

# EnKo Translation

자연어 처리를 이용한 번역기

지도교수: 양희경 교수님

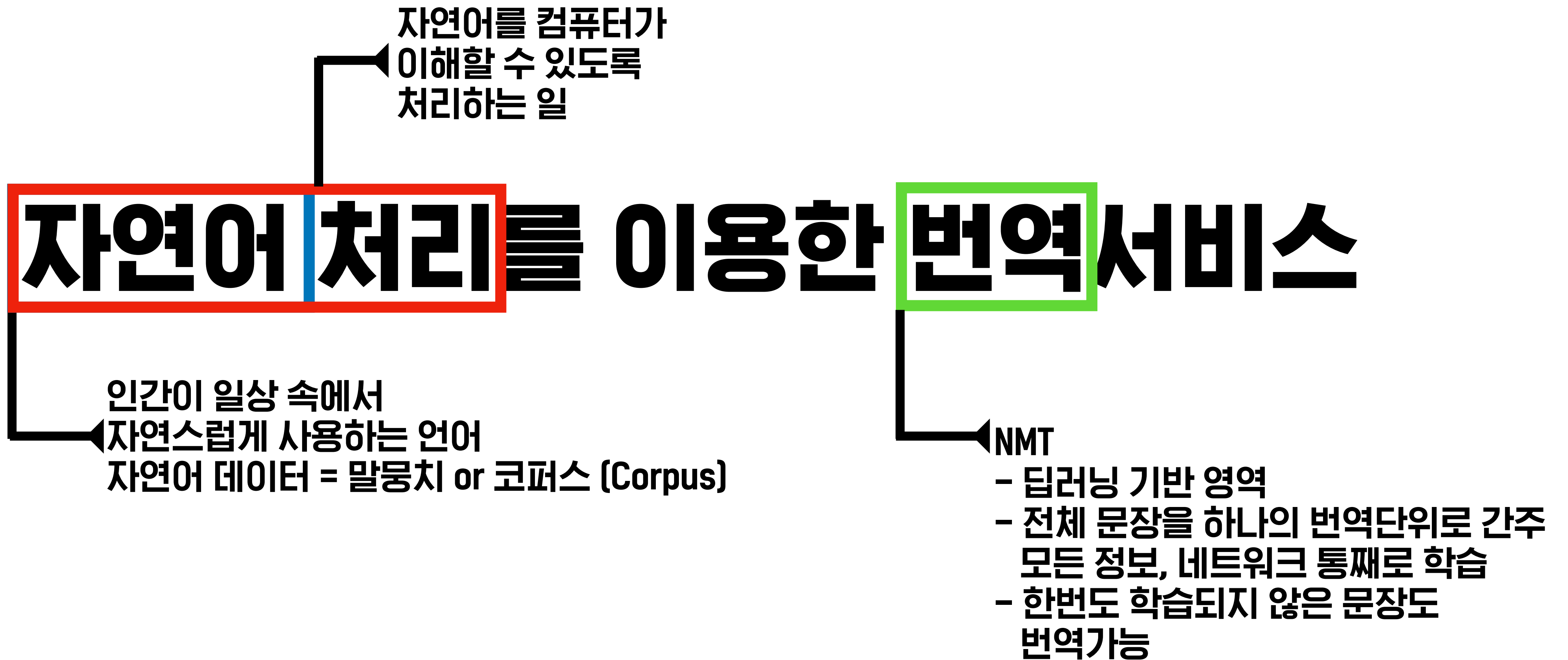
팀장: 서현수

팀원: 김부용, 김현중, 최다경

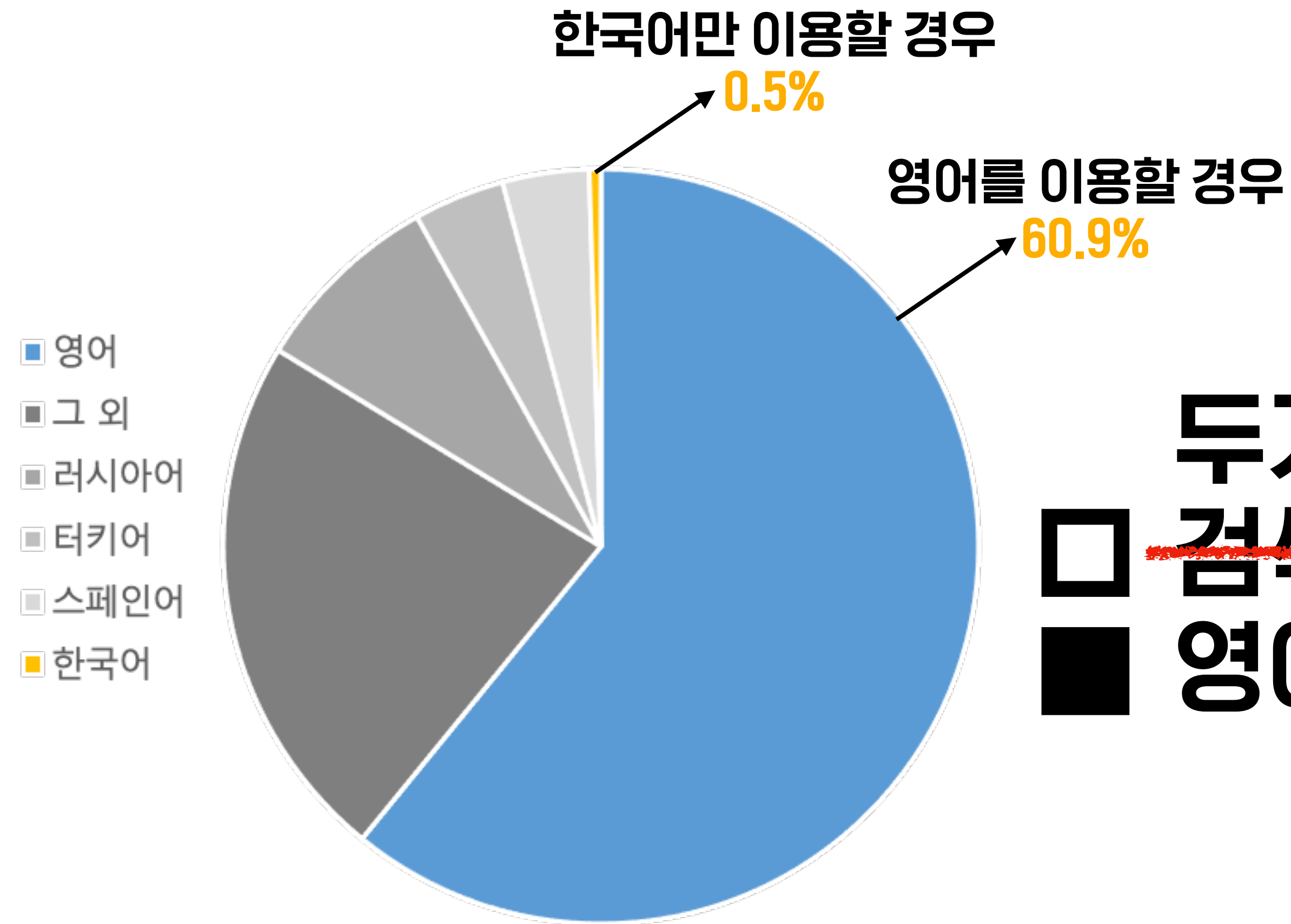
# 목차 및 발표순서

- 주제 - 자연어 처리를 이용한 번역기
- 필요성
- 기술조사 - RNN, Attention, Transformer 기술과 오픈소스
- 특정회사의 관심기술 - 네이버의 파파고 API
- 기술시연
- 사업화전략
- 향후 발전계획
- 역할 및 일정

