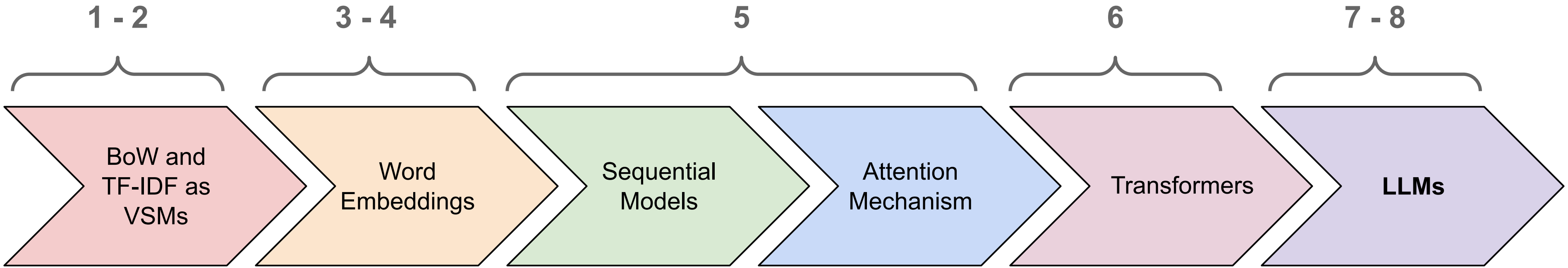
Code.Hub

The first Hub for Developers

# Large Language Models

Thanos Tagaris

# NLP timeline up till today...




- The first direction the research community took was to scale their Transformers as much as they could.
- These large-scale models became really proficient in all sorts of NLP tasks out of the box.
- These extremely large scale Transformers are referred to these days as **Large Language Models (LLMs)**

Code.Hub

# Transformers

- Previously we discussed the transformer architecture.
- Now, we’ll see how this transformer can be trained to produce the LLMs we are familiar with.

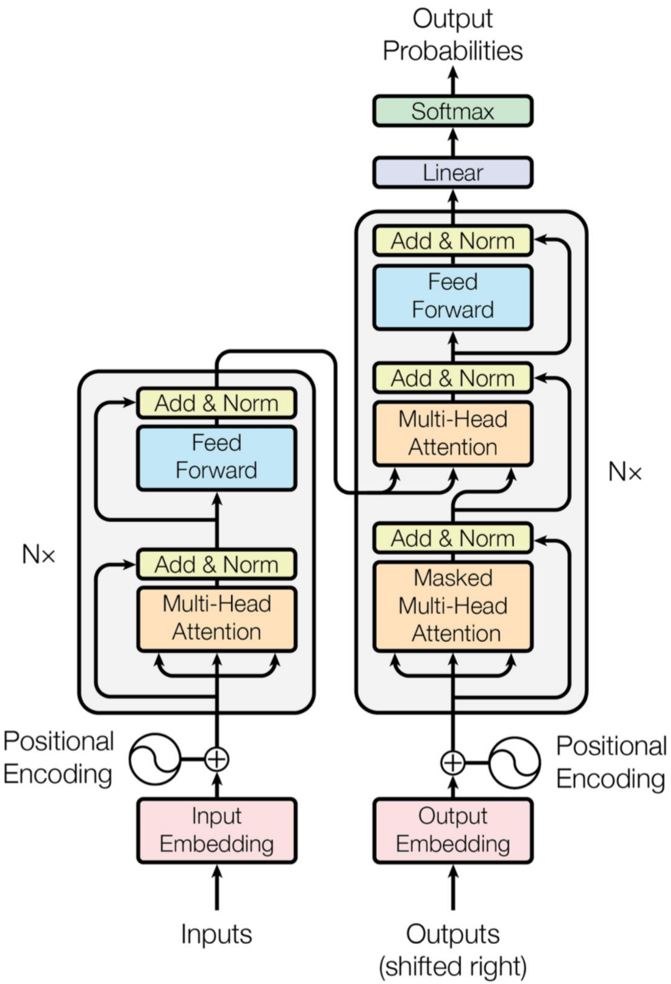



Figure 1: The Transformer - model architecture.

