Natural Language Processing II

Transformers, NLP tasks and Huggingface

Seongyun Lee

sy-lee@korea.ac.kr

Artificial Intelligence in KU (AIKU)

Department of Computer Science and Engineering, Korea University

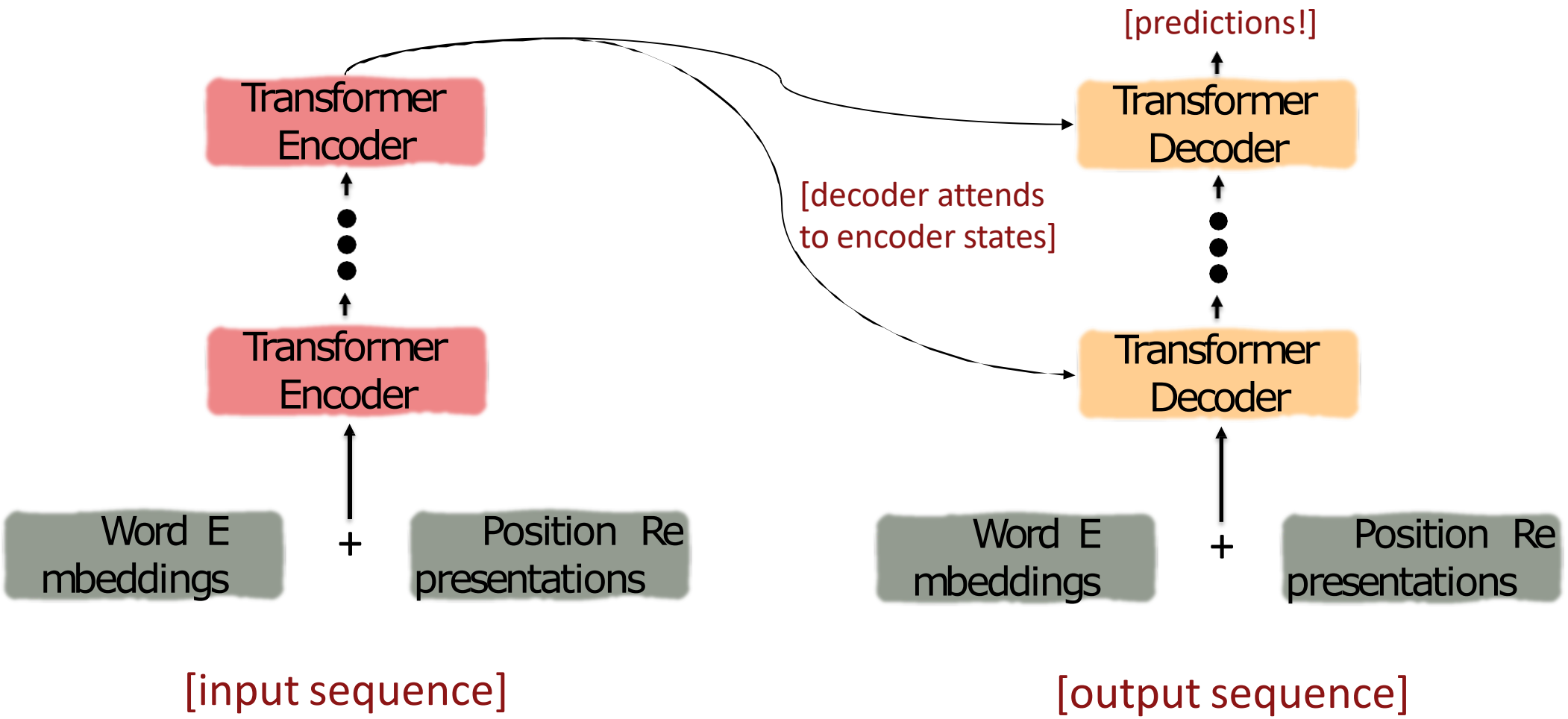
AIKU

Recap

- Week 4: Basic concepts of NLP from Word2Vec to Transformer
- Week 6: Diverse Transformer-family models, NLP tasks and huggingface 🤗

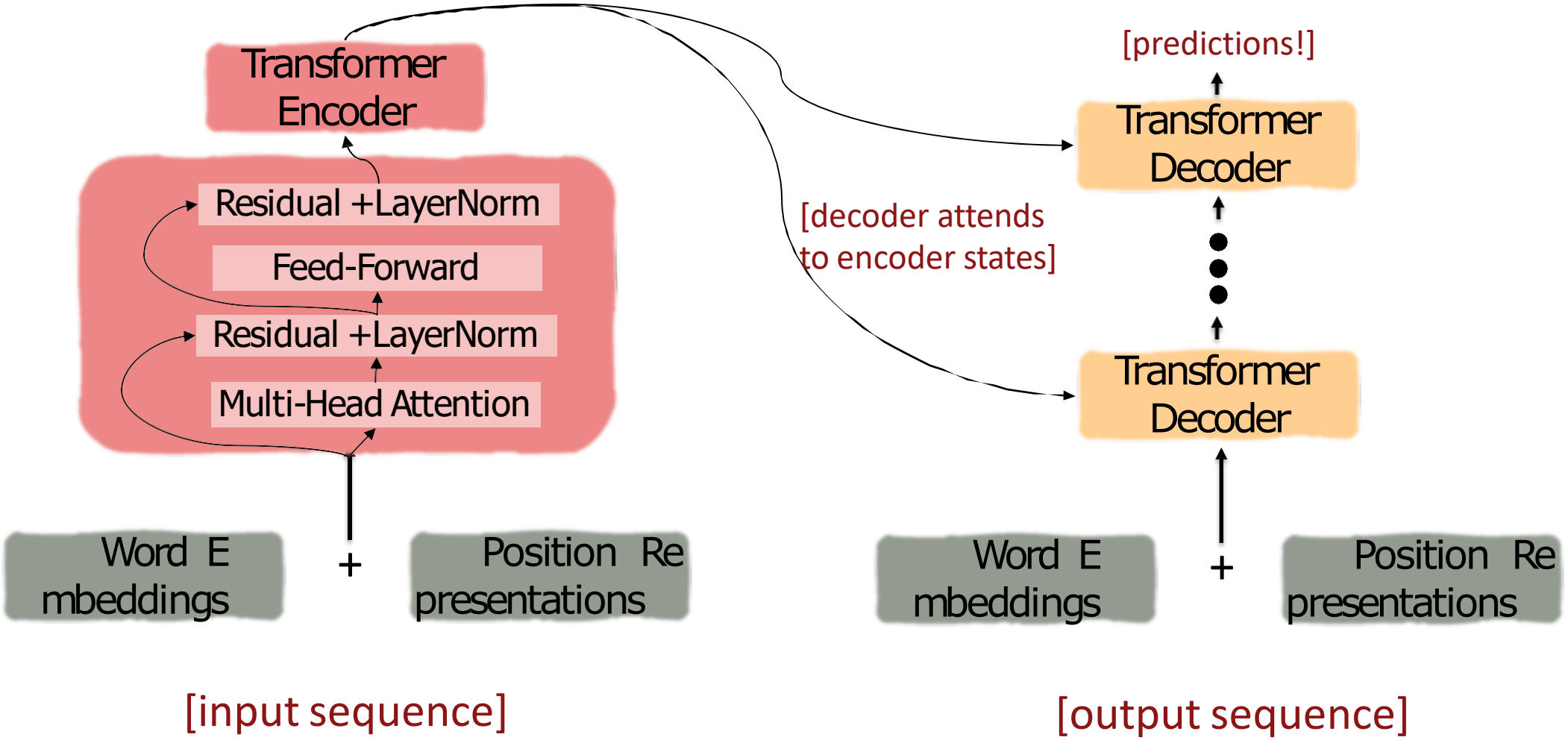
The Transformer Encoder-Decoder

Looking back at the whole model, zooming in on an Encoder block:



The Transformer Encoder-Decoder

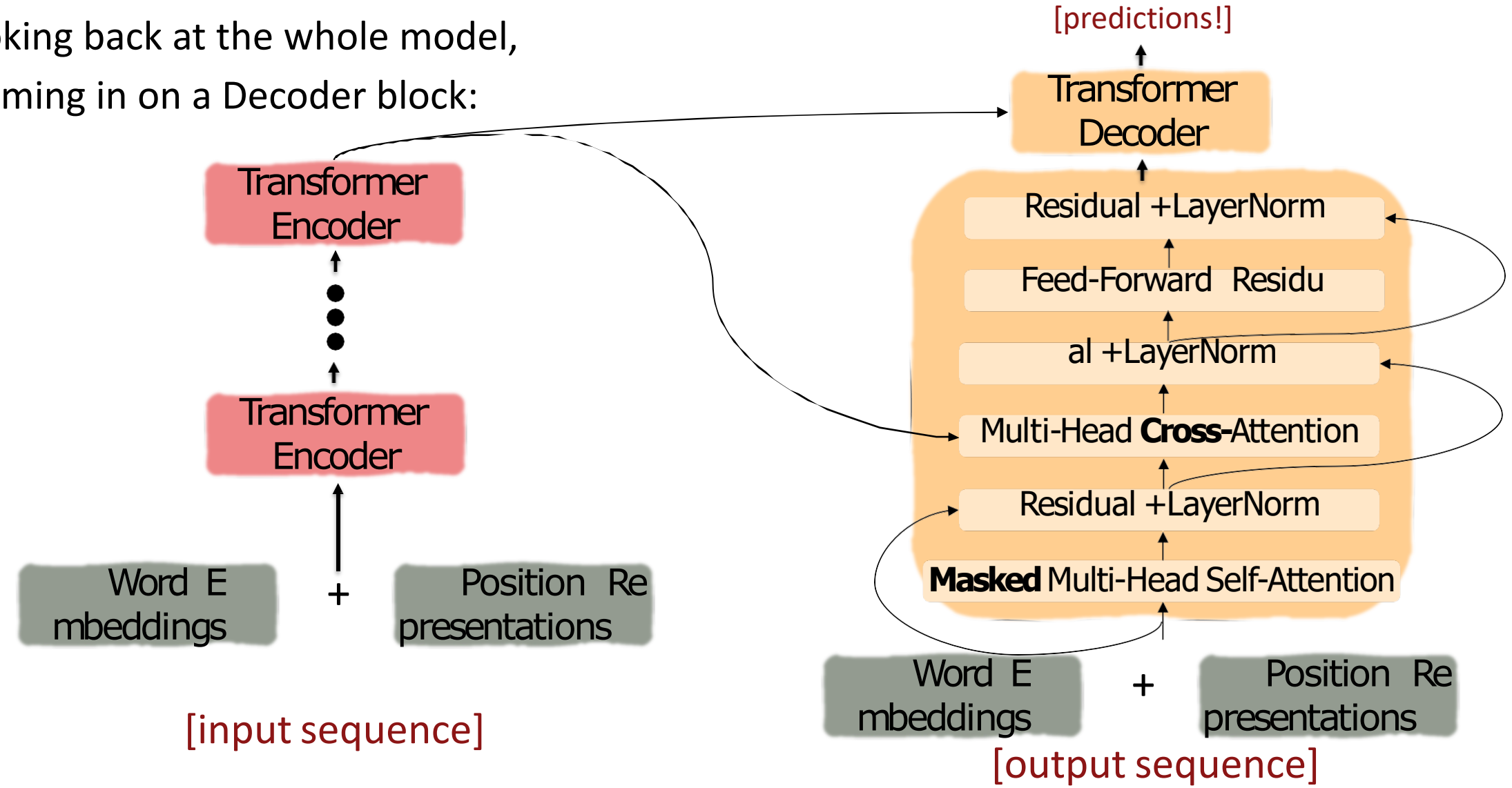
Looking back at the whole model, zooming in on an Encoder block:



The Transformer Encoder-Decoder

Looking back at the whole model,

zooming in on a Decoder block:

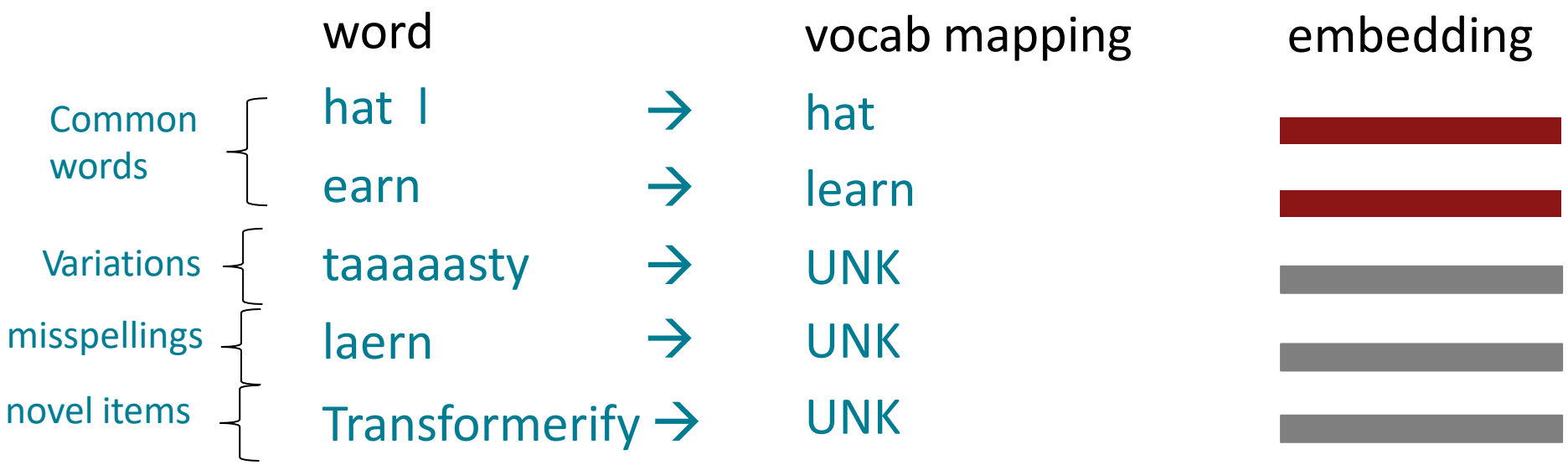


Word structure and subword models

Let's take a look at the assumptions we've made about a language's vocabulary.

We assume a fixed vocab of tens of thousands of words, built from the training set.

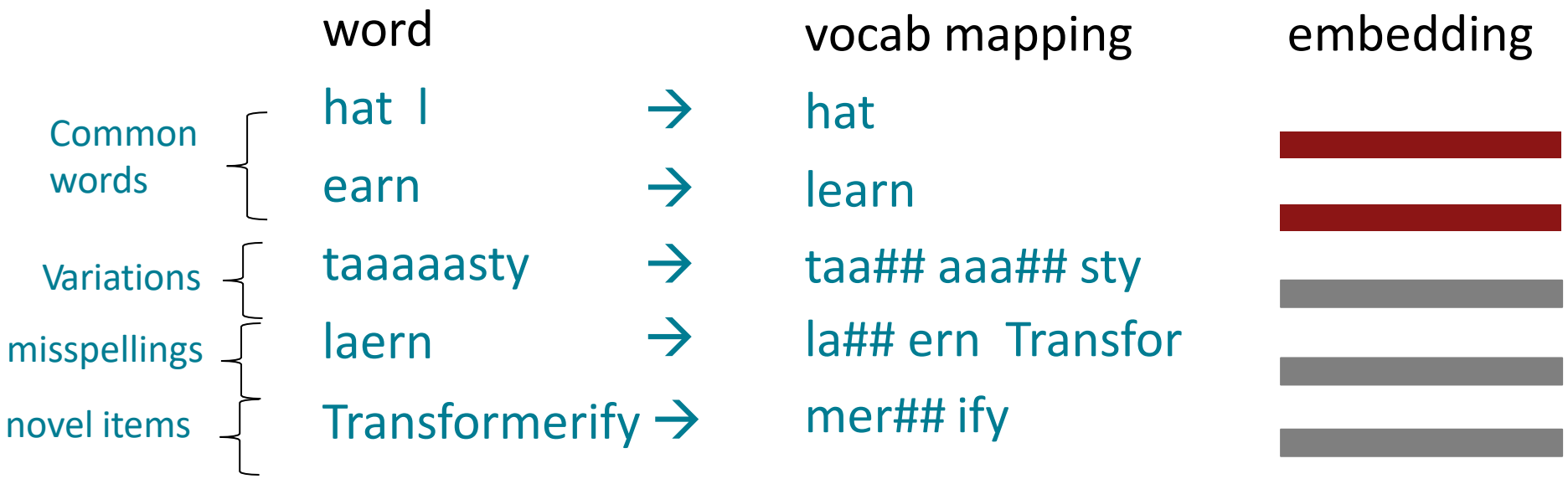
All *novel* words seen at test time are mapped to a single UNK.



