AI VIETNAM
All-in-One Course
(TA Session)

End-to-end Question Answering

Project - P1

Dinh-Thang Duong - TA

Year 2024

Outline

- Introduction
- Question Answering
- Classification Approach
- Extractive Approach
- Question

Introduction

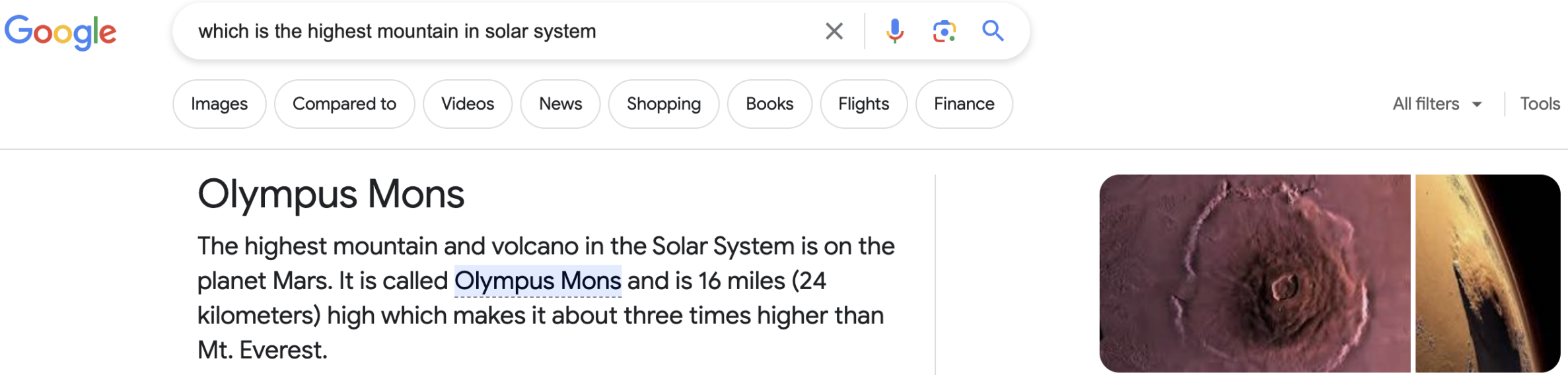
Introduction

❖ Getting Started

Most famous search engines

Introduction

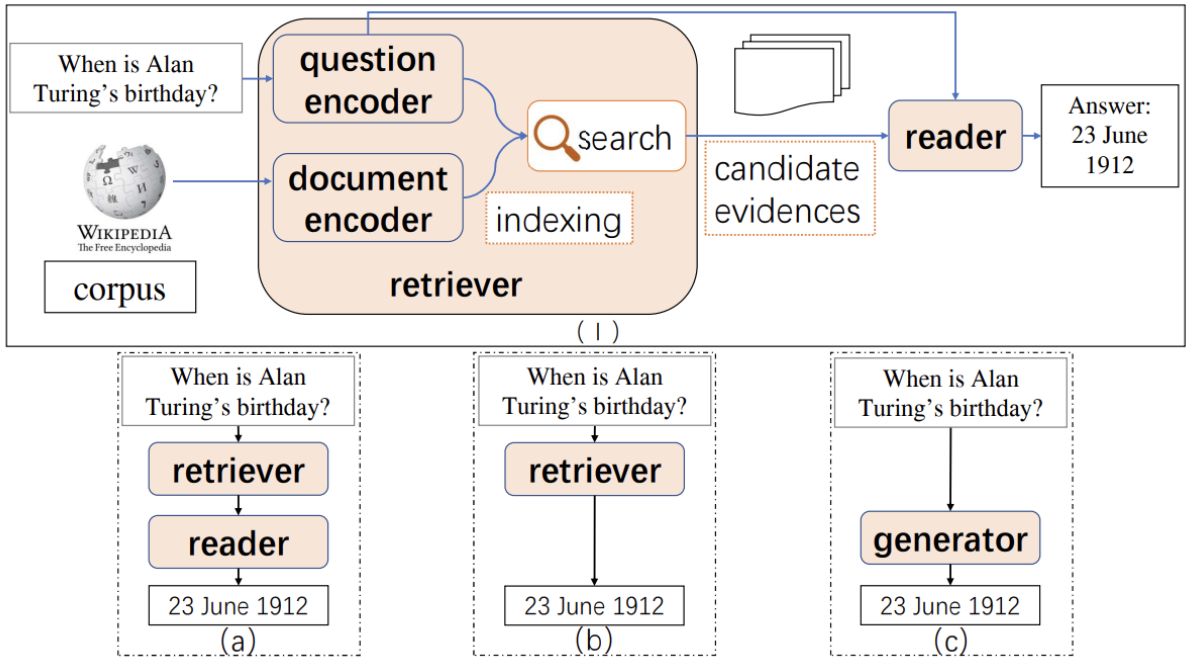
❖ Getting Started



Answer from Google for the query question: Open-domain Question Answering

Introduction

❖ What is Open-domain Question Answering?

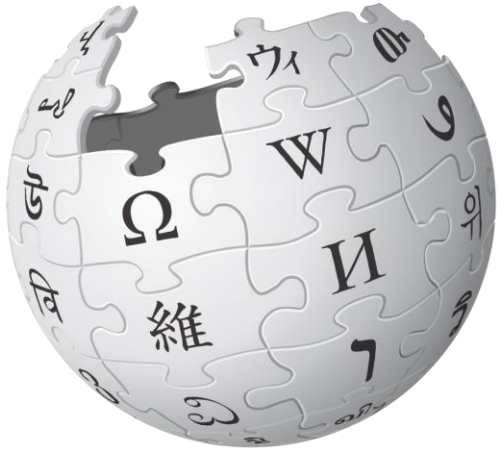


Open-domain Question Answering (ODQA): Refers to the task of providing answers to questions over a broad range of topics, leveraging vast corpora of unstructured text without being restricted to a specific domain.

Image source: https://arxiv.org/pdf/2211.07886.pdf

Introduction

❖ What is Open-domain Question Answering?



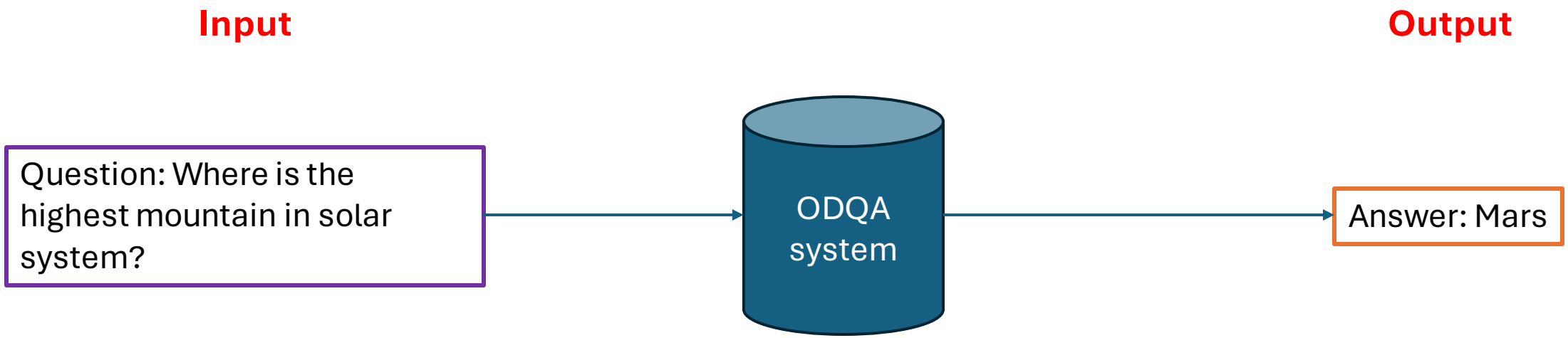
Dataset Facts

Dataset BookCorpus	
Instances Per Dataset 7,185 unique books, 11,038 total	
Motivation	
Original Authors	Zhu and Kiros et al. (2015) [39]
Original Use Case	Sentence embedding
Funding	Google, Samsung, NSERC, CIFAR, ONR
Composition	
Sample or Complete	Sample, ≈2% of smashwords.com in 2014
Missing Data	98 empty files, ≤655 truncated files
Sensitive Information	Author email addresses
Collection	

ODQA systems often embody the end-to-end (E2E) question answering approach, seamlessly integrate the entire process from interpreting natural language queries to retrieving relevant contexts, then extracting/generating concise answers.

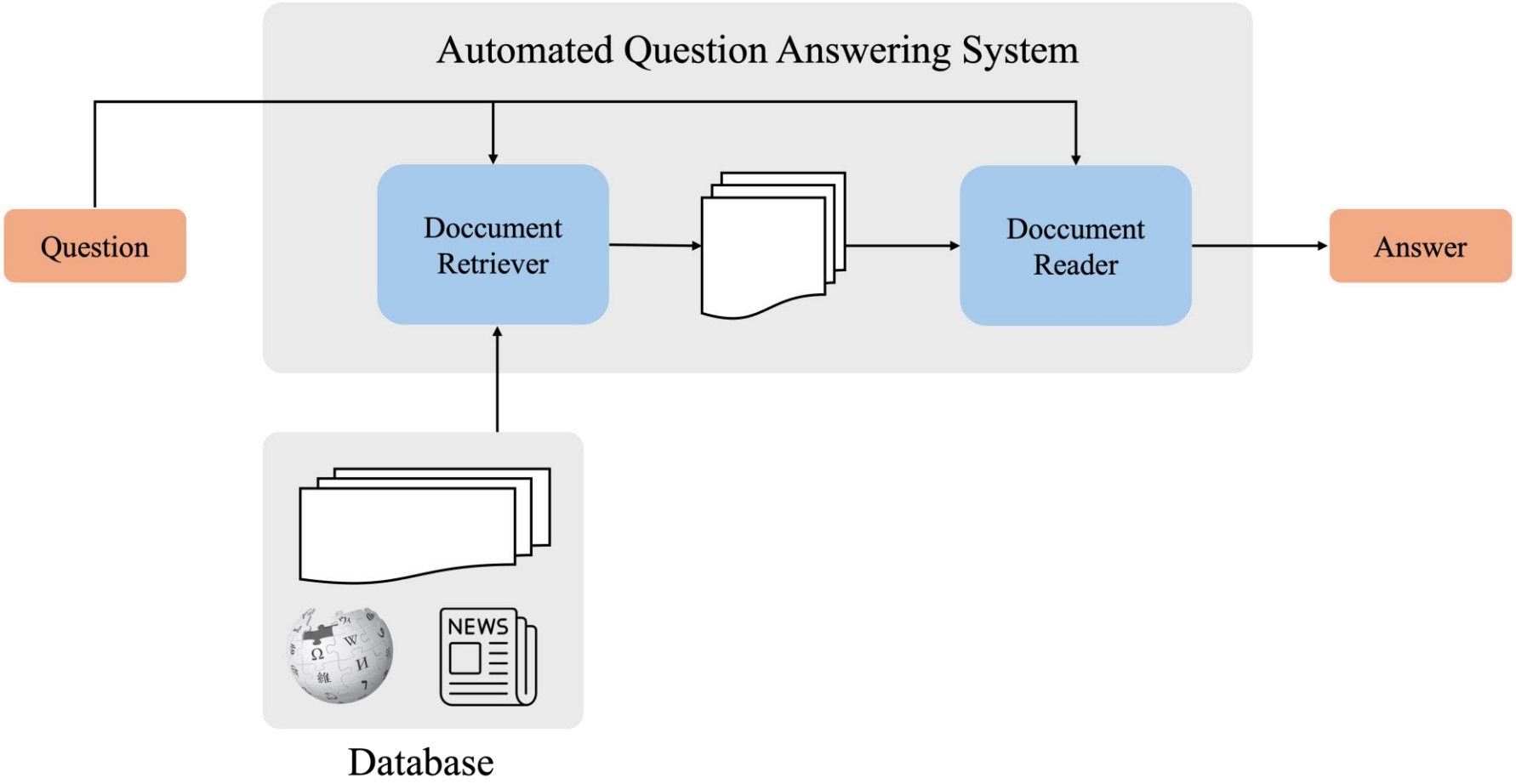
Introduction

❖ End-to-end Question Answering I/O



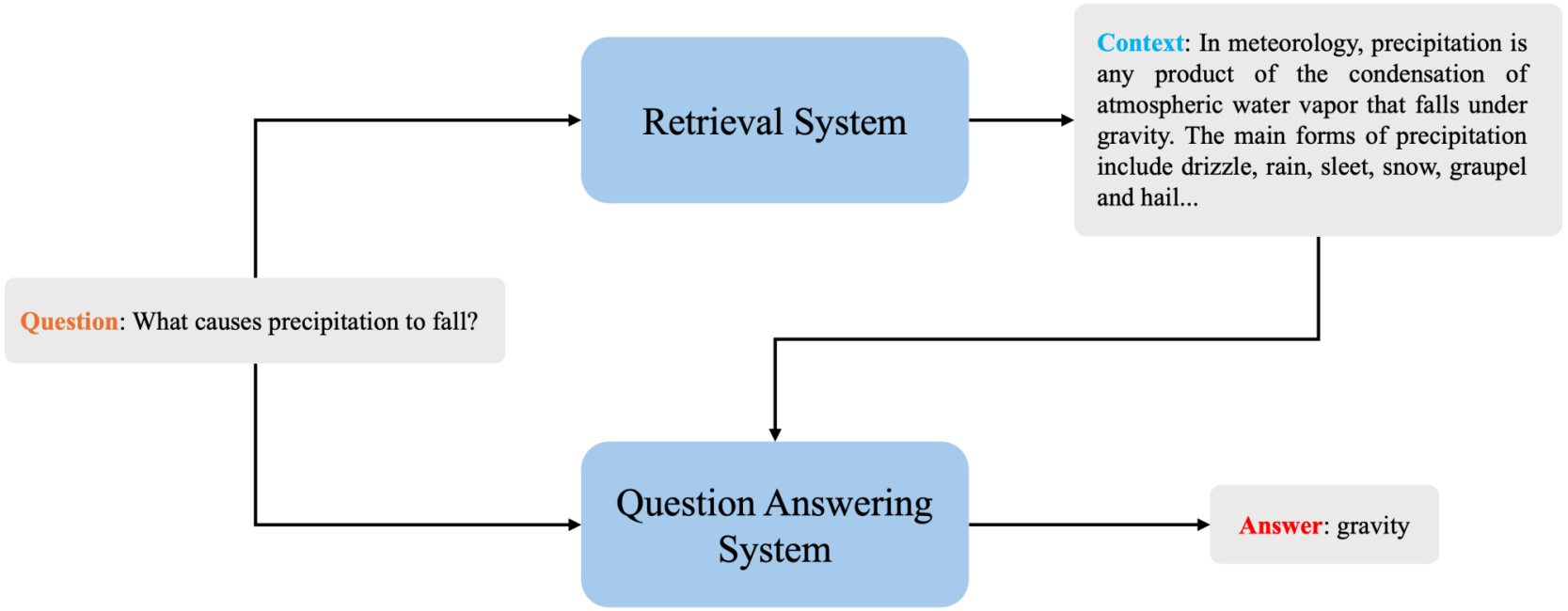
Introduction

❖ End-to-end Question Answering System I/O



Introduction

❖ End-to-end Question Answering System

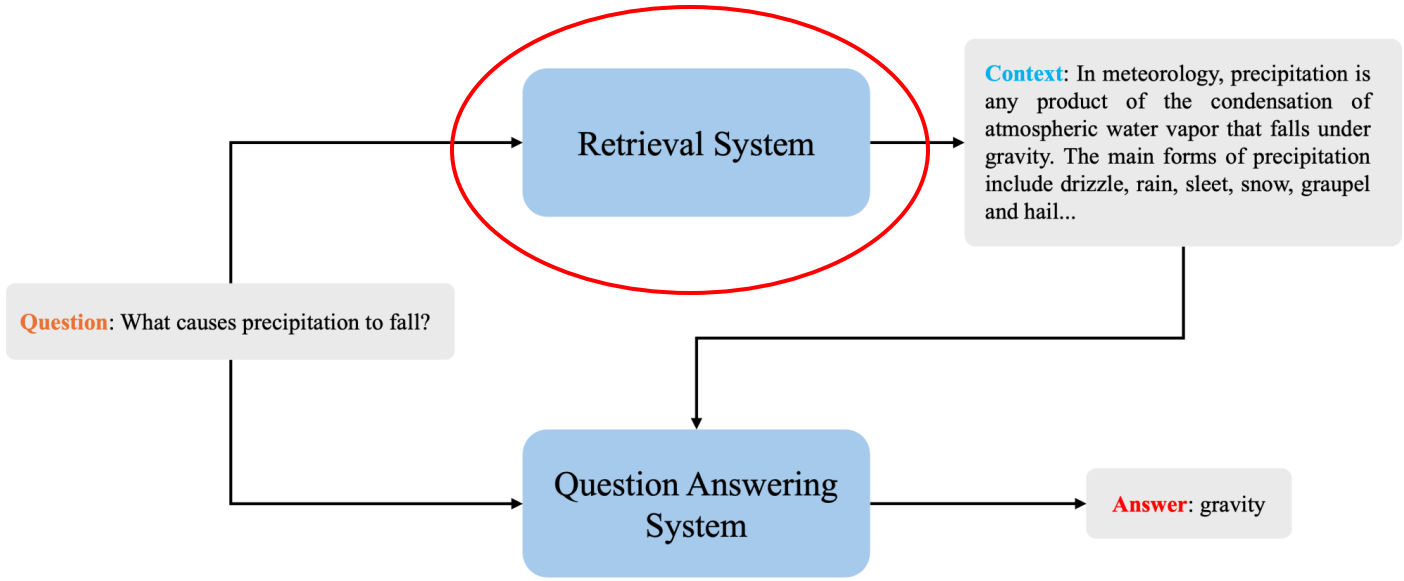


End-to-end QA system consists of:

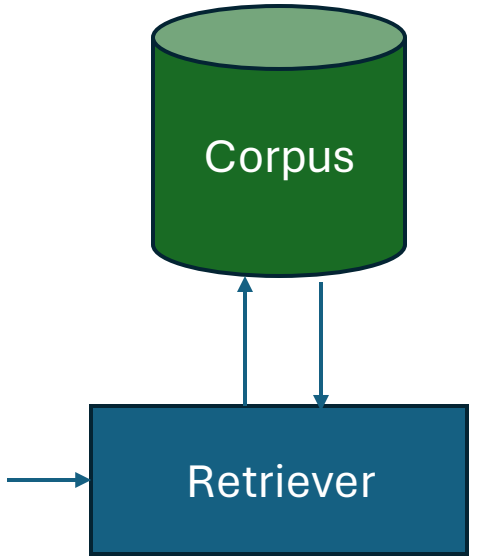
1. Retriever
2. Reader

Introduction

❖ End-to-end Question Answering System: Retriever



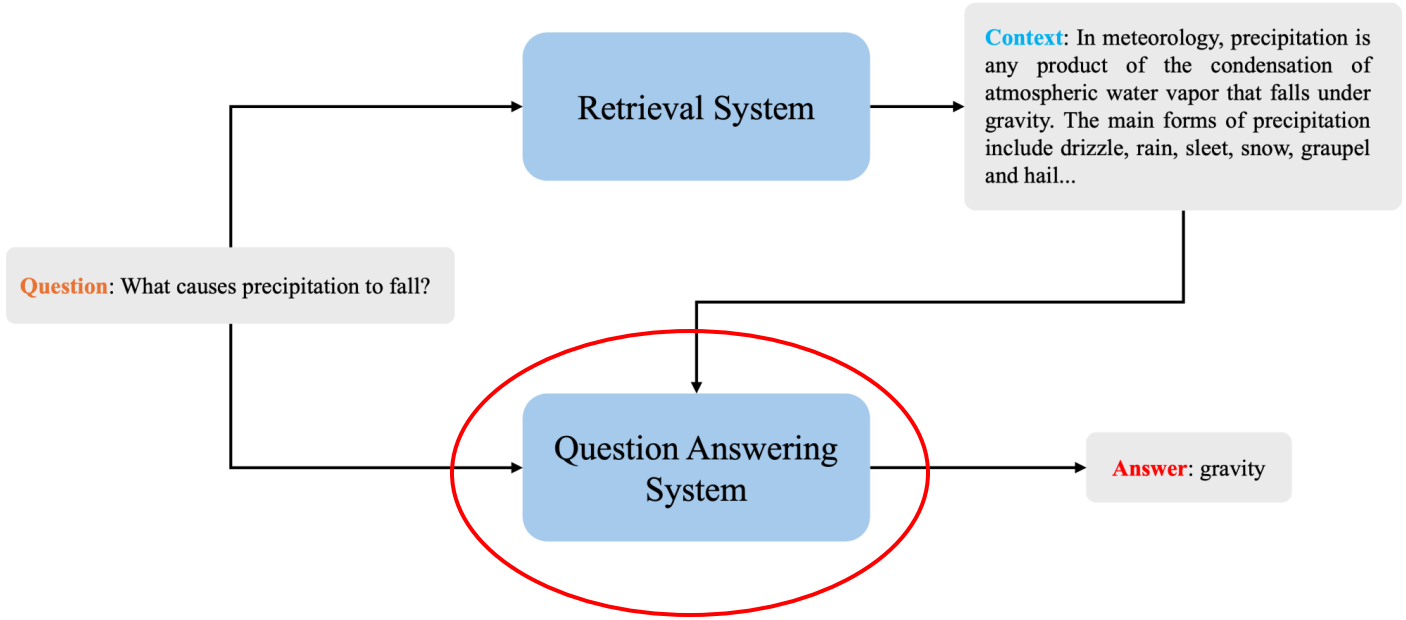
Query: Where is the highest mountain in solar system?