# Model type

- **Encoder-only**
  - good for generating a representation of a document
  - e.g. BERT, RoBERTa, Albert
- **Encoder-decoder**
  - good for usages where output **heavily relies on input** (machine translation, text summarization, etc.)
  - e.g. Flan-T5, BART
- **Decoder-only**
  - good for text generation
  - e.g. LLaMA series, Falcon, GPT series, LaMDA

# Model type

- **Encoder-only**
  - good for generating a representation of a document
  - e.g. BERT, RoBERTa, Albert
- **Encoder-decoder**
  - good for usages where output **heavily relies on input** (machine translation, text summarization, etc.)
  - e.g. Flan-T5, BART

Since text generation applications are the most widely developed the past years, we’ll focus on decoder-only architectures from now on

- **Decoder-only**
  - good for text generation
  - e.g. LLaMA series, Falcon, GPT series, LaMDA

Code.Hub

# Contents



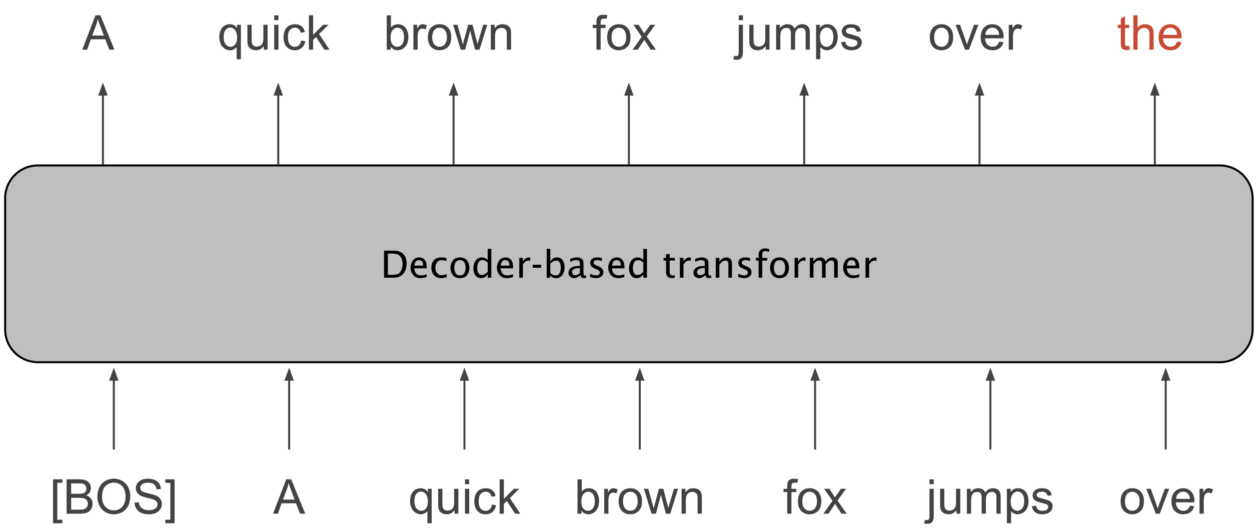
Code.Hub

# Part 1. Training

## Unsupervised Pretraining

# Pre-training

- To pre-train a LLM a self-supervised learning technique is employed called **Language Modelling**
- The task is to simply **predict the next word in a sequence**



# Pre-training

- Note that training is inherently **unsupervised**, so we can use any data we can find!
- LMs are trained on lots of GB of data **scraped from the web**
  - e.g. wikipedia, quora, books.
- After training LMs can generate texts **autoregressively**
  - Start with an input prompt and ask the LM to predict the next token
  - Then append this predicted token to the end of the prompt and ask it to predict the next token
  - Repeat this process until an [EOS] token is generated

# So what does pretraining teach?

The Parthenon is a former temple located in _______, Greece.

Trivia

# So what does pretraining teach?

He sat in the front seat of ___ car.

Syntax

# So what does pretraining teach?

The man grabbed _____ arm in pain.

Corefernce

# So what does pretraining teach?

The shelter was full of stray ___ and dogs.

Lexical semantics

# So what does pretraining teach?

I’m never going to watch another Tarantino movie again, the last movie I watched was _______.

Sentiment

Code.Hub

# So what does pretraining teach?

… several key aspects of a language

That’s why we call it **Language Modelling**!

# Part 1. Training

## From LM to assistant

# Is pretraining enough?

- After pretraining our transformer is a proper Language Model, capable of understanding several key aspects of a language.
- What if we wanted our model to follow our instructions.

| | |
|---|---|
| Prompt | *Explain the moon landing to a 6 year old in a few sentences.* |

# Is pretraining useful?

- In reality this is an illformed question from the beginning.
- Pretraining is extremely useful for producing a very good base model, which we can adapt to perform other tasks (one of which was the previous example)
- What we have after this intensive pretraining is a **foundational model**
  - GPT-3 is such a foundational model
  - It was adapted to follow user instructions to produce ChatGPT

# How to repurpose a LM as an assistant

- LMs are good at producing coherent output, but are **not inherently aligned with user intent**
- What we are more interested in is a **multitask assistant**!