Word structure and subword models

Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

In the worst case, words are split into as many subwords as they have characters.

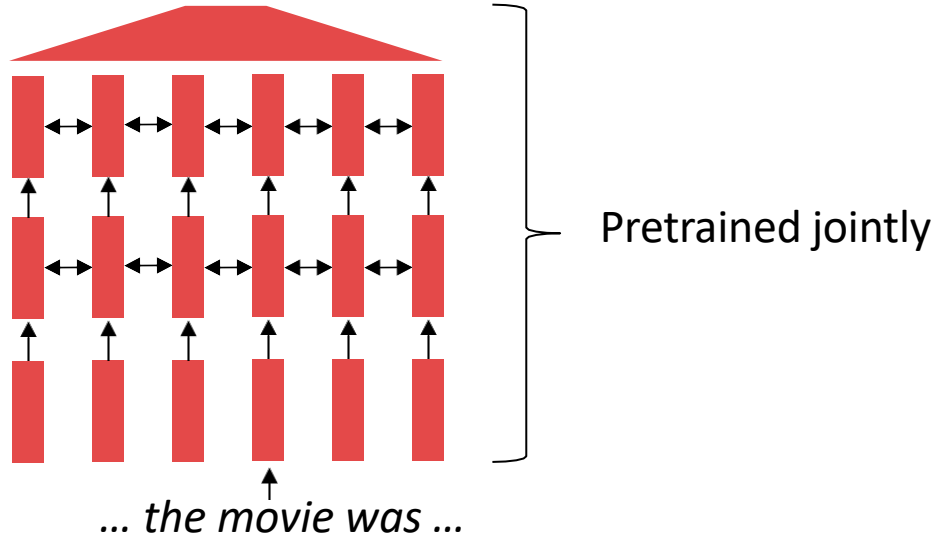


Where we're going: pretraining whole models

In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via **pretraining**.
- Pretraining methods hide parts of the input from the model, and then train the model to reconstruct those parts.

- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.
  - **probability distributions** over language that we can sample from



[This model has learned how to represent entire sentences through pretraining]

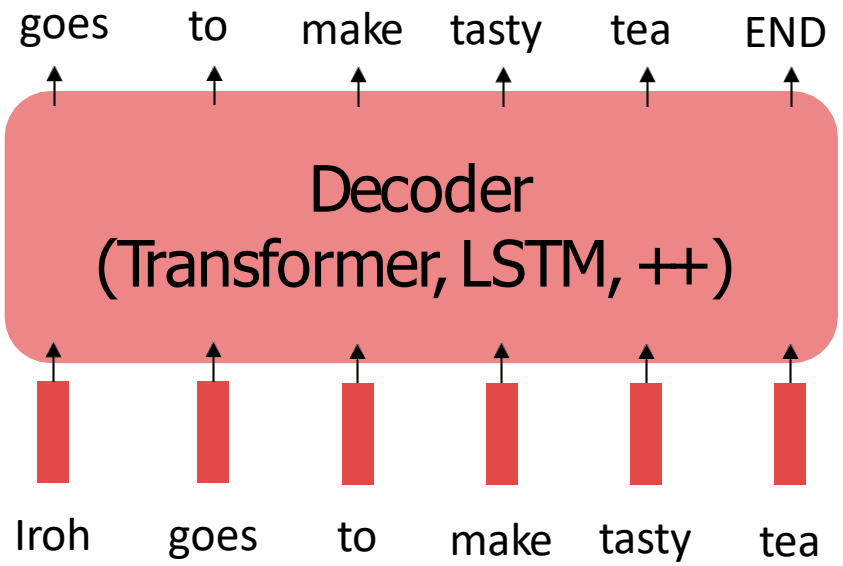
Pretraining through language modeling

Recall the **language modeling** task:

- Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution over words given their past contexts.
- There's lots of data for this! (In English.)

Pretraining through language modeling:

- Train a neural network to perform language modeling on a large amount of text.
- Save the network parameters.

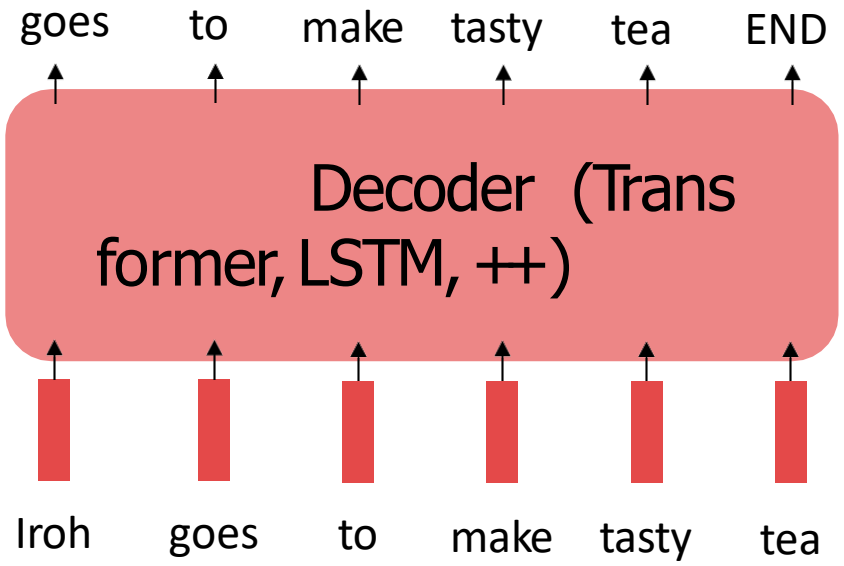


The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

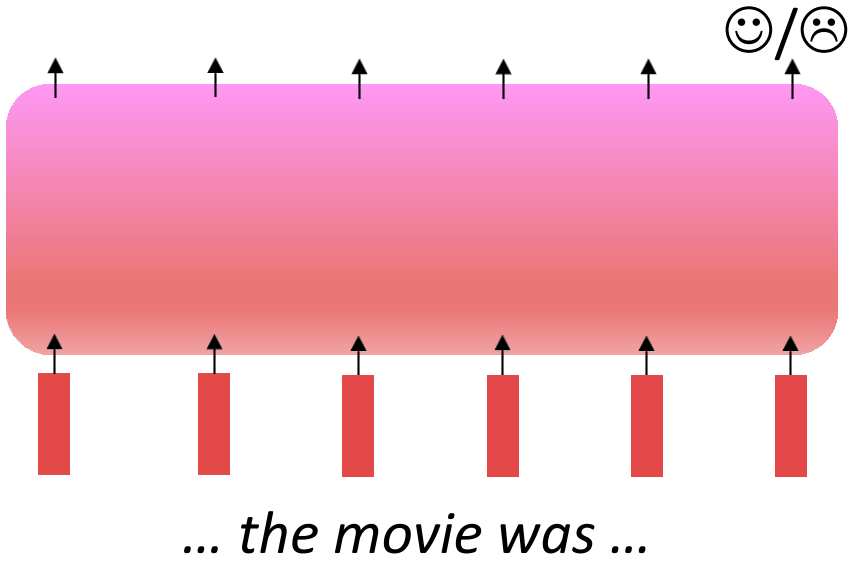
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



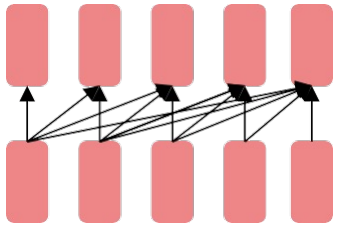
Step 2: Finetune (on your task)

Not many labels; adapt to the task!



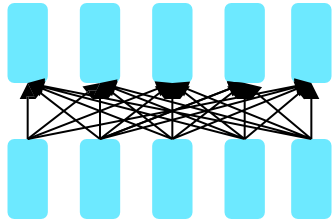
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



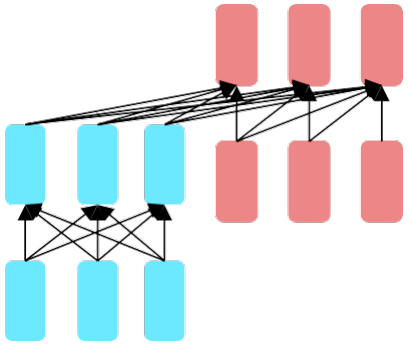
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa

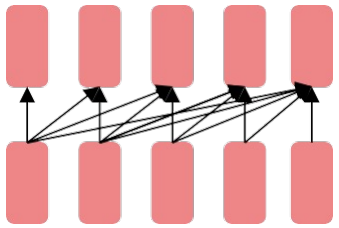


Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

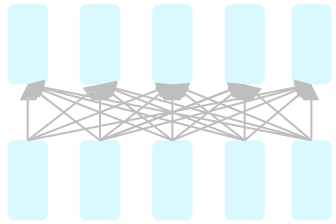
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



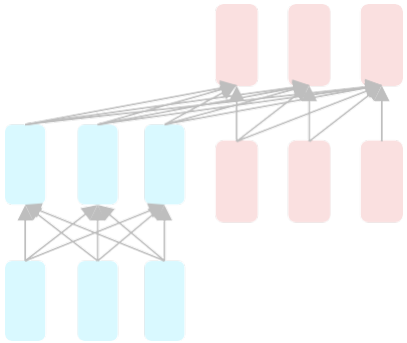
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Pretraining decoders

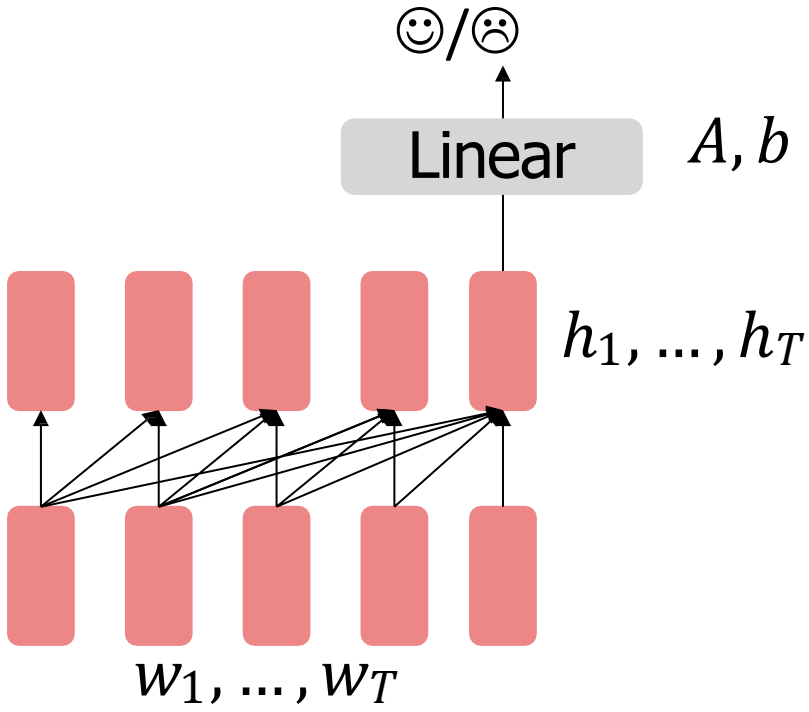
When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t | w_{1:t\%1})$.

We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$$
$$y \sim Ah_T + b$$

Where A and b are randomly initialized and specified by the downstream task.

Gradients backpropagate through the whole network.



[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

Pretraining decoders

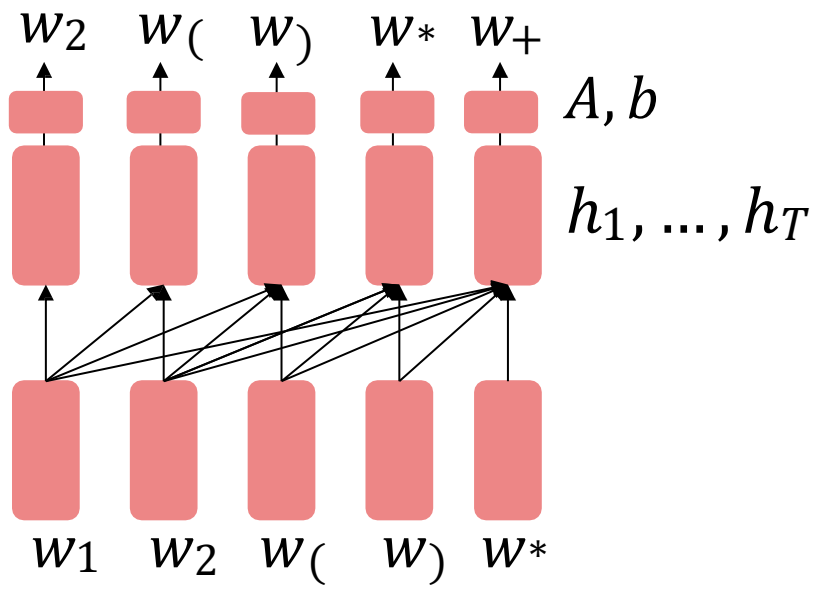
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_!(w_t|w_{1:t\%1})$!

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pret raining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$$
$$w_t \sim Ah_{t\%1} + b$$

Where A, b were pretrained in the language model!



[Note how the linear layer has been pretrained.]