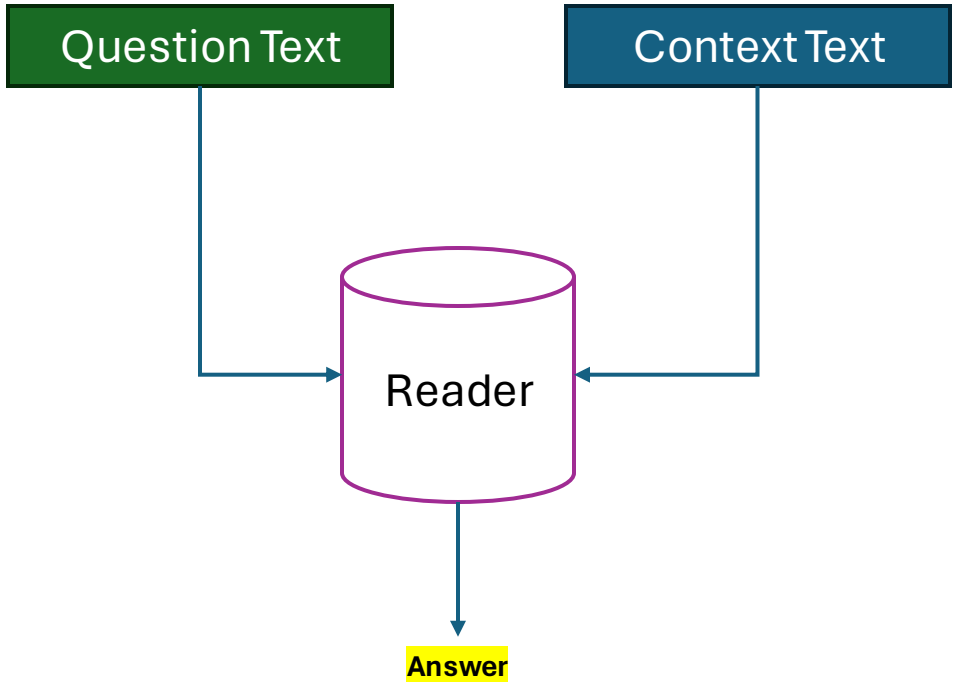


Rank	Document ID
1	Document_4
2	Document_52
3	Document_96
...	...

Introduction

❖ End-to-end Question Answering System: Reader





Introduction

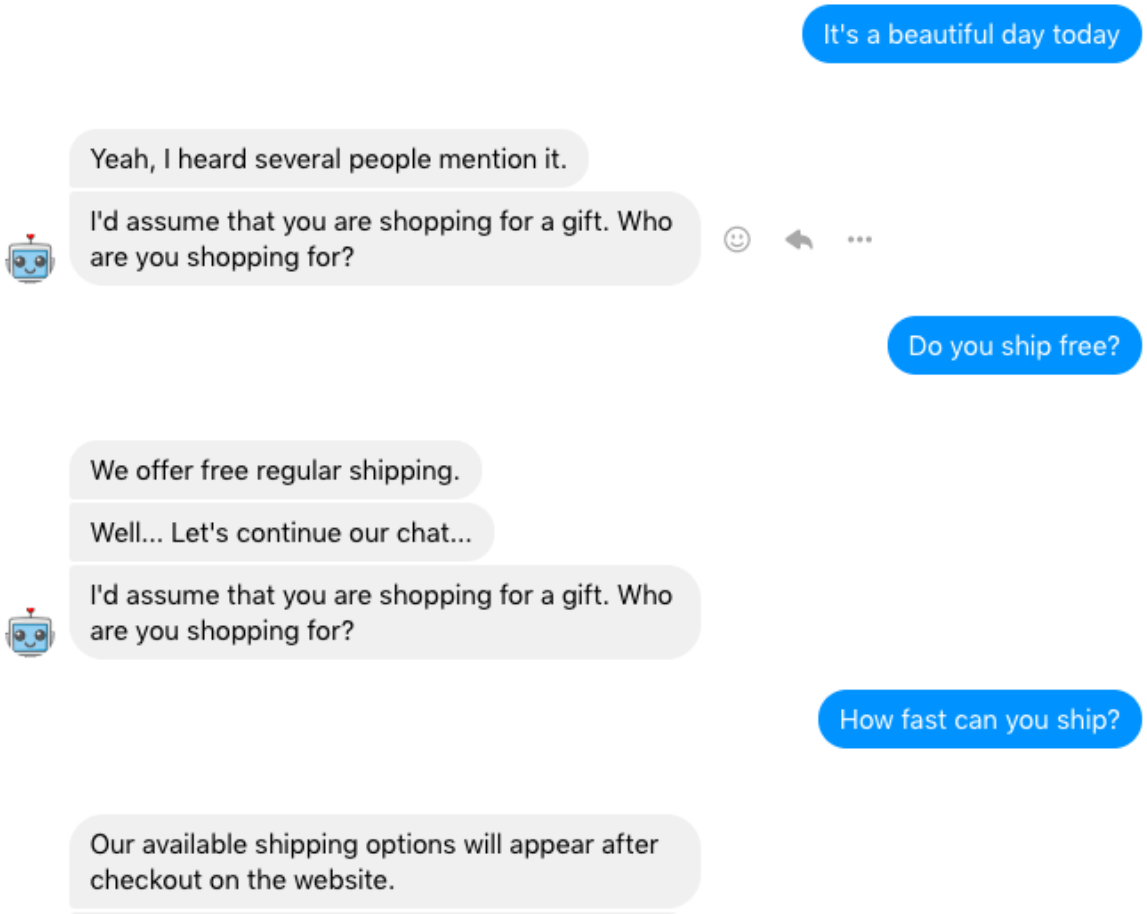
❖ End-to-end Question Answering: Applications

Application: Quick search

https://www.nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/17-question-answering

Introduction

❖ End-to-end Question Answering: Applications



Application: Chatbot

https://juji.io/blog/building-a-smart-chatbot-in-a-few-minutes-to-answer-free-text-questions/

Introduction

❖ End-to-end Question Answering: Applications

Record Date: 08/09/98

08/31/96 ascending aortic root replacement with homograft with omentopexy. The patient continued to be hemodynamically stable making good progress. Physical examination: BMI: 33.4 Obese, high risk. Pulse: 60. resp. rate: 18

Question: Has the patient ever had an abnormal BMI?

Answer: BMI: 33.4 Obese, high risk

Question: When did the patient last receive a homograft replacement ?

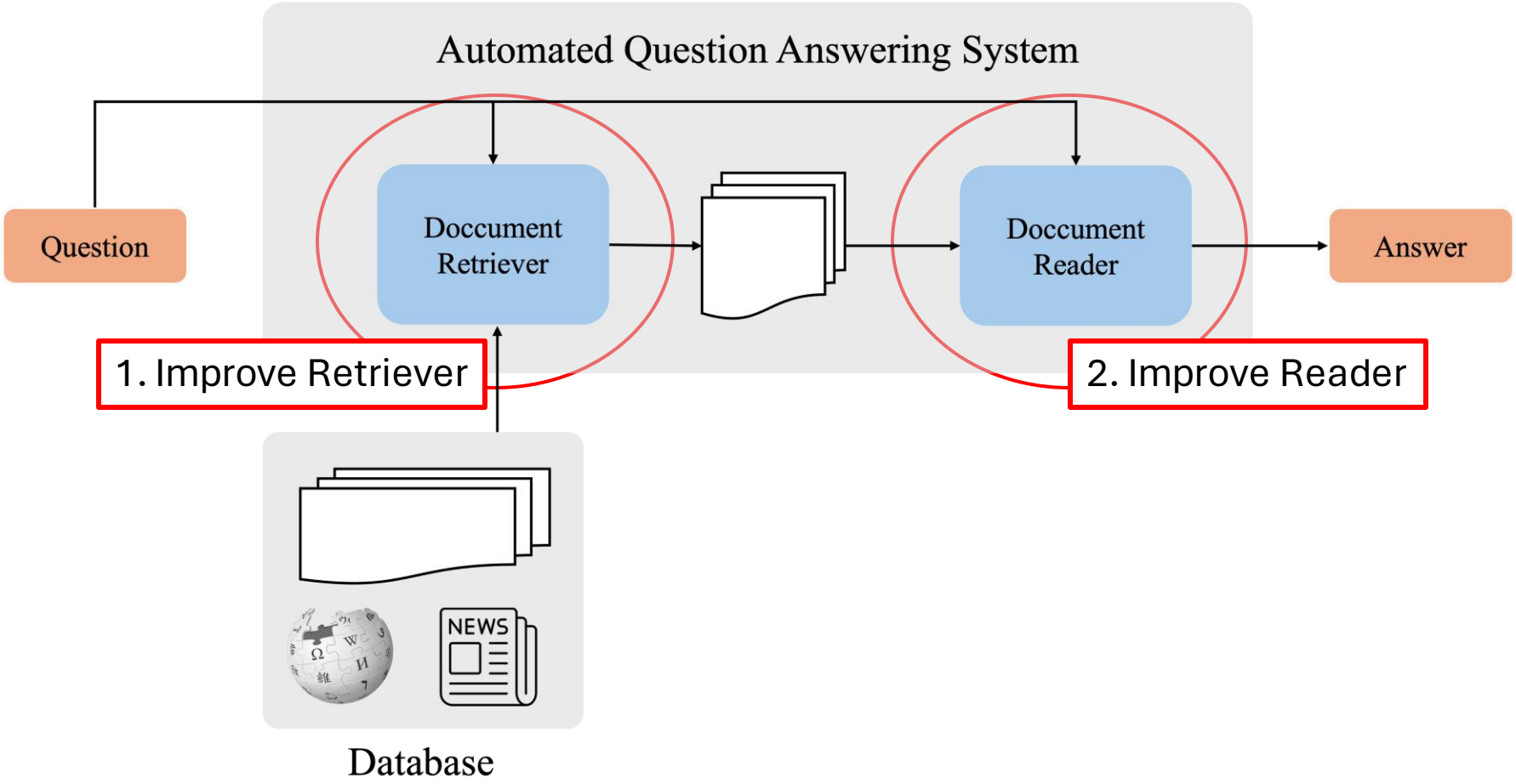
Answer: 08/31/96 ascending aortic root replacement with homograft with omentopexy.

Application: Healthcare

https://paperswithcode.com/dataset/emrqa

Introduction

❖ End-to-end Question Answering: Challenges



Introduction

❖ End-to-end Question Answering: Challenges

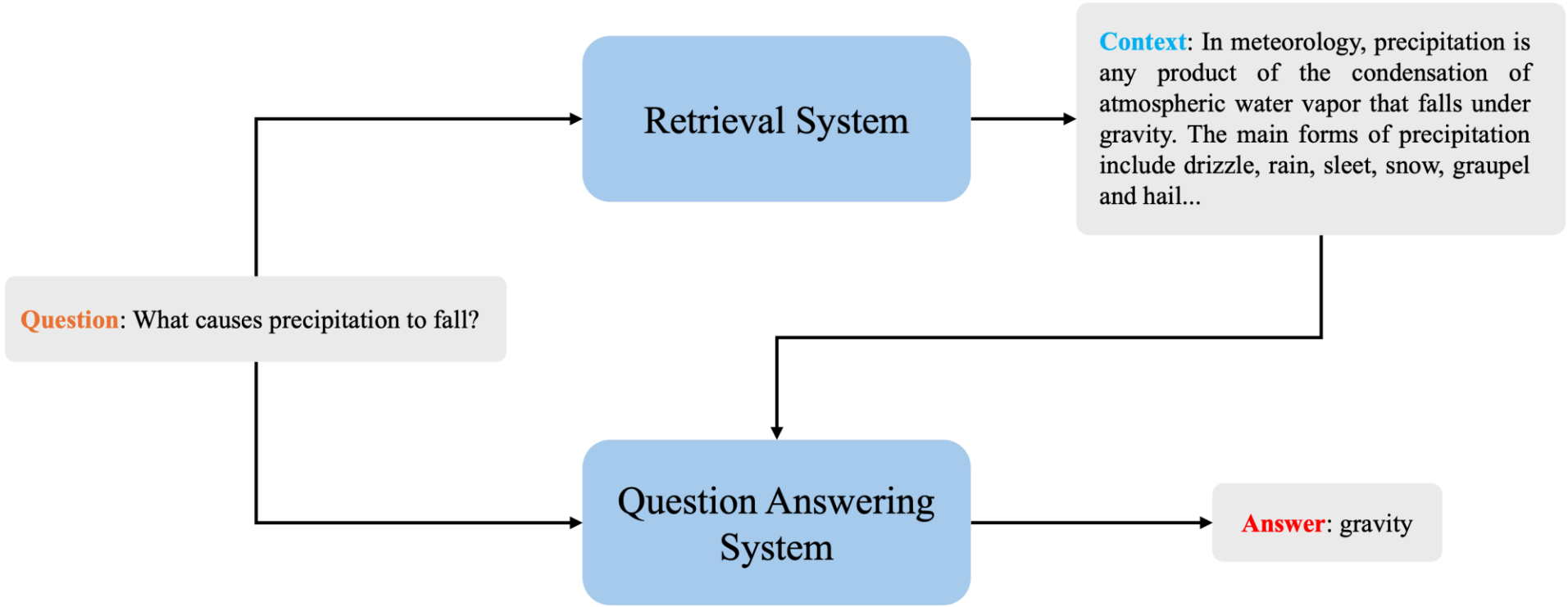
Question	Category	Answer(s)
How	Factoid	Biomedical Entity Names, Number(s), short expression
	Summary	Phrase, Paragraph, short text summarization
Why	Summary	Phrase, Paragraph, short text summarization
Where	Factoid	Biomedical Entity Names, Number(s), short expression
Which	Factoid	Biomedical Entity Names, Number(s), short expression
	Summary	Phrase, Paragraph, short text summarization
What	Factoid	Biomedical Entity Names, Number(s), short expression
	Summary	Phrase, Paragraph, short text summarization
Yes/No	Yes/No	Yes or No

https://www.researchgate.net/figure/Question-Categories-and-their-Exc

Introduction

❖ Project Description

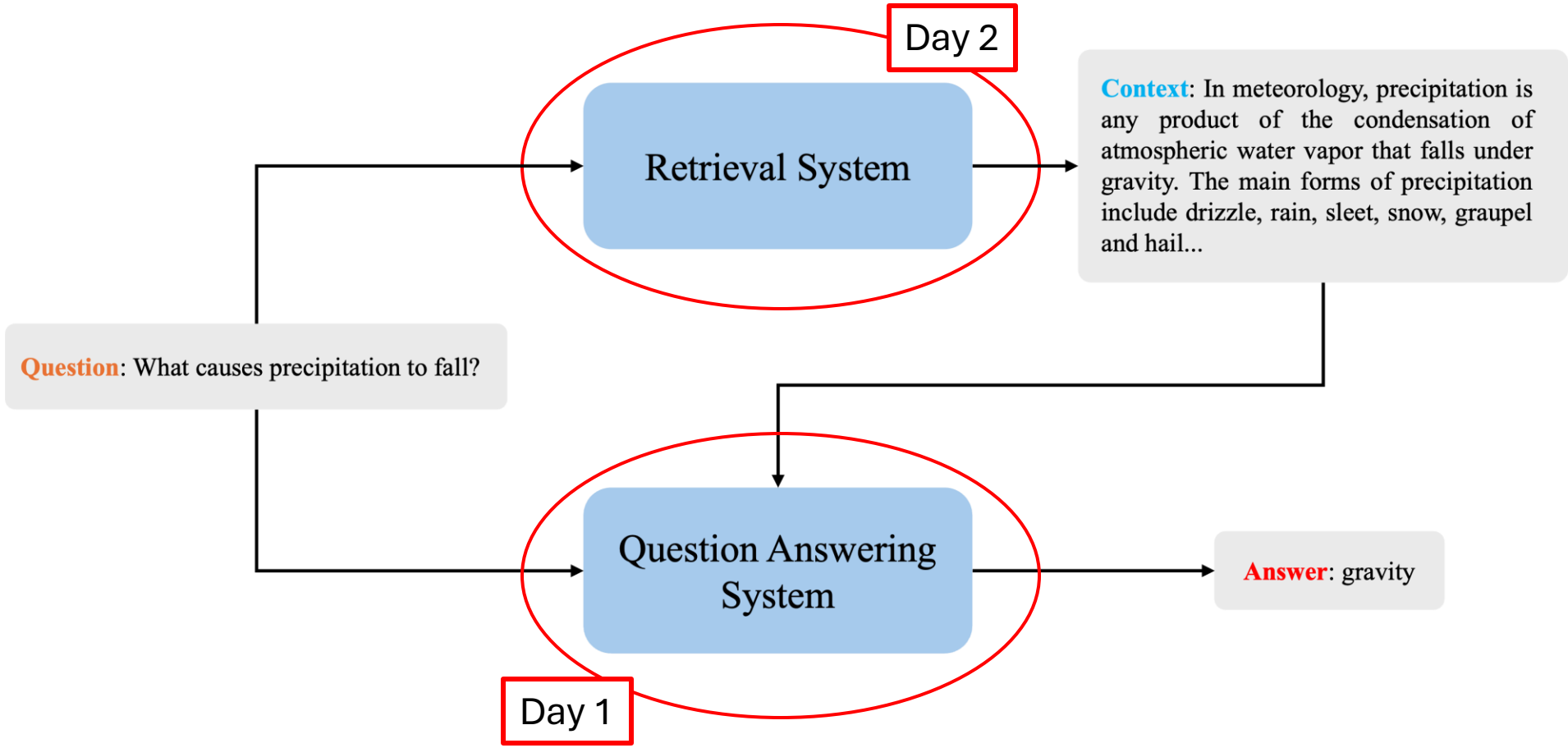
Description: In this project, we will build an end-to-end QA system that can retrieve relevant contexts, extract best answer for a given question. The domain of the question can be varied in different topics.



Introduction

❖ Project Description

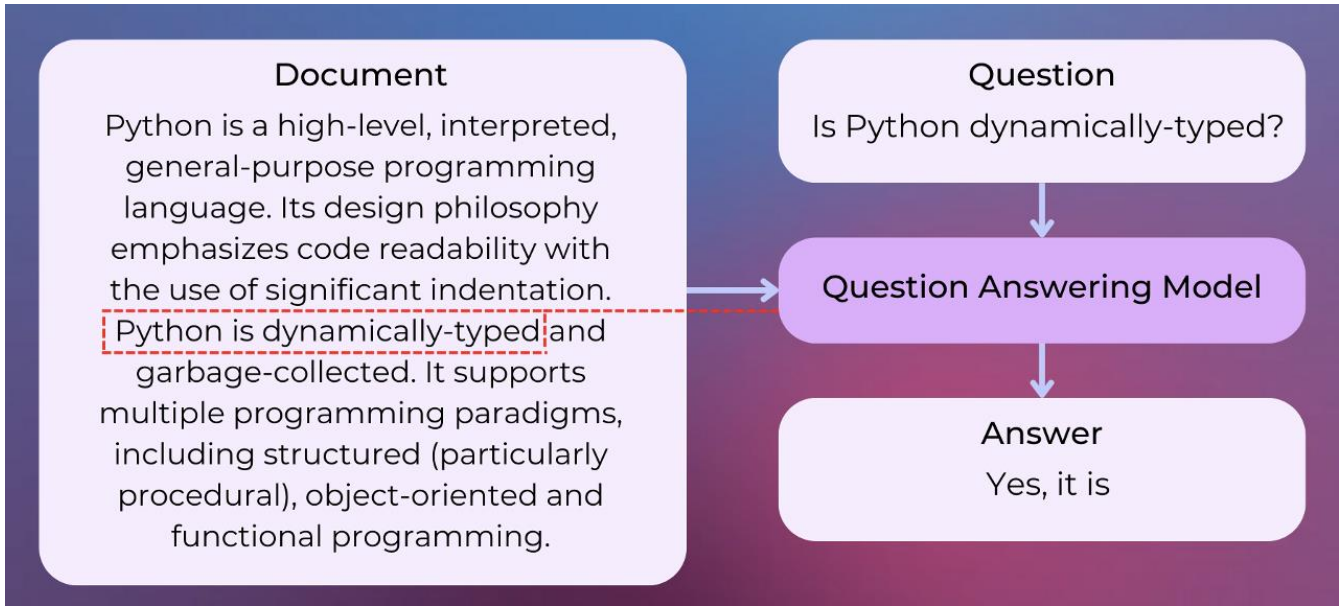
Description: In this project, we will build an end-to-end QA system that can retrieve relevant contexts, extract best answer for a given question. The domain of the question can be varied in different topics.