# 주제



# 필요성



두가지 선택지

□ ~~검색을 위해 영어를 공부~~

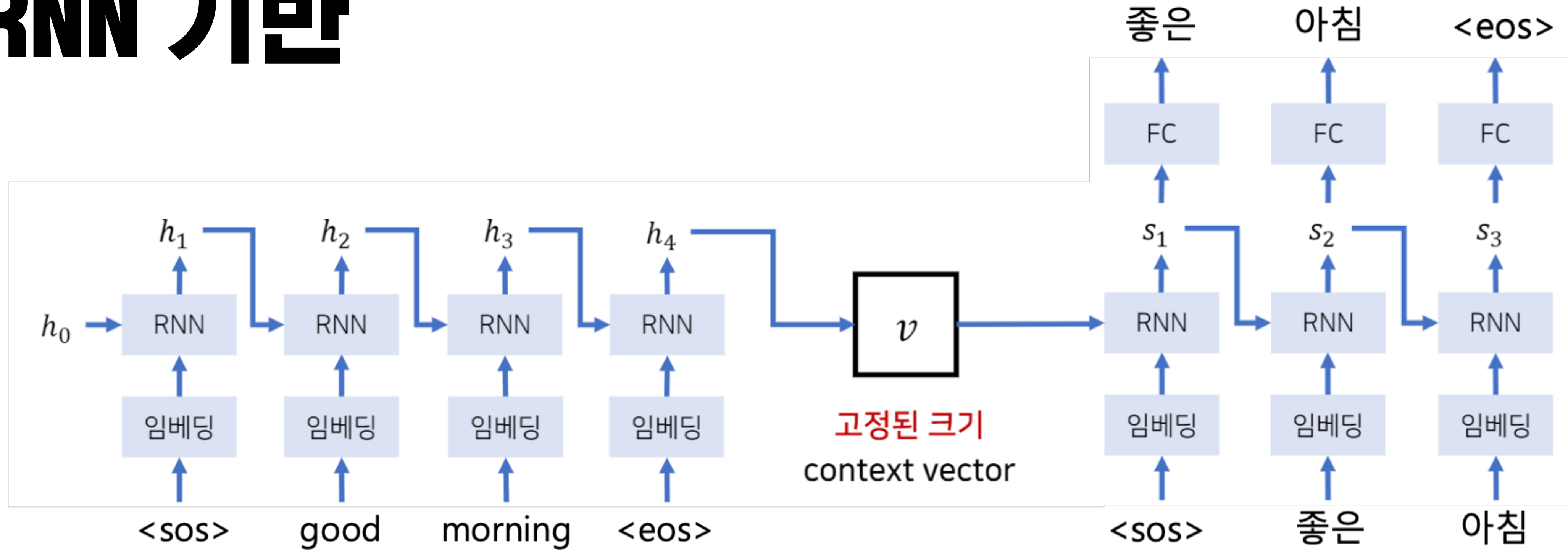
■ 영어 공부는 컴퓨터에게 맡기기

언어별 웹사이트 정보량  
출처: W3Techs.com, 2021 년 4 월 8 일 기준

# 기술조사

– RNN, ATTENTION, TRANSFORMER

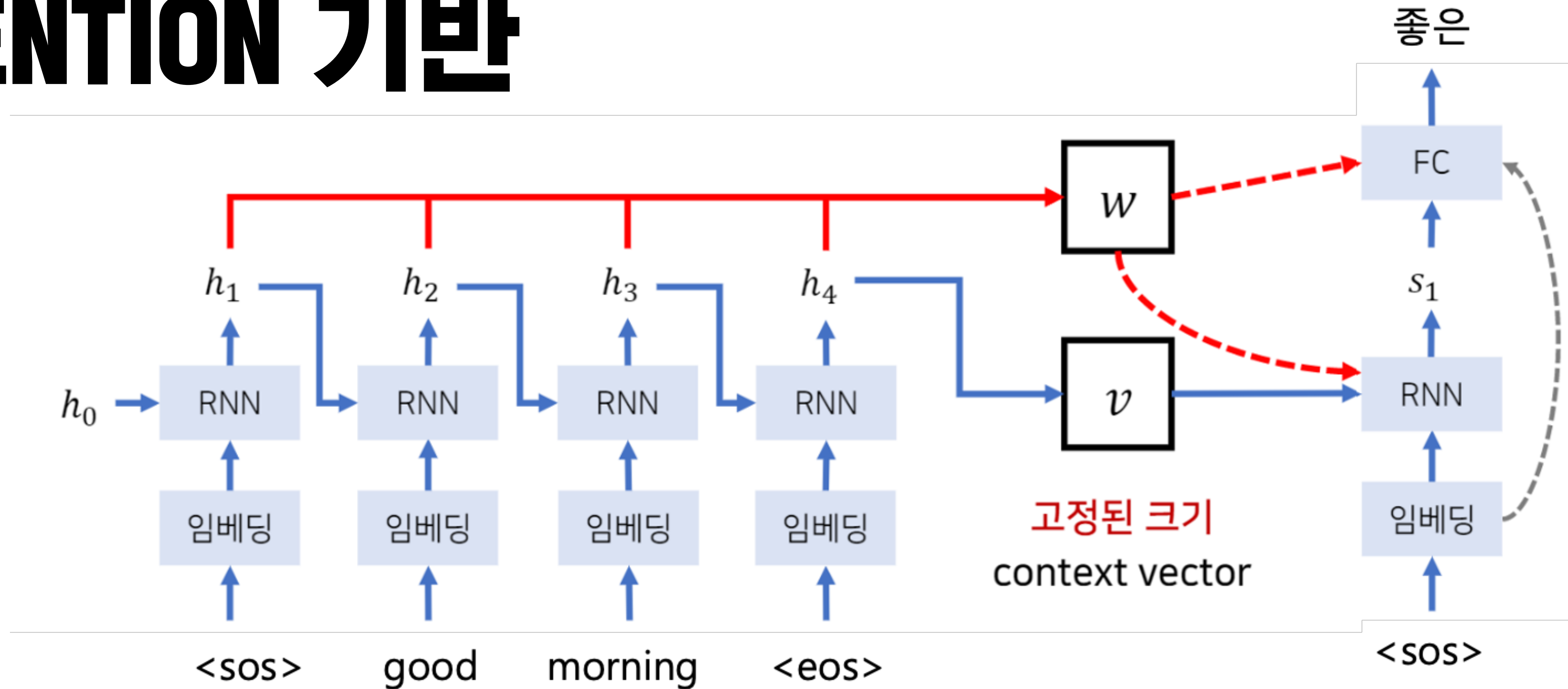
# RNN 기반



**Encoder:** 입력 문장을 RNN을 이용하여 context vector로 변환시켜 정보를 압축  
**Decoder:** context vector로부터 번역 결과를 추론

**한계점:** context vector가 고정된 크기를 가지기 때문에  
많은 정보 -> 정보손실, 성능 저하