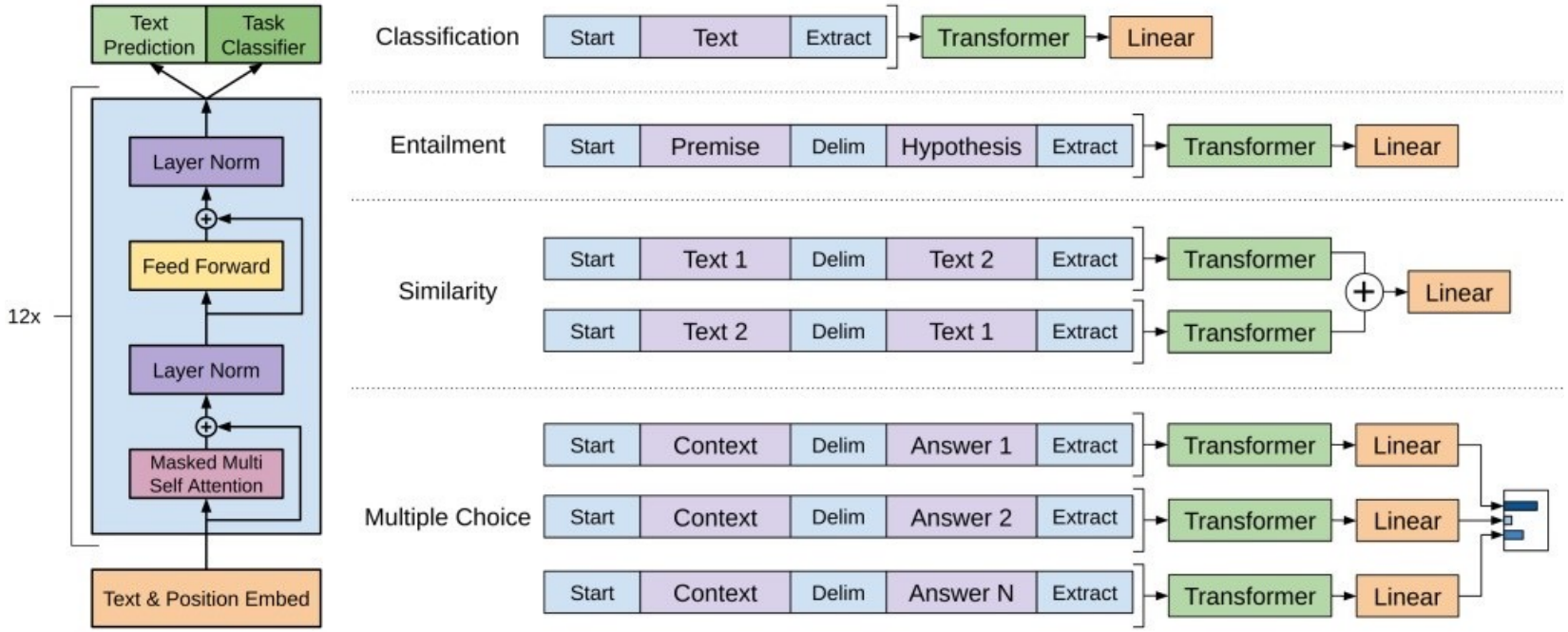
Generative Pretrained Transformer (GPT) [Radford et al., 2018]

2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

How do we format inputs to our decoder for **finetuning tasks?**



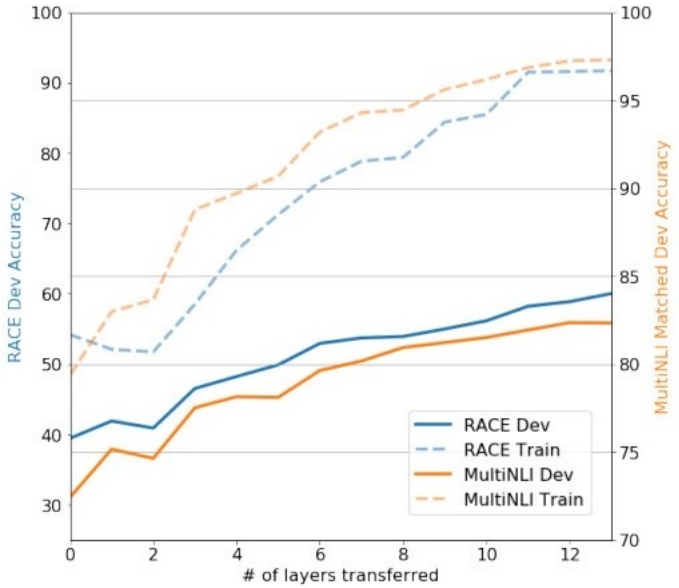
The linear classifier is applied to the representation of the [EXTRACT] token.

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

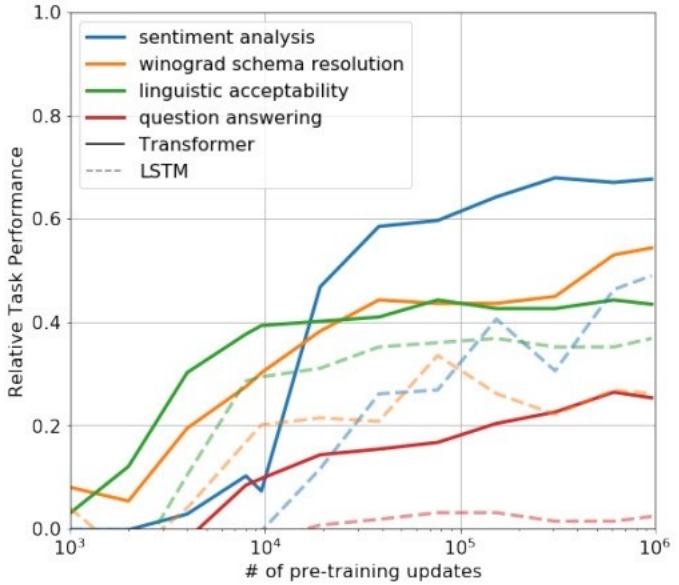
GPT results on various *natural language inference* datasets.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

Examining the Effect of Pretraining in GPT [Radford et al., 2018]



As more layers are transferred, performance improves on RACE (a large- scale reading comprehension dataset) and MultiNLI.



Zero-shot performance of Transformer vs. LSTM as a function of the # of pre-training updates.

Increasingly convincing generations (GPT2) [Radford et al., 2018]

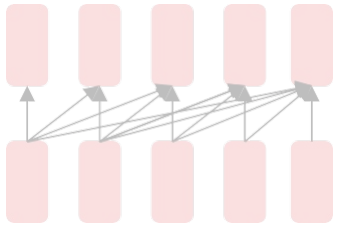
We mentioned how pretrained decoders can be used **in their capacities as language models.**

GPT-2, a larger version of GPT trained on more data, was shown to produce relatively convincing samples of natural language.

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

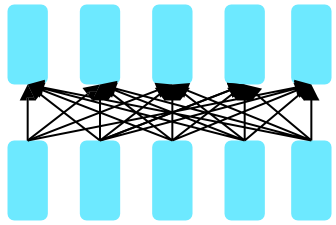
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



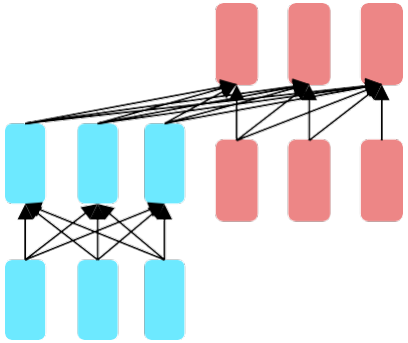
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

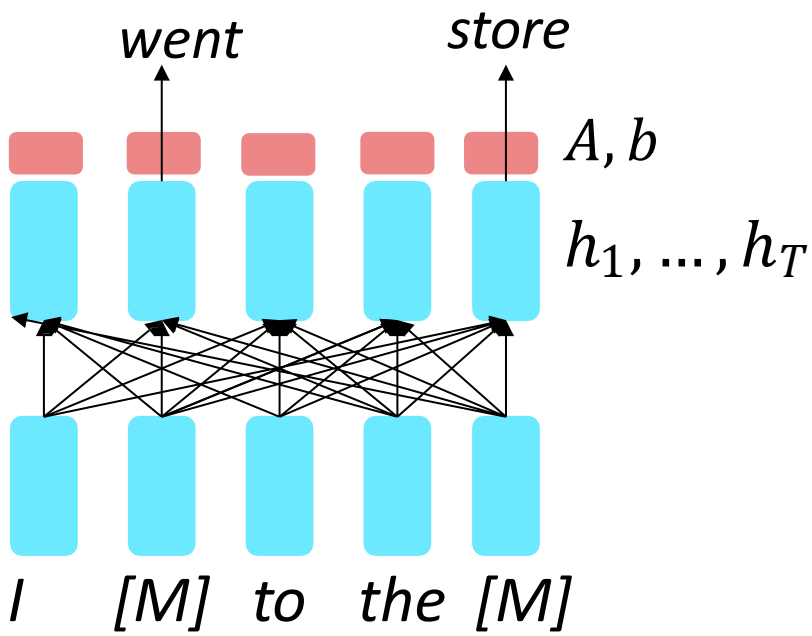
- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Pretraining encoders: what pretraining objective to use?

So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called **Masked LM**.



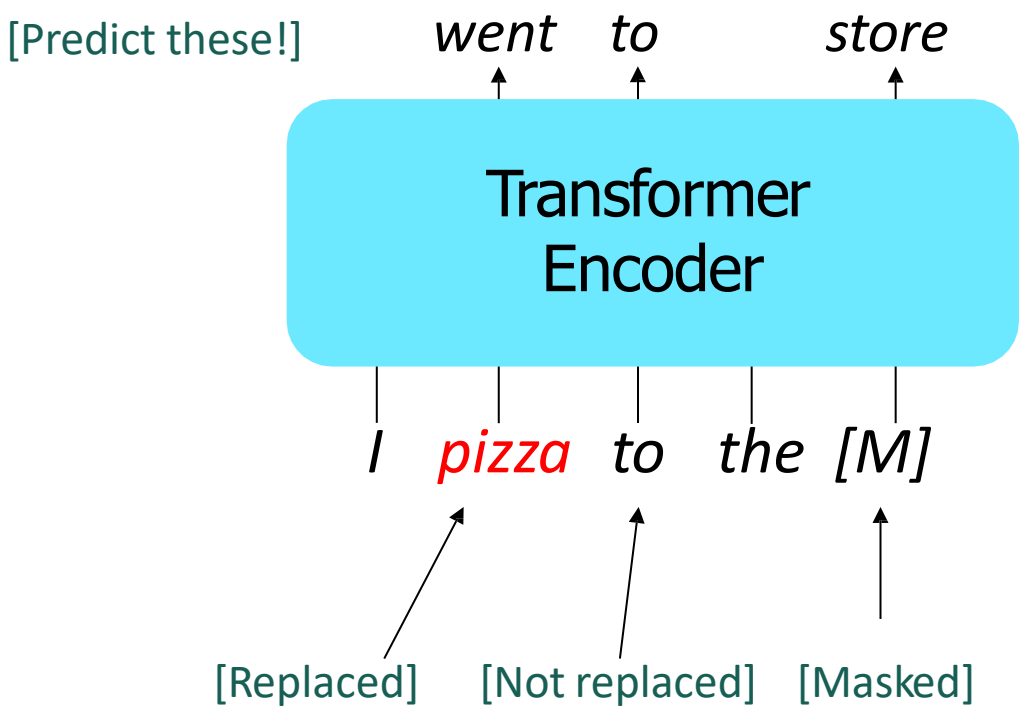
[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the "Masked LM" objective, open-sourced their model as the tensor2tensor library, and **released the weights of their pretrained Transformer (BERT)**.

Some more details about Masked LM for BERT:

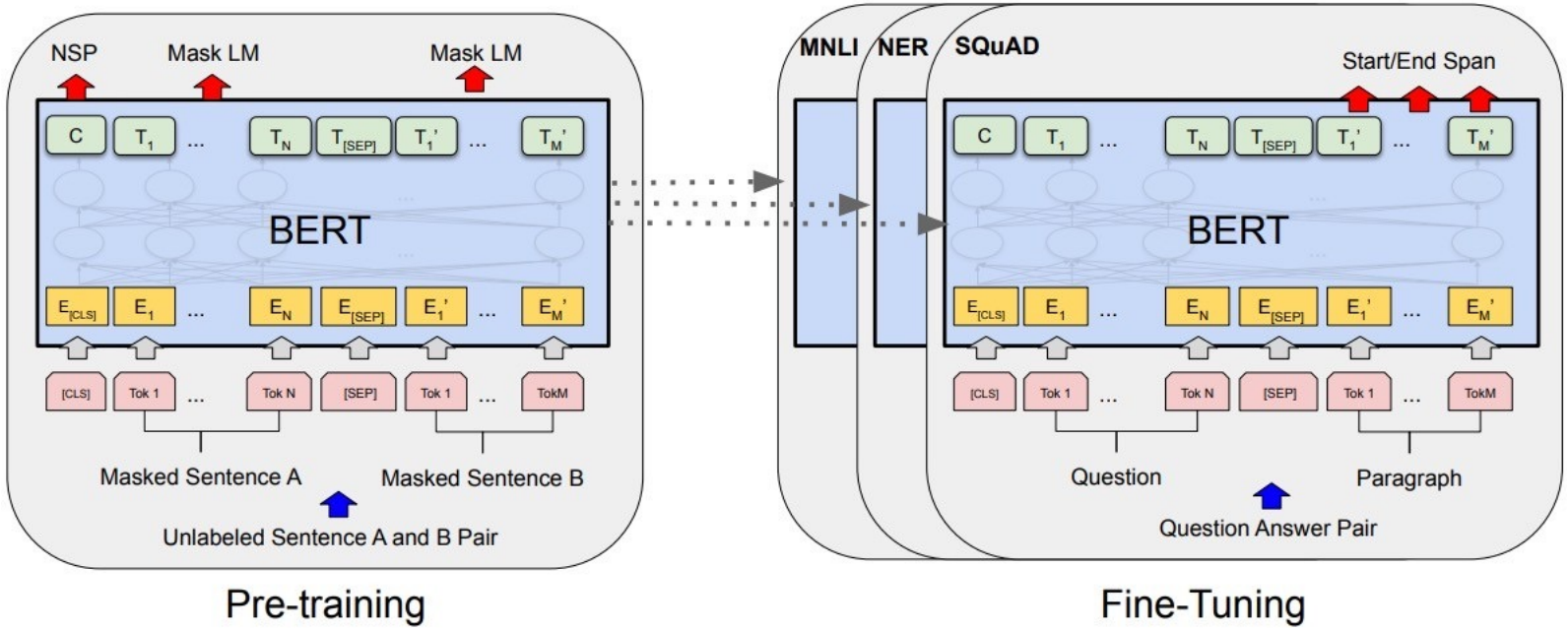
- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

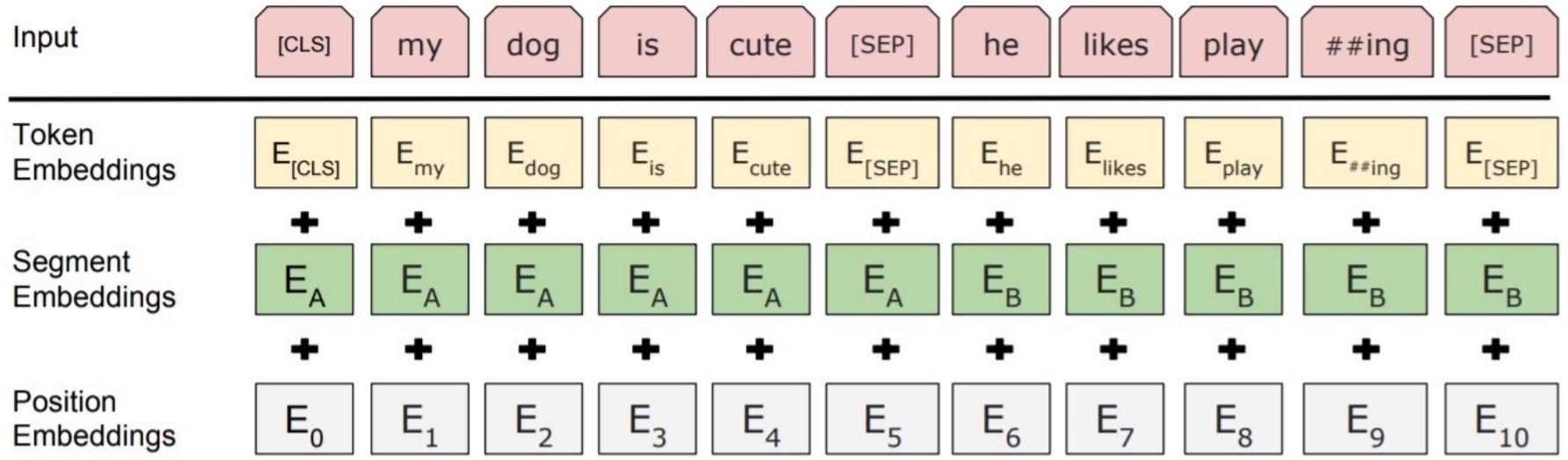
- **Unified Architecture:** As shown below, there are minimal differences bet ween the pre-training architecture and the fine-tuned version for each do wnstream task.