Question Answering

Question Answering

❖ Introduction



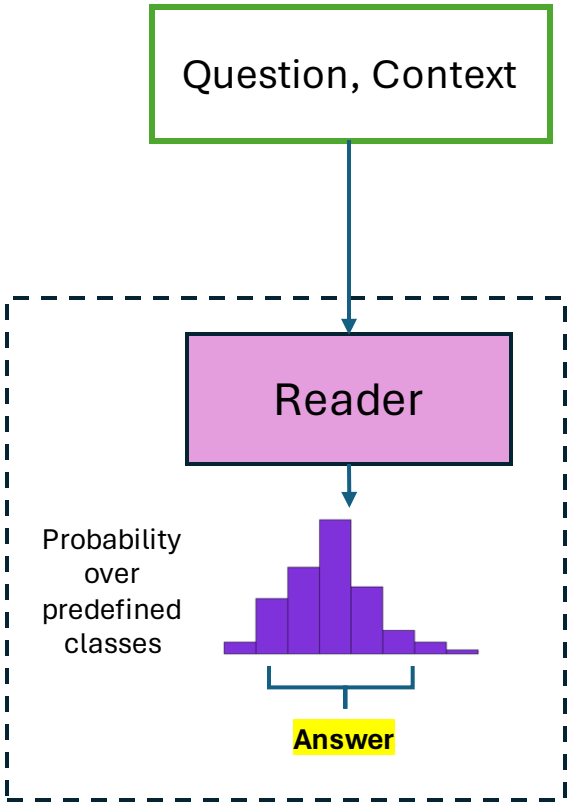
Question Answering (QA): A fundamental task in NLP, aiming to develop systems capable of understanding and responding to human-posed questions in natural language.

https://www.nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/17-question-answering

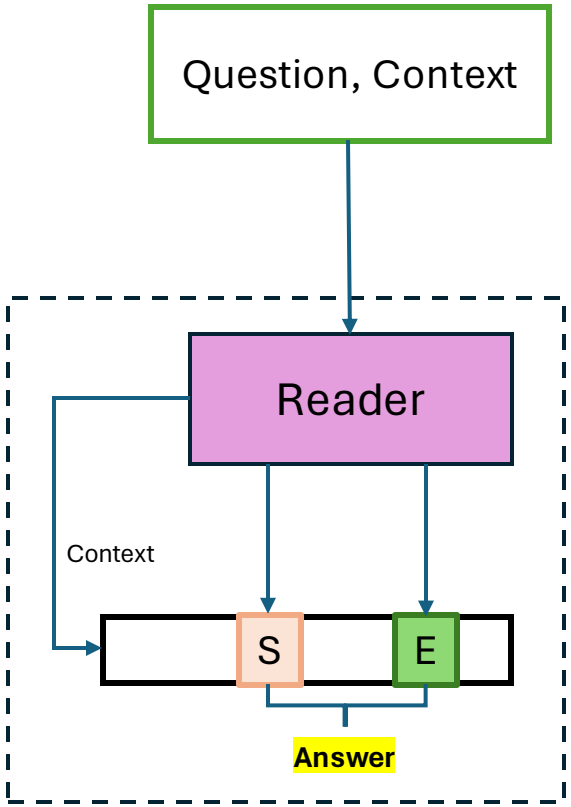
Question Answering

❖ Type of Reader

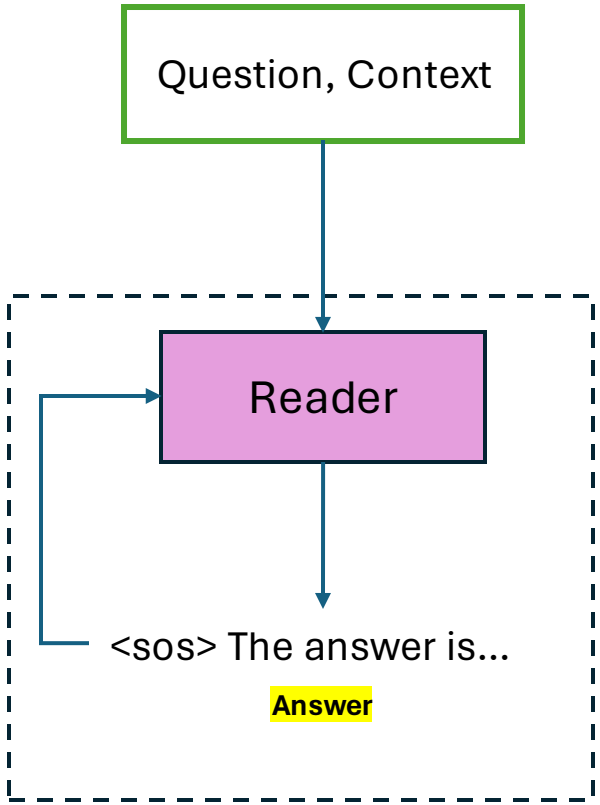
I. Classification Approach



II. Extractive Approach



III. Generative Approach



Question Answering

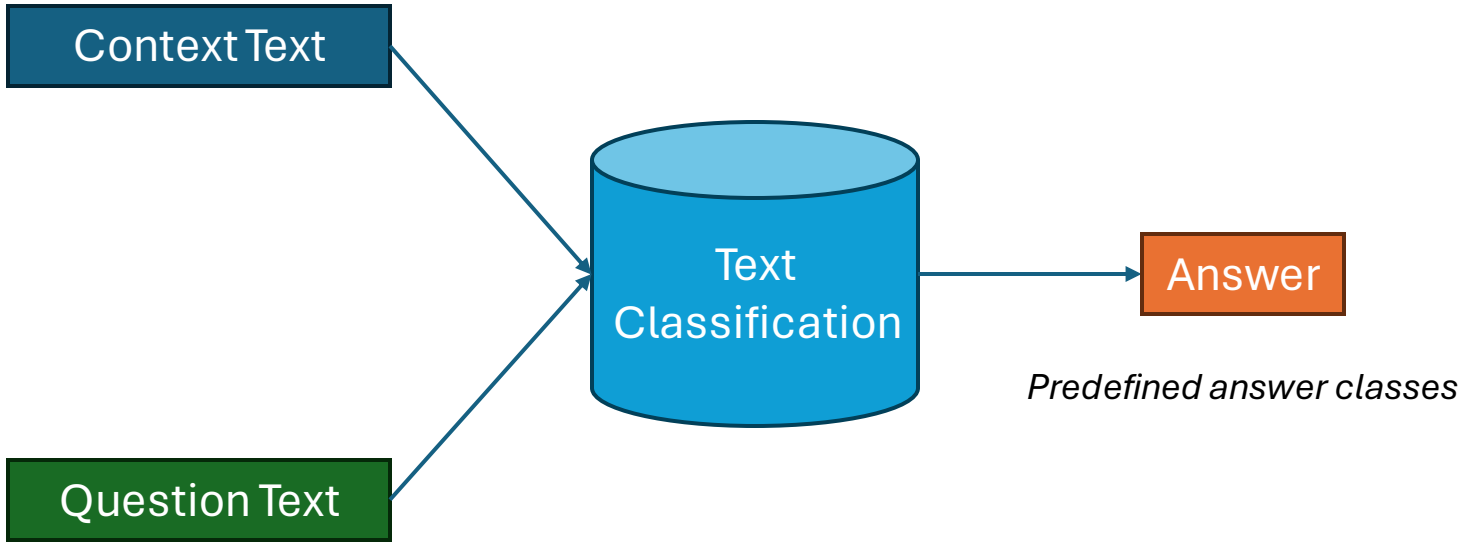
❖ Classification Approach

Short answers

Give short answers to the following questions:

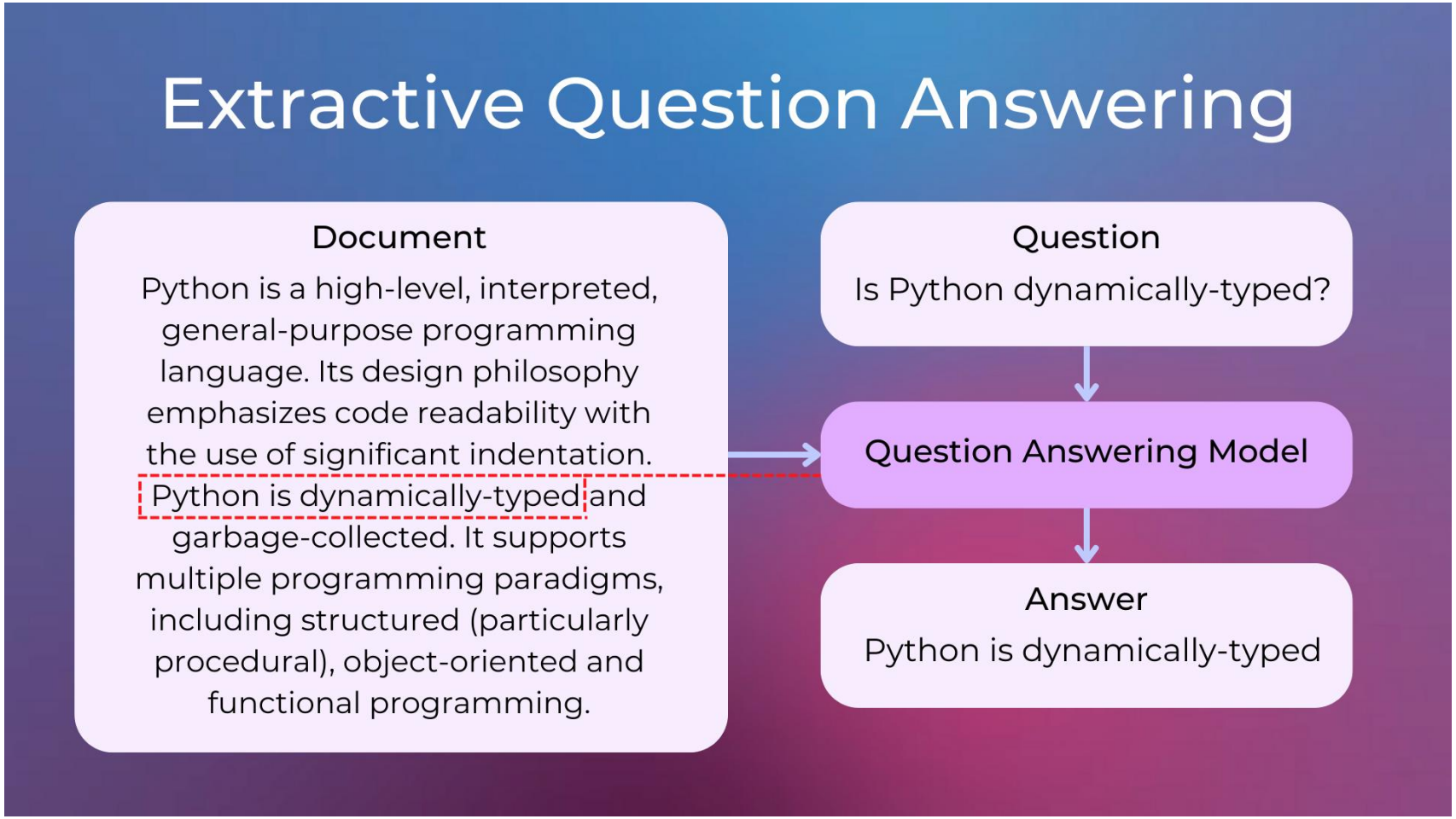
	YES	NO
1. Does Tom work in a bank?		
2. Do Fred and Anna live with their parents?		
3. Has Jack broken his leg?		
4. Are the two men following us?		
5. Are they going to buy a new car?		
6. Did Tom buy a jumper yesterday?		
7. Should he go home?		
8. Did they get married 5 years ago?		
9. Have they been married for 5 years?		
10. Had we seen this film on TV before?		
11. Has Jack eaten all the cookies?		
12. Were you running in the evening?		
13. Was it dark?		
14. Are they going to build a new school?		
15. Can they play the piano very well?		
16. May I go out?		
17. Would you do it?		
18. Is she writing a history essay?		
19. Did Susan go to a party with John?		
20. Will you be going to the baker's?		
21. Has he been to the gym twice this week?		
22. Has he found a dog in the park?		
23. Shall we go home?		
24. Is Tom going to cook today?		
25. Has John been travelling to China?		
26. Will Claire help you do the homework?		
27. Must she do the washing now?		
28. Was she reading at 5 p.m?		
29. Has Kate got a brother?		
30. Had they finished?		

iSLCollective.com



Question Answering

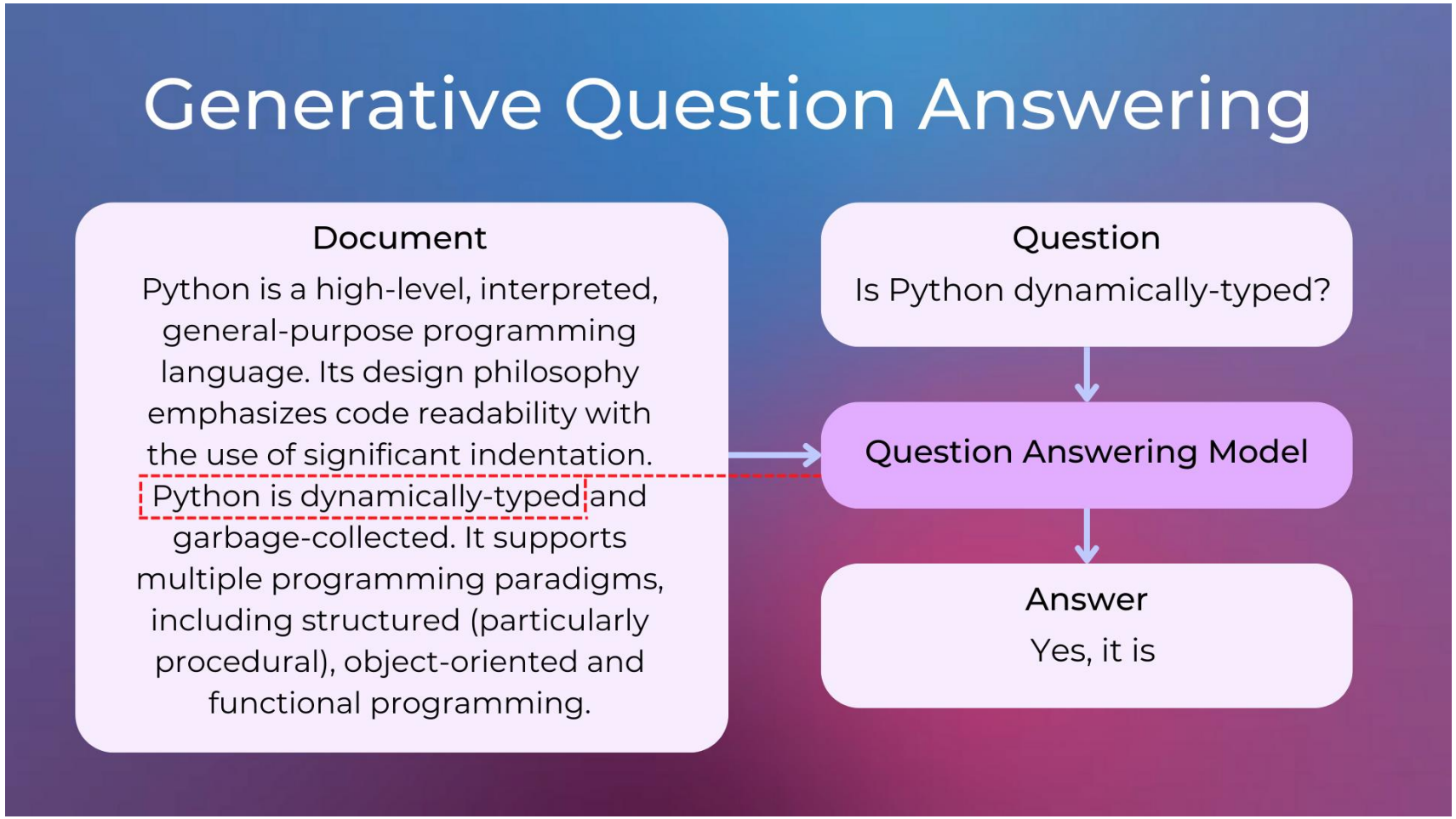
❖ Extractive Approach



https://www.nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/17-question-answering

Question Answering

❖ Generative Approach



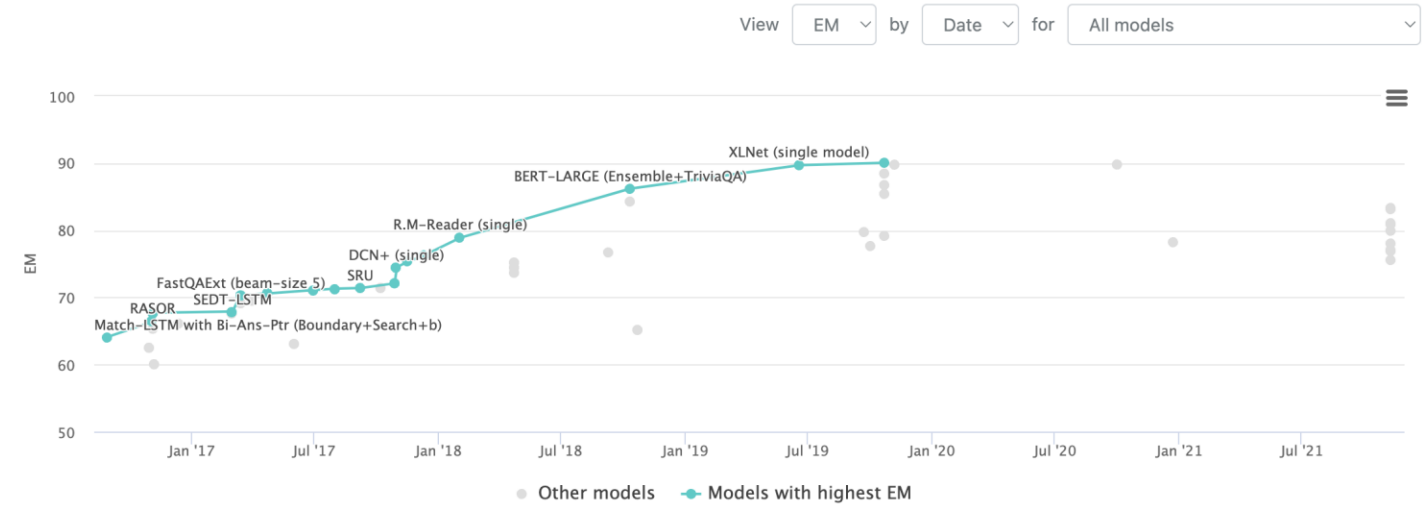
https://www.nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/17-question-answering

Question Answering

❖ Benchmarks

Question Answering on SQuAD1.1 dev

Leaderboard Dataset



Filter: LSTM DCN Transformer untagged

Edit Leaderboard

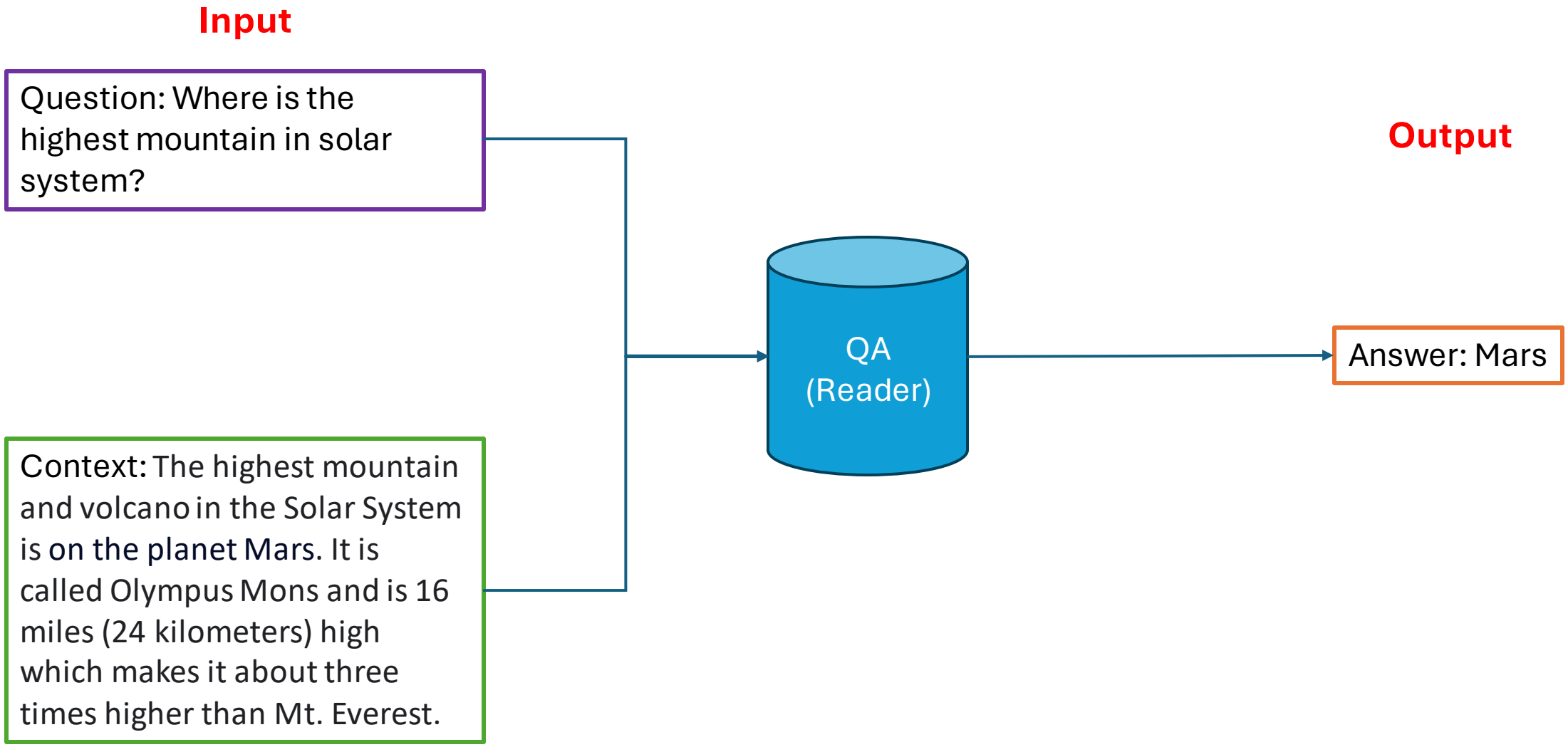
Rank	Model	EM	F1	Extra Training Data	Paper	Code	Result	Year	Tags
1	**T5-11B**	90.06	95.64	✓	Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer			2019	
2	**LUKE**	89.8	95	✓	LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention			2020	

https://paperswithcode.com/sota/question-answering-on-squad11-dev

Classification Approach

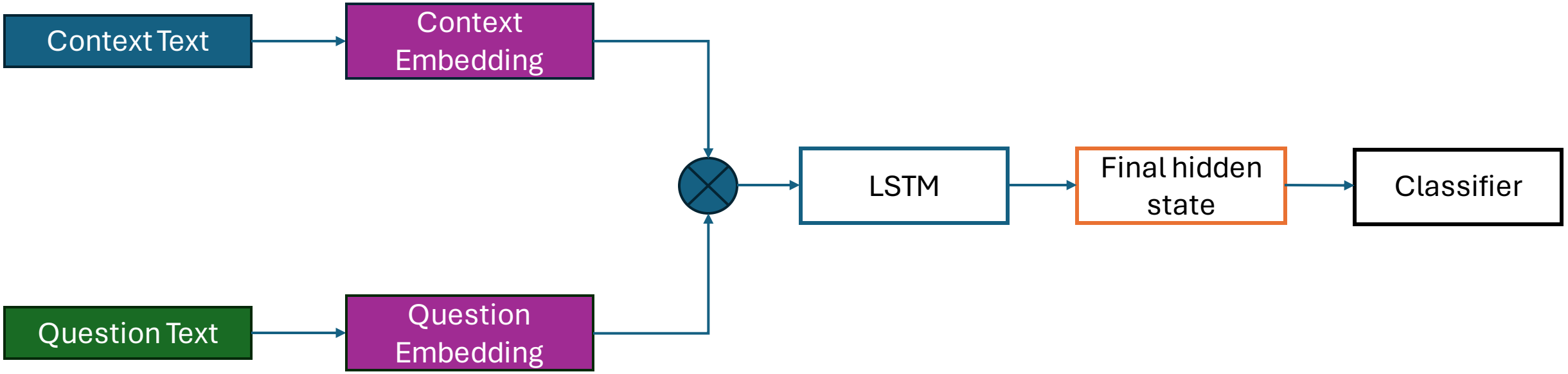
Question Answering

❖ Reader: Classification



Classification Approach

❖ Model Design



Classification Approach

❖ Step 1: Import libraries and create temp dataset

```python
import torch
import torch.nn as nn
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import Dataset, DataLoader
```

```python
qa_dataset = [
    {
        'context': 'My name is AIVN and I am from Vietnam.',
        'question': 'What is my name?',
        'answer': 'AIVN'
    },
    {
        'context': 'I love painting and my favorite artist is Vincent Van Gogh.',
        'question': 'What is my favorite activity?',
        'answer': 'painting'
    },
    {
        'context': 'I am studying computer science at the University of Tokyo.',
        'question': 'What am I studying?',
        'answer': 'computer science'
    }
]
```

Classification Approach

❖ Step 1: Import libraries and create temp dataset

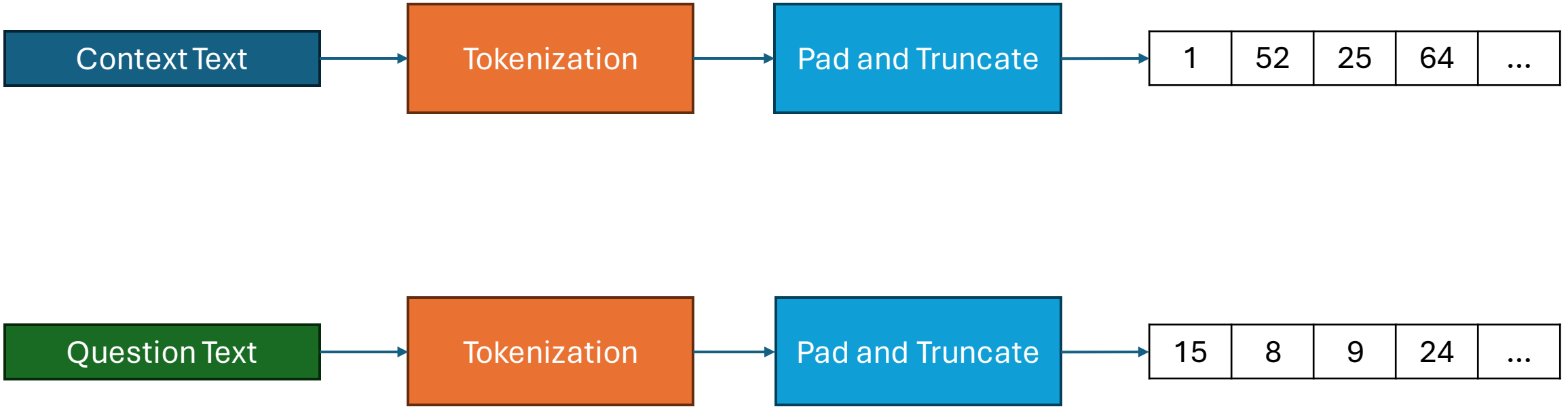
```python
qa_dataset = [
    {
        'context': 'My name is AIVN and I am from Vietnam.',
        'question': 'What is my name?',
        'answer': 'AIVN'
    },
    {
        'context': 'I love painting and my favorite artist is Vincent Van Gogh.',
        'question': 'What is my favorite activity?',
        'answer': 'painting'
    },
    {
        'context': 'I am studying computer science at the University of Tokyo.',
        'question': 'What am I studying?',
        'answer': 'computer science'
    }
]
```

Input:
- context: Paragraph that contains answer.
- question: The query answerable question.