# ATTENTION 기반

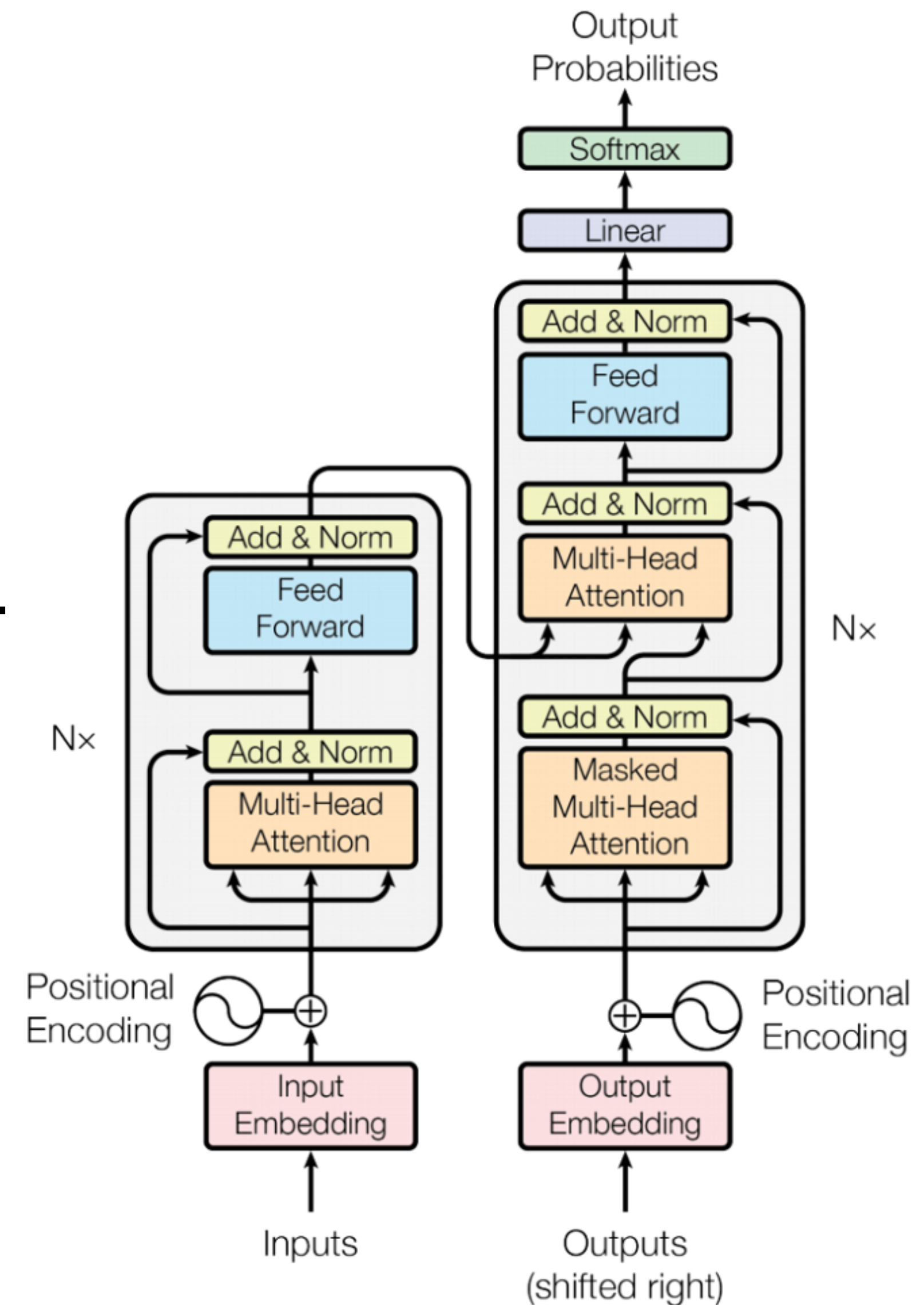


Decoder에서 출력 단어를 예측하는 때 시점(time stamp)마다  
Encoder의 입력 문장의 hidden states를 참고

단, 전체 입력 문장을 전부 동일한 비율로 참고하는 것이 아닌  
해당 시점에서 예측해야 할 단어와 연관이 있는  
입력 단어 부분에 가중치를 높여 참고

# TRANSFORMER 기반

기존의 seq2seq의 구조인 Encoder - Decoder 구조를 따르지만  
RNN, CNN을 사용하지 않고 Attention만으로 구현한 모델