Prompt *Explain the moon landing to a 6 year old in a few sentences.*

Completion GPT-3

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

InstructGPT

```
People went to the moon, and they took pictures of what they saw, and sent them
back to the earth so we could all see them.
```

# How to repurpose a LM as an assistant

- We know that LMs have learned the necessary information to be useful
- The only question is …

… how do we go from this to this?

Prompt *Explain the moon landing to a 6 year old in a few sentences.*

Completion GPT-3

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

InstructGPT

```
People went to the moon, and they took pictures of what they saw, and sent them
back to the earth so we could all see them.
```

# How to repurpose a LM as an assistant

Three options:

A. **Prompting**
   - construct a prompt to extract relevant information from the LM

B. **Supervised fine-tuning**
   - continue the model's training in a supervised manner on pairs of (prompt, desired_output)

C. **Human feedback**
   - use human feedback to evaluate the prompt's outputs and teach it to produce more desirable outputs

# Part 1. Training

## From LM to assistant

A. Prompt Engineering

# Zero/Few shot capabilities of LLMs

- It has been shown that Large LMs exhibit latent **zero** and **few-shot capabilities**
- The latter is also referred to as **in-context learning**
- e.g. imagine writing the following prompt to a LLM

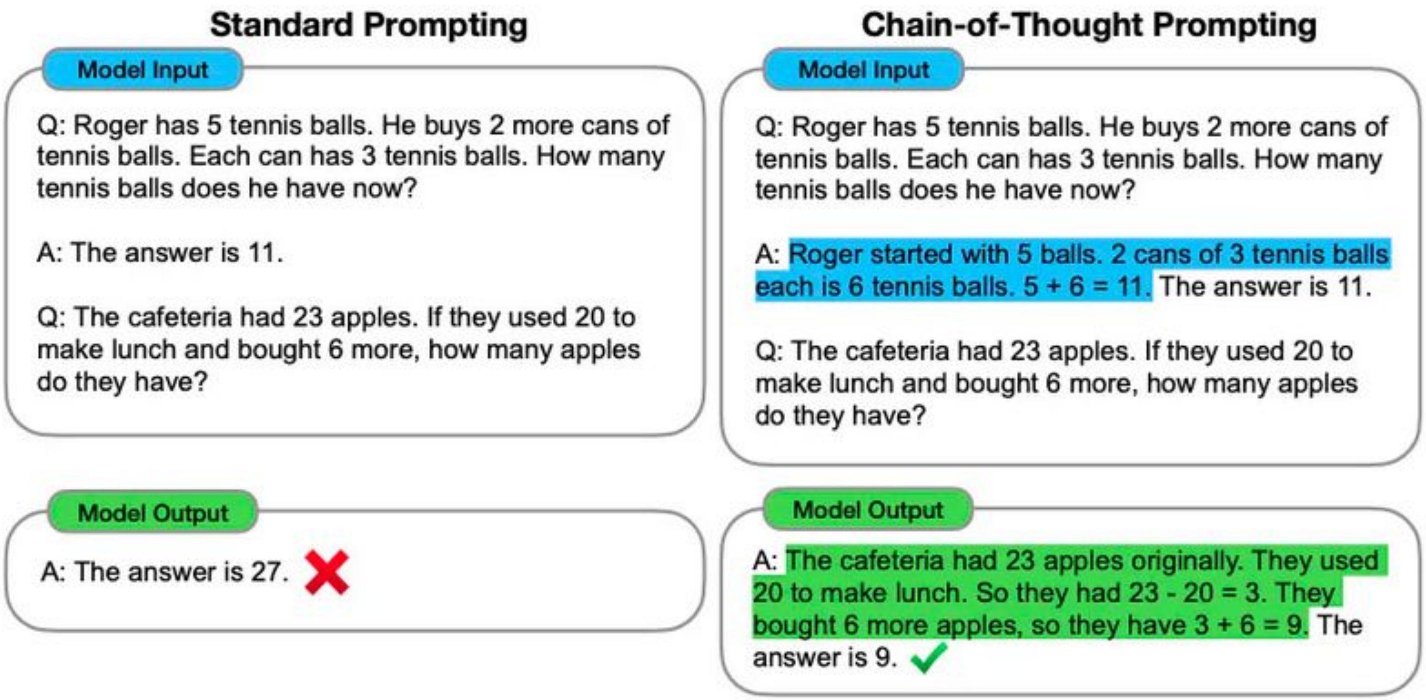
```
thanks -> merci

hello -> bonjour

goodbye ->
```