Output:
- answer: Answer span.

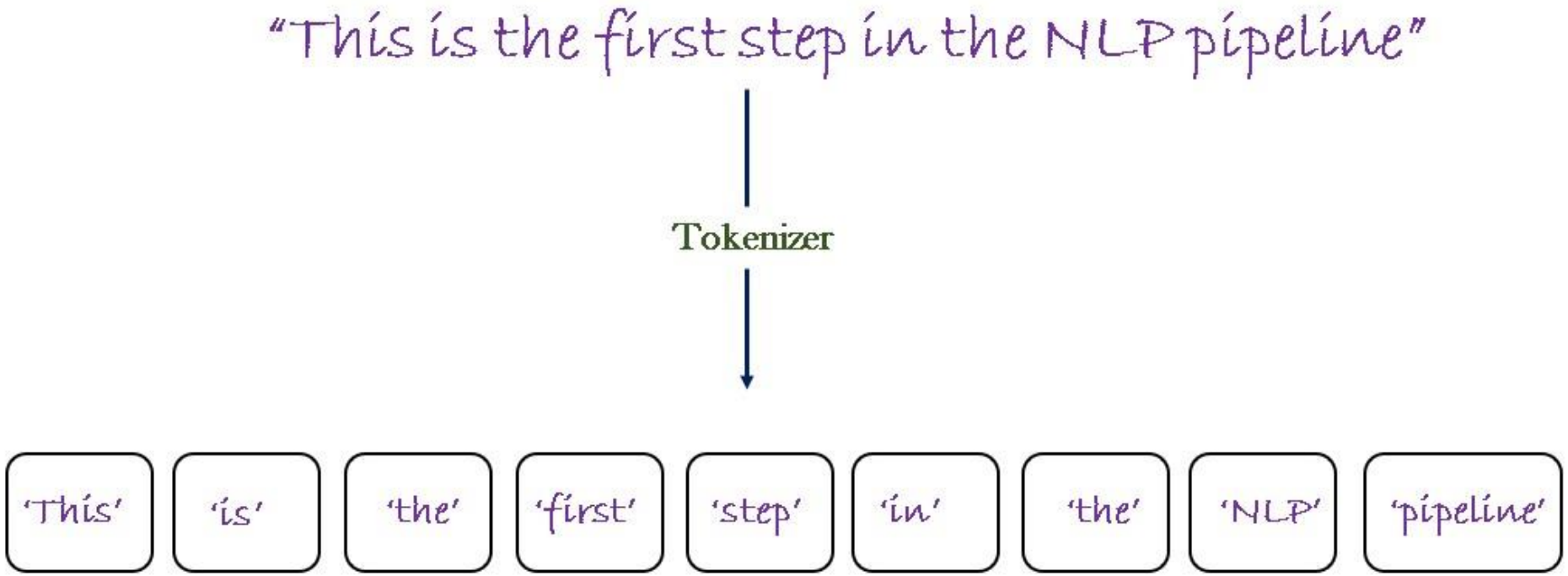
Classification Approach

❖ Step 2: Vectorization



Classification Approach

❖ Step 2: Vectorization



https://wisdomml.in/tokenization-in-natural-language-processing/

Classification Approach

❖ Step 2: Vectorization

```
1 # Define tokenizer function
2 tokenizer = get_tokenizer('basic_english')
3
4 text = 'I love AIVN'
5 tokenizer(text)
```

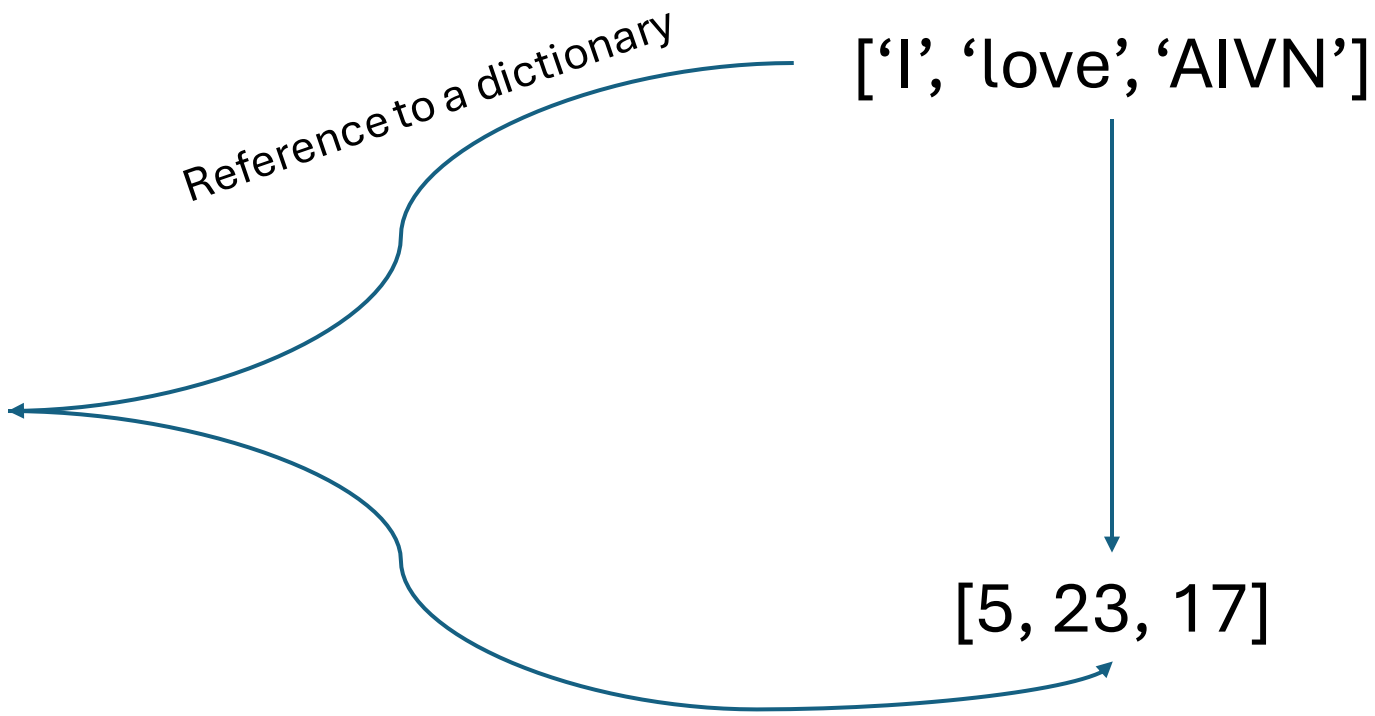
```
['i', 'love', 'aivn']
```

1. text = “I love AIVN”
2. tokenizer(text)
3. tokenized_text -> [‘I’, ‘love’, ‘AIVN’]

Classification Approach

❖ Step 2: Vectorization

```
{'vincent': 32,
 'vietnam': 31,
 'van': 30,
 'university': 29,
 'what': 11,
 '<sep>': 4,
 '<bos>': 2,
 'of': 24,
 'am': 10,
 'my': 7,
 'is': 6,
 'at': 19,
 '.': 8,
 'gogh': 22,
 '<eos>': 3,
 '<pad>': 1,
 'computer': 20,
 'painting': 25,
 'and': 12,
 '<unk>': 0,
 'artist': 18,
 'favorite': 13,
 'studying': 15,
 'i': 5,
```



Classification Approach

❖ Step 2: Vectorization

```python
# Define tokenizer function
tokenizer = get_tokenizer('basic_english')

# Create a function to yield list of tokens
def yield_tokens(data):
    for item in data:
        yield tokenizer(item['context'] + ' ' + item['question'])

# Create vocabulary
vocab = build_vocab_from_iterator(
    yield_tokens(qa_dataset),
    specials=['<unk>', '<pad>', '<bos>', '<eos>', '<sep>']
)
vocab.set_default_index(vocab['<unk>'])
vocab.get_stoi()
```

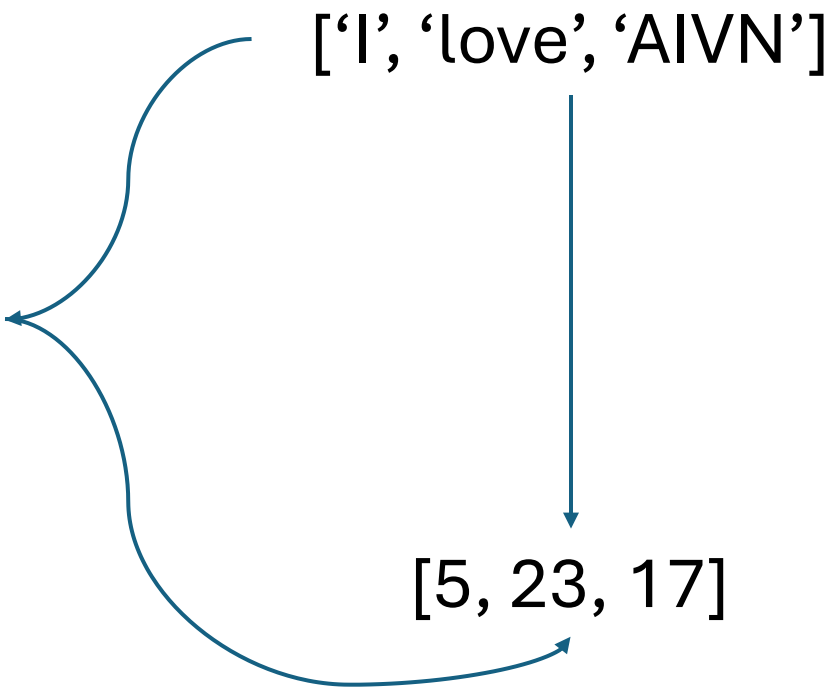
Build vocabulary

```
{'vincent': 32,
 'vietnam': 31,
 'van': 30,
 'university': 29,
 'what': 11,
 '<sep>': 4,
 '<bos>': 2,
 'of': 24,
 'am': 10,
 'my': 7,
 'is': 6,
 'at': 19,
 '.': 8,
 'gogh': 22,
 '<eos>': 3,
 '<pad>': 1,
 'computer': 20,
 'painting': 25,
 'and': 12,
 '<unk>': 0,
 'artist': 18,
 'favorite': 13,
 'studying': 15,
 'i': 5,
```

Classification Approach

❖ Step 2: Vectorization

```
{'vincent': 32,
 'vietnam': 31,
 'van': 30,
 'university': 29,
 'what': 11,
 '<sep>': 4,
 '<bos>': 2,
 'of': 24,
 'am': 10,
 'my': 7,
 'is': 6,
 'at': 19,
 '.': 8,
 'gogh': 22,
 '<eos>': 3,
 '<pad>': 1,
 'computer': 20,
 'painting': 25,
 'and': 12,
 '<unk>': 0,
 'artist': 18,
 'favorite': 13,
 'studying': 15,
 'i': 5,
```




```python
# Define tokenizer function
tokenizer = get_tokenizer('basic_english')

text = 'I love AIVN'
tokens = tokenizer(text)
tokens = [vocab[token] for token in tokens]
print(tokens)
```

```
[5, 23, 17]
```

Classification Approach

❖ Step 2: Vectorization

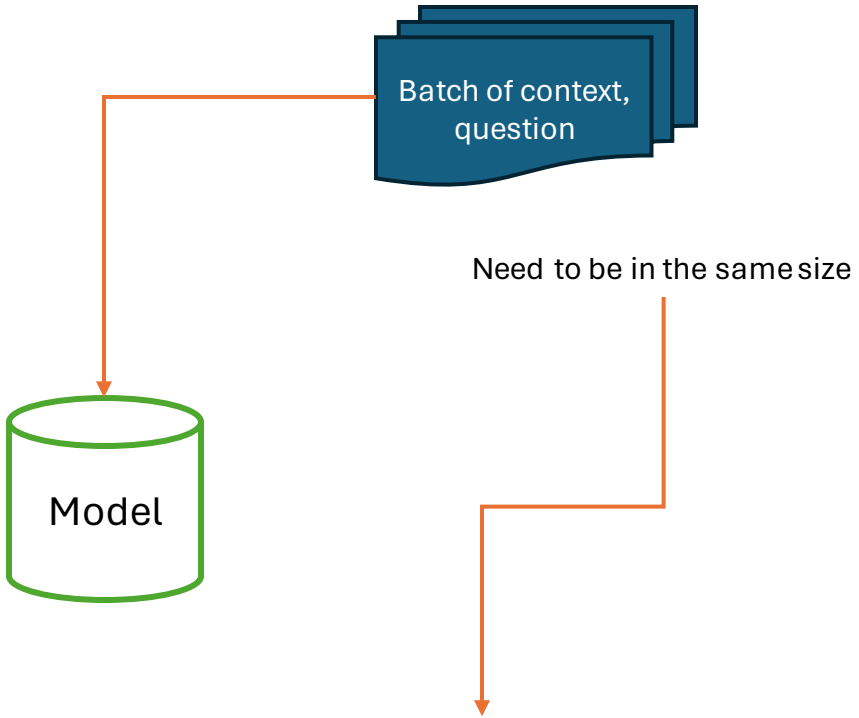
```
'context': 'My name is AIVN and I am from Vietnam.
'question': 'What is my name?',
'answer': 'AIVN'


'context': 'I love painting and my favorite artist
'question': 'What is my favorite activity?',
'answer': 'painting'


'context': 'I am studying computer science at the
'question': 'What am I studying?',
'answer': 'computer science'
```

```python
classes = set([item['answer'] for item in qa_dataset])
classes_to_idx = {
    cls_name: idx for idx, cls_name in enumerate(classes)
}
idx_to_classes = {
    idx: cls_name for idx, cls_name in enumerate(classes)
}
print(idx_to_classes)
```

```
{0: 'computer science', 1: 'painting', 2: 'AIVN'}
```

Classification Approach

❖ Step 2: Vectorization



1. Input Text = “Hello World”

Hello	World	<pad>	<pad>	<pad>

(padding)

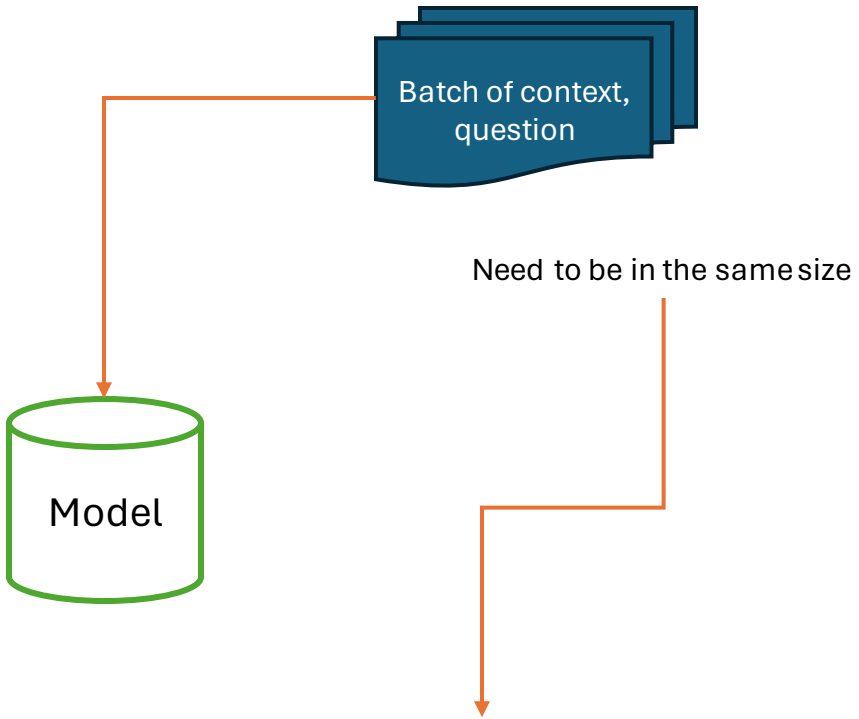
2. Input Text = “My name is AIVN I am from Vietnam”

My	name	is	AIVN	I

(truncate)

Classification Approach

❖ Step 2: Vectorization




```python
def pad_and_truncate(input_ids, max_seq_len):
    if len(input_ids) > max_seq_len:
        input_ids = input_ids[:max_seq_len]
    elif len(input_ids) < max_seq_len:
        input_ids += [PAD_IDX] * (max_seq_len - len(input_ids))

    return input_ids
```

```python
MAX_SEQ_LEN = 5
text = 'I love AIVN'
tokenized_text = tokenizer(text)
tokens = [vocab[token] for token in tokenized_text]
print(tokens)
padded_tokens = pad_and_truncate(tokens, MAX_SEQ_LEN)
print(padded_tokens)
```

```
[5, 23, 17]
[5, 23, 17, 1, 1]
```

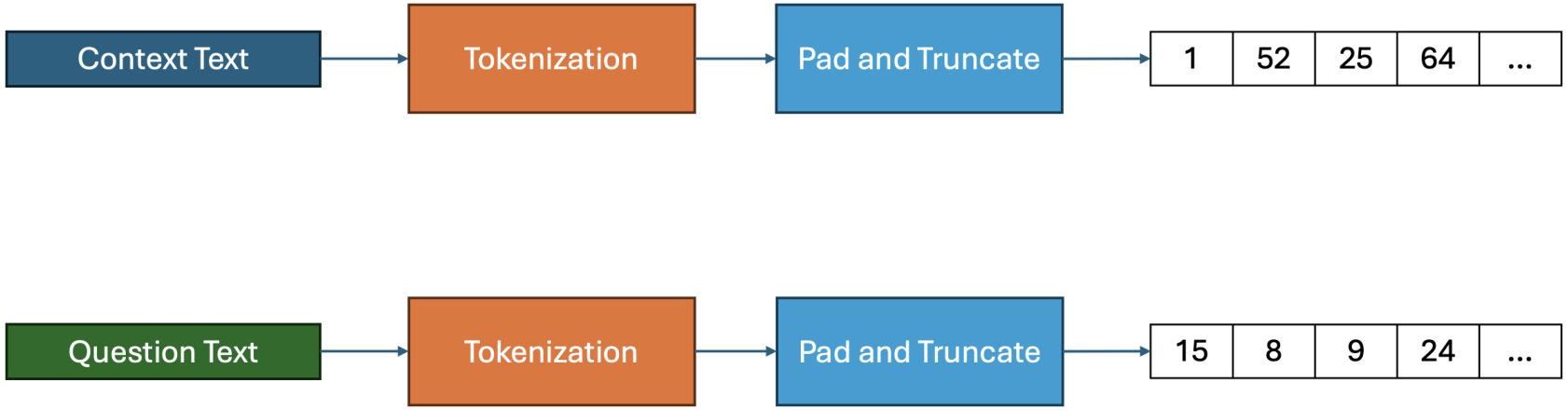
Classification Approach

❖ Step 2: Vectorization

```
13 def vectorize(question, context):
14     input_question_ids = [vocab[token] for token in tokenizer(question)]
15     input_context_ids = [vocab[token] for token in tokenizer(context)]
16 
17     input_question_ids = pad_and_truncate(input_question_ids, MAX_QUESTION_LEN)
18     input_context_ids = pad_and_truncate(input_context_ids, MAX_CONTEXT_LEN)
19 
20     input_question_ids = torch.tensor(input_question_ids, dtype=torch.long)
21     input_context_ids = torch.tensor(input_context_ids, dtype=torch.long)
22 
23     return input_question_ids, input_context_ids
```

Classification Approach

❖ Step 2: Vectorization




```
input_question_ids, input_context_ids = vectorize(
    qa_dataset[0]['context'],
    qa_dataset[0]['question']
)
print(input_question_ids)
print(input_context_ids)
print(classes_to_idx[qa_dataset[0]['answer']])
```

```
tensor([ 7, 14,  6, 17, 12,  5, 10, 21, 31,  8])
tensor([11,  6,  7, 14,  9,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1])
2
```

Classification Approach

❖ Step 3: Create datasets

```
'context': 'My name is AIVN and I am from Vietnam.',
'question': 'What is my name?',
'answer': 'AIVN'


'context': 'I love painting and my favorite artist is
'question': 'What is my favorite activity?',
'answer': 'painting'


'context': 'I am studying computer science at the Univ
'question': 'What am I studying?',
'answer': 'computer science'
```

```python
class QADataset(Dataset):
    def __init__(self, data):
        self.data = data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        item = self.data[idx]
        question_text = item['question']
        context_text = item['context']

        input_question_ids, input_context_ids = vectorize(
            question_text, context_text
        )

        answer_text = item['answer']
        answer_id = classes_to_idx[answer_text]
        answer_id = torch.tensor(answer_id, dtype=torch.long)

        return input_question_ids, input_context_ids, answer_id
```

Classification Approach

❖ Step 3: Create datasets

```python
def decode(input_ids):
    return ' '.join([vocab.lookup_token(token) for token in input_ids])
```