[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

- The pretraining input to BERT was two separate contiguous chunks of text:



- BERT was trained to predict whether one chunk follows the other or is randomly sampled.
  - Later work has argued this “next sentence prediction” is not necessary.

[Devlin et al., 2018, Liu et al., 2019]

BERT: Bidirectional Encoder Representations from Transformers

Details about BERT

- Two models were released:
  - BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
  - BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.
- Trained on:
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)
- Pretraining is expensive and impractical on a single GPU.
  - BERT was pretrained with 64 TPU chips for a total of 4 days.
  - (TPUs are special tensor operation acceleration hardware)
- Finetuning is practical and common on a single GPU
  - “Pretrain once, finetune many times.”

[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

BERT was massively popular and hugely versatile; finetuning BERT led to new state-of-the-art results on a broad range of tasks.

- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI**: natural language inference over question answering data
- **SST-2**: sentiment analysis
- **CoLA**: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B**: semantic textual similarity
- **MRPC**: microsoft paraphrase corpus
- **RTE**: a small natural language inference corpus

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

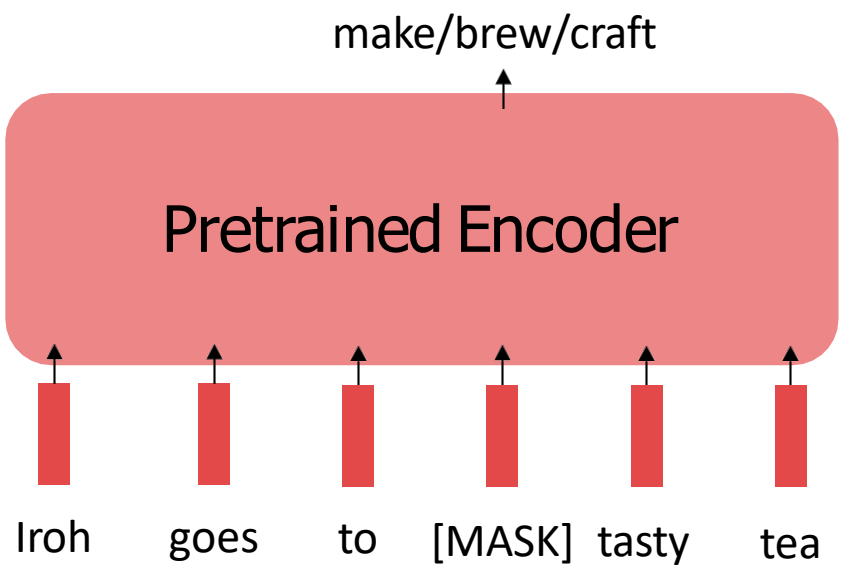
Note that $\text{BERT}_{\text{BASE}}$ was chosen to have the same number of parameters as OpenAI GPT.

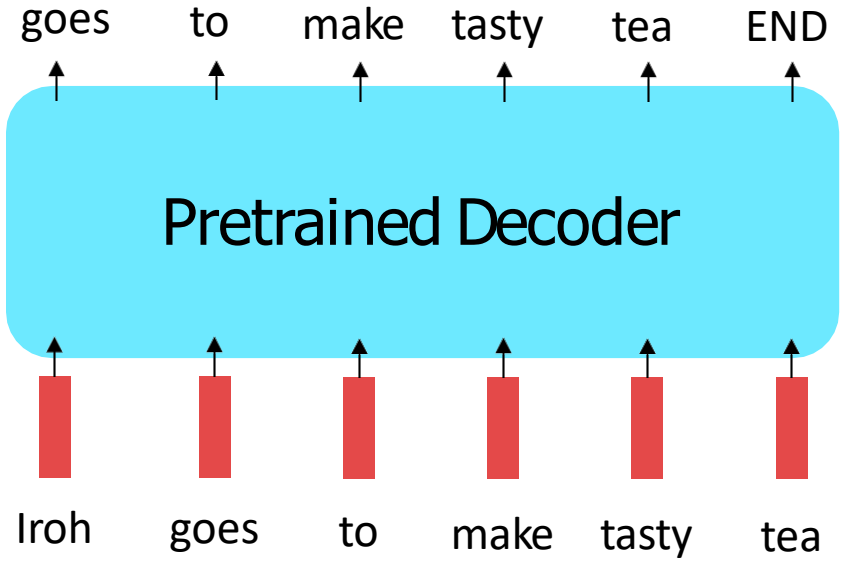
[Devlin et al., 2018]

Limitations of pretrained encoders

Those results looked great! Why not used pretrained encoders for everything?

If your task involves generating sequences, consider using a pretrained decoder; BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation m ethods.



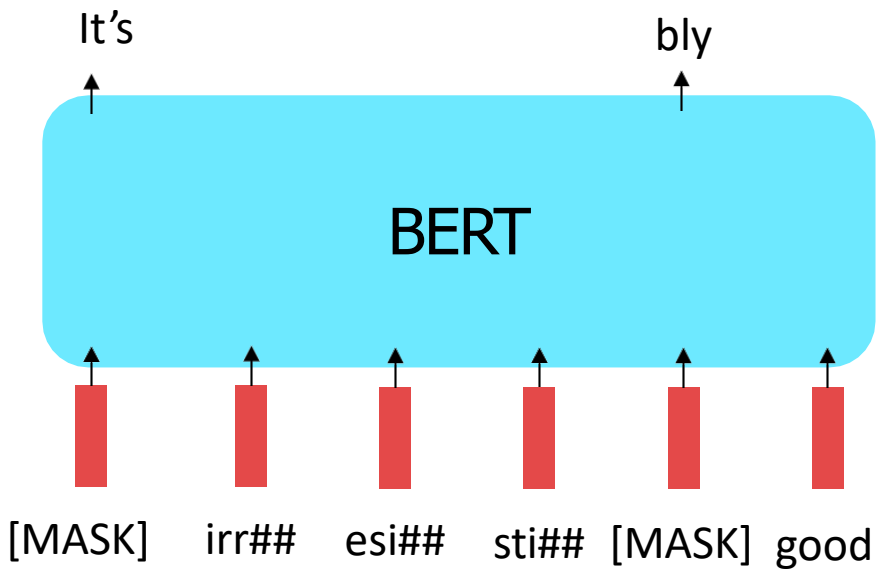


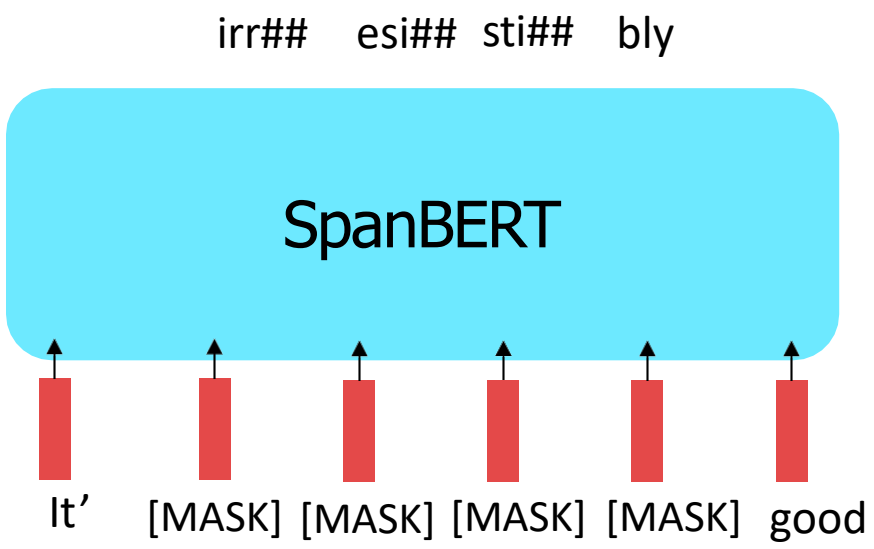
Extensions of BERT

You'll see a lot of BERT variants like RoBERTa, SpanBERT, +++

Some generally accepted improvements to the BERT pretraining formula:

- RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
- SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task





[Liu et al., 2019; Joshi et al., 2020]

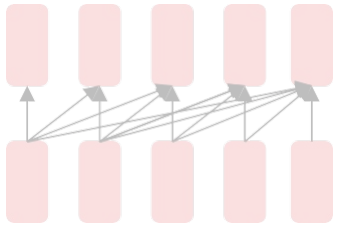
Extensions of BERT

A takeaway from the RoBERTa paper: more compute, more data can improve pretraining even when not changing the underlying Transformer encoder.

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

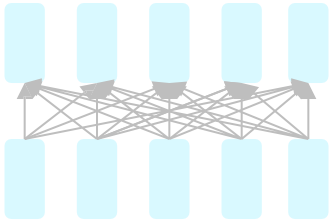
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



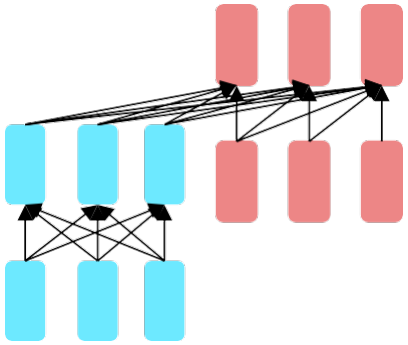
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

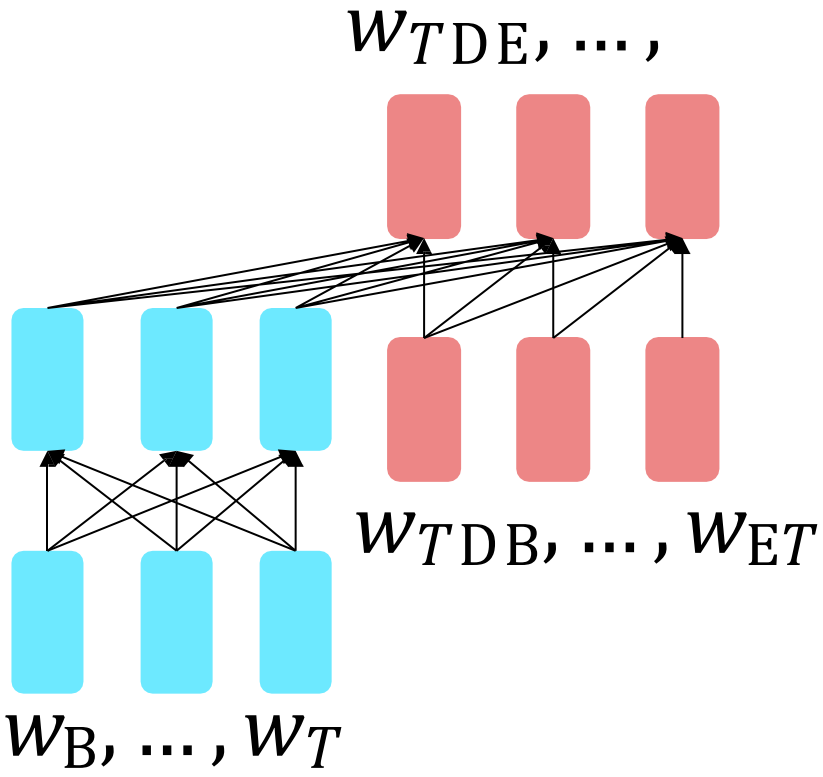
- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Pretraining encoder-decoders: what pretraining objective to use?

For **encoder-decoders**, we could do something like **language modeling**, but where a prefix of every input is provided to the encoder and is not predicted.

$$h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T)$$
$$h_{T,\ 1}, \ldots, h_2 = Decoder(w_1, \ldots, w_T, h_1, \ldots, h_T)$$
$$y_{\text{i}} \sim Aw_{\text{i}} + b, i > T$$

The **encoder** portion benefits from bidirecti onal context; the **decoder** portion is used to train the whole model through language m odeling.



[Raffel et al., 2018]

Pretraining encoder-decoders: what pretraining objective to use?

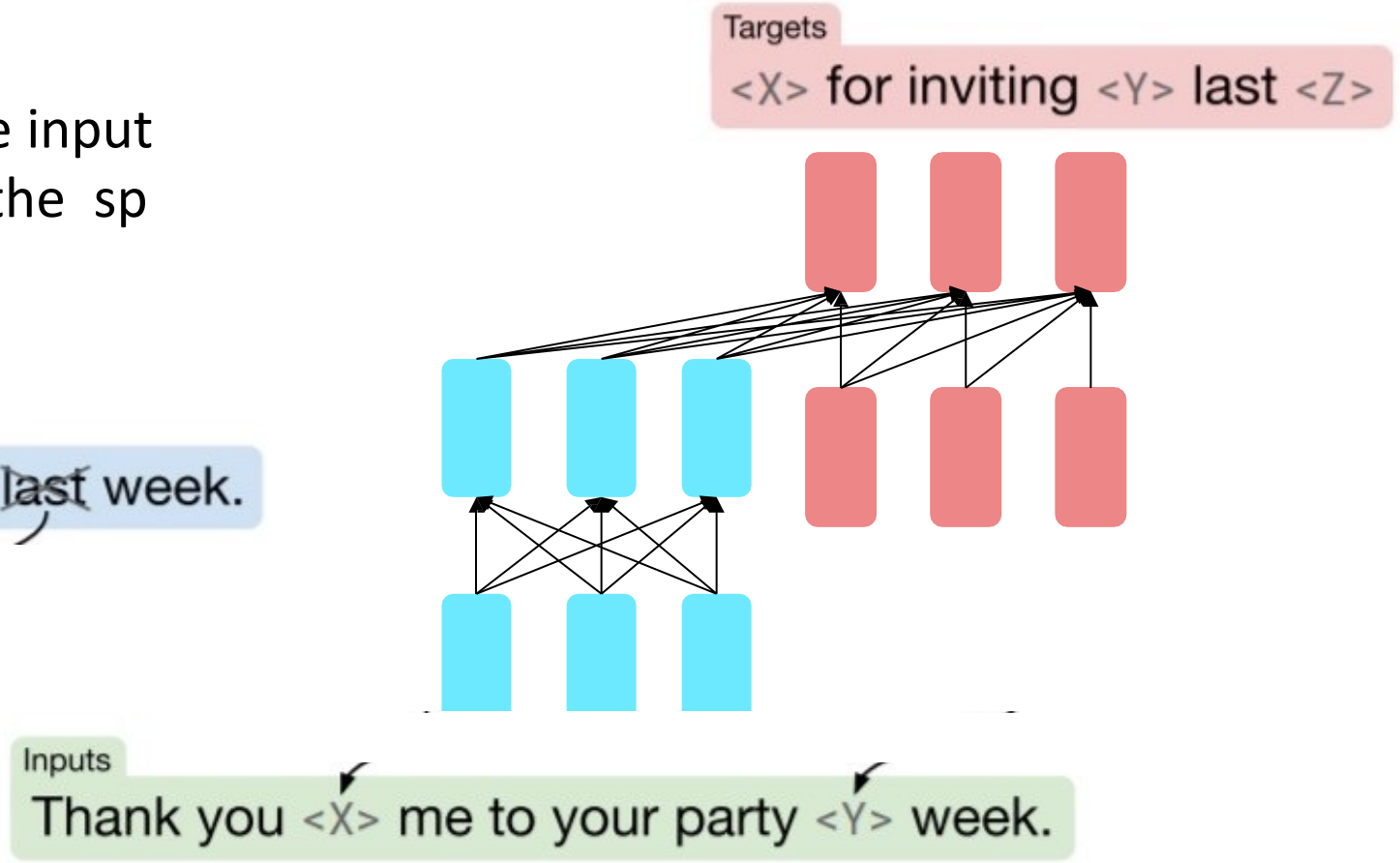
What Raffel et al., 2018 found to work best was **span corruption.** Their model: **T5**.