- The LLM would try to predict the most probable next tokens which in this case might be `"au revoir"`

# Chain-of-Thought



# Part 1. Training

## From LM to assistant

### B. Supervised Fine-Tuning (SFT)

# Supervised Fine-Tuning

- We have a very good model that doesn't do *exactly* what we want?
- What if we go to our bread and butter in Deep Learning: fine-tuning!
- Process:
  1. Collect examples of (prompt, desired output) over multiple tasks
     - can either pay people to write these
     - or look for such datasets in the web (e.g. questions/answers)
  2. Fine-tune the LM on these examples in a supervised manner
  3. Profit

# Issues with SFT

- Very promising approach on paper, simple and straightforward
- But…
  - no right answer on creative tasks (e.g. “write me a story about …”) → hard to train for such tasks
  - not all mistakes are equally bad!

    Let’s say our (x, y) pair is:

    ("Where is the parthenon located?", "The parthenon is located in Athens.")

    Model output 1: "The parthenon is located in Greece".

    Model output 2: "The parthenon is located in Thessaloniki".

    Both outputs would be penalized equally
- Mismatch between training objective and human preferences

Code.Hub

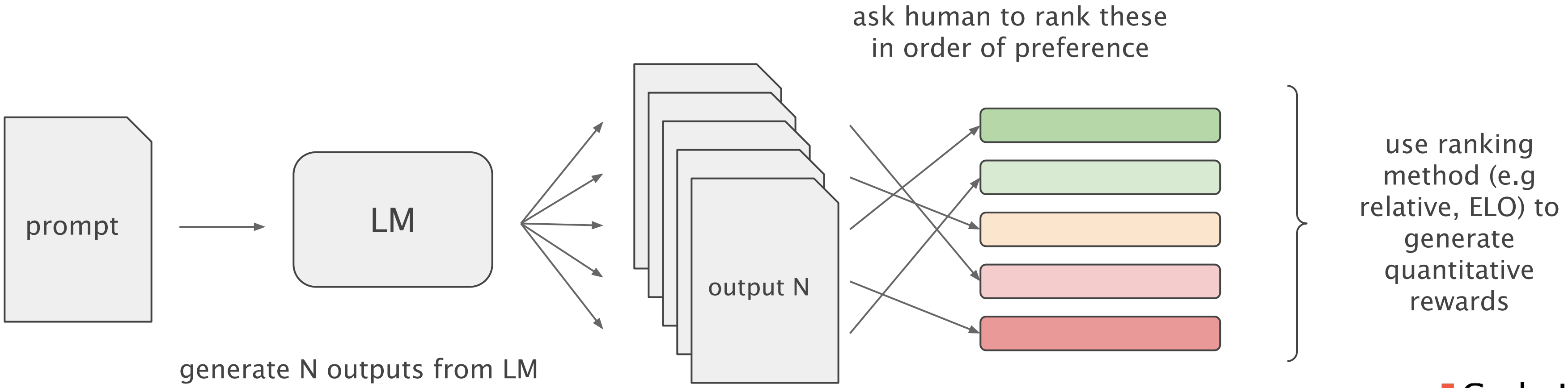
# Part 1. Training

## From LM to assistant

### C. Reinforcement Learning from Human Feedback (RLHF)

# Human Feedback

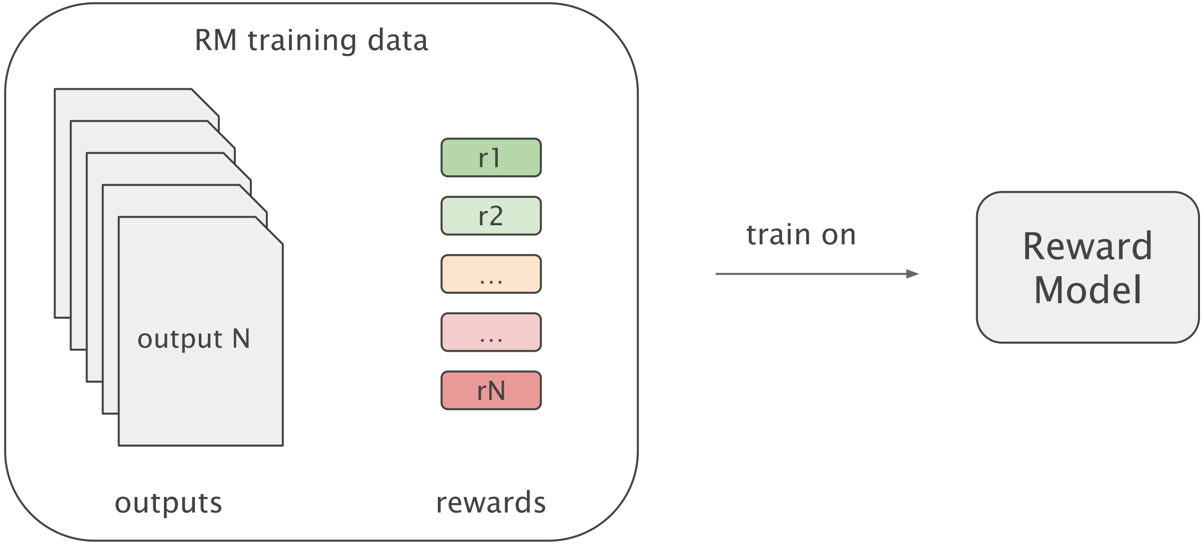