# 오픈소스

- Self Attention, Multi Head Attention  
Encoder, Decoder, Positional Encoding  
Transformer

# Self Attention

```
class SelfAttention(nn.Module):  
    def __init__(self):  
        super(SelfAttention, self).__init__()  
        self.matmul = torch.matmul  
        self.softmax = torch.softmax
```

```
    def forward(self, query, key, value, mask=None):
```

```
        key_transpose = torch.transpose(key, -2, -1)
```

```
        matmul_result = self.matmul(query, key_transpose)
```

```
        d_k = key.size()[-1]
```

```
        attention_score = matmul_result / math.sqrt(d_k)
```

```
        if mask is not None:
```

```
            attention_score = attention_score.masked_fill(mask == 0, -1e20)
```

```
        softmax_attention_score = self.softmax(attention_score, dim=-1)
```

```
        result = self.matmul(softmax_attention_score, value)
```

```
    return result, softmax_attention_score
```

# (batch, head num, d k, token)

Query 벡터와 Key 벡터의 행렬곱

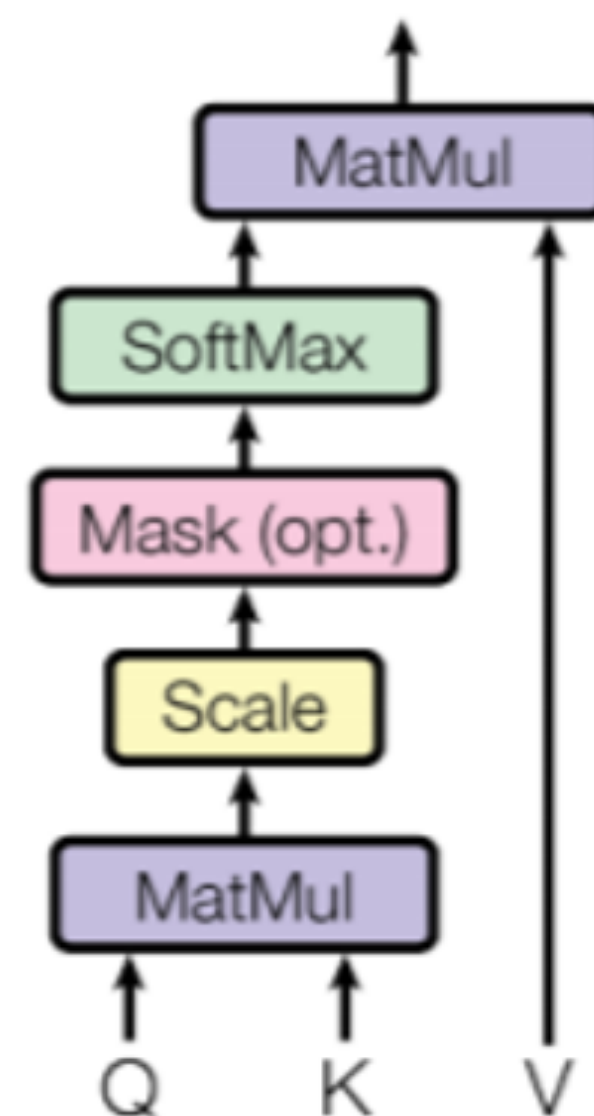
Scale: key의 제곱근으로 나눠줌

확률값과 value 벡터를 가중합

$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V$$

=

Z



Q: Query, 현재 시점의 token을 의미

K: Key, Attention을 구하고자 하는 대상 token을 의미

V: Value, Attention을 구하고자 하는 대상 token을 의미

# Multi Head Attention

```
class MultiHeadAttention(nn.Module):
    def __init__(self, head_num = 8 , d_model = 512, dropout = 0.1):
        super(MultiHeadAttention, self).__init__()

        # print(d_model % head_num)
        # assert d_model % head_num != 0 # d_model % head_num == 0 이 아닌 경우 에러메세지 발생

        self.head_num = head_num
        self.d_model = d_model
        self.d_k = self.d_v = d_model // head_num

        self.w_q = nn.Linear(d_model, d_model)
        self.w_k = nn.Linear(d_model, d_model)
        self.w_v = nn.Linear(d_model, d_model)
        self.w_o = nn.Linear(d_model, d_model)

        self.self_attention = SelfAttention()
        self.dropout = nn.Dropout(p=dropout)

    def forward(self, query, key, value, mask = None):
        if mask is not None:
            # Same mask applied to all h heads.
            mask = mask.unsqueeze(1)

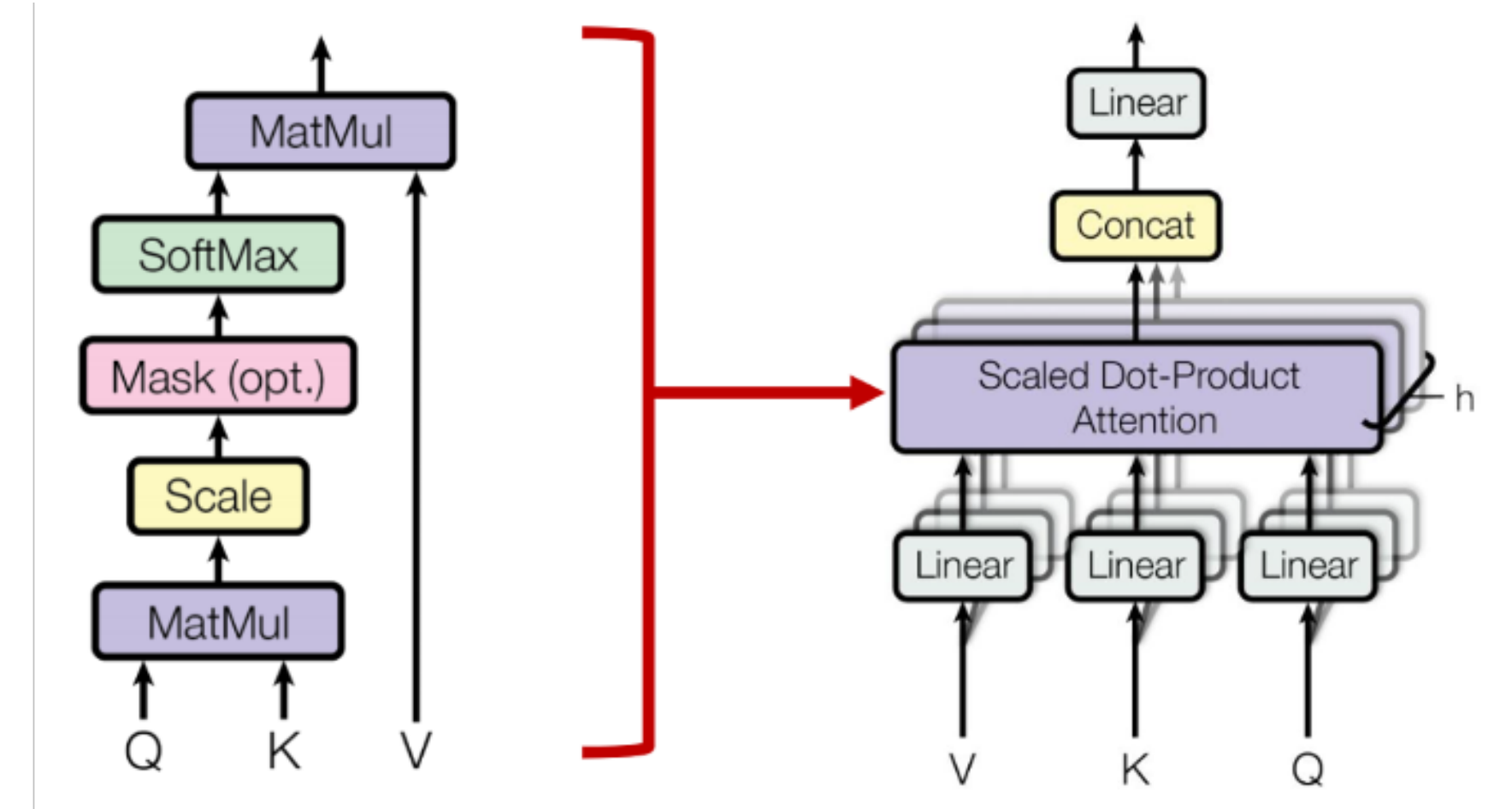
        batche_num = query.size(0)

        query = self.w_q(query).view(batche_num, -1, self.head_num, self.d_k).transpose(1, 2)
        key = self.w_k(key).view(batche_num, -1, self.head_num, self.d_k).transpose(1, 2)
        value = self.w_v(value).view(batche_num, -1, self.head_num, self.d_k).transpose(1, 2)

        attention_result, attention_score = self.self_attention(query, key, value, mask)

        attention_result = attention_result.transpose(1,2).contiguous().view(batche_num, -1, self.head_num * self.d_k)

        return self.w_o(attention_result)
```