Replace different-length spans from the input with unique placeholders; decode out the sp ans that were removed!

Original text

Thank you ~~for inviting~~ me to your party ~~last~~ week.

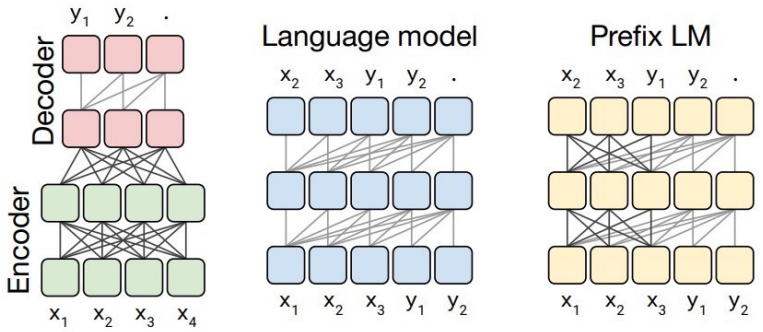
This is implemented in text preproc essing: it's still an objective that loo ks like **language modeling** at the dec oder side.



[Raffel et al., 2018]

Pretraining encoder-decoders: what pretraining objective to use?

Raffel et al., 2018 found encoder-decoders to work better than decoders for their tasks, and span corruption (denoising) to work better than language modeling.

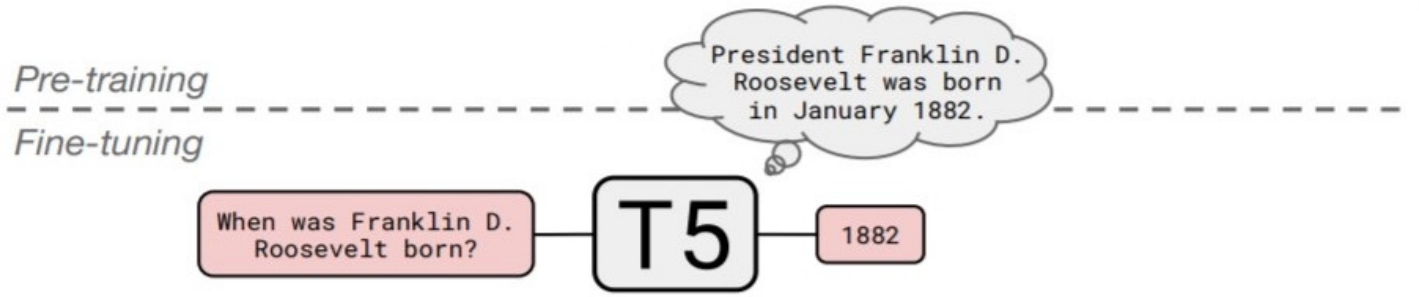


Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

[Raffel et al., 2018]

Pretraining encoder-decoders: what pretraining objective to use?

A fascinating property of T5: it can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.



	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

NQ: Natural Questions

WQ: WebQuestions T

QA: Trivia QA

All “open-domain” versions

[Raffel et al., 2018]

GPT-3, In-context learning, and very large models

So far, we've interacted with pretrained models in two ways:

- Sample from the distributions they define (maybe providing a prompt)
- Fine-tune them on a task we care about, and then take their predictions.

Emergent behavior: Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters.

GPT-3 has 175 billion parameters.

GPT-3, In-context learning, and very large models

Very large language models seem to perform some kind of learning without gradient steps simply from examples you provide within their contexts.

The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing the task to a certain extent.

Input (prefix within a single Transformer decoder context):

“ thanks -> merci

hello -> bonjour

mint -> menthe

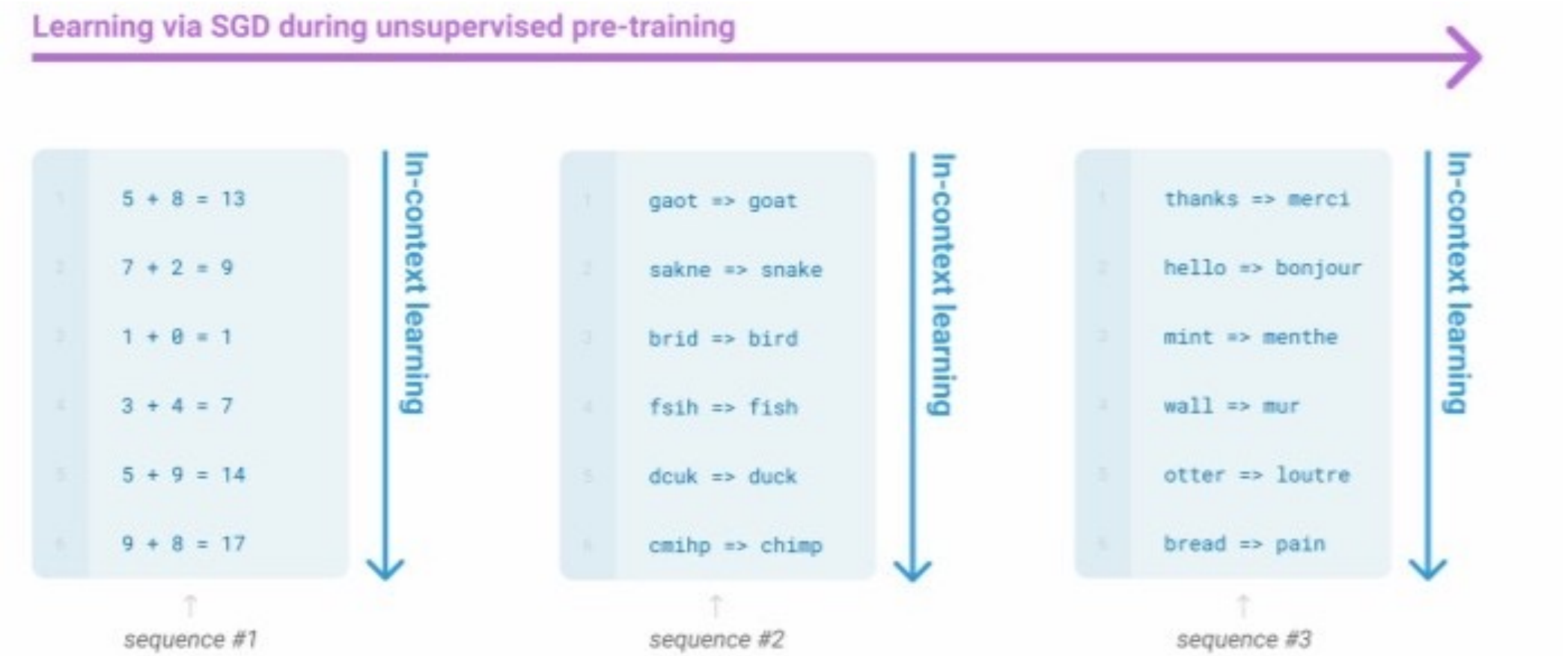
otter -> ”

Output (conditional generations):

loutre…”

GPT-3, In-context learning, and very large models

Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

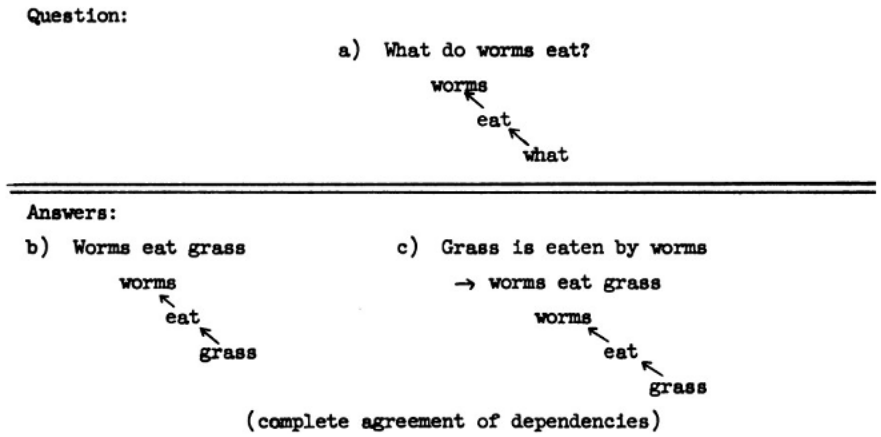


What is question answering?

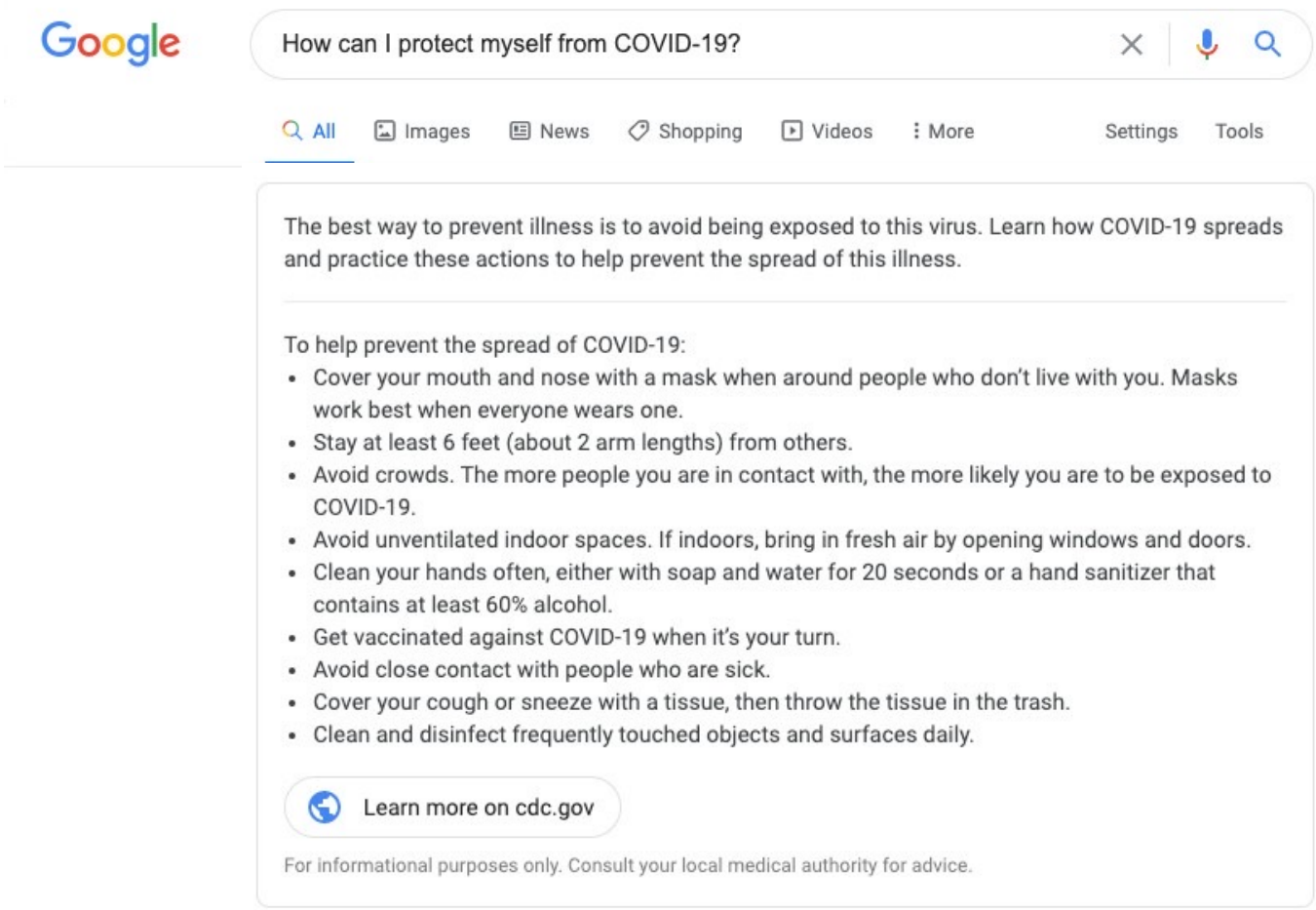
The goal of question answering is to build systems that **automatically** answer questions posed by humans in a **natural language**

The earliest QA systems dated back to 1960s!

(Simmons et al., 1964)



Applications



Reading Comprehension

Reading comprehension = comprehend a passage of text and answer questions about its content (P, Q) A

Tesla was the fourth of five children. He had an older brother named Dane and three sisters, Milka, Angelina and Marica. Dane was killed in a horse-riding accident when Nikola was five. In 1861, Tesla attended the "Lower" or "Primary" School in Smiljan where he studied German, arithmetic, and religio n. In 1862, the Tesla family moved to Gospić, Austrian Empir e, where Tesla's father worked as a pastor. Nikola completed "Lower" or "Primary" School, followed by the "Lower Real Gy mnasium" or "Normal School."

Q: What language did Tesla study while in school?

A: German

Stanford question answering dataset (SQuAD)

- 100k annotated (passage, question, answer) triples

  Large-scale supervised datasets are also a key ingredient for training effective neural models for reading comprehension!

- Passages are selected from English Wikipedia, usually 100~150 words.
- Questions are crowd-sourced.
- Each answer is a short segment of text (or span) in the passage.

  This is a limitation— not all the questions can be answered in this way!

- SQuAD still remains the most popular reading comprehens ion dataset; it is "almost solved" today and the state-of-the-art exceeds the estimated human performance.

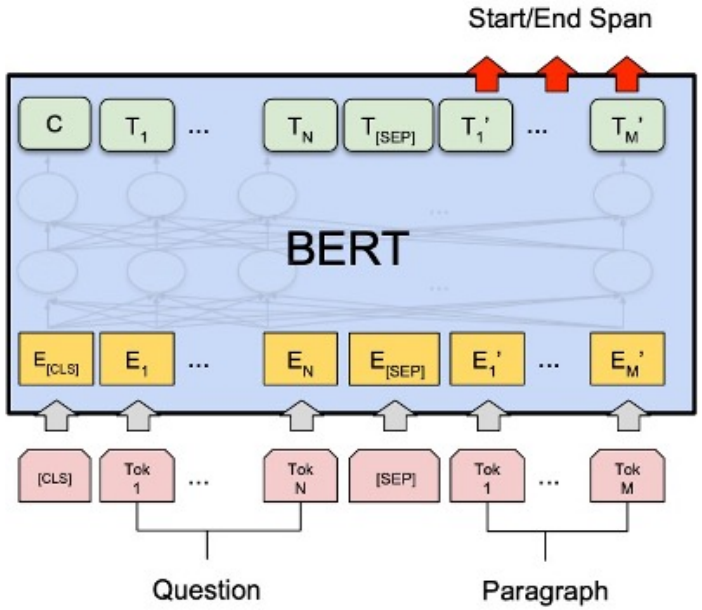
In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

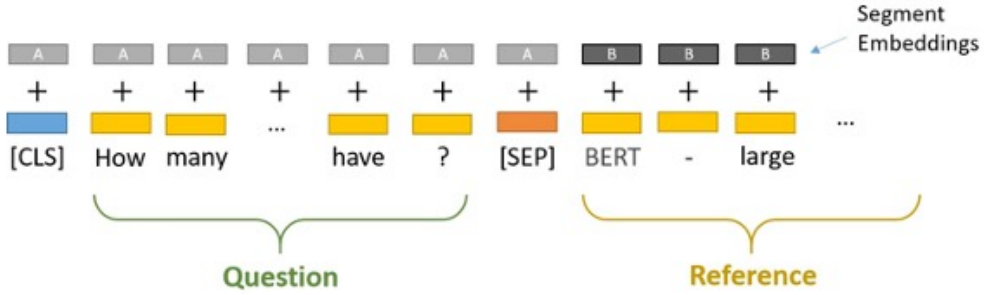
BERT for reading comprehension



Question = Segment A

Passage = Segment B

Answer = predicting two endpoints in segment B



Question: How many parameters does BERT-large have?

