은 모든 문서에 대한 beam에서 나타나지 않을 수 있음

가설 y의 확률을 추정하기 위해,
beam에 y가 나타나지 않는 각 문서 z에 대한 추가 forward pass를 실행하고,
generator 확률을 pη(z|x)와 곱하고 marginal들에 대한 beam 전체의 확률을 합산함.

이 decoding 절차를 **Thorough Decoding**이라 함.

*RAG-Sequence는 모든 문서에 대해 해당 문장이 나올 확률을 다 더해야함.
따라서 확률 합산을 위해 한 번 더 계산하겠다는 의미

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2

2.5 Decoding

2. RAG-Sequence

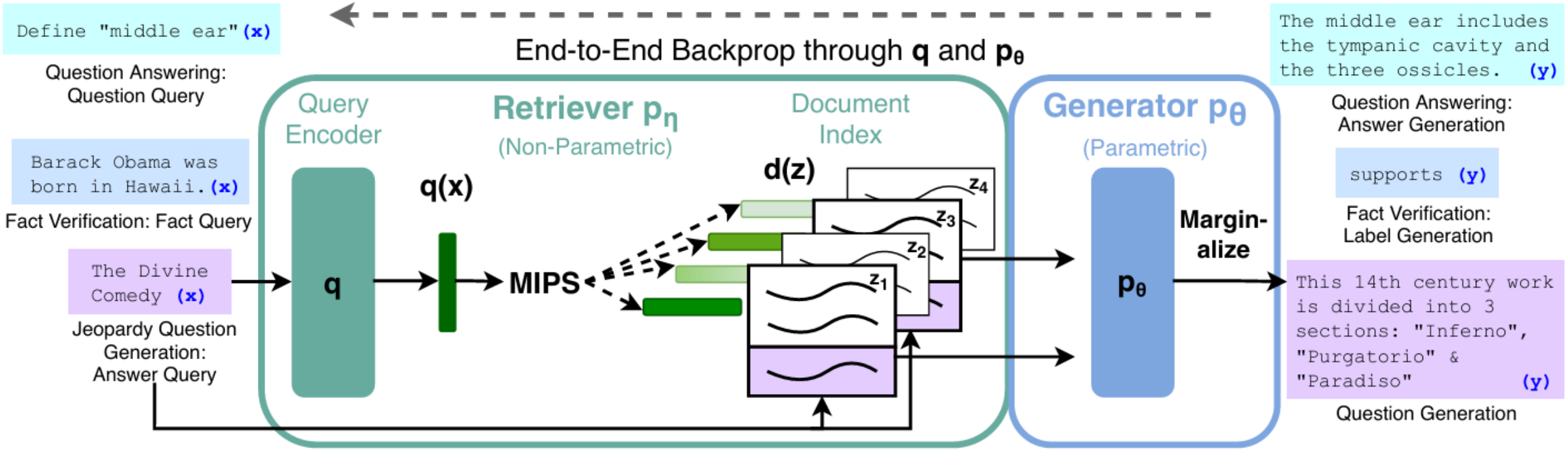
더 긴 결과 sequence에 대하여,
y의 크기가 커질 수 있으며 이에 따라 **여러 번의 forward pass**가 필요할 수 있다.

더 효율적인 decoding을 위해, y가 beam search에서 생성되지 않은 경우,
pθ(y|x,zi) ≈ 0로 추가(further) 근사를 할 수 있음.
이 방법은 후보 집합 y가 생성된 후 추가 forward pass를 실행할 필요가 없음

이 decoding 절차를 **Fast Decoding**이라 함.

*token으로 쪼갤 수 없는거랑 한 번의 beam search로 해결 못하는거랑 무슨 상관이지?
-> RAG-Sequence는 고른 문서를 끝까지 보고 문장 전체(y)를 완성함.
중간에 token별로 확률을 합칠 수 없으니, 각 문서별로 따로 beam search를 해서 최선의 결과를 만들어야 함.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks 2.2



1. 왼쪽 부분 : 입력(x) 및 질의(Query)
2. Query Encoder q를 통과하여 dense Representation인 q(x) 벡터로 변환
3. 가운데 부분 : Retriever pη *(Non-Parametric)*
4. 오른쪽 부분 : Generator pθ (Parametric)
5. 상단 화살표 : 학습의 흐름. 오차를 계산 -> Retriever(pη)와 Generator(pθ) 양쪽으로 모두 전달
-> End-to-end로 η, θ 업데이트