Scaled Dot-Product Attention

Multi-Head Attention

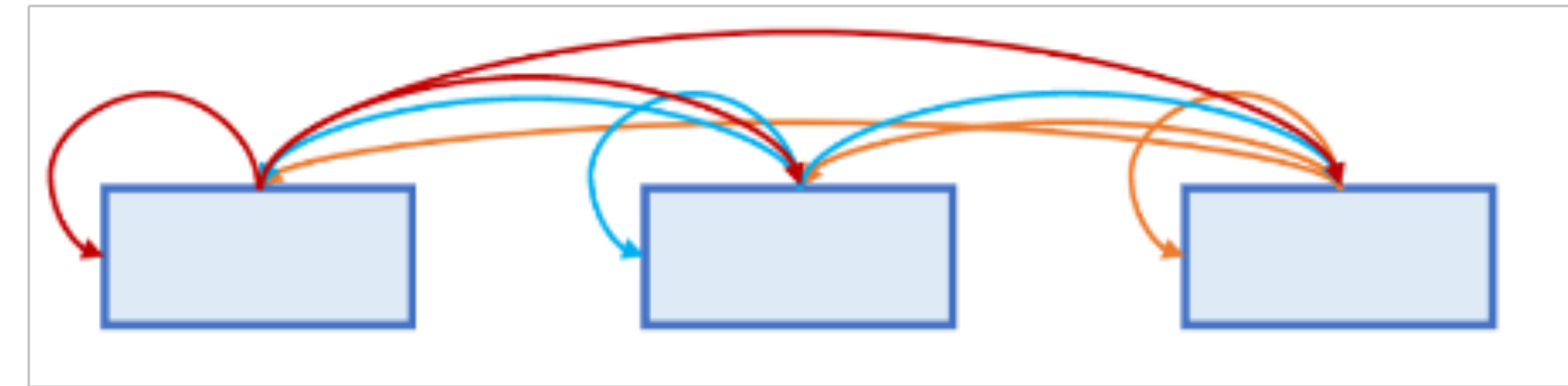
**Self-Attention을 8번 진행하여  
각각 다르게 표현된 8개의 행렬을  
하나의 행렬로 합침**

# Encoder, Decoder

```
class Encoder(nn.Module):
    def __init__(self, d_model, head_num, dropout):
        super(Encoder, self).__init__()
        self.multi_head_attention = MultiHeadAttention(d_model= d_model, head_num= head_num)
        self.residual_1 = ResidualConnection(d_model, dropout=dropout)

        self.feed_forward = FeedForward(d_model)
        self.residual_2 = ResidualConnection(d_model, dropout=dropout)

    def forward(self, input, mask):
        x = self.residual_1(input, lambda x: self.multi_head_attention(x, x, x, mask))
        x = self.residual_2(x, lambda x: self.feed_forward(x))
        return x
```



**Encoder Self Attention**

```
class Decoder(nn.Module):
    def __init__(self, d_model, head_num, dropout):
        super(Decoder, self).__init__()
        self.masked_multi_head_attention = MultiHeadAttention(d_model= d_model, head_num= head_num)
        self.residual_1 = ResidualConnection(d_model, dropout=dropout)

        self.encoder_decoder_attention = MultiHeadAttention(d_model= d_model, head_num= head_num)
        self.residual_2 = ResidualConnection(d_model, dropout=dropout)

        self.feed_forward = FeedForward(d_model)
        self.residual_3 = ResidualConnection(d_model, dropout=dropout)

    def forward(self, target, encoder_output, target_mask, encoder_mask):
        # target, x, target_mask, input_mask
        x = self.residual_1(target, lambda x: self.masked_multi_head_attention(x, x, x, target_mask))
        x = self.residual_2(x, lambda x: self.encoder_decoder_attention(x, encoder_output, encoder_output, encoder_mask))
        x = self.residual_3(x, self.feed_forward)

        return x
```