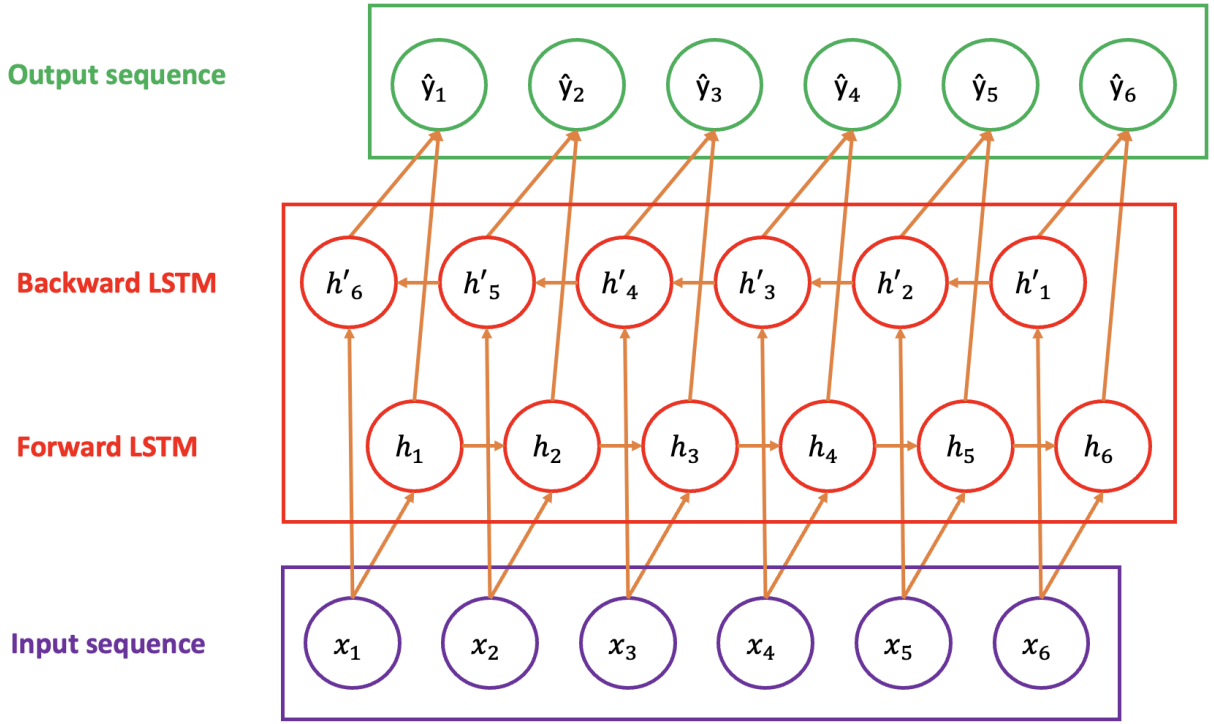
```python
train_dataset = QADataset(qa_dataset)
train_loader = DataLoader(train_dataset, batch_size=2, shuffle=True)
```

```python
for batch in train_loader:
    input_question_ids, input_context_ids, answer_id = batch
    print(input_question_ids, input_context_ids, answer_id)
```

```
tensor([[11, 10,  5, 15,  9,  1,  1,  1,  1,  1],
        [11,  6,  7, 13, 16,  9,  1,  1,  1,  1]]) tensor([[ 5, 10, 15, 20, 26, 19, 27, 29, 24, 28,  8,  1,  1,  1,  1],
        [ 5, 23, 25, 12,  7, 13, 18,  6, 32, 30, 22,  8,  1,  1,  1]]) tensor([0, 1])
tensor([[11,  6,  7, 14,  9,  1,  1,  1,  1,  1]]) tensor([[ 7, 14,  6, 17, 12,  5, 10, 21, 31,  8,  1,  1,  1,  1,  1]]) tensor([2])
```

Classification Approach

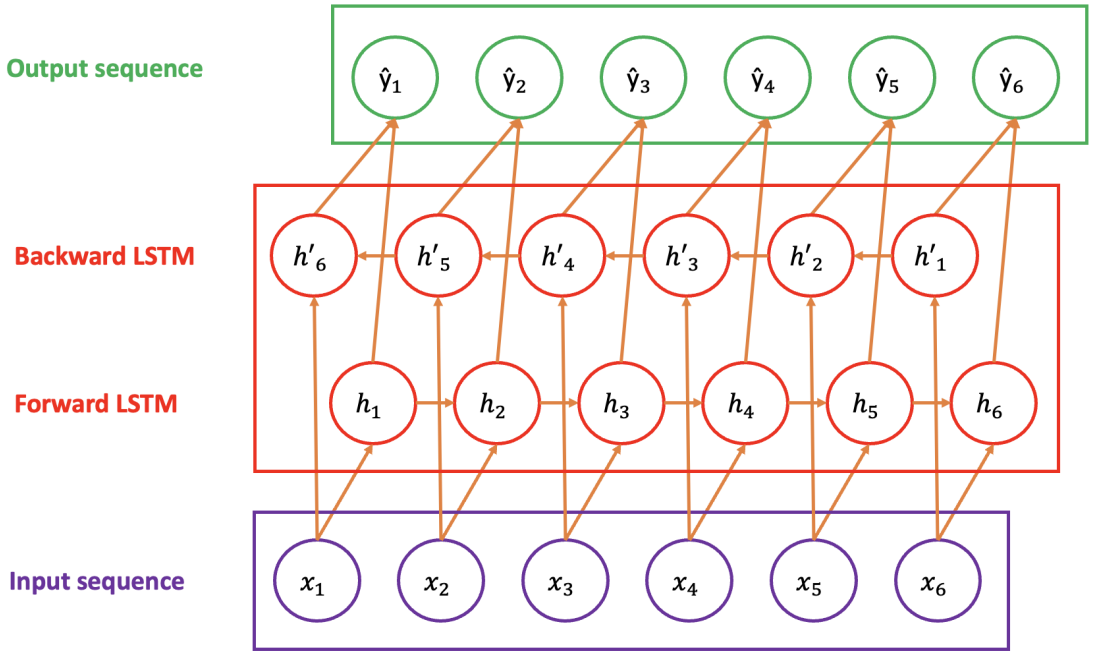
❖ Step 4: Create models




```python
class QAModel(nn.Module):
    def __init__(self,
        vocab_size, embedding_dim, hidden_size,
        n_layers, n_classes
    ):
        super(QAModel, self).__init__()
        self.question_embedding = nn.Embedding(
            vocab_size, embedding_dim
        )
        self.context_embedding = nn.Embedding(
            vocab_size, embedding_dim
        )

        self.lstm = nn.LSTM(
            embedding_dim * 2, hidden_size,
            num_layers=n_layers,
            batch_first=True,
            bidirectional=True
        )
        self.fc = nn.Linear(hidden_size * 2, n_classes)

    def forward(self, question, context):
        question_embed = self.question_embedding(question)
        context_embed = self.context_embedding(context)

        combined = torch.cat(
            (question_embed, context_embed),
            dim=1
        )

        lstm_out, _ = self.lstm(combined)
        lstm_out = lstm_out[:, -1, :]

        out = self.fc(lstm_out)

        return out
```

Classification Approach

❖ Step 4: Create models




```python
# Model parameters
EMBEDDING_DIM = 64
HIDDEN_SIZE = 128
VOCAB_SIZE = len(vocab)
N_LAYERS = 2
N_CLASSES = len(classes)

model = QAModel(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_SIZE, N_LAYERS, N_CLASSES)

input_context = torch.randint(0, 10, size=(1, MAX_CONTEXT_LEN))
input_question = torch.randint(0, 10, size=(1, MAX_QUESTION_LEN))
model.eval()
with torch.no_grad():
    logits = model(input_question, input_context)

print(logits.shape)
```

```
torch.Size([1, 3])
```

Classification Approach

❖ Step 5: Training model

```python
LR = 1e-3
EPOCHS = 20
optimizer = torch.optim.Adam(
    model.parameters(), lr=LR
)
criterion = nn.CrossEntropyLoss()

model.train()
for _ in range(EPOCHS):
    for idx, (input_question_ids, input_context_ids, answer_id) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs = model(input_question_ids, input_context_ids)
        loss = criterion(outputs, answer_id)
        loss.backward()
        optimizer.step()
        print(loss.item())
```

```
1.0841989517211914
1.2927165031433105
1.1452460289001465
1.0640251636505127
1.1059435606002808
1.052100419998169
1.0566418170928955
1.1575090885162354
1.1141493320465088
0.9826964139938354
1.1025049686431885
0.9516065716743469
0.994110643863678
1.120270013809204
1.0778357982635498
0.8329105377197266
1.0211389064788818
0.7478755712509155
0.9395896196365356
0.6099604964256287
0.5574547052383423
1.0696161985397339
0.7166918516159058
0.29412445425987244
0.5896788239479065
0.18702475726604462
0.4493691027164459
0.10719136893749237
0.3353314697742462
0.05937677249312401
0.05708907172083855
0.4555579721927643
```

Classification Approach

❖ Step 6: Test

```python
model.eval()
with torch.no_grad():
    sample = qa_dataset[0]
    context, question, answer = sample.values()
    question_ids, context_ids = vectorize(question, context)
    question_ids = question_ids.unsqueeze(0)
    context_ids = context_ids.unsqueeze(0)
    outputs = model(question_ids, context_ids)
    _, predicted = torch.max(outputs.data, 1)
    print(f'Context: {context}')
    print(f'Question: {question}')
    print(f'Prediction: {idx_to_classes[predicted.numpy()[0]]}')
```

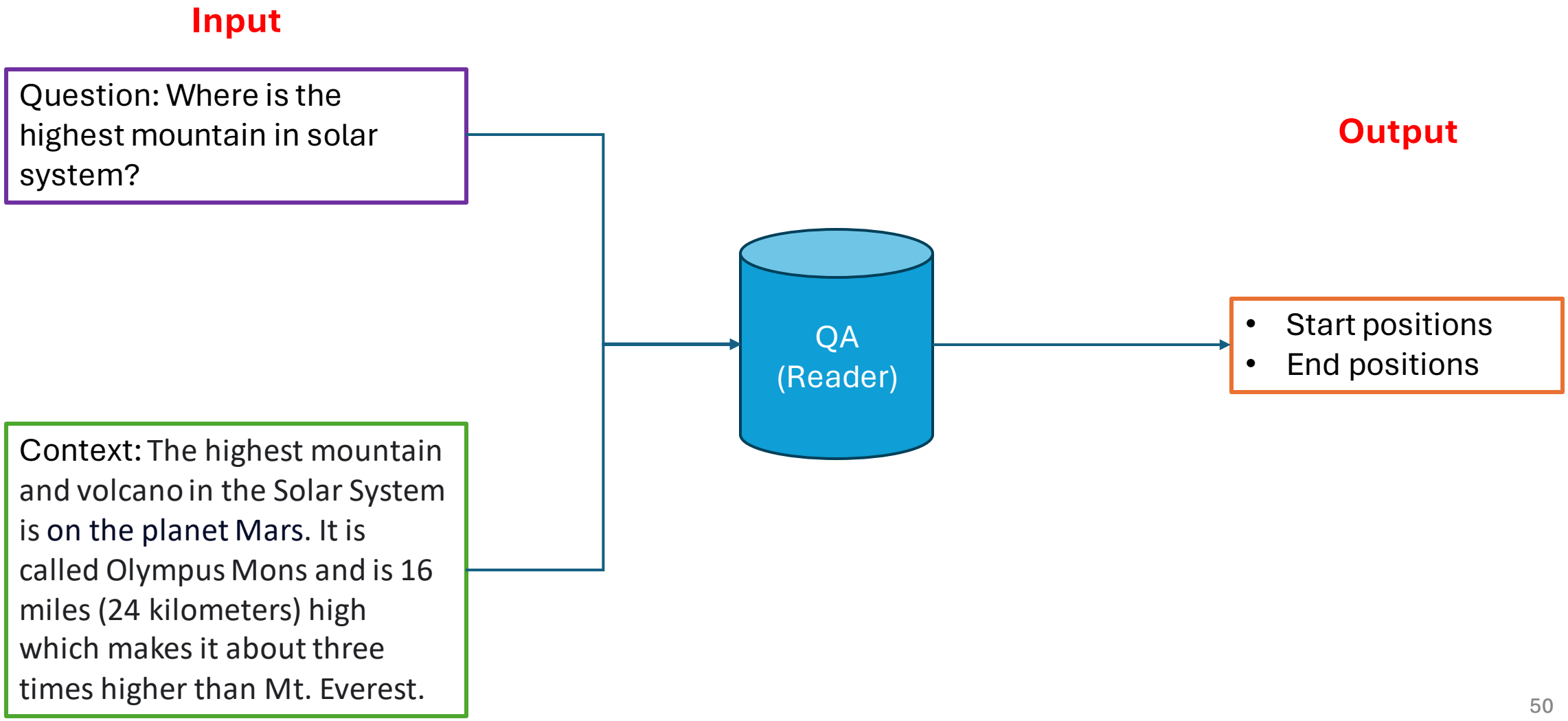
```
{
'computer science': 0,
'painting': 1,
'AIVN': 2
}
```

```
Context: My name is AIVN and I am from Vietnam.
Question: What is my name?
Prediction: AIVN
```

Extractive Approach

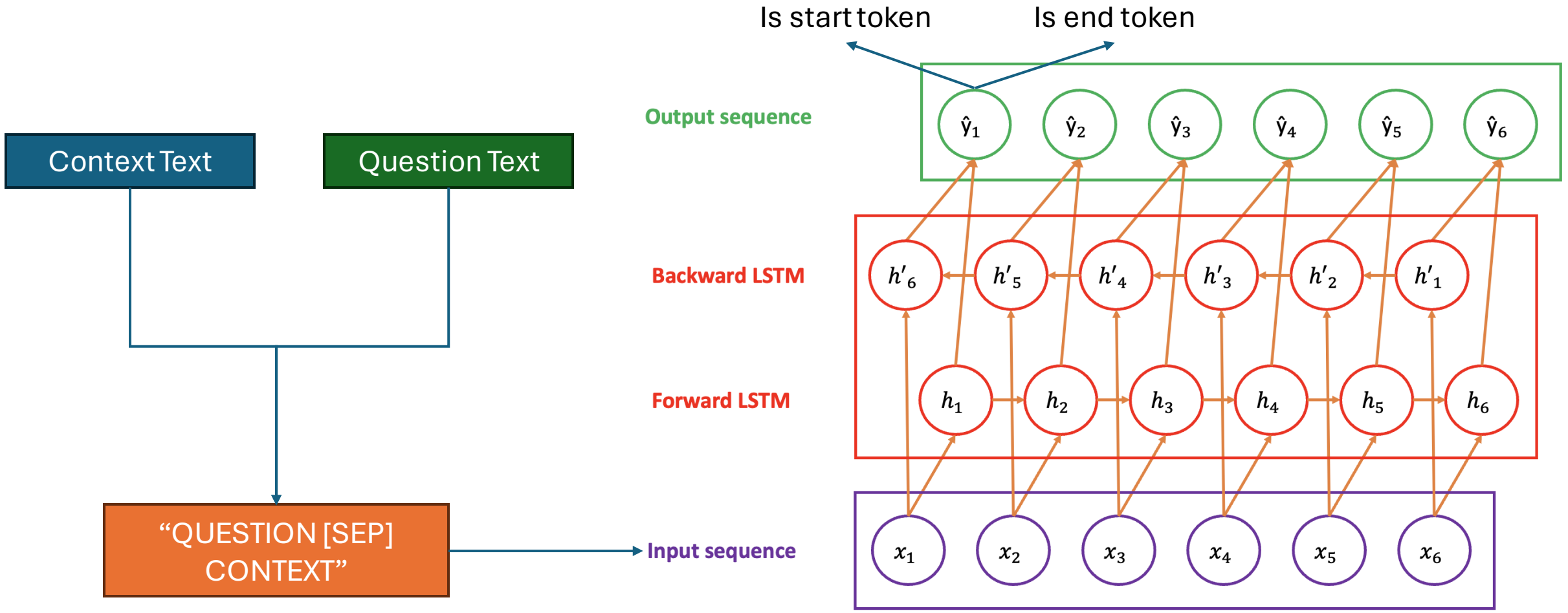
Extractive Approach

❖ Reader: Extractive Extraction



Extractive Approach

❖ Model Design



Extractive Approach

❖ Model Design

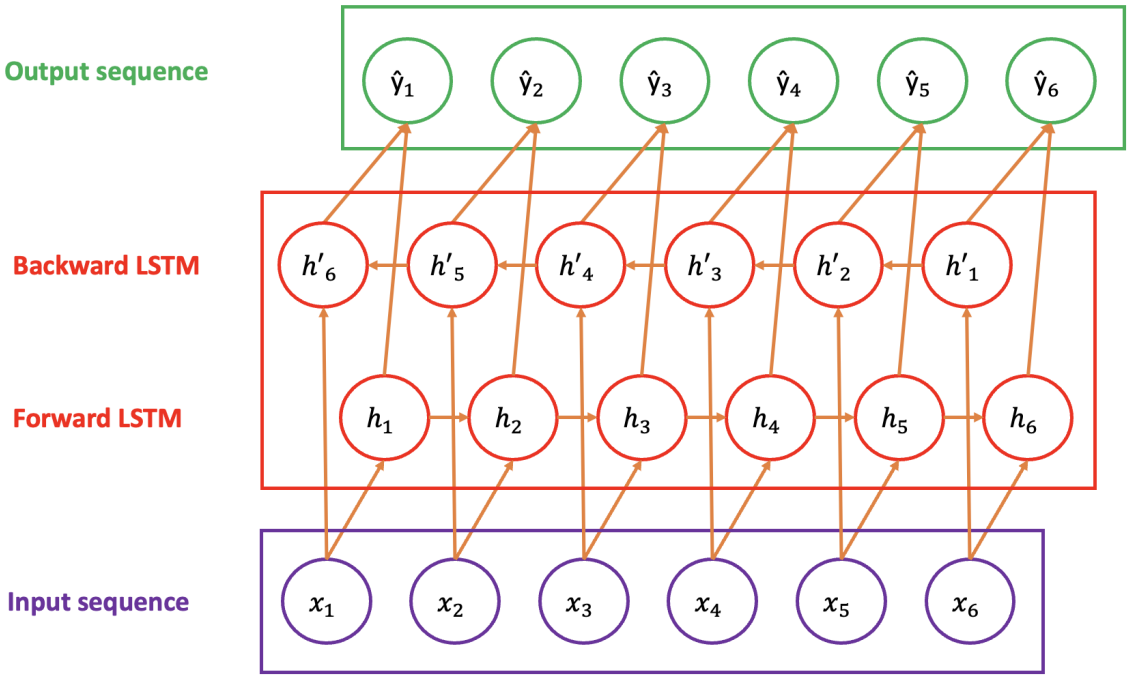
```
1 INPUT_QUESTION = 'What is my name?'
2 INPUT_CONTEXT = 'My name is AI Vietnam and I live in Vietnam.'
3 pipe(question=INPUT_QUESTION, context=INPUT_CONTEXT)
```

```
{'score': 0.97179114818573, 'start': 11, 'end': 21, 'answer': 'AI Vietnam'}
```

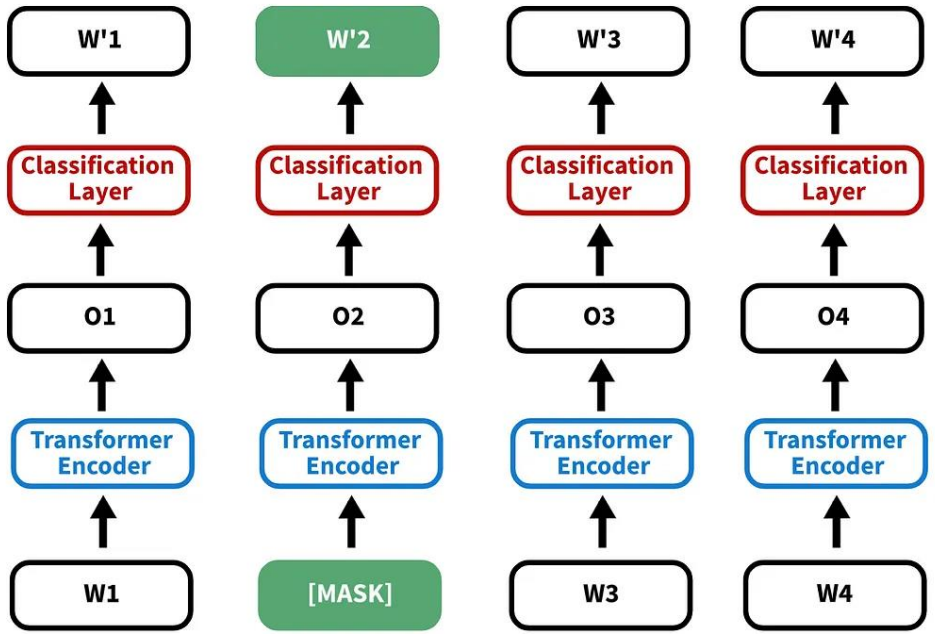
The answer span starts from index 11 to 21 in context text

Extractive Approach

❖ Model Design



LSTM-based



Transformer Encoder-based

https://towardsdatascience.com/how-to-answer-questions-with-machine-learning-6c21357a44fc

Extractive Approach

❖ LSTM-based Step 1: Import libraries and create temp datasets

```python
import torch
import torch.nn as nn
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import Dataset, DataLoader
```

```
'context': 'My name is AIVN and I am from Vietnam.',
'question': 'What is my name?',
'answer': 'AIVN'


'context': 'I love painting and my favorite artist is Vincent Van Gogh.',
'question': 'What is my favorite activity?',
'answer': 'painting'


'context': 'I am studying computer science at the University of Tokyo.',
'question': 'What am I studying?',
'answer': 'computer science'


'context': 'My favorite book is "To Kill a Mockingbird" by Harper Lee.',
'question': 'What is my favorite book?',
'answer': '"To Kill a Mockingbird"'
```

Extractive Approach

❖ LSTM-based Step 2: Build vocabulary

```python
# Define tokenizer function
tokenizer = get_tokenizer('basic_english')

# Create a function to yield list of tokens
def yield_tokens(data):
    for item in data:
        yield tokenizer(item['context'] + ' <sep> ' + item['question'])

# Create vocabulary
vocab = build_vocab_from_iterator(
    yield_tokens(qa_dataset),
    specials=['<unk>', '<pad>', '<bos>', '<eos>', '<sep>']
)
vocab.set_default_index(vocab['<unk>'])
vocab.get_stoi()
```