Reference Text: BERT-large is really big... it has 24 layers and an embedding size of 1,024, for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple minutes to download to your Colab instance.

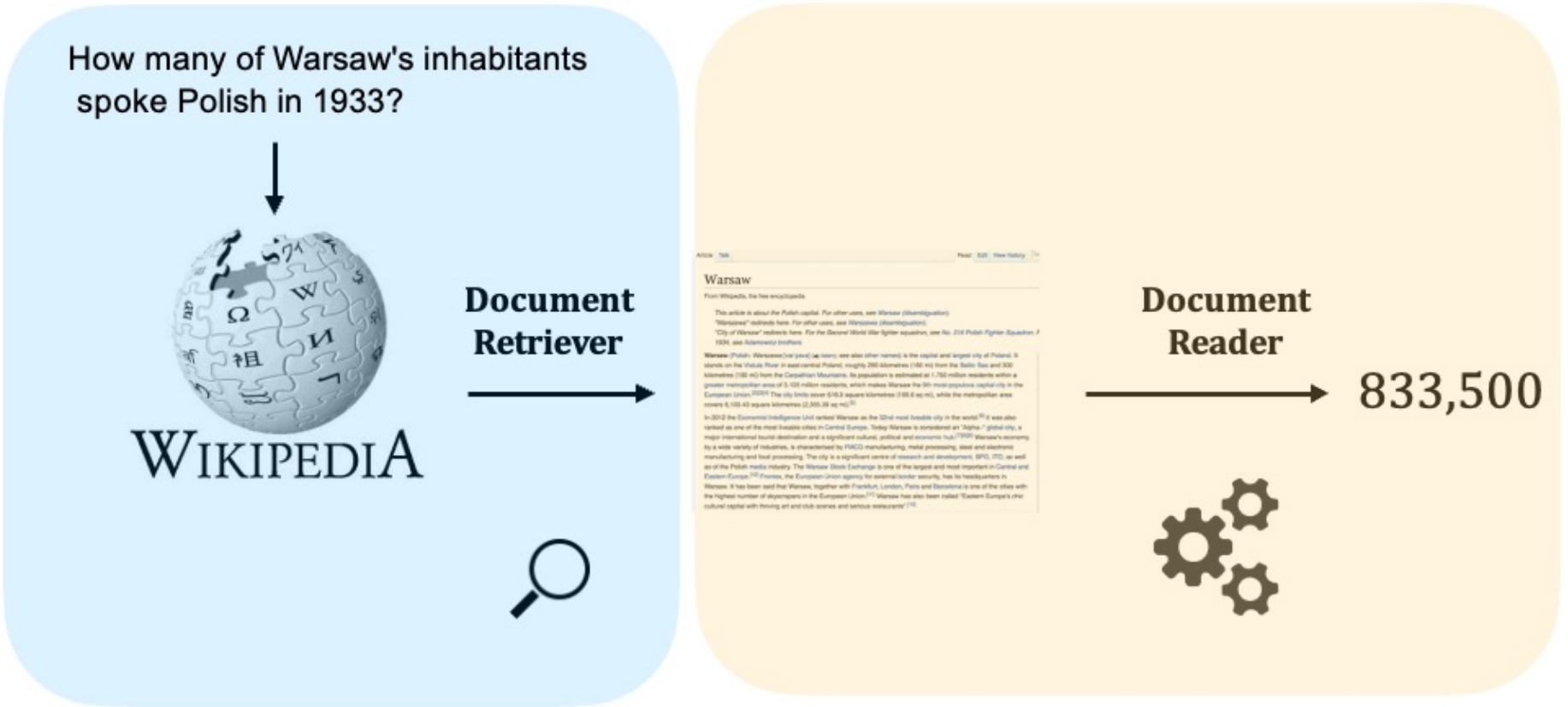
Image credit: https://mccormickml.com/

Open-domain question answering

- Different from reading comprehension, we don't assume a given passage.
- Instead, we only have access to a large collection of documents (e.g., Wikipedia). We don't know where the answer is located, and the goal is to return the answer for any o pen-domain questions.
- Much more challenging and a more practical problem!

*In contrast to **closed-domain** systems that deal with questions under a specific domain (medicine, technical support).*

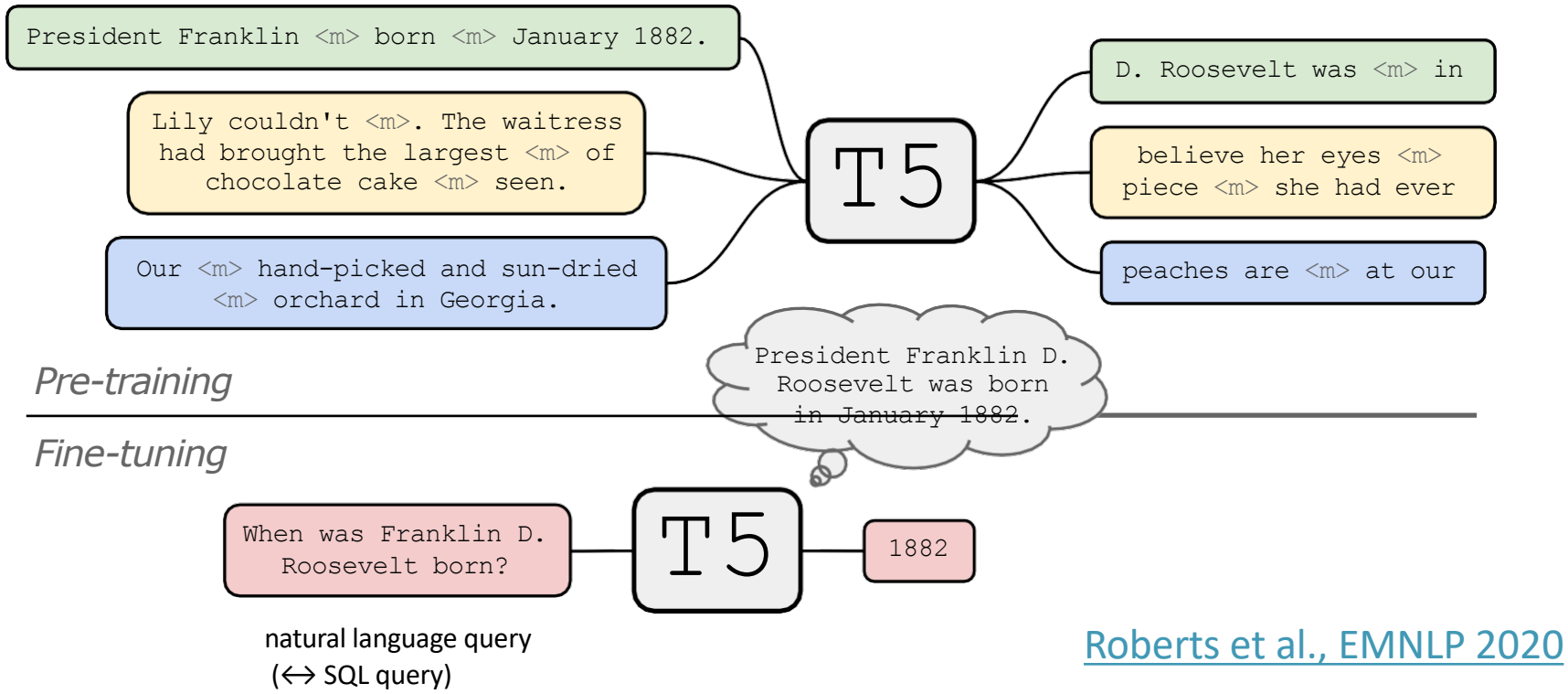
Retriever-reader framework



https://github.com/facebookresearch/DrQA

How to query language models as knowledge bases

- Pretrain LM over unstructured text and then query with natural language.



Roberts et al., EMNLP 2020

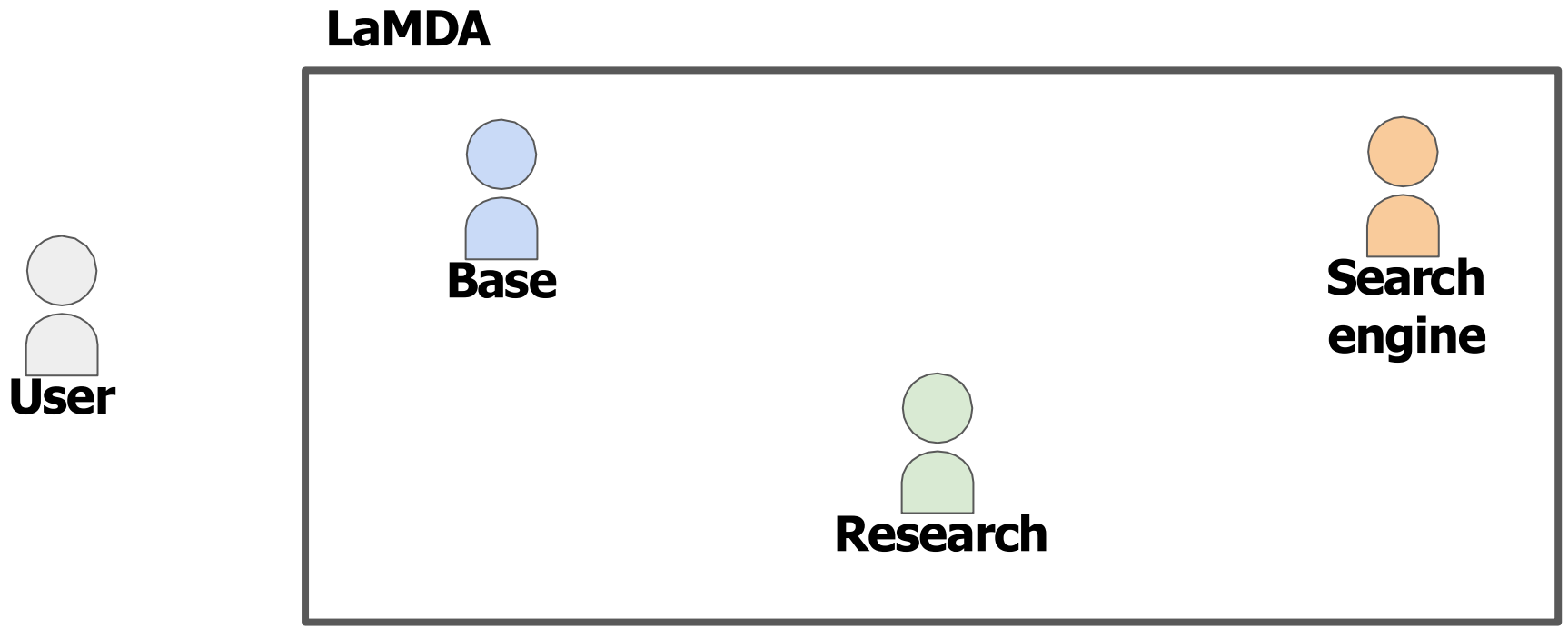
Using an external tool (**LaMDA**: Thoppilan et al, 2022)

- LaMDA is an **open-domain chatbot**: you can talk to it about anything.
- The basic version of the model is simply a Transformer decoder:
  - Input = previous turns in the conversation
  - Output = a new utterance
- This basic model sometimes generates **factually incorrect claims**.

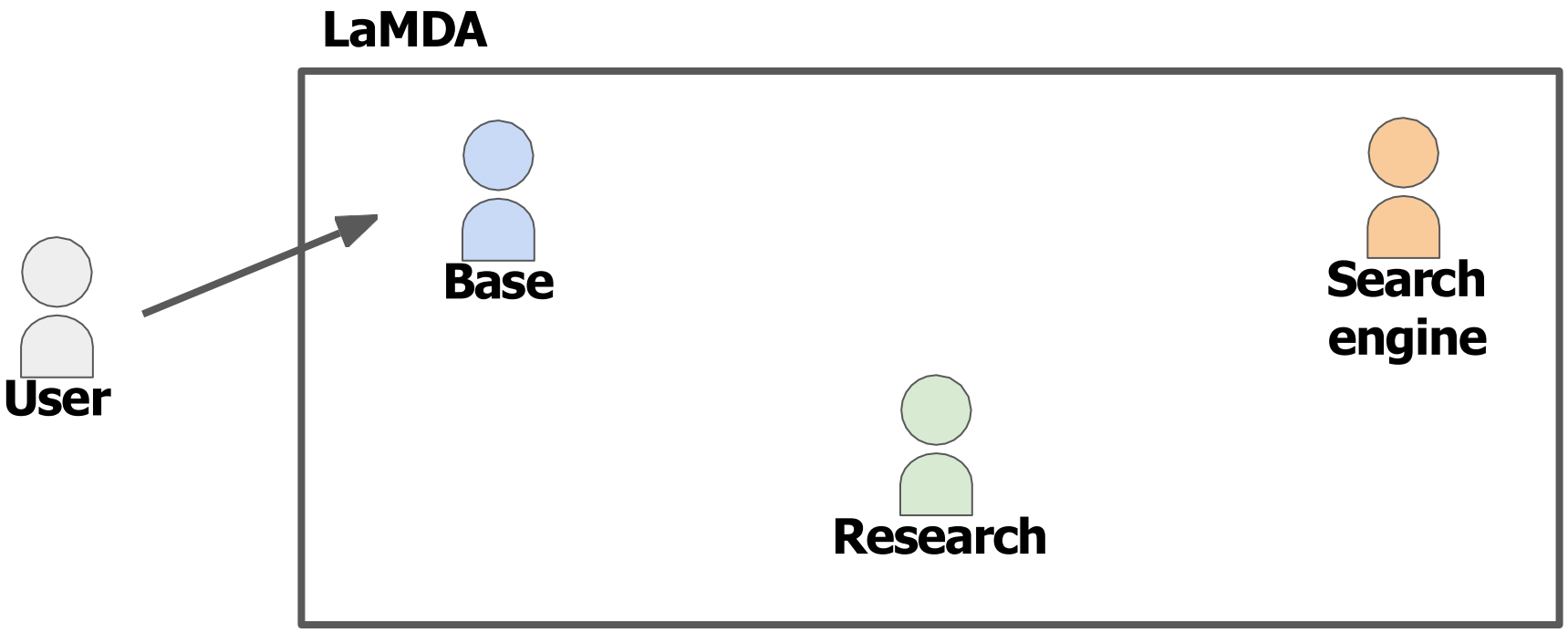
User: What do you think of Rosalie Gascoigne's sculptures?

LaMDA: They're great, and I love how her work changed through her life. I like her later work more than her earlier ones. Her influence is also super interesting - did you know she was one of the artists that inspired Miró?