**Masked Multi-Head Attention**



**Encoder - Decoder Attention**



# Positional Encoding

```
class Embeddings(nn.Module):
    def __init__(self, vocab_num, d_model):
        super(Embeddings, self).__init__()
        self.emb = nn.Embedding(vocab_num, d_model)
        self.d_model = d_model
    def forward(self, x):
        return self.emb(x) * math.sqrt(self.d_model)

class PositionalEncoding(nn.Module):
    def __init__(self, max_seq_len, d_model, dropout=0.1):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        pe = torch.zeros(max_seq_len, d_model)

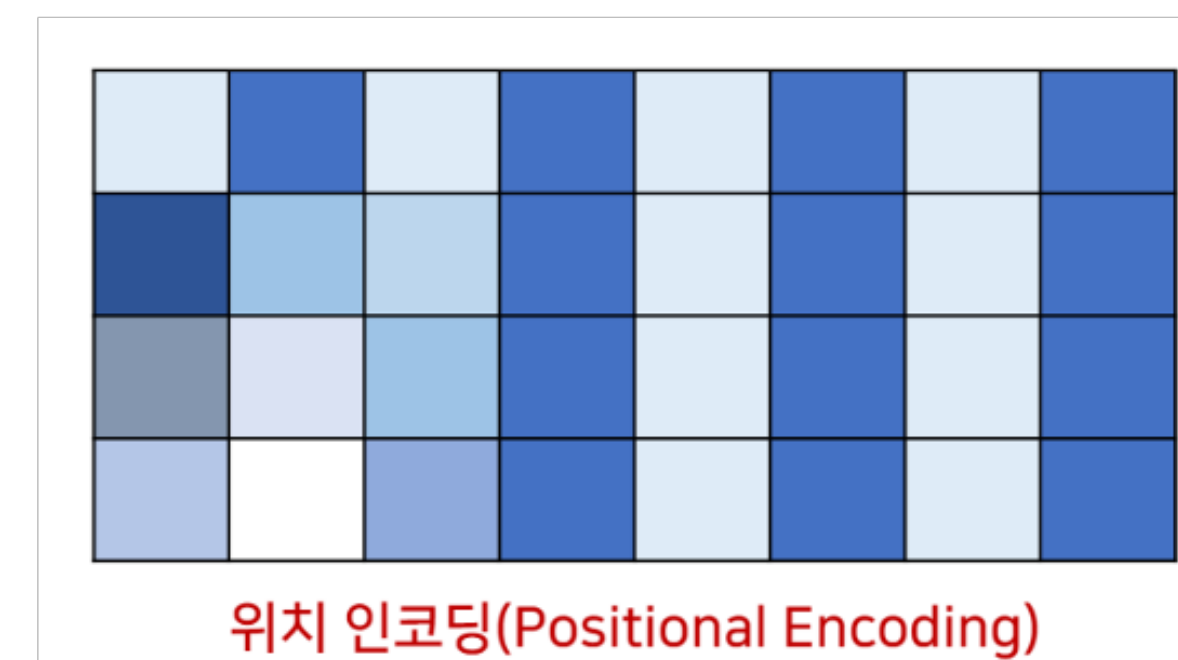
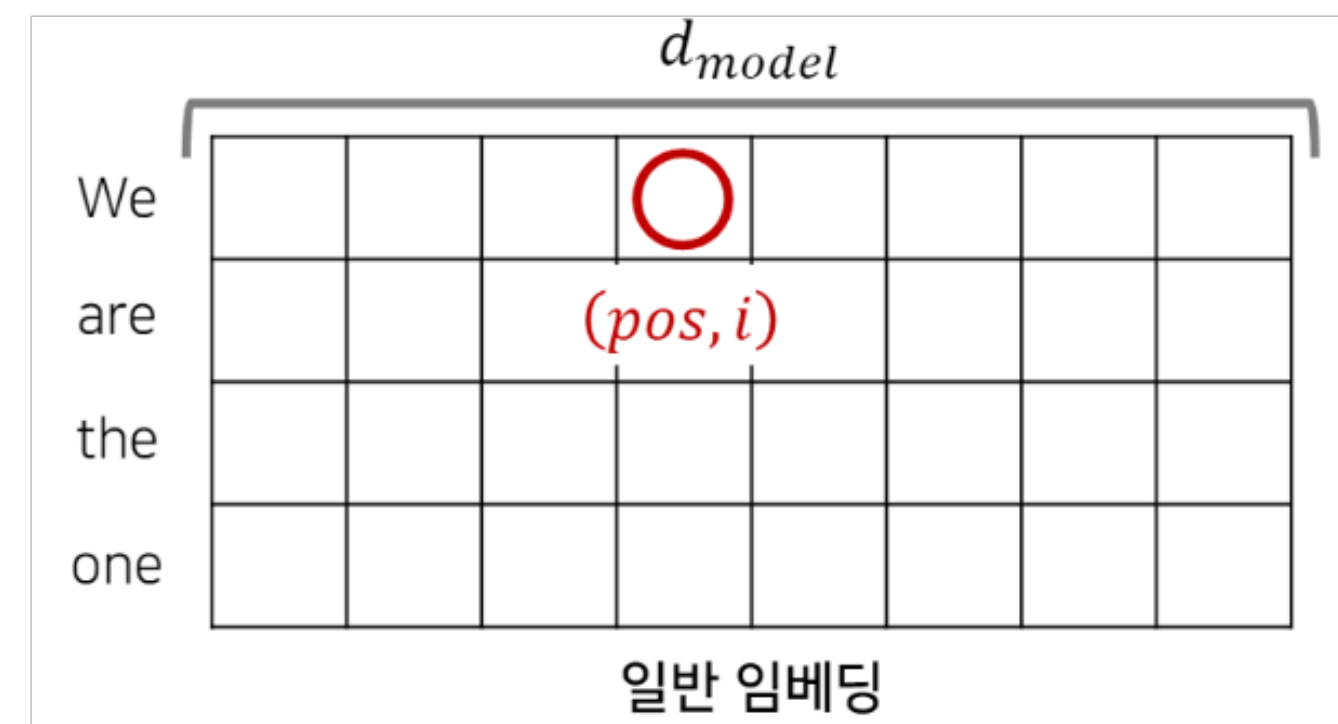
        position = torch.arange(0, max_seq_len).unsqueeze(1)
        base = torch.ones(d_model // 2).fill_(10000)
        pow_term = torch.arange(0, d_model, 2) / torch.tensor(d_model, dtype=torch.float32)
        div_term = torch.pow(base, pow_term)

        pe[:, 0::2] = torch.sin(position / div_term)
        pe[:, 1::2] = torch.cos(position / div_term)

        pe = pe.unsqueeze(0)

        # pe를 학습되지 않는 변수로 등록
        self.register_buffer('positional_encoding', pe)

    def forward(self, x):
        x = x + Variable(self.positional_encoding[:, :x.size(1)], requires_grad=False)
        return self.dropout(x)
```



