EE-508: Hardware Foundations for Machine Learning

Transformers – Part 3

University of Southern California

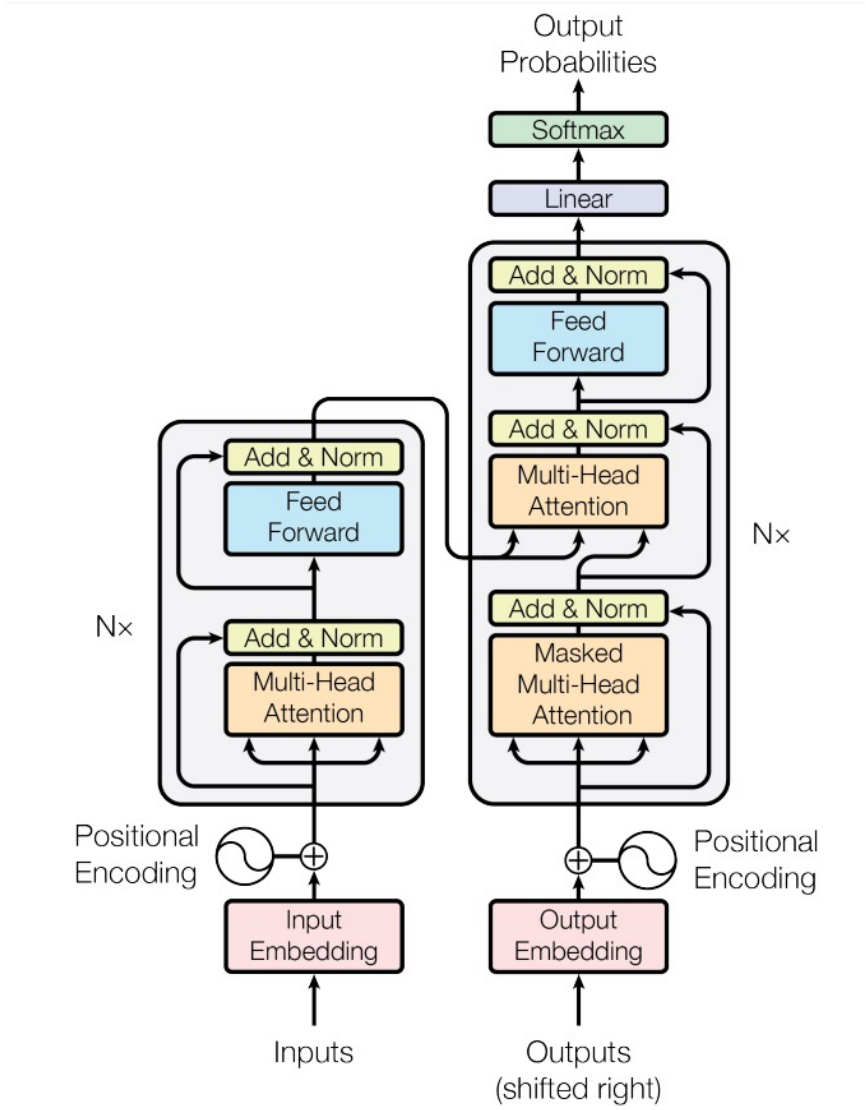
Ming Hsieh Department of Electrical and Computer Engineering

Instructors:
Arash Saifhashemi

Training Transformers

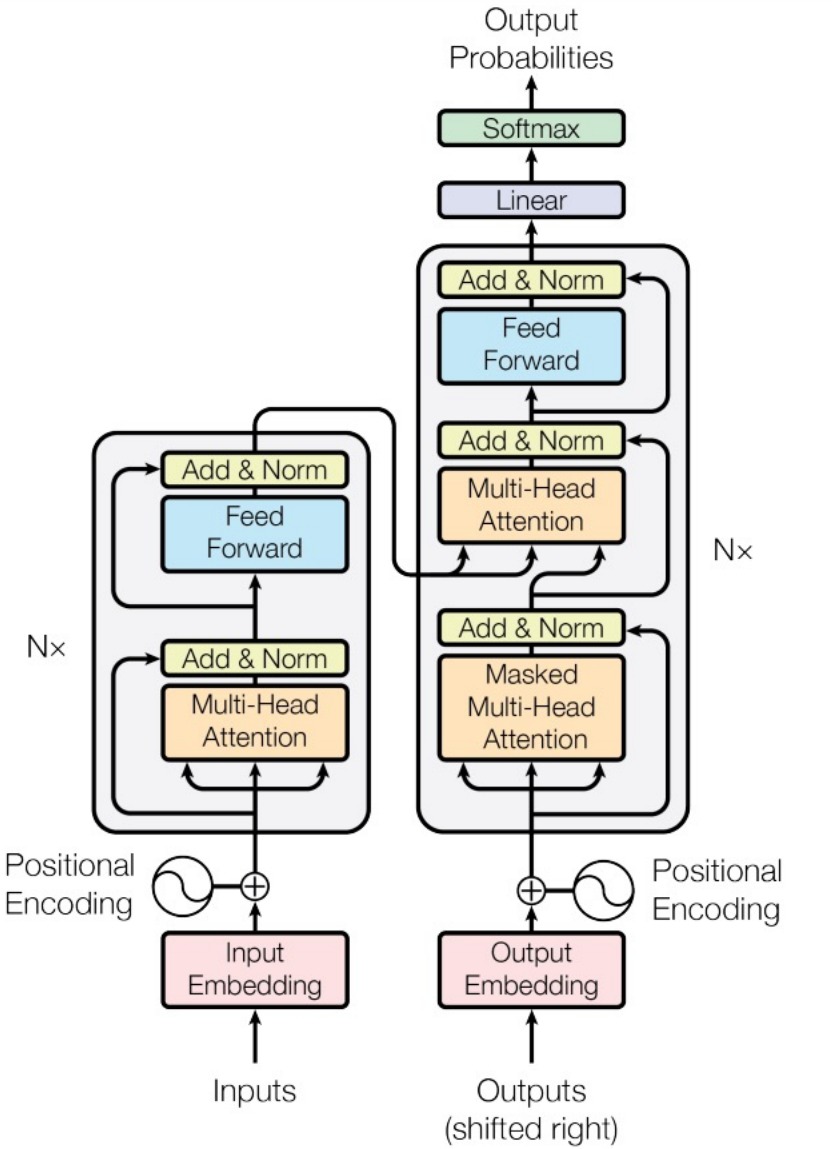
Training Transformers

- Training
  - Pass the input sequence through **encoder**.
  - **Decoder** block generates the output sequence.
  - Use **backpropagation** and a **loss** function like cross-entropy to optimize the model.



Training Transformers

- Training
  - Pass the input sequence through **encoder**.
  - **Decoder** block generates the output sequence.
  - Use **backpropagation** and a **loss** function like cross-entropy to optimize the model.
- Pretraining:
  - Perform unsupervised pretraining on