Using a search engine to improve factuality

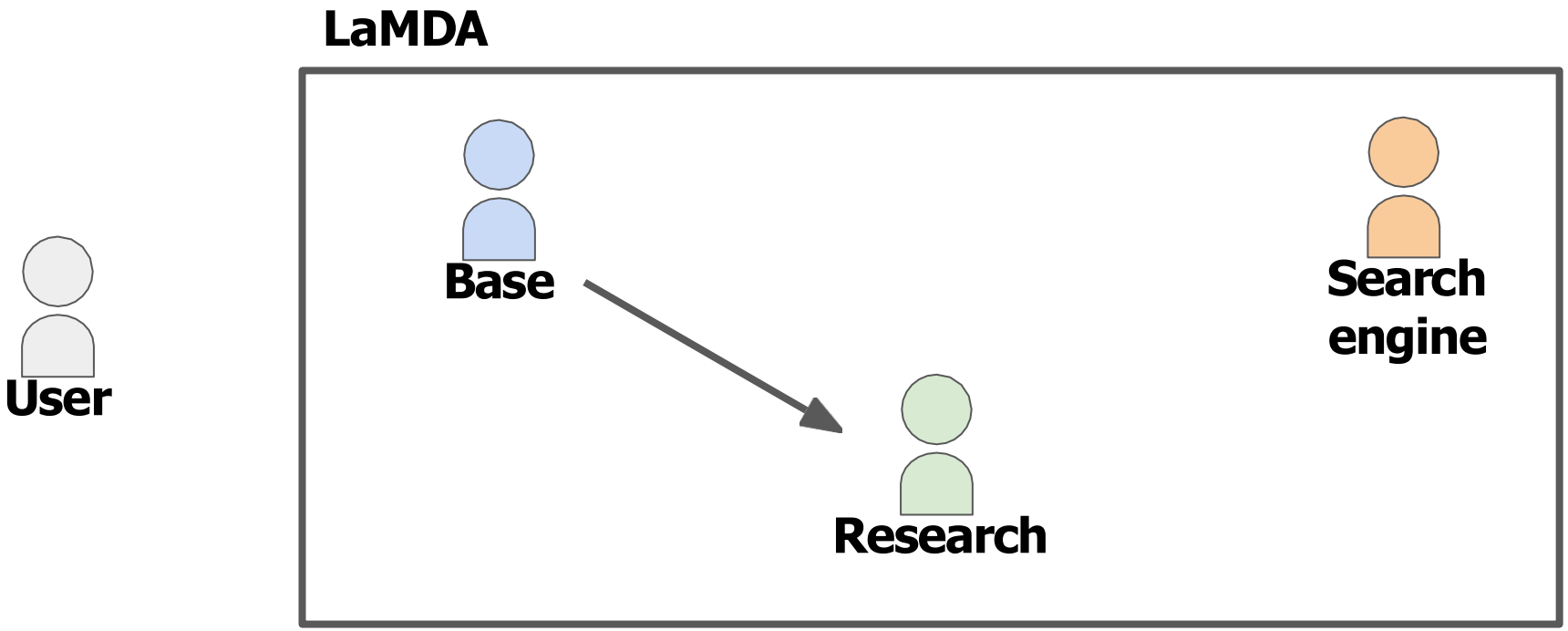


Using a search engine to improve factuality



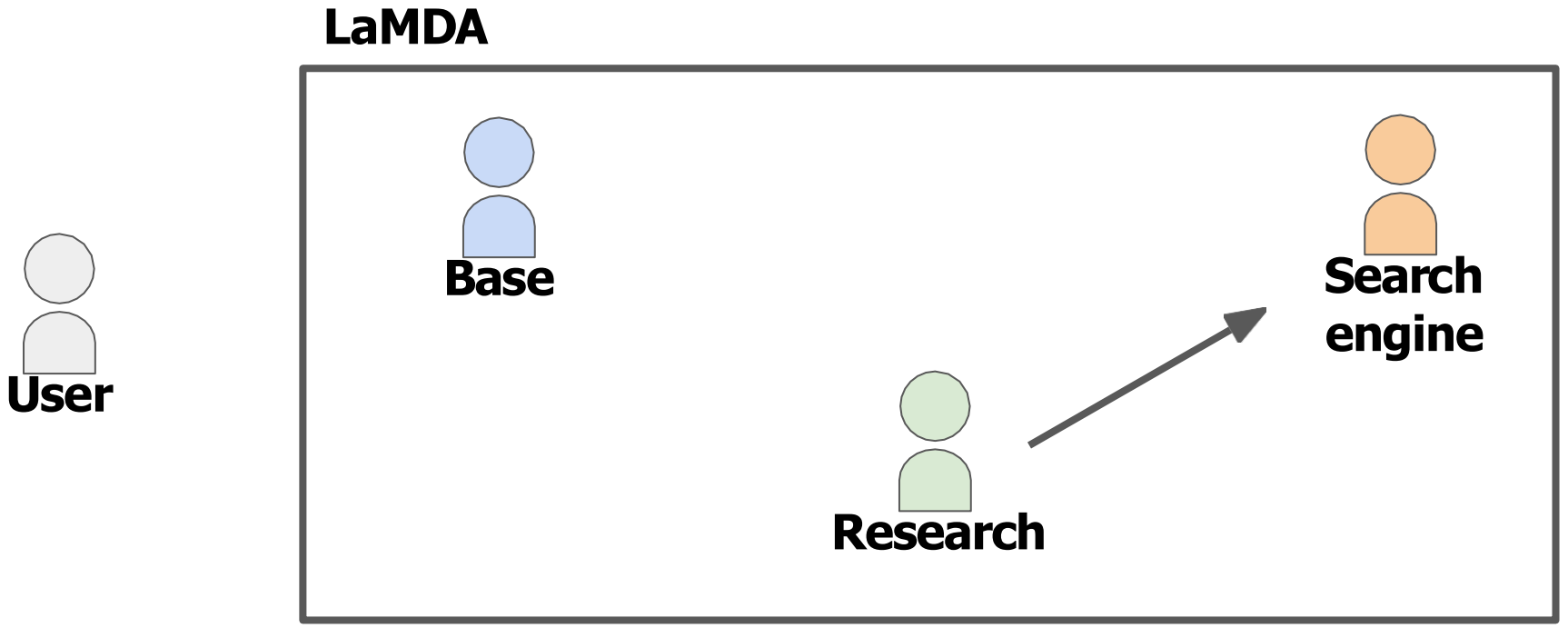
User to Base: When was the Eiffel Tower built?

Using a search engine to improve factuality



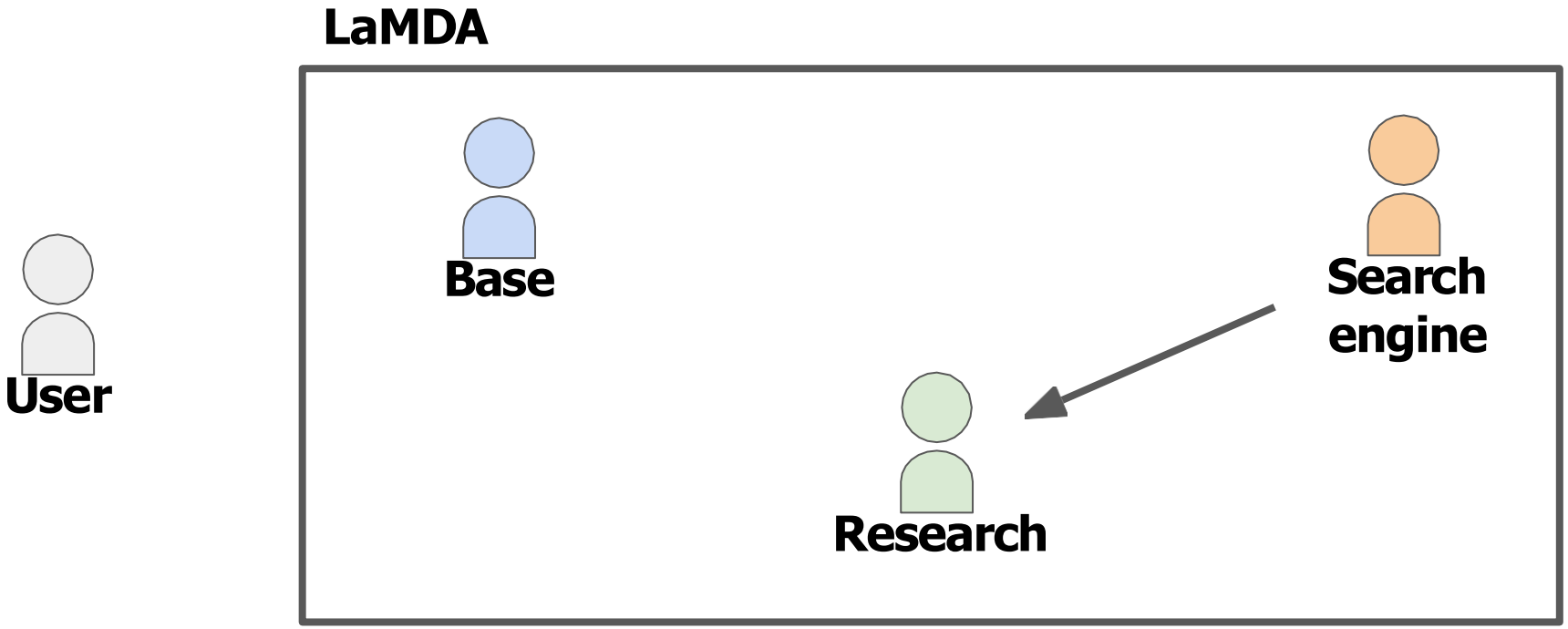
Base to Research: It was constructed in **1887**.

Using a search engine to improve factuality



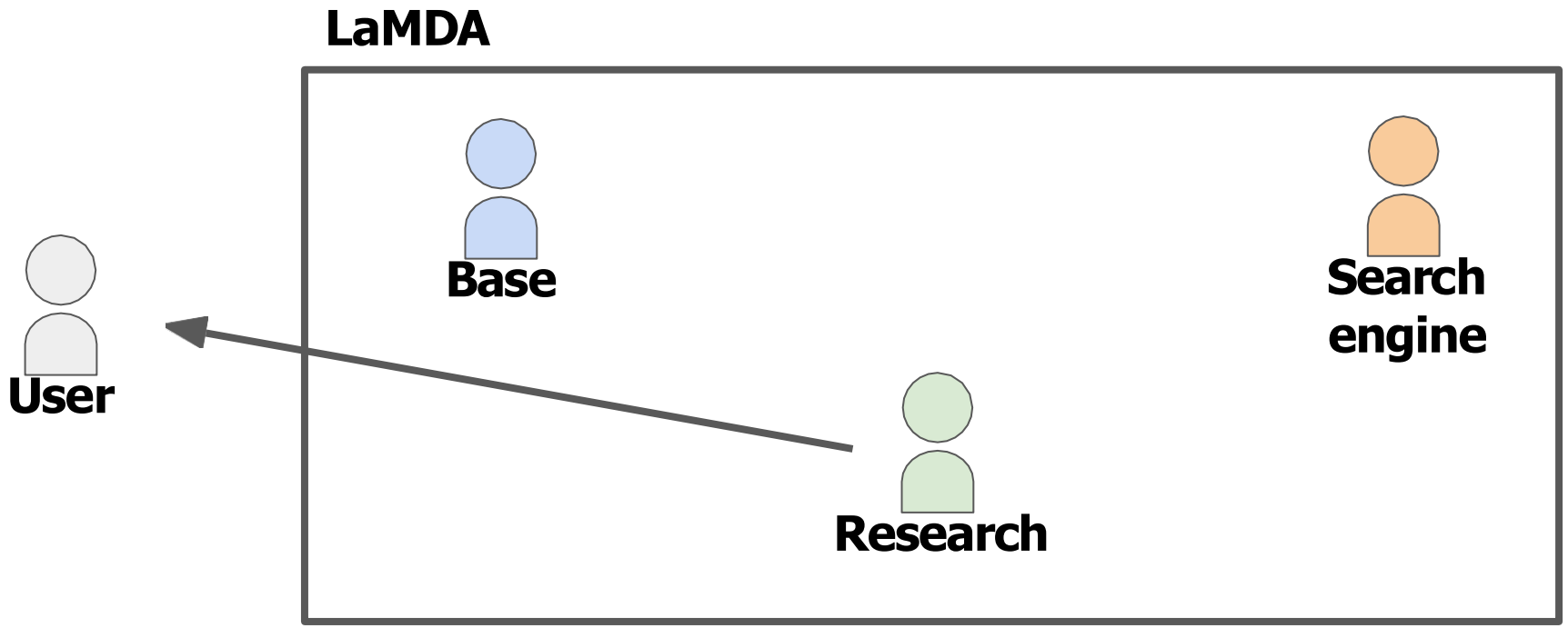
Research to Search: Eiffel Tower construction date

Using a search engine to improve factuality



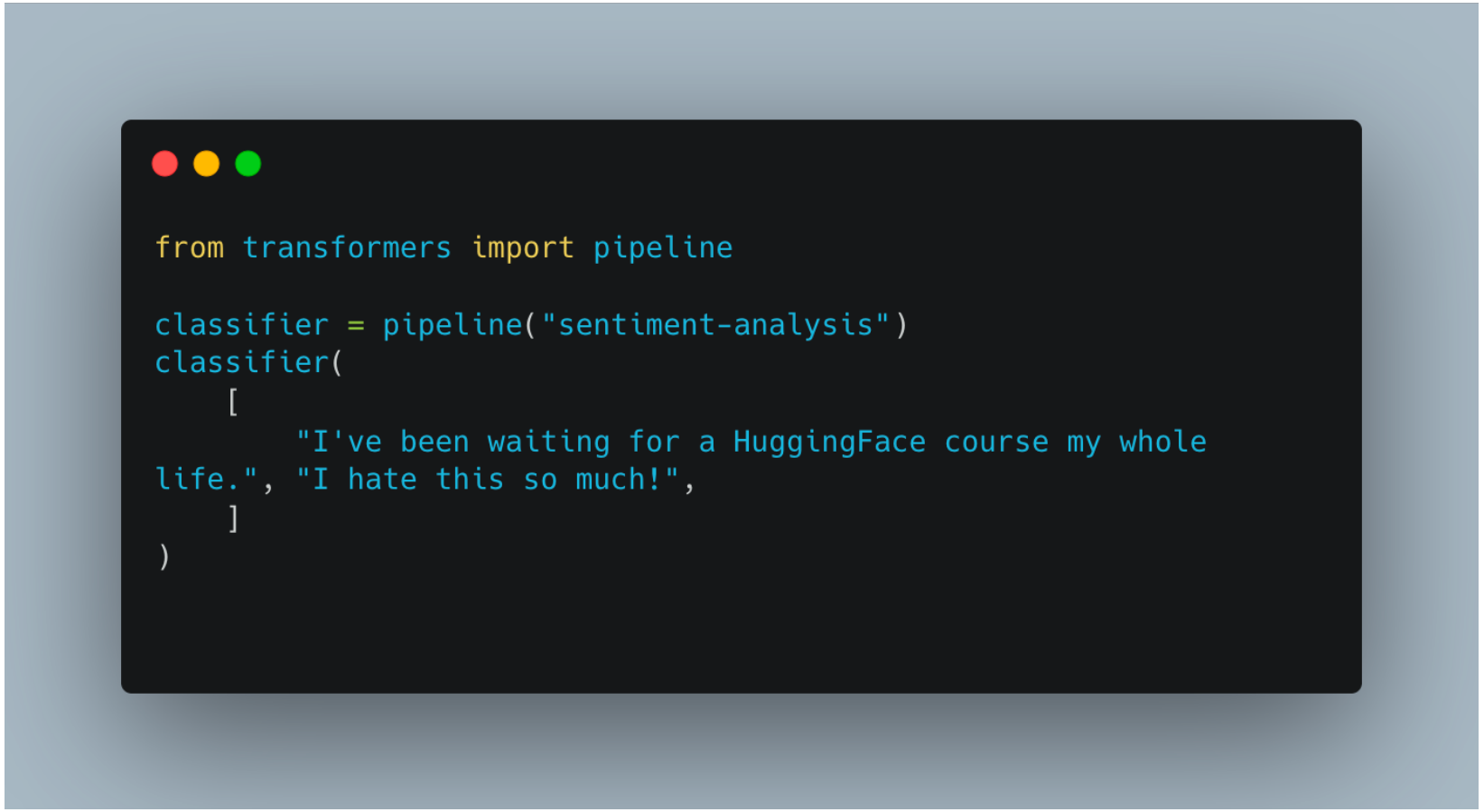
Search to Research: 15 essential things to know about the Eiffel Tower | The Eiffel Tower was built in record time: 2 years, 2 months and 5 days. From late January 1887 to **March 31, 1889**. In addition to the technical … | https://www.toureiffel.paris

Using a search engine to improve factuality



Research to User: Work started on it in January 1887 and it was opened in March 1889.