각 단어의 위치와 시퀀스 내의 다른 단어 간의  
위치 차이에 대한 정보를 포함하고 있는  
positional encoding을 사용해 정규화

# Transformer

```
class Transformer(nn.Module):
    def __init__(self, vocab_num, d_model, max_seq_len, head_num, dropout, N):
        super(Transformer, self).__init__()
        self.embedding = Embeddings(vocab_num, d_model)
        self.positional_encoding = PositionalEncoding(max_seq_len, d_model)

        self.encoders = clones(Encoder(d_model=d_model, head_num=head_num, dropout=dropout), N)
        self.decoders = clones(Decoder(d_model=d_model, head_num=head_num, dropout=dropout), N)

        self.generator = Generator(d_model, vocab_num)

    def forward(self, input, target, input_mask, target_mask, labels=None):
        x = self.positional_encoding(self.embedding(input))
        for encoder in self.encoders:
            x = encoder(x, input_mask)

        target = self.positional_encoding(self.embedding(target))
        for decoder in self.decoders:
            # target, encoder_output, target_mask, encoder_mask
            target = decoder(target, x, target_mask, input_mask)

        lm_logits = self.generator(target)
        loss = None
        if labels is not None:
            # Shift so that tokens < n predict n
            shift_logits = lm_logits[..., :-1, :].contiguous()
            shift_labels = labels[..., 1:].contiguous()
            # Flatten the tokens
            loss_fct = CrossEntropyLoss(ignore_index=0)
            loss = loss_fct(shift_logits.view(-1), shift_labels.view(-1))

        return lm_logits, loss
```

```
def encode(self, input, input_mask):
    x = self.positional_encoding(self.embedding(input))
    for encoder in self.encoders:
        x = encoder(x, input_mask)
    return x

def decode(self, encode_output, encoder_mask, target, target_mask):
    target = self.positional_encoding(self.embedding(target))
    for decoder in self.decoders:
        # target, encoder_output, target_mask, encoder_mask
        target = decoder(target, encode_output, target_mask, encoder_mask)

    lm_logits = self.generator(target)

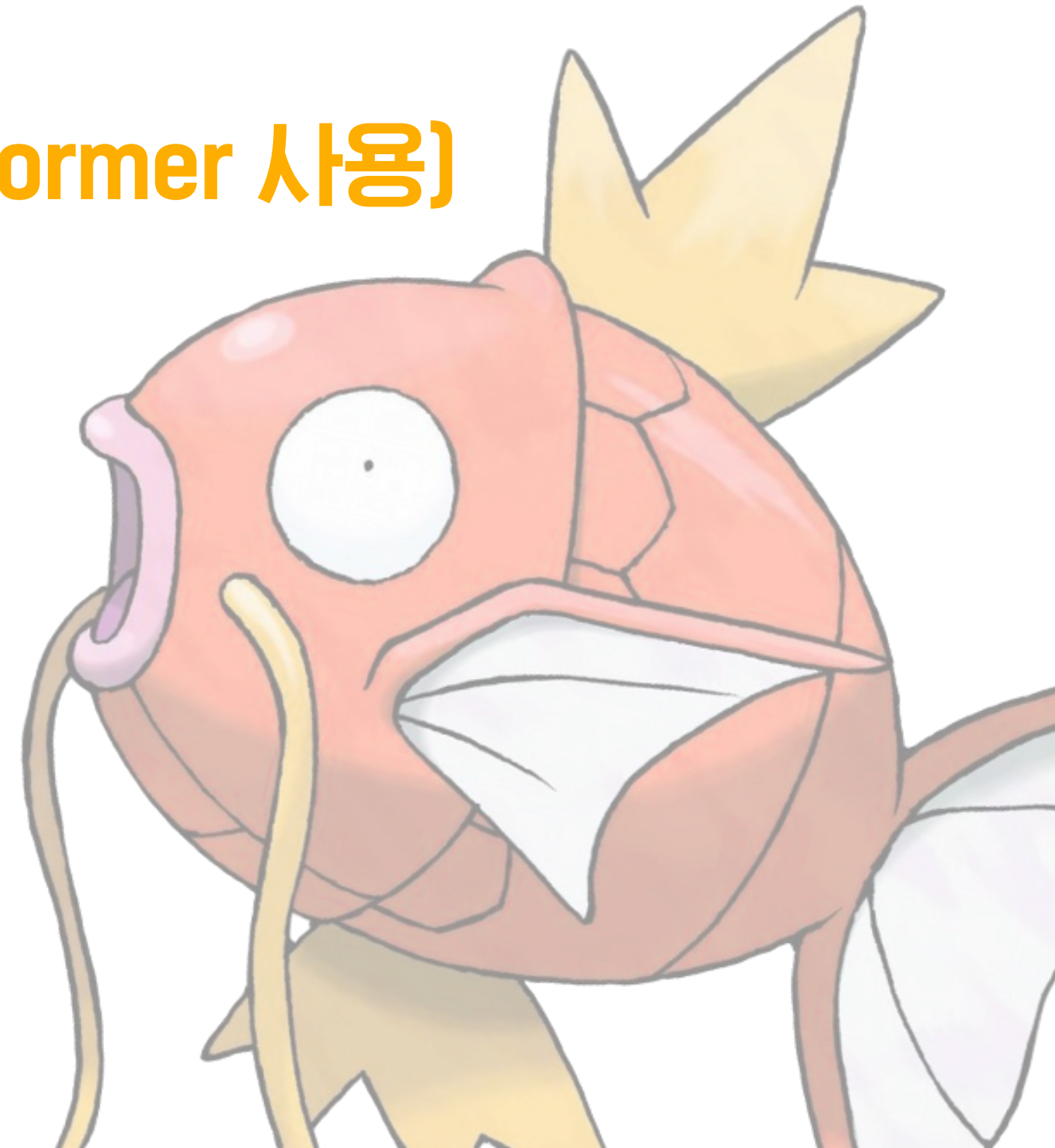
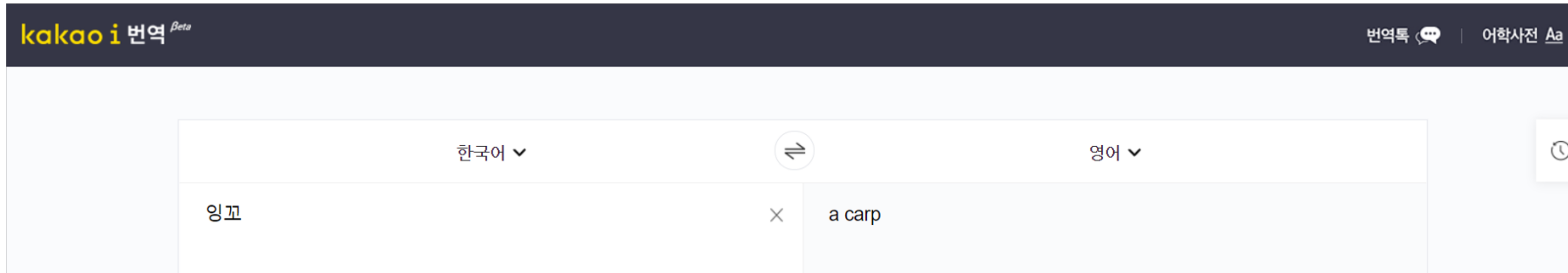
    return lm_logits
```