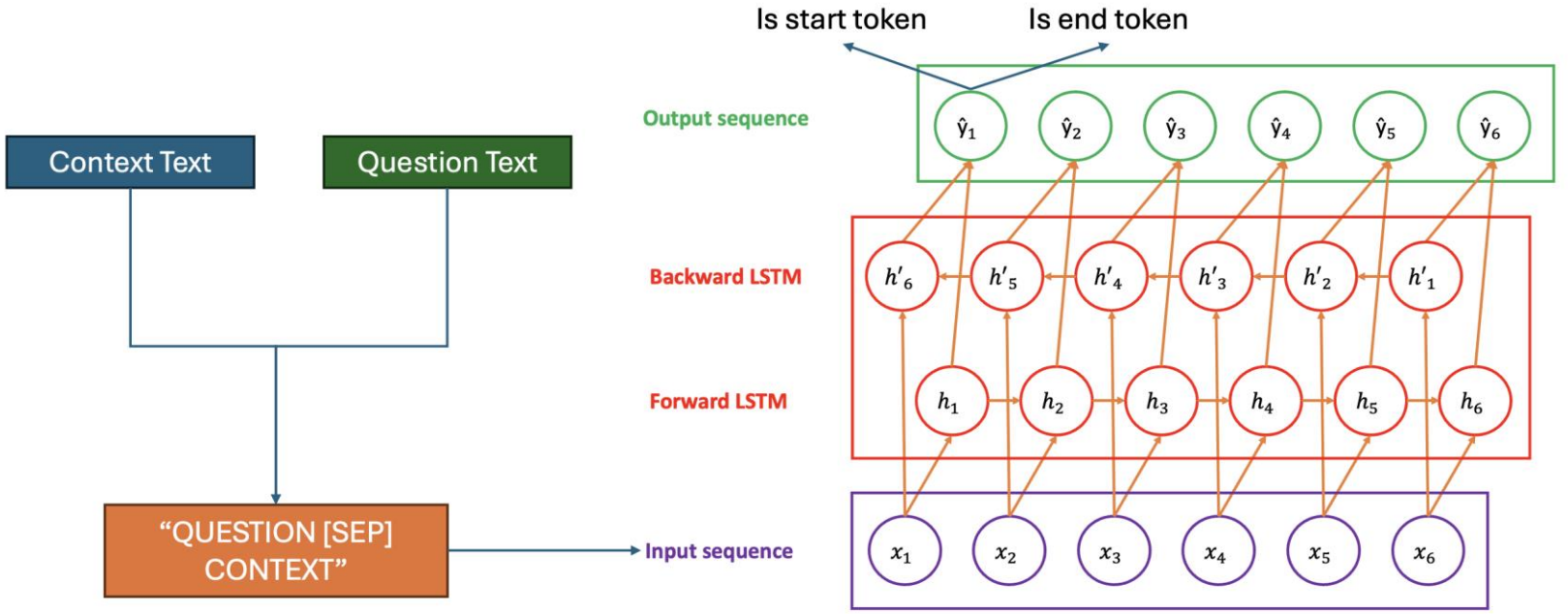
```
{'to': 24,
 ',': 25,
 'pet': 21,
 'who': 61,
 'gogh': 39,
 'the': 23,
 'fetch': 37,
 'play': 52,
 'van': 56,
 'now': 19,
 'was': 59,
 'a': 14,
 'name': 13,
 'aivn': 27,
 'i': 5,
 'studying': 22,
 'and': 15,
 'where': 60,
 '<unk>': 0,
 'favorite': 11,
```

Extractive Approach

❖ LSTM-based Step 3: Create vectorize function

We combined context and question into 1 sequence:

```
'context': 'My name is AIVN and I
am from Vietnam.',
'question': 'What is my name?'

-> What is my name ? <sep> My name
is AIVN and I am from Vietnam .
```



Extractive Approach

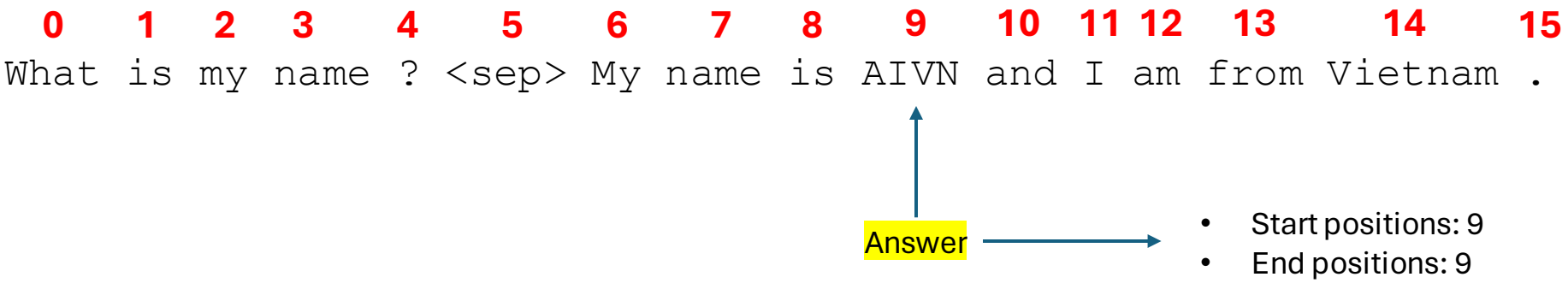
❖ LSTM-based Step 3: Create vectorize function

```
'context': 'My name is AIVN and I am from Vietnam.',
'question': 'What is my name?'
```

-> What is my name ? <sep> My name is AIVN and I am from Vietnam .

```
vectorize(
    qa_dataset[0]['question'],
    qa_dataset[0]['context'],
    qa_dataset[0]['answer'],
)
```

```
(tensor([10,  6,  7, 13,  9,  4,  7, 13,  6, 27, 15,  5, 12, 38, 57,  8,  1,  1,
         1,  1,  1,  1]),
 tensor(9),
 tensor(9))
```

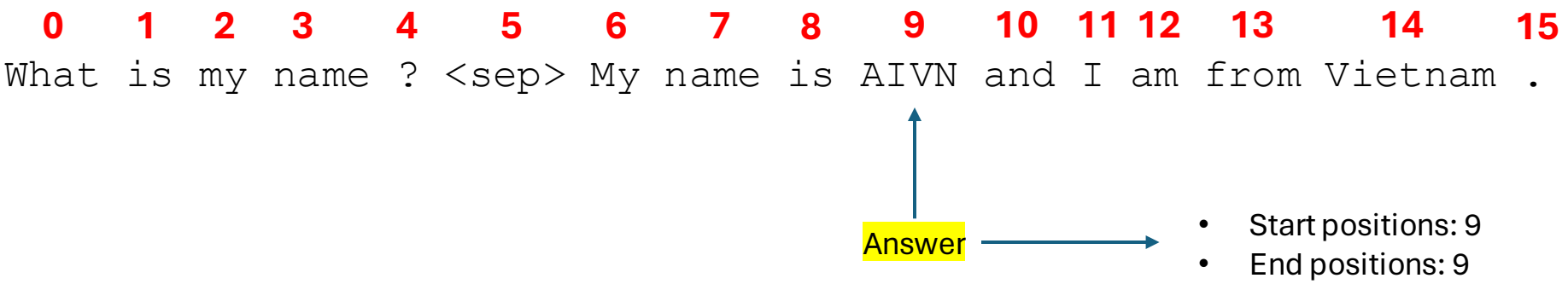


Extractive Approach

❖ LSTM-based Step 3: Create vectorize function

```python
MAX_SEQ_LENGTH = 22
PAD_IDX = vocab['<pad>']

def pad_and_truncate(input_ids, max_seq_len):
    if len(input_ids) > max_seq_len:
        input_ids = input_ids[:max_seq_len]
    elif len(input_ids) < max_seq_len:
        input_ids += [PAD_IDX] * (max_seq_len - len(input_ids))

    return input_ids
```

```python
def vectorize(question, context, answer):
    input_text = question + ' <sep> ' + context
    input_ids = [vocab[token] for token in tokenizer(input_text)]
    input_ids = pad_and_truncate(input_ids, MAX_SEQ_LENGTH)

    answer_ids = [vocab[token] for token in tokenizer(answer)]
    start_positions = input_ids.index(answer_ids[0])
    end_positions = start_positions + len(answer_ids) - 1

    input_ids = torch.tensor(input_ids, dtype=torch.long)
    start_positions = torch.tensor(start_positions, dtype=torch.long)
    end_positions = torch.tensor(end_positions, dtype=torch.long)

    return input_ids, start_positions, end_positions
```



Extractive Approach

❖ LSTM-based Step 4: Create datasets

```
'context': 'My name is AIVN and I am from Vietnam.',
'question': 'What is my name?',
'answer': 'AIVN'


'context': 'I love painting and my favorite artist is Vi
'question': 'What is my favorite activity?',
'answer': 'painting'


'context': 'I am studying computer science at the Univer
'question': 'What am I studying?',
'answer': 'computer science'
```

```python
class QADataset(Dataset):
    def __init__(self, data):
        self.data = data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        item = self.data[idx]
        question_text = item['question']
        context_text = item['context']
        answer_text = item['answer']

        input_ids, start_positions, end_positions = vectorize(
            question_text, context_text, answer_text
        )

        return input_ids, start_positions, end_positions
```

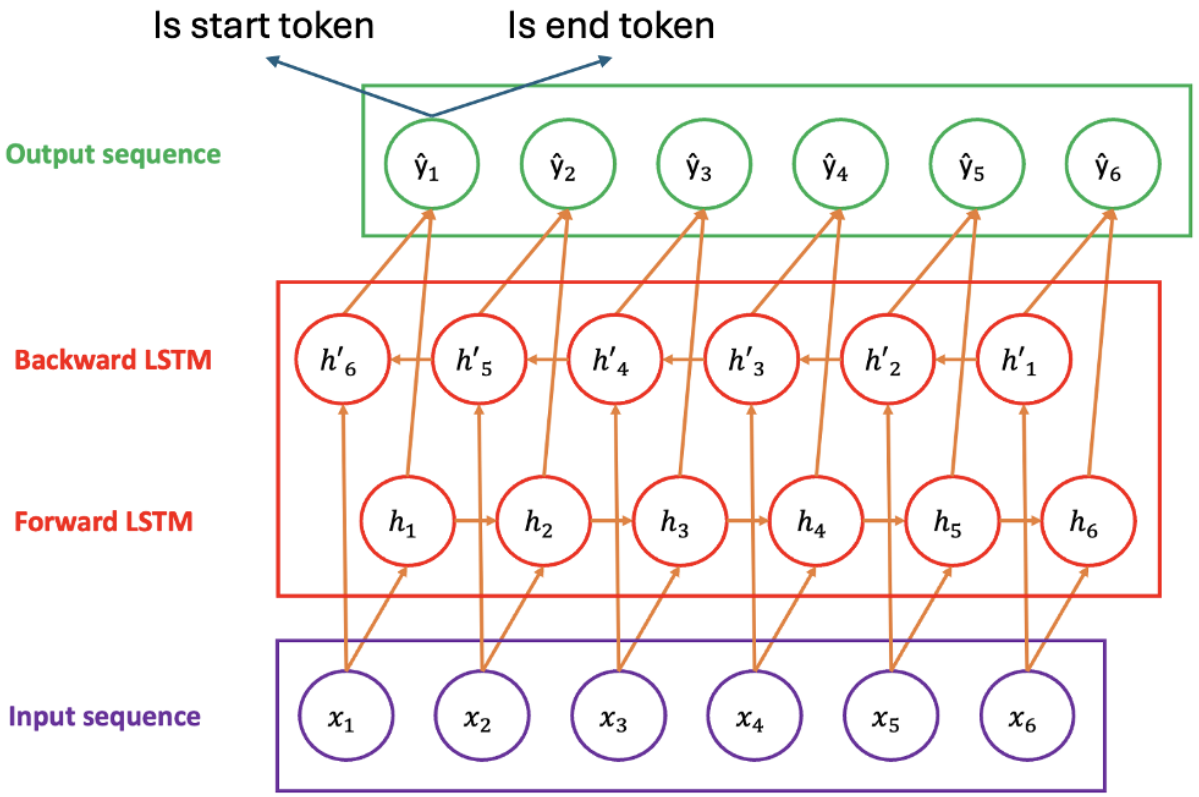
Extractive Approach

❖ LSTM-based Step 4: Create datasets

```
tensor([10,  6,  7, 13,  9,  4,  7, 13,  6, 27, 15,  5, 12, 38, 57,  8,  1,  1,
         1,  1,  1,  1])
what is my name ? <sep> my name is aivn and i am from vietnam . <pad> <pad> <pad> <pad> <pad> <pad>
aivn
tensor([10,  6,  7, 11, 26,  9,  4,  5, 44, 50, 15,  7, 11, 28,  6, 58, 56, 39,
         8,  1,  1,  1])
what is my favorite activity ? <sep> i love painting and my favorite artist is vincent van gogh . <pad> <pad> <pad>
painting
tensor([10, 12,  5, 22,  9,  4,  5, 12, 22, 34, 53, 29, 23, 55, 20, 54,  8,  1,
         1,  1,  1,  1])
what am i studying ? <sep> i am studying computer science at the university of tokyo . <pad> <pad> <pad> <pad> <pad>
computer science
tensor([10,  6,  7, 11, 16,  9,  4,  7, 11, 16,  6, 24, 42, 14, 47, 32, 40, 43,
         8,  1,  1,  1])
what is my favorite book ? <sep> my favorite book is to kill a mockingbird by harper lee . <pad> <pad> <pad>
to kill a mockingbird
tensor([10,  6, 23, 13, 20,  7, 21,  9,  4,  5, 41, 14, 21, 36, 48, 46, 61, 45,
        24, 52, 37,  8])
what is the name of my pet ? <sep> i have a pet dog named max who loves to play fetch .
max
tensor([60, 35,  5, 18, 19,  9,  4,  5, 59, 30, 17, 51, 25, 31, 19,  5, 18, 17,
        49, 62, 33,  8])
where do i live now ? <sep> i was born in paris , but now i live in new york city .
new york city
```

Extractive Approach

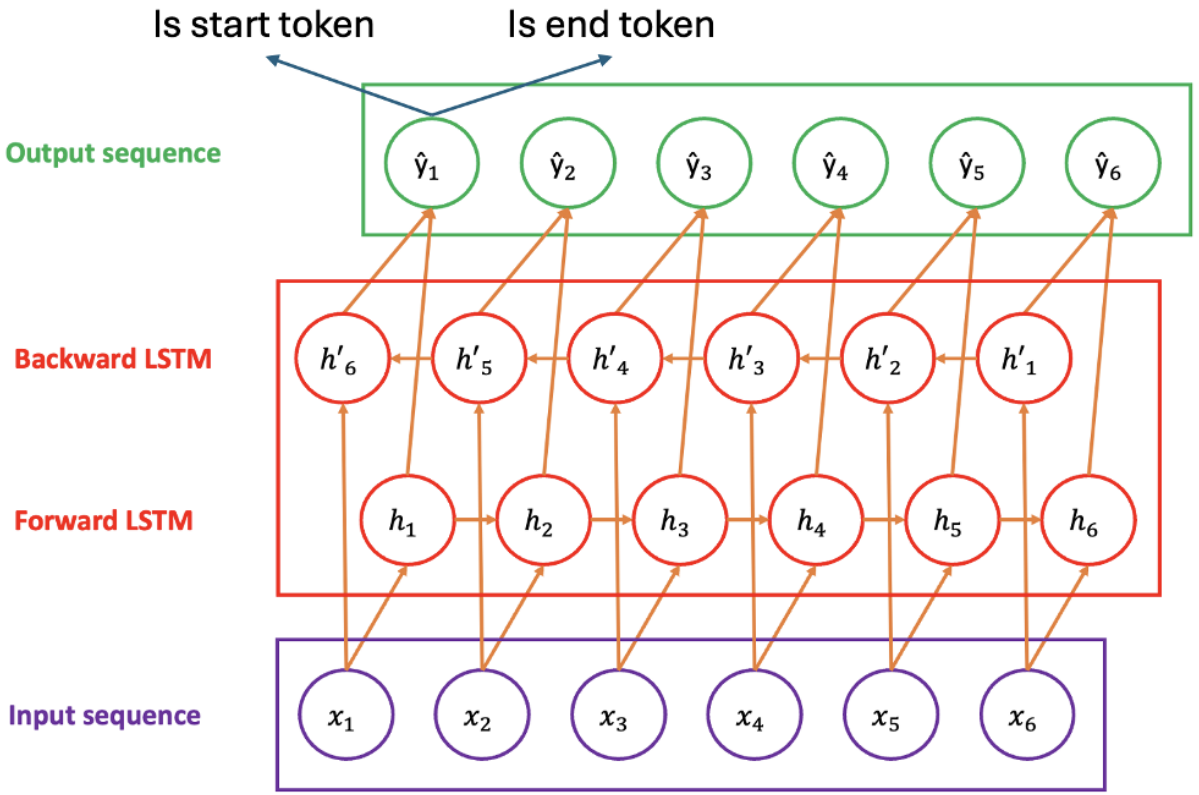
❖ LSTM-based Step 5: Create models




```python
class QAModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_size, n_layers):
        super(QAModel, self).__init__()
        self.input_embedding = nn.Embedding(vocab_size, embedding_dim)

        self.lstm = nn.LSTM(
            embedding_dim, hidden_size,
            num_layers=n_layers,
            batch_first=True,
            bidirectional=True
        )

        self.start_linear = nn.Linear(hidden_size * 2, 1)
        self.end_linear = nn.Linear(hidden_size * 2, 1)

    def forward(self, text):
        input_embedded = self.input_embedding(text)
        lstm_out, _ = self.lstm(input_embedded)

        start_logits = self.start_linear(lstm_out).squeeze(-1)
        end_logits = self.end_linear(lstm_out).squeeze(-1)

        return start_logits, end_logits
```

Extractive Approach

❖ LSTM-based Step 5: Create models




```
28 # Model parameters
29 EMBEDDING_DIM = 64
30 HIDDEN_SIZE = 128
31 VOCAB_SIZE = len(vocab)
32 N_LAYERS = 2
33
34 model = QAModel(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_SIZE, N_LAYERS)
35
36 input = torch.randint(0, 10, size=(1, 10))
37 print(input.shape)
38 model.eval()
39 with torch.no_grad():
40     start_logits, end_logits = model(input)
41
42 print(start_logits.shape)
```

```
torch.Size([1, 10])
torch.Size([1, 10])
```

Extractive Approach

❖ LSTM-based Step 6: Training

```
LR = 1e-3
optimizer = torch.optim.Adam(model.parameters(), lr=LR)
criterion = nn.CrossEntropyLoss()
```

```
EPOCHS = 15

model.train()
for _ in range(EPOCHS):
    for idx, (input_ids, start_positions, end_positions) in enumerate(train_loader):
        optimizer.zero_grad()
        start_logits, end_logits = model(input_ids)
        start_loss = criterion(start_logits, start_positions)
        end_loss = criterion(end_logits, end_positions)
        total_loss = (start_loss + end_loss) / 2
        total_loss.backward()
        optimizer.step()
        print(total_loss.item())
```

```
3.095088005065918
3.091099262237549
3.053279399871826
2.9391098022460938
2.967653274536133
2.9471912384033203
2.8976731300354004
2.7128355503082275
2.7618019580841064
2.649993419647217
2.5278143882751465
2.4534716606140137
2.2229530811309814
2.2061705589294434
2.1098806858062744
1.8857851028442383
1.6195350885391235
1.6607139110565186
1.2966654300689697
1.49912428855896
1.1670032739639282
1.1337673664093018
0.9600555300712585
1.0188782215118408
0.8088362812995911
0.9213032126426697
0.6407122015953064
0.7290103435516357
0.5646023154258728
0.5509380102157593
0.325381338596344
```

Extractive Approach

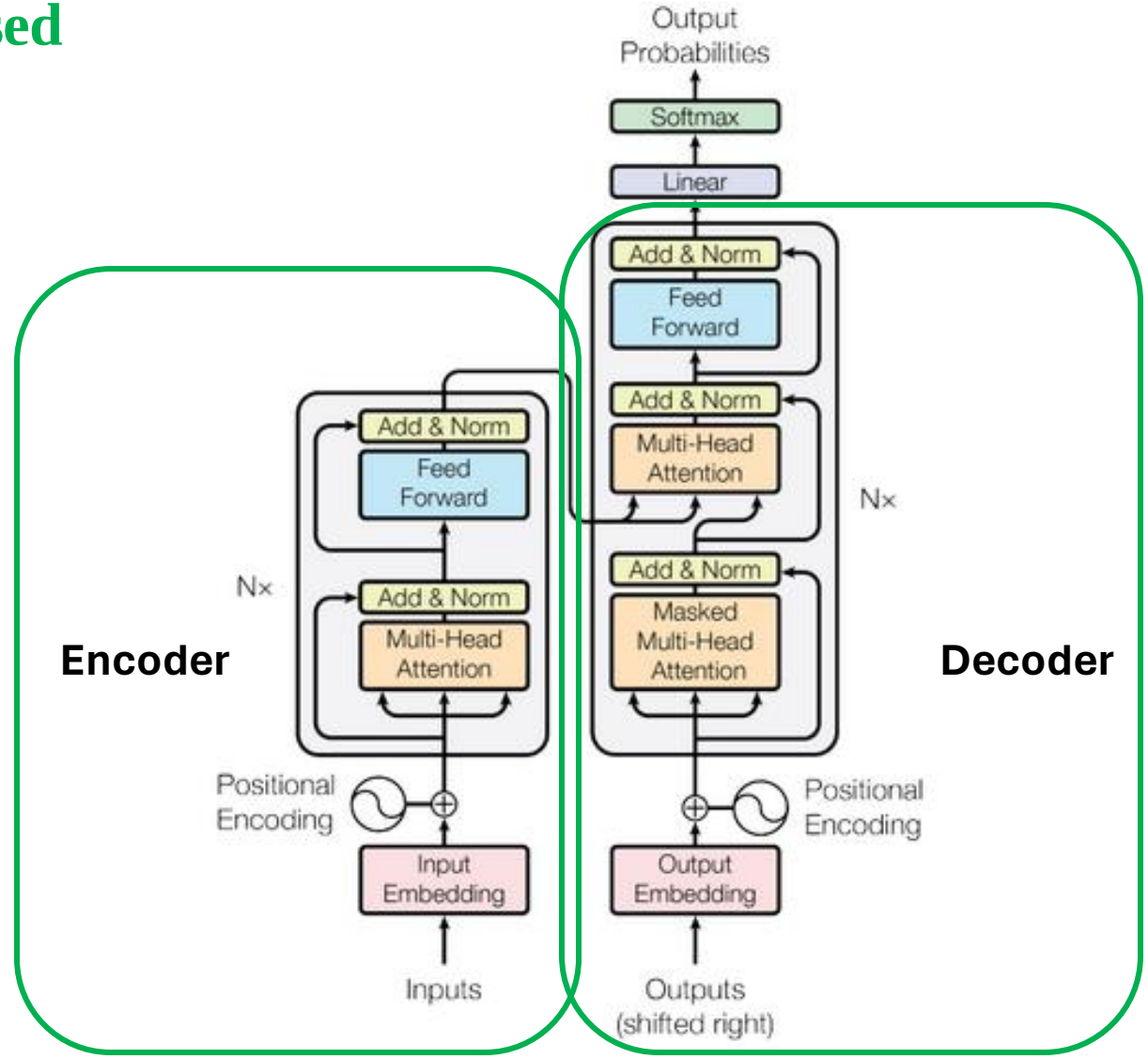
❖ LSTM-based Step 7: Test

```python
model.eval()
with torch.no_grad():
    sample = qa_dataset[1]
    context, question, answer = sample.values()
    input_ids, start_positions, end_positions = vectorize(question, context, answer)
    input_ids = input_ids.unsqueeze(0)
    start_logits, end_logits = model(input_ids)

    offset = len(tokenizer(question)) + 2
    start_position = torch.argmax(start_logits, dim=1).numpy()[0]
    end_position = torch.argmax(end_logits, dim=1).numpy()[0]

    start_position -= offset
    end_position -= offset

    start_position = max(start_position, 0)
    end_position = min(end_position, len(tokenizer(context)) - 1)

    if end_position >= start_position:
        # Extract the predicted answer span
        context_tokens = tokenizer(context)
        predicted_answer_tokens = context_tokens[start_position:end_position + 1]
        predicted_answer = ' '.join(predicted_answer_tokens)
    else:
        predicted_answer = ''
```

```
Context: I am studying computer science at the University of Tokyo.
Question: What am I studying?
Start position: 3
End position: 4
Answer span: computer science
```