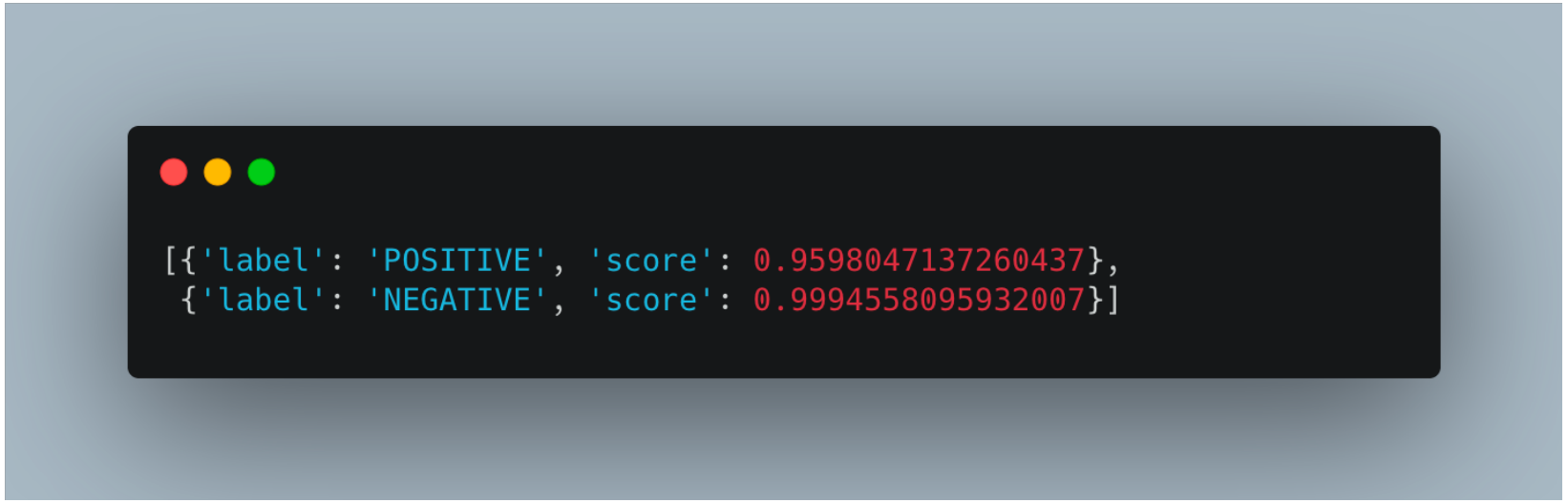
Huggingface 🤗



```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
classifier(
    [
        "I've been waiting for a HuggingFace course my whole
life.", "I hate this so much!",
    ]
)
```

Huggingface 🤗



Huggingface 🤗

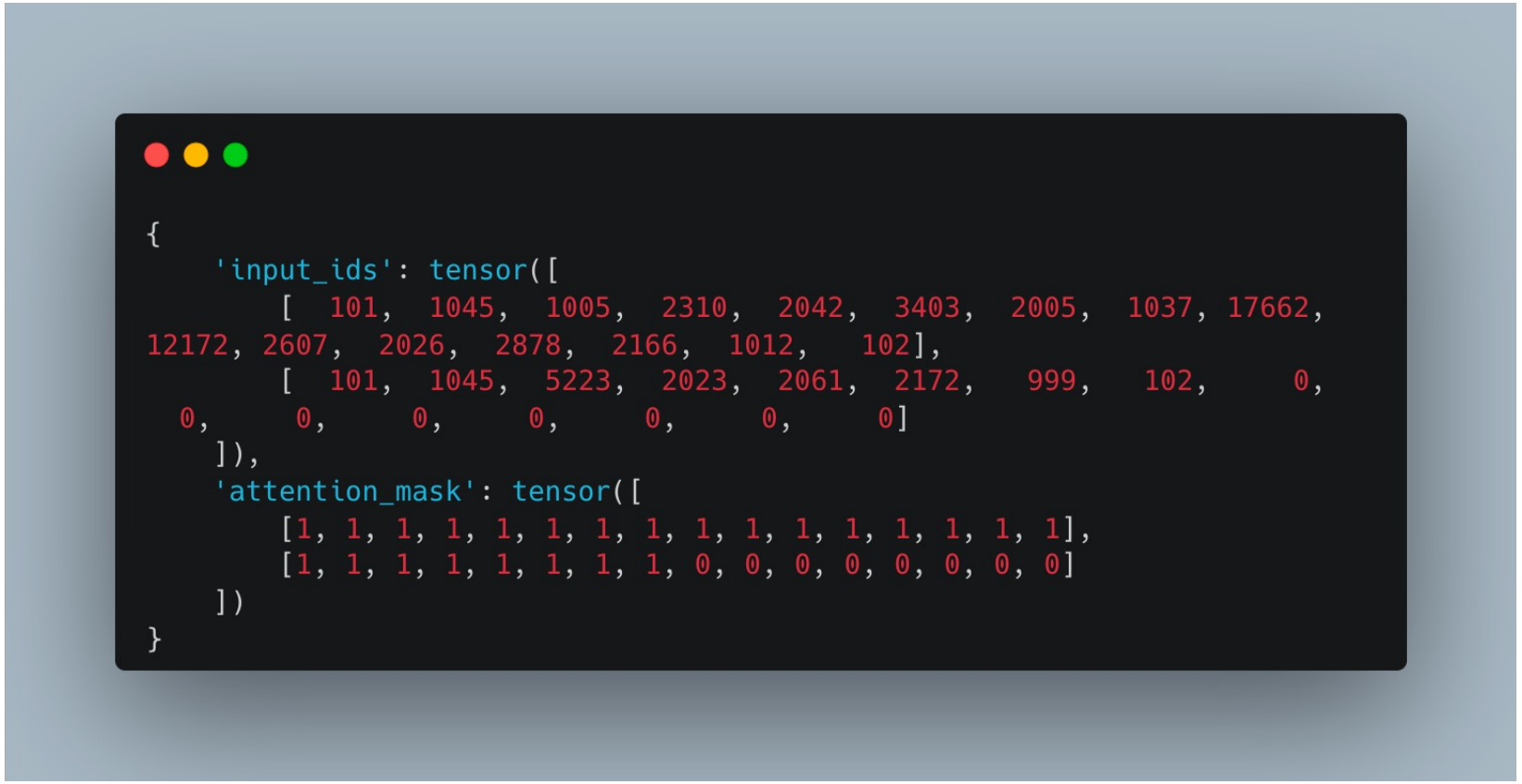
```python
from transformers import AutoTokenizer

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

Huggingface 🤗

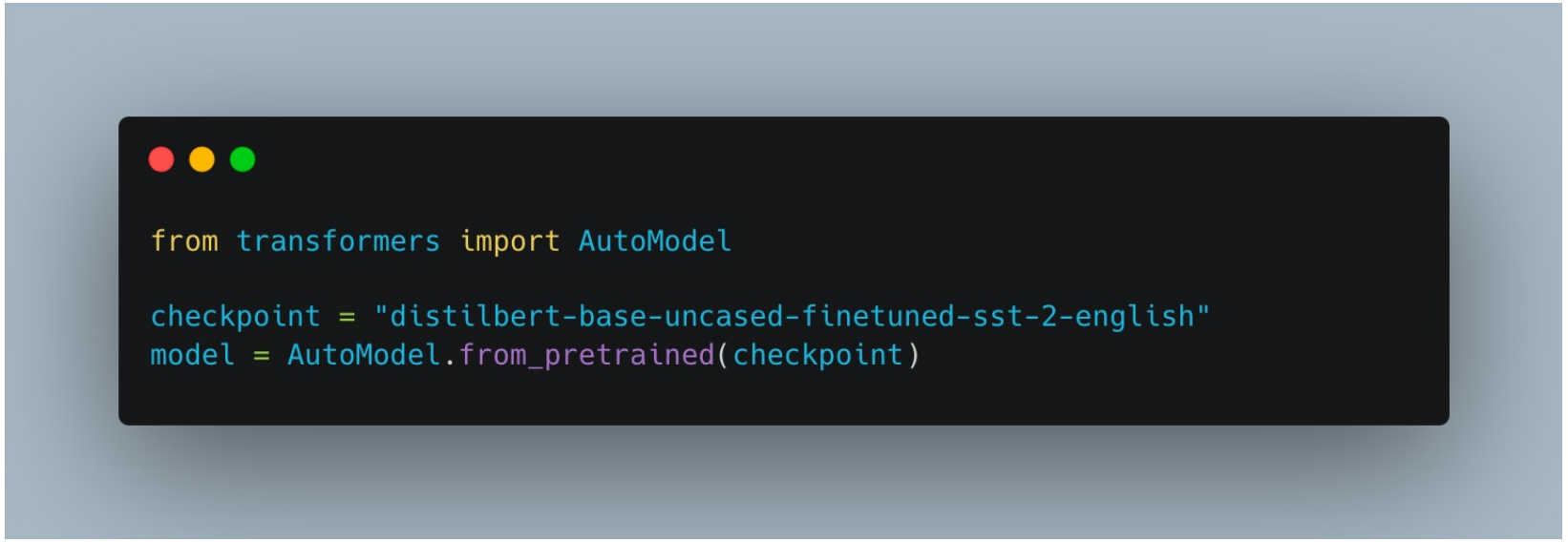
```
raw_inputs = [
    "I've been waiting for a HuggingFace course my whole life.",
    "I hate this so much!",
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True,
return_tensors="pt")
print(inputs)
```

Huggingface 🤗



```
{
    'input_ids': tensor([
        [  101,  1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662,
12172, 2607,  2026,  2878,  2166,  1012,   102],
        [  101,  1045,  5223,  2023,  2061,  2172,   999,   102,     0,
  0,     0,     0,     0,     0,     0,     0]
    ]),
    'attention_mask': tensor([
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
    ])
}
```

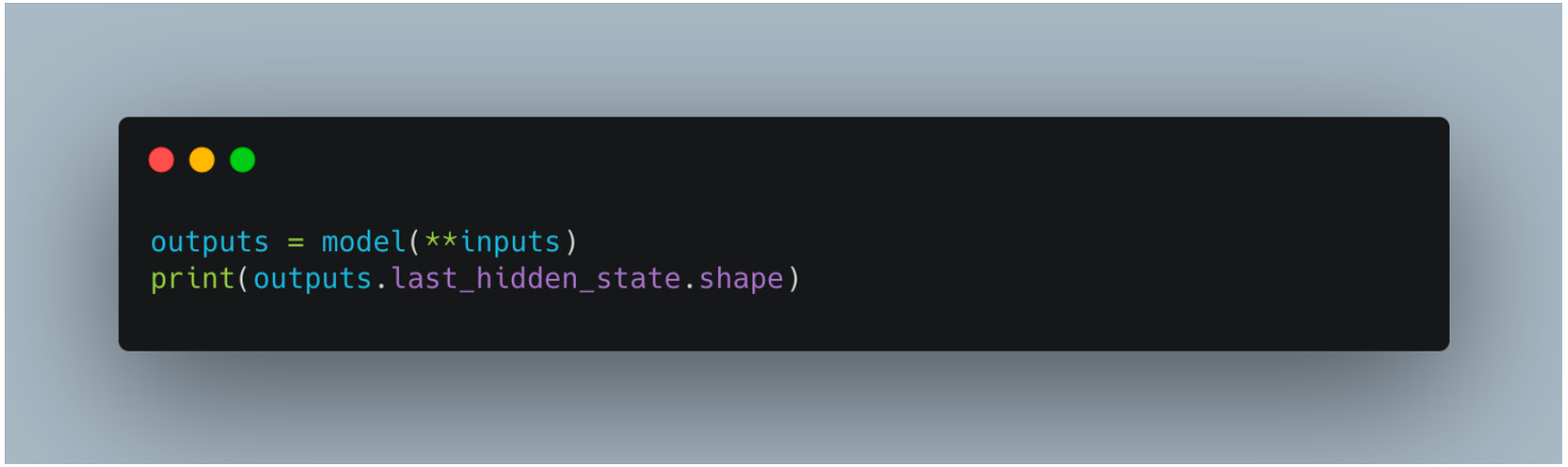
Huggingface 🤗



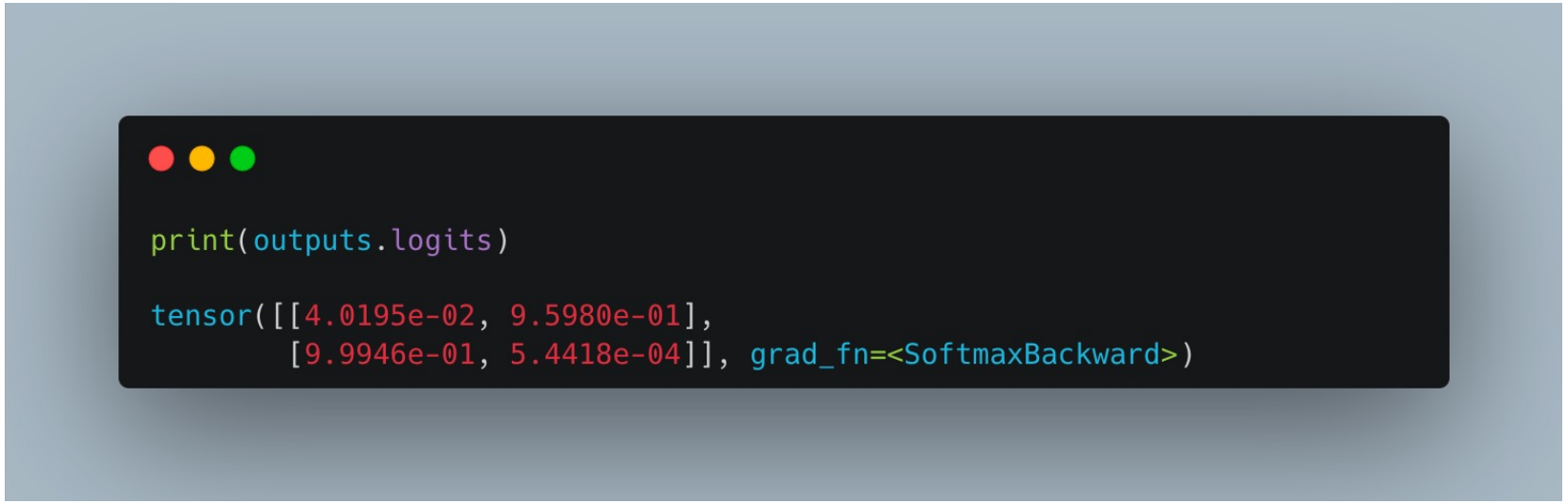
```python
from transformers import AutoModel

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModel.from_pretrained(checkpoint)
```

Huggingface 🤗



Huggingface 🤗



```
print(outputs.logits)

tensor([[4.0195e-02, 9.5980e-01],
        [9.9946e-01, 5.4418e-04]], grad_fn=<SoftmaxBackward>)
```

Huggingface 🤗

```
import torch

predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
print(predictions)

tensor([[4.0195e-02, 9.5980e-01],
        [9.9946e-01, 5.4418e-04]], grad_fn=<SoftmaxBackward>)
```

Huggingface 🤗

```
print(model.config.id2label)

{0: 'NEGATIVE', 1: 'POSITIVE'}
```

Thank you!
Q & A