# 특정 회사의 관심기술 - 네이버의 파파고 API

**파파고:** 국내 대표 AI 통번역 서비스

## 선택 이유

1. 네이버는 개발자 컨퍼런스인 'DEVVIEW'를 정기적으로 개최해 현황을 발표, 접근 자료가 많음
2. 학습자를 타겟으로 개발되기 시작한 번역기
3. 네이버의 방대한 데이터셋, 오픈소스가 아닌 자체 개발 모델(Transformer 사용)
4. 국내 다른 번역 서비스와 비교해봤을 때 좋은 성능



기술시연



# En&Ko Translation

## 펭귄에 대한 검색 결과

[펭귄 - 나무위키](#)

7 days ago — 조강 펭귄목의 새. 한자로는 '인조(人鳥)'라고 한다. 키는 40~120cm이고, 몸은 뚱뚱한 방추형이고 날개는 지느러미 모양으로 짧고 작게 변화하여 ...

[아델리펭귄 - 나무위키](#)

10 May 2021

[펭귄 - 위키백과, 우리 모두의 백과사전](#)

구글에서 더 보기

## Penguin에 대한 검색 결과

[Penguin Random House India: Home](#)

Home of (almost) all your favourite books.

[Penguin - Wikipedia](#)

Penguins are a group of aquatic flightless birds. They live almost exclusively in the Southern Hemisphere, with only one species, the Galápagos penguin, found ...

[Google Penguin - Wikipedia](#)

구글에서 더 보기

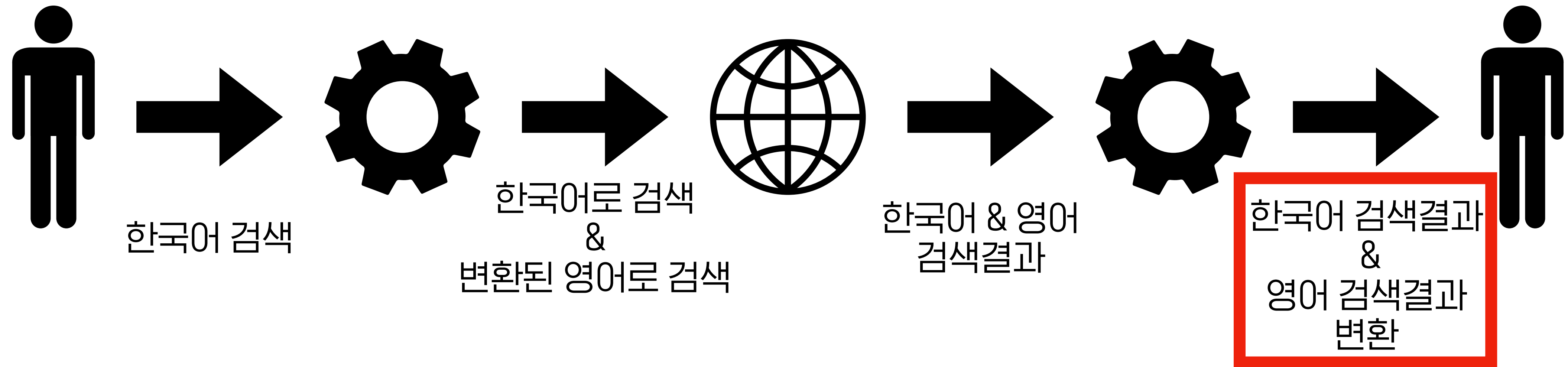


한글 검색어를 입력해주세요.

clear



# 사업화전략 - 배너광고



일반 웹사이트: 한국어 검색결과만 사용자에게 제공  
-> 해외광고주가 해당 웹사이트를 이용한 배너광고를 할 이유가 없음

**BUT !!!**

한국어 & 영어 검색결과(한국어로 변환된) 사용자에게 제공  
-> 해외광고주가 해당 웹사이트를 이용한 배너광고를 할 이유 충분  
-> N x 본래 수익

# 향후 발전 계획

영어가 아닌 **다른 언어로의 확장**

자체모델을 **커스텀**한다면 사용자가 원하는 주제에 맞는  
학습데이터를 사용한 **맞춤형 모델로** 제공해 **만족도 상승**

쓸모 없는 **검색결과를 분류**해 번역처리 되지 않도록 **필터링**

.....

# 역할 및 일정계획

역할	담당
Light Survey	공통
기술 조사 및 코드 분석	서현수, 최다경
PPT 제작, 발표	김현중
웹 사이트 구현	김부용

	1회차	2회차	3회차	4회차
프로젝트 계획 & 팀 역할 분배				
기업 조사 & 사업화 전략 구상				
데이터 수집 & 기술 조사				
기술 구현 & 프로토타입 제작				

감사합니다