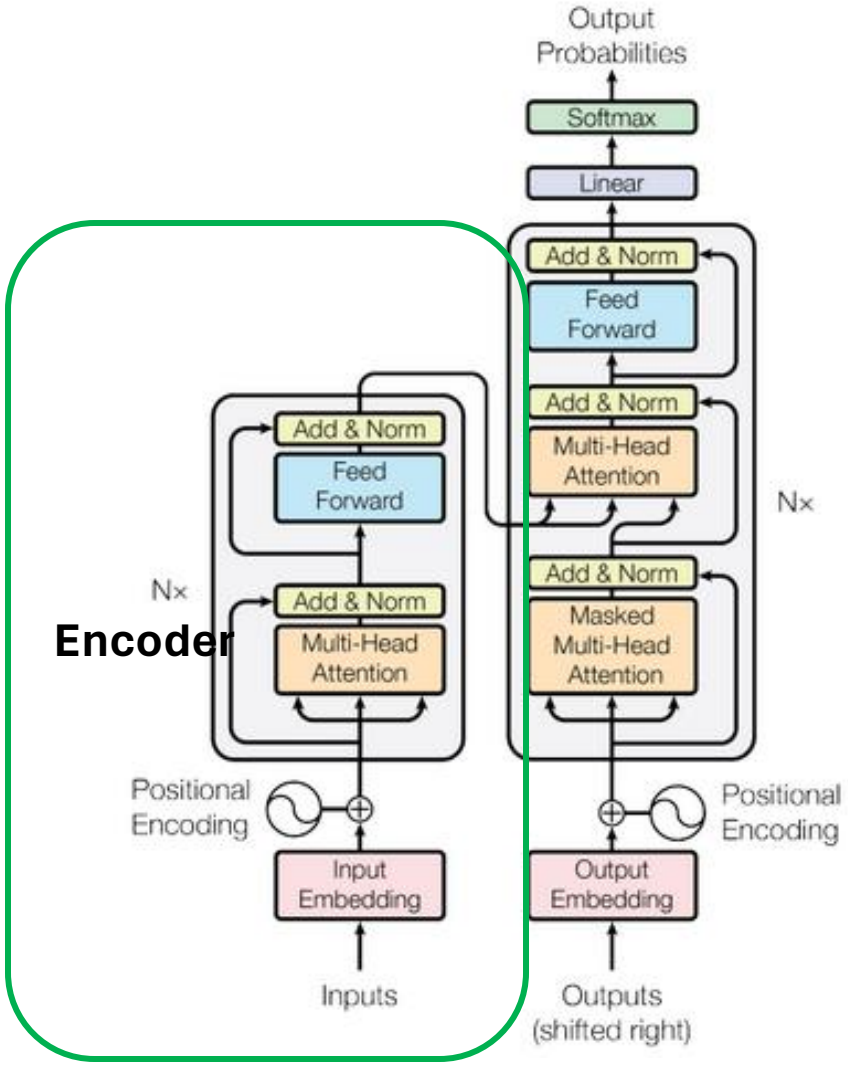
Extractive Approach

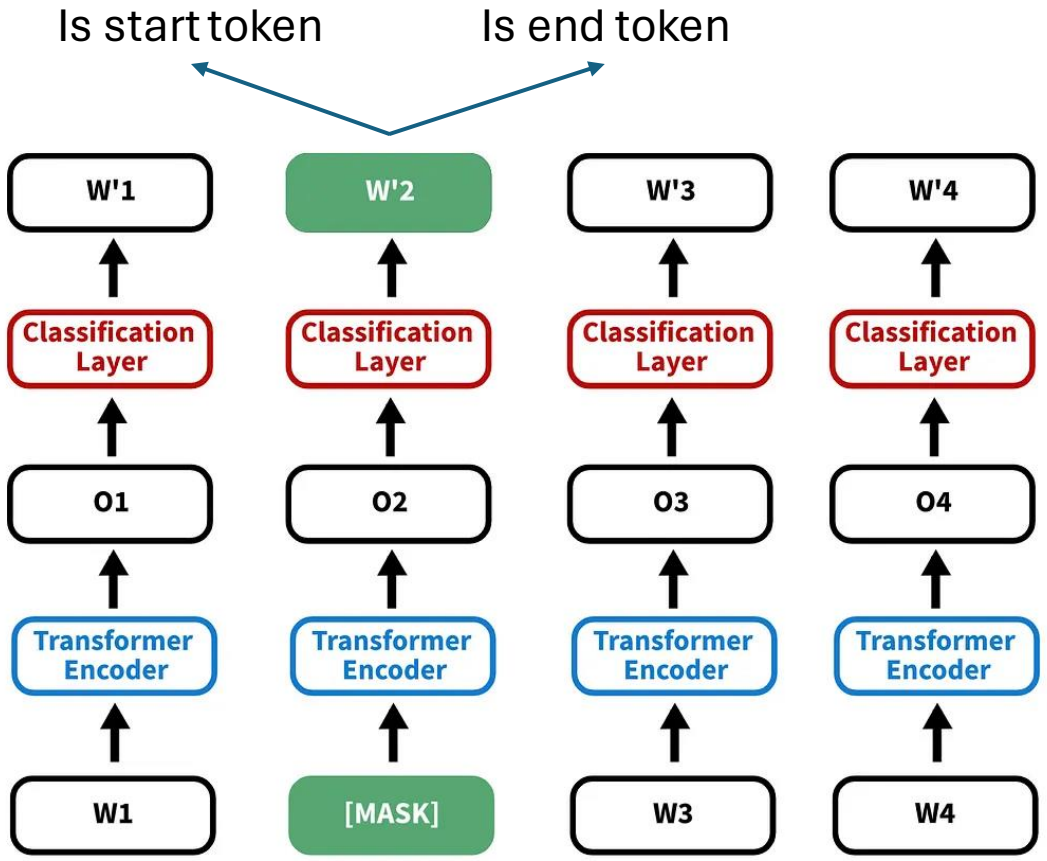
❖ Transformer-based



Extractive Approach

❖ Transformer-based





https://arxiv.org/abs/1907.11692

https://towardsdatascience.com/how-to-answer-questions-with-machine-learning-6c21357a44fc

Extractive Approach

❖ Transformer-based Step 1: Import libraries and create temp dataset

```python
import torch
import torch.nn as nn
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import Dataset, DataLoader
```

```
'context': 'My name is AIVN and I am from Vietnam.',
'question': 'What is my name?',
'answer': 'AIVN'


'context': 'I love painting and my favorite artist is Vincent Van Gogh.',
'question': 'What is my favorite activity?',
'answer': 'painting'


'context': 'I am studying computer science at the University of Tokyo.',
'question': 'What am I studying?',
'answer': 'computer science'


'context': 'My favorite book is "To Kill a Mockingbird" by Harper Lee.',
'question': 'What is my favorite book?',
'answer': '"To Kill a Mockingbird"'
```

Extractive Approach

❖ Transformer-based Step 2: Build vocabulary

```python
# Define tokenizer function
tokenizer = get_tokenizer('basic_english')

# Create a function to yield list of tokens
def yield_tokens(data):
    for item in data:
        yield tokenizer('<cls> ' + item['context'] + ' <sep> ' + item['question'])

# Create vocabulary
vocab = build_vocab_from_iterator(
    yield_tokens(qa_dataset),
    specials=['<unk>', '<pad>', '<bos>', '<eos>', '<sep>', '<cls>']
)
vocab.set_default_index(vocab['<unk>'])
vocab.get_stoi()
```

```
{'to': 24,
 ',': 25,
 'pet': 21,
 'who': 61,
 'gogh': 39,
 'the': 23,
 'fetch': 37,
 'play': 52,
 'van': 56,
 'now': 19,
 'was': 59,
 'a': 14,
 'name': 13,
 'aivn': 27,
 'i': 5,
 'studying': 22,
 'and': 15,
 'where': 60,
 '<unk>': 0,
 'favorite': 11,
```

Extractive Approach

❖ Transformer-based Step 2: Build vocabulary

<cls> what do I do ? <sep> my name is aivn and i am from vietnam . <pad> <pad> <pad> <pad> <pad> <pad>

What is the purpose of <cls> token?

In case the question is non answerable, we can label start and end positions at <cls> token

<cls> what do I do ? <sep> my name is aivn and i am from vietnam . <pad> <pad> <pad> <pad> <pad> <pad>

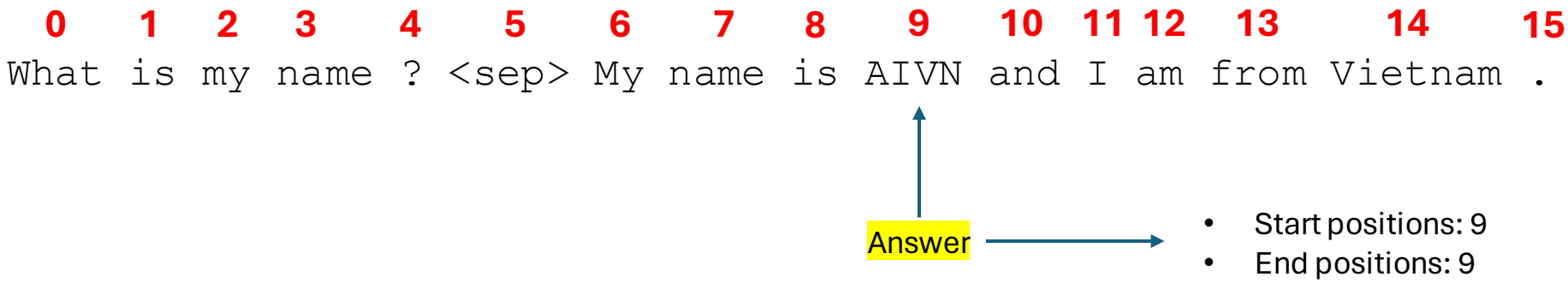
- Start position: 0
- End position: 0

Extractive Approach

❖ Trasnformer-based Step 3: Create vectorize function

```python
MAX_SEQ_LENGTH = 22
PAD_IDX = vocab['<pad>']

def pad_and_truncate(input_ids, max_seq_len):
    if len(input_ids) > max_seq_len:
        input_ids = input_ids[:max_seq_len]
    elif len(input_ids) < max_seq_len:
        input_ids += [PAD_IDX] * (max_seq_len - len(input_ids))

    return input_ids
```

```python
def vectorize(question, context, answer):
    input_text = question + ' <sep> ' + context
    input_ids = [vocab[token] for token in tokenizer(input_text)]
    input_ids = pad_and_truncate(input_ids, MAX_SEQ_LENGTH)

    answer_ids = [vocab[token] for token in tokenizer(answer)]
    start_positions = input_ids.index(answer_ids[0])
    end_positions = start_positions + len(answer_ids) - 1

    input_ids = torch.tensor(input_ids, dtype=torch.long)
    start_positions = torch.tensor(start_positions, dtype=torch.long)
    end_positions = torch.tensor(end_positions, dtype=torch.long)

    return input_ids, start_positions, end_positions
```



Extractive Approach

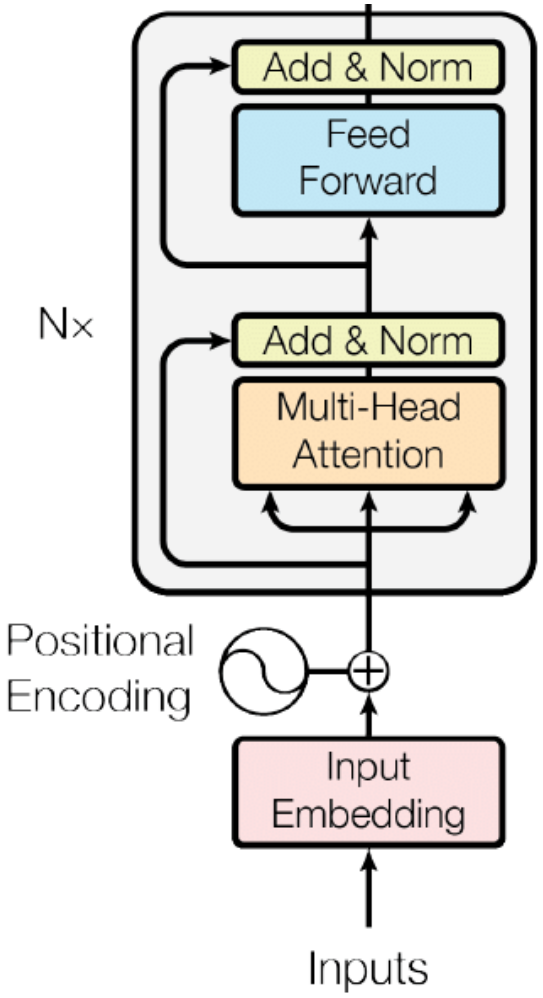
❖ Transformer-based Step 4: Create datasets

```
'context': 'My name is AIVN and I am from Vietnam.',
'question': 'What is my name?',
'answer': 'AIVN'


'context': 'I love painting and my favorite artist is Vi
'question': 'What is my favorite activity?',
'answer': 'painting'


'context': 'I am studying computer science at the Univer
'question': 'What am I studying?',
'answer': 'computer science'
```

```python
class QADataset(Dataset):
    def __init__(self, data):
        self.data = data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        item = self.data[idx]
        question_text = item['question']
        context_text = item['context']
        answer_text = item['answer']

        input_ids, start_positions, end_positions = vectorize(
            question_text, context_text, answer_text
        )

        return input_ids, start_positions, end_positions
```

Extractive Approach

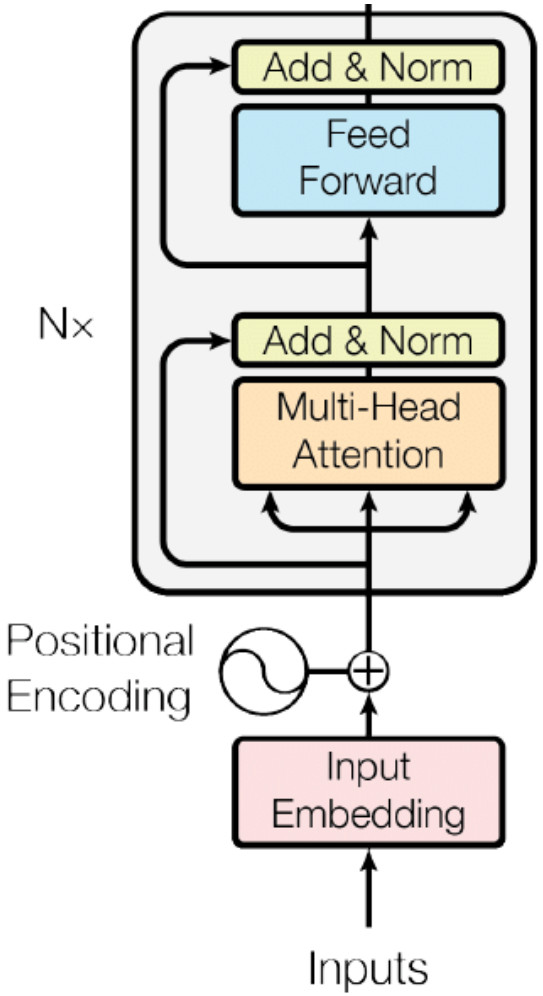
❖ Transformer-based Step 5: Create model




```python
import math
import torch.nn as nn
import torch.optim as optim

class TransformerBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, ff_dim):
        super().__init__()
        self.attn = nn.MultiheadAttention(embed_dim=embed_dim,
                                          num_heads=num_heads)
        self.ffn = nn.Linear(in_features=embed_dim,
                             out_features=ff_dim)
        self.layernorm_1 = nn.LayerNorm(normalized_shape=embed_dim)
        self.layernorm_2 = nn.LayerNorm(normalized_shape=embed_dim)

    def forward(self, query, key, value):
        attn_output, _ = self.attn(query, key, value)
        out_1 = self.layernorm_1(query + attn_output)
        ffn_output = self.ffn(out_1)
        x = self.layernorm_2(out_1 + ffn_output)

        return x
```

Extractive Approach

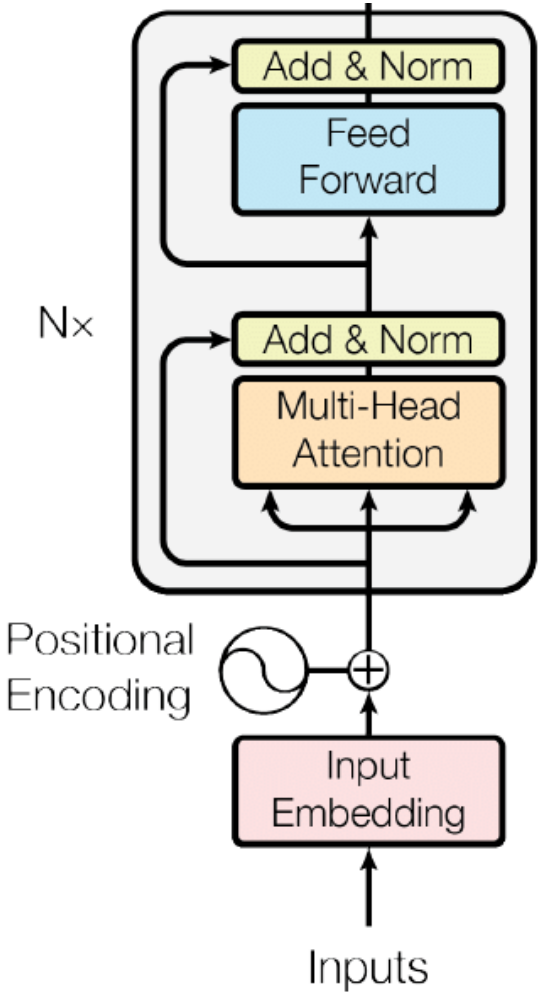
❖ Transformer-based Step 5: Create model




```python
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        super(PositionalEncoding, self).__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(0), :]

        return x
```

Extractive Approach

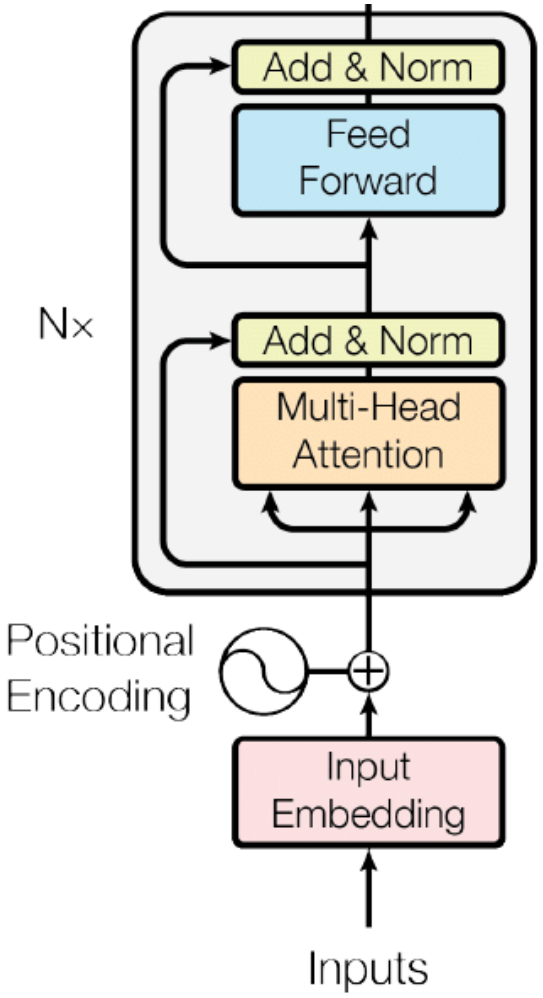
❖ Transformer-based Step 5: Create model




```python
class QAModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, n_heads, ff_dim, max_len):
        super(QAModel, self).__init__()
        self.input_embedding = nn.Embedding(vocab_size, embedding_dim)
        self.pos_encoder = PositionalEncoding(embedding_dim, max_len)
        self.transformer = TransformerBlock(embedding_dim, n_heads, ff_dim)

        self.start_linear = nn.Linear(ff_dim, 1)
        self.end_linear = nn.Linear(ff_dim, 1)

    def forward(self, text):
        input_embedded = self.input_embedding(text)
        input_embedded = self.pos_encoder(input_embedded)
        transformer_out = self.transformer(input_embedded, input_embedded, input_embedded)
        start_logits = self.start_linear(transformer_out).squeeze(-1)
        end_logits = self.end_linear(transformer_out).squeeze(-1)

        return start_logits, end_logits
```

Extractive Approach

❖ Transformer-based Step 5: Create model




```python
# Model parameters
EMBEDDING_DIM = 128
FF_DIM = 128
N_HEADS = 1
VOCAB_SIZE = len(vocab)

model = QAModel(VOCAB_SIZE, EMBEDDING_DIM, N_HEADS, FF_DIM, MAX_SEQ_LEN)

input = torch.randint(0, 10, size=(1, 10))
print(input.shape)
model.eval()
with torch.no_grad():
    start_logits, end_logits = model(input)

print(start_logits.shape)
```

```
torch.Size([1, 10])
torch.Size([1, 10])
```