- Let's say we have some way of obtaining qualitative human feedback for a LM's outputs
- We could then optimize the LM on this feedback signal
- How do we get an reliable human feedback signal though?




Code.Hub

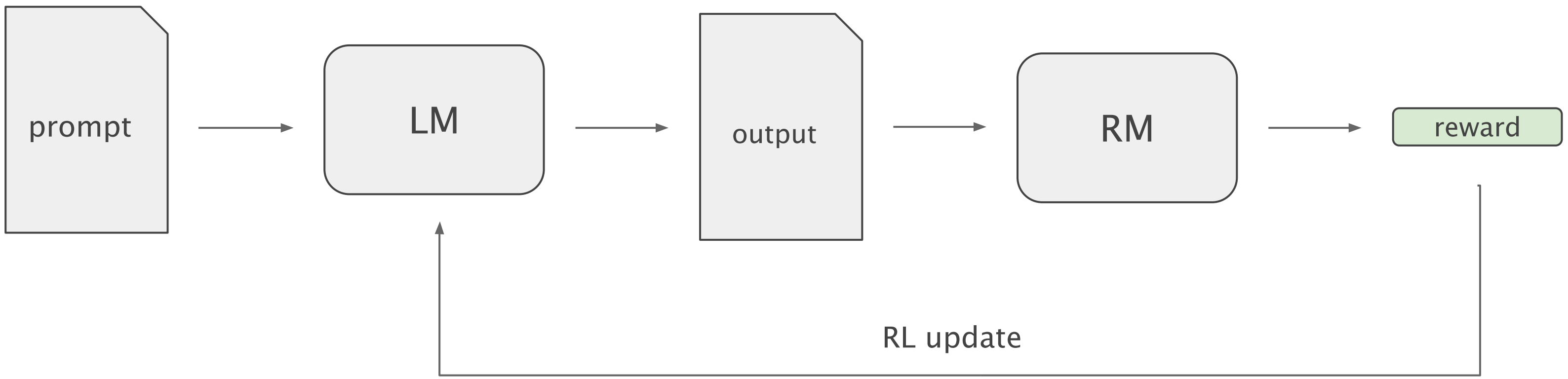
# Reward Model

- Now that we have some (output, reward) pairs, we want to scale things up
- Train another LM (we'll call this the Reward Model) to predict the reward from a given output



# Reinforcement Learning

- The previously trained Reward Model (RM) is capable of predicting how a human would rate the output of a LM
- The only thing that remains is to fine-tune the LM on the human feedback
- We'll use an RL algorithm to train do this (usually PPO)




Code.Hub

# RLHF recap

- RLHF Process:
  1. Collect RM training data
     - get some input prompts that interest us and ask our LM to produce several outputs for each
     - ask humans to rank these outputs in order of preference
     - obtain a quantitative reward from this ranking
  2. Train the RM
     - RM is another LM that accepts a text input and outputs a scalar value
     - train this in a supervised manner on the (output, reward) pairs of step 1
  3. Fine-tune the LM
     - generate outputs from input prompts
     - ask the RM to produce a reward from these outptus
     - use RL to train the LM to maximize reward