Extractive Approach

❖ Transformer-based Step 6: Training

```
LR = 1e-3
optimizer = torch.optim.Adam(model.parameters(), lr=LR)
criterion = nn.CrossEntropyLoss()
```

```
EPOCHS = 10

model.train()
for _ in range(EPOCHS):
    for idx, (input_ids, start_positions, end_positions) in enumerate(train_loader):
        optimizer.zero_grad()
        start_logits, end_logits = model(input_ids)
        start_loss = criterion(start_logits, start_positions)
        end_loss = criterion(end_logits, end_positions)
        total_loss = (start_loss + end_loss) / 2
        total_loss.backward()
        optimizer.step()
        print(total_loss.item())
```

```
3.152693748474121
3.077603340148926
2.7068653106689453
2.728649139404297
1.8173941373825073
2.026475429534912
1.5392624139785767
1.4260454177856445
1.4798178672790527
1.1923370361328125
0.6881036758422852
1.7173633575439453
0.7657170295715332
0.29344385862350464
0.9570809602737427
0.44978058338165283
0.46998921036720276
0.2717154324054718
0.21450546383857727
0.09800814092159271
0.1609918177127838
0.1384696513414383
0.13001765310764313
0.07135705649852753
0.022294631227850914
0.04425870627164841
0.10749465227127075
0.06528482586145401
0.030680760741233826
0.09406206011772156
```

Extractive Approach

❖ Transformer-based Step 7: Test

```python
model.eval()
with torch.no_grad():
    sample = qa_dataset[1]
    context, question, answer = sample.values()
    input_ids, start_positions, end_positions = vectorize(question, context, answer)
    input_ids = input_ids.unsqueeze(0)
    start_logits, end_logits = model(input_ids)

    offset = len(tokenizer(question)) + 2
    start_position = torch.argmax(start_logits, dim=1).numpy()[0]
    end_position = torch.argmax(end_logits, dim=1).numpy()[0]

    start_position -= offset
    end_position -= offset

    start_position = max(start_position, 0)
    end_position = min(end_position, len(tokenizer(context)) - 1)

    if end_position >= start_position:
        # Extract the predicted answer span
        context_tokens = tokenizer(context)
        predicted_answer_tokens = context_tokens[start_position:end_position + 1]
        predicted_answer = ' '.join(predicted_answer_tokens)
    else:
        predicted_answer = ''
```

```
Context: I love painting and my favorite artist is Vincent Van Gogh.
Question: What is my favorite activity?
Start position: 2
End position: 2
Answer span: painting
```

Extractive Approach

❖ Training with a proper dataset

Dataset Viewer — Auto-converted to Parquet | API | View in Dataset Viewer

Split (2)
train · 87.6k rows

Search this dataset

id string · *lengths* 24 – 24	title string · *lengths* 3 – 59	context string · *lengths* 151 – 3.71k	question string · *lengths* 1 – 25.7k	answers sequence
5733be284776f41900661182	University_of_Notre_Dame	Architecturally, the school has a…	To whom did the Virgin Mary…	{ "text": ["Saint…
5733be284776f4190066117f	University_of_Notre_Dame	Architecturally, the school has a…	What is in front of the Notre Dame Mai…	{ "text": ["a copper statue o…
5733be284776f41900661180	University_of_Notre_Dame	Architecturally, the school has a…	The Basilica of the Sacred heart at…	{ "text": ["the Main Building"…
5733be284776f41900661181	University_of_Notre_Dame	Architecturally, the school has a…	What is the Grotto at Notre Dame?	{ "text": ["a Marian place of…
5733be284776f4190066117e	University_of_Notre_Dame	Architecturally, the school has a…	What sits on top of the Main Building…	{ "text": ["a golden statue o…
5733bf84d058e614000b61be	University_of_Notre_Dame	As at most other universities, Notr…	When did the Scholastic Magazin…	{ "text": ["September 1876…

‹ Previous 1 2 3 ... 876 Next ›

Training with SQuAD dataset

Extractive Approach

❖ Step 1: Import and download dataset

```
1 import torch
2 import torch.nn as nn
3 from torchtext.data.utils import get_tokenizer
4 from torchtext.vocab import build_vocab_from_iterator
5 from torch.nn.utils.rnn import pad_sequence
6 from torch.utils.data import Dataset, DataLoader
7 from datasets import load_dataset
8
9 qa_dataset = load_dataset('squad', split='train').shard(num_shards=40, index=0)
10 qa_dataset
```

```
Dataset({
    features: ['id', 'title', 'context', 'question', 'answers'],
    num_rows: 2190
})
```

Extractive Approach

❖ Step 2: Build vocabulary

```python
import re

def text_normalize(text):
    text = re.sub(r'[^\w\s]', ' ', text)

    return text

# Define tokenizer function
tokenizer = get_tokenizer('basic_english')

# Create a function to yield list of tokens
def yield_tokens(data):
    for item in data:
        yield tokenizer(
            text_normalize(item['context']) + ' <sep> ' + text_normalize(item['question'])
        )


# Create vocabulary
vocab = build_vocab_from_iterator(
    yield_tokens(qa_dataset),
    specials=['<unk>', '<pad>', '<bos>', '<eos>', '<sep>']
)
vocab.set_default_index(vocab['<unk>'])
vocab.get_stoi()
```

```
'wanton': 26314,
'waned': 26312,
'waltzes': 26310,
'wallachia': 26306,
'walked': 26305,
'walid': 26304,
'walachia': 26303,
'waiters': 26301,
'wahab': 26296,
'wagram': 26295,
'wagon': 26294,
'wag': 26290,
'wafer': 26289,
'wadis': 26288,
'wacht': 26286,
'wac': 26285,
'vən': 26283,
'vélodrome': 26281,
'vying': 26278,
'vy': 26277,
'vuitton': 26273,
'vreemdeling': 26271,
'voyagers': 26269,
'vorticity': 26265,
'vorpommern': 26263,
'vonore': 26260,
'vom': 26258,
```

Extractive Approach

❖ Step 3: Create vectorize function

```python
MAX_SEQ_LEN = 512
PAD_IDX = vocab['<pad>']

def pad_and_truncate(input_ids, max_seq_len):
    if len(input_ids) > max_seq_len:
        input_ids = input_ids[:max_seq_len]
    elif len(input_ids) < max_seq_len:
        input_ids += [PAD_IDX] * (max_seq_len - len(input_ids))

    return input_ids
```

```python
def vectorize(question, context, answer):
    input_text = text_normalize(question) + ' <sep> ' + text_normalize(context)
    input_ids = [vocab[token] for token in tokenizer(input_text)]
    input_ids = pad_and_truncate(input_ids, MAX_SEQ_LEN)

    answer_ids = [vocab[token] for token in tokenizer(text_normalize(answer))]
    try:
        start_positions = input_ids.index(answer_ids[0])
        end_positions = start_positions + len(answer_ids) - 1
    except:
        start_positions = 0
        end_positions = 0

    input_ids = torch.tensor(input_ids, dtype=torch.long)
    start_positions = torch.tensor(start_positions, dtype=torch.long)
    end_positions = torch.tensor(end_positions, dtype=torch.long)

    return input_ids, start_positions, end_positions
```

Extractive Approach

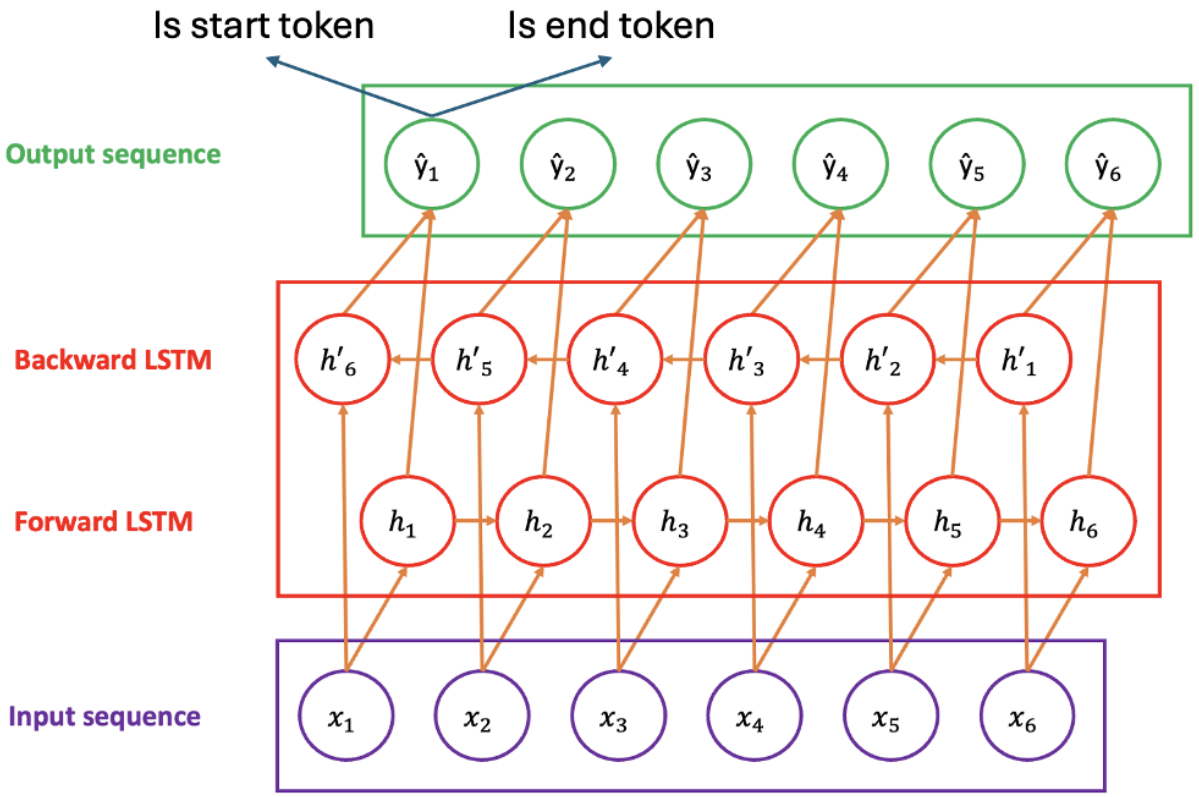
❖ Step 4: Create datasets

context string · *lengths* 151 3.71k	question string · *lengths* 1 25.7k	answers sequence
Architecturally, the school has a…	To whom did the Virgin Mary…	{ "text": ["Saint…
Architecturally, the school has a…	What is in front of the Notre Dame Mai…	{ "text": ["a copper statue o…
Architecturally, the school has a…	The Basilica of the Sacred heart at…	{ "text": ["the Main Building"…
Architecturally, the school has a…	What is the Grotto at Notre Dame?	{ "text": ["a Marian place of…
Architecturally, the school has a…	What sits on top of the Main Building…	{ "text": ["a golden statue o…
As at most other universities, Notr…	When did the Scholastic Magazin…	{ "text": ["September 1876…

```python
class QADataset(Dataset):
    def __init__(self, data):
        self.data = data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        item = self.data[idx]
        question_text = item['question']
        context_text = item['context']
        answer_text = item['answer']

        input_ids, start_positions, end_positions = vectorize(
            question_text, context_text, answer_text
        )

        return input_ids, start_positions, end_positions
```

Extractive Approach

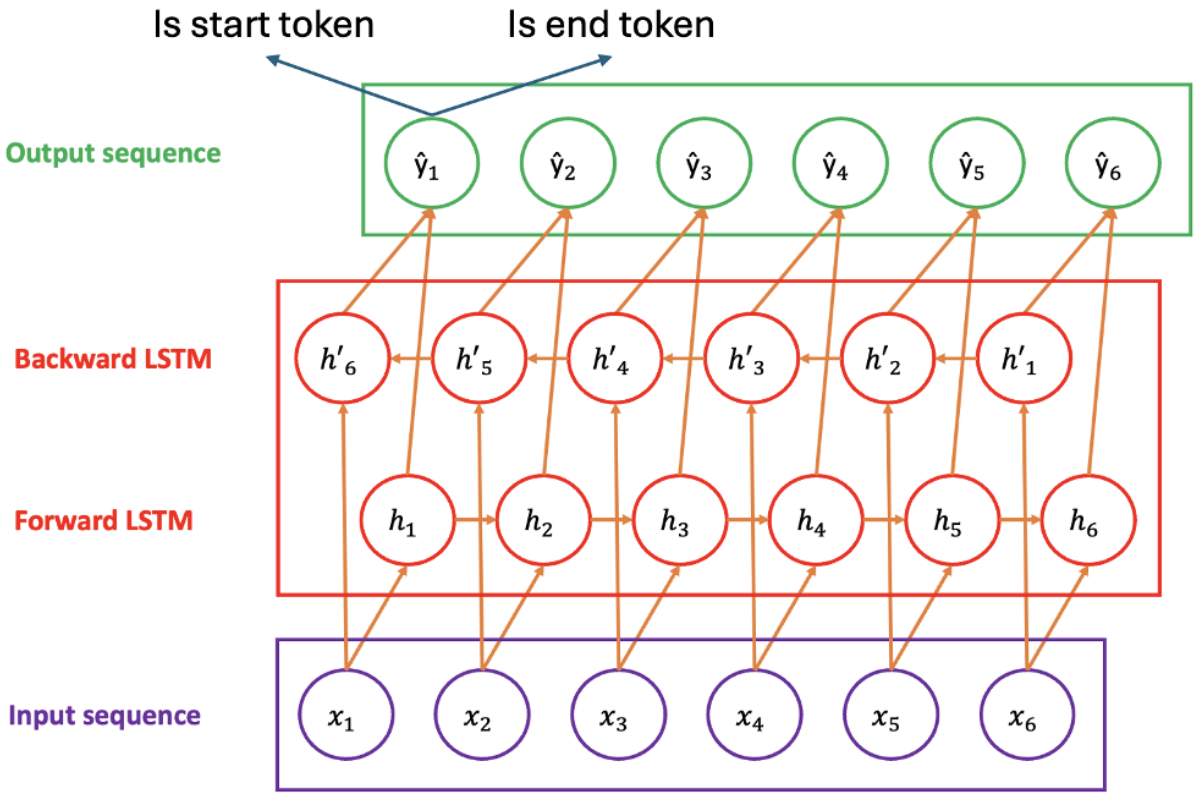
❖ Step 5: Create models




```python
class QAModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_size, n_layers):
        super(QAModel, self).__init__()
        self.input_embedding = nn.Embedding(vocab_size, embedding_dim)

        self.lstm = nn.LSTM(
            embedding_dim, hidden_size,
            num_layers=n_layers,
            batch_first=True,
            bidirectional=True
        )

        self.start_linear = nn.Linear(hidden_size * 2, 1)
        self.end_linear = nn.Linear(hidden_size * 2, 1)

    def forward(self, text):
        input_embedded = self.input_embedding(text)
        lstm_out, _ = self.lstm(input_embedded)

        start_logits = self.start_linear(lstm_out).squeeze(-1)
        end_logits = self.end_linear(lstm_out).squeeze(-1)

        return start_logits, end_logits
```

Extractive Approach

❖ Step 5: Create models




```python
# Model parameters
EMBEDDING_DIM = 64
HIDDEN_SIZE = 128
VOCAB_SIZE = len(vocab)
N_LAYERS = 3
device = 'cuda' if torch.cuda.is_available() else 'cpu'

model = QAModel(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_SIZE, N_LAYERS).to(device)

input = torch.randint(0, 10, size=(1, 10)).to(device)
print(input.shape)
model.eval()
with torch.no_grad():
    start_logits, end_logits = model(input)

print(start_logits.shape)
```

```
torch.Size([1, 10])
torch.Size([1, 10])
```

Extractive Approach

❖ Step 6: Training

```python
from tqdm import tqdm
EPOCHS = 10

model.train()
for epoch in tqdm(range(EPOCHS)):
    train_losses = []
    for idx, (input_ids, start_positions, end_positions) in enumerate(train_loader):
        input_ids = input_ids.to(device)
        start_positions = start_positions.to(device)
        end_positions = end_positions.to(device)
        optimizer.zero_grad()
        start_logits, end_logits = model(input_ids)
        start_loss = criterion(start_logits, start_positions)
        end_loss = criterion(end_logits, end_positions)
        total_loss = (start_loss + end_loss) / 2
        total_loss.backward()
        optimizer.step()
        train_losses.append(total_loss.item())
    train_loss = sum(train_losses) / len(train_losses)
    print(f'EPOCH {epoch + 1}\tTraining Loss: {train_loss}')
```

```
 10%|▉         | 1/10 [00:03<00:34,  3.88s/it]EPOCH 1   Training Loss: 4.849793354670207
 20%|██        | 2/10 [00:08<00:32,  4.07s/it]EPOCH 2   Training Loss: 4.337740341822307
 30%|███       | 3/10 [00:11<00:27,  3.95s/it]EPOCH 3   Training Loss: 3.609094884660509
 40%|████      | 4/10 [00:15<00:23,  3.91s/it]EPOCH 4   Training Loss: 2.471651620335049
 50%|█████     | 5/10 [00:19<00:19,  3.98s/it]EPOCH 5   Training Loss: 1.360072414080302
 60%|██████    | 6/10 [00:23<00:15,  3.96s/it]EPOCH 6   Training Loss: 0.8433425062232547
 70%|███████   | 7/10 [00:28<00:12,  4.23s/it]EPOCH 7   Training Loss: 0.5627094805240631
 80%|████████  | 8/10 [00:33<00:08,  4.36s/it]EPOCH 8   Training Loss: 0.4058462844954597
 90%|█████████ | 9/10 [00:37<00:04,  4.20s/it]EPOCH 9   Training Loss: 0.27234171041184
100%|██████████| 10/10 [00:40<00:00,  4.09s/it]EPOCH 10 Training Loss: 0.17832869581050342
```

Extractive Approach

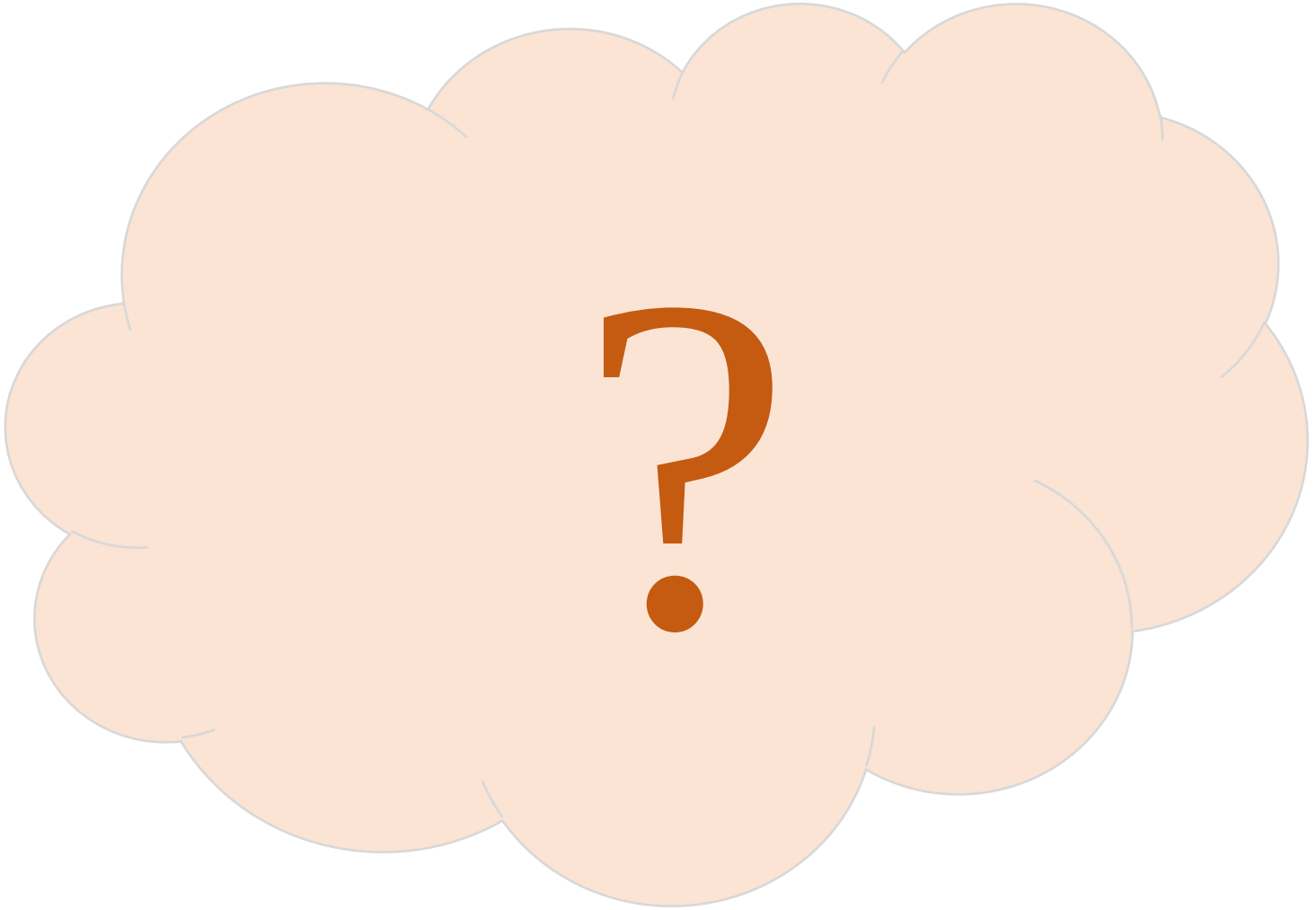
❖ Step 7: Test

```python
model.eval()
with torch.no_grad():
    context = 'Jane is a student and she is from Canada'
    question = 'Where is Jane from?'
    answer = 'Canada'
    context = text_normalize(context)
    question = text_normalize(question)
    answer = text_normalize(answer)
    input_ids, start_positions, end_positions = vectorize(question, context, answer)
    input_ids = input_ids.to(device)
    start_positions = start_positions.to(device)
    end_positions = end_positions.to(device)
    input_ids = input_ids.unsqueeze(0)
    start_logits, end_logits = model(input_ids)

    offset = len(tokenizer(question)) + 1
    start_position = torch.argmax(start_logits, dim=1).cpu().numpy()[0]
    end_position = torch.argmax(end_logits, dim=1).cpu().numpy()[0]

    start_position -= offset
    end_position -= offset

    start_position = max(start_position, 0)
    end_position = min(end_position, len(tokenizer(context)) - 1)

    if end_position >= start_position:
        # Extract the predicted answer span
        context_tokens = tokenizer(context)
        predicted_answer_tokens = context_tokens[start_position:end_position + 1]
        predicted_answer = ' '.join(predicted_answer_tokens)
    else:
        predicted_answer = ''
```

```
Context: Jane is a student and she is from Canada
Question: Where is Jane from
Start position: 8
End position: 8
Answer span: canada
Canada
```

```
Context: The city and surrounding area suffered the bulk of the economic damage
Question: On what date did the World Trade Center PATH begin operation
Start position: 126
End position: 128
Answer span: july 19 1909
Label: July 19  1909
```

Question




Thank you!