# Part 1. Training

## From LM to assistant

## Conclusion

# Which method to choose?

- We saw 3 possible ways of going from LM to assistant; each has their pros and cons
- Which one should we choose?
- These methods are **not** mutually exclusive!
- In fact modern chatbots (e.g. ChatGPT) were trained with all of these
    1. **SSL** for pretraining (i.e. language modelling)
    2. **SFT** afterwards to condition LM more towards instructions
    3. **RLHF** to make its outputs closer to human preferences
- Additionally, during inference we can use advanced **prompt engineering** techniques (e.g. chain of thought) to get even better results

# Foundational models

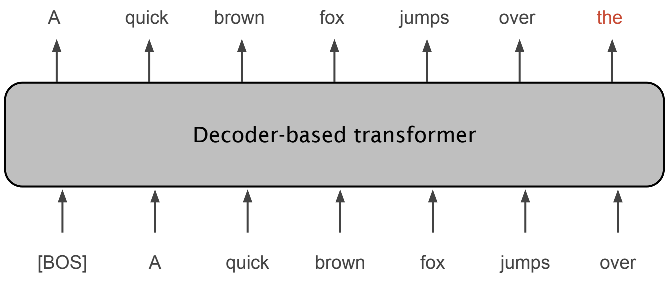
- We saw 3 possible ways of going from LM to assistant; each has their pros and cons
- Which one should we choose?
- These methods are **not** mutually exclusive!
- In fact modern chatbots (e.g. ChatGPT) were trained with all of these
  1. **SSL** for pretraining (i.e. language modelling)
  2. **SFT** afterwards to condition LM more towards instructions
  3. **RLHF** to make its outputs closer to human preferences
- Additionally, during inference we can use advanced **prompt engineering** techniques (e.g. chain of thought) to get even better results

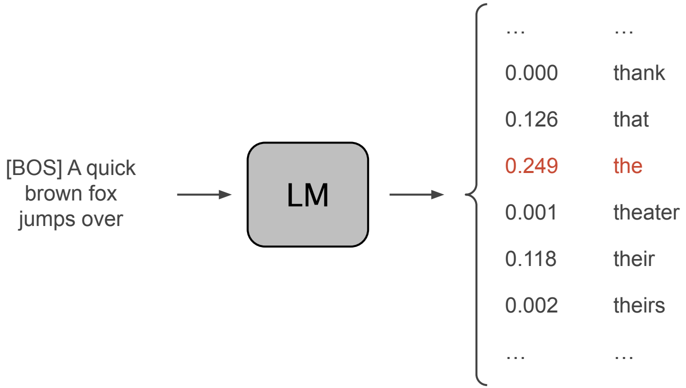
# Part 1. Training

## Generating text

# How do LMs generate text?

- We've talked a lot about how LMs are trained but never actually discussed how they can generate text
- The figure to the right is a bit of an oversimplification
- In fact no LM outputs tokens; instead **they output a probability distribution over all tokens in the vocabulary**
- So how was the token "the" selected out of the mix?

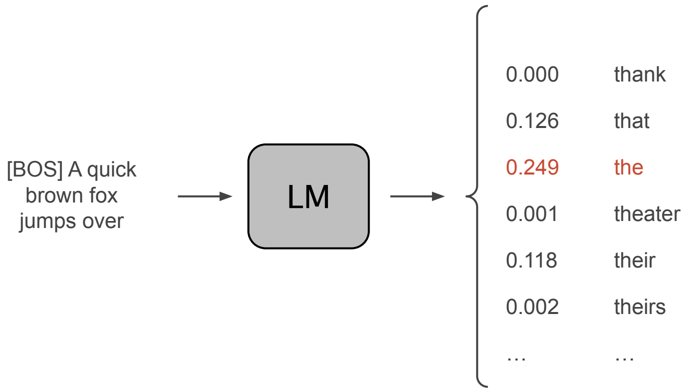


# Decoding

The process of selecting the output tokens

- Does not change trainable parameters
- Can have noticeable effect in quality of outputs
- How the tokens will be selected based on their probabilities is referred to as the **decoding strategy**



# Decoding strategies

- **Greedy search**: select the token with the highest probability
- **Beam search**: keep the most likely N hypotheses; choose the one with the highest overall probability
- **Sampling**: randomly pick a word according to its probability
- **Top-k sampling**: similar to above, but only consider top-k tokens
- **Top-p (nucleus) sampling**: sample from the smallest set of words whose cumulative probability exceeds p
- **Contrastive search**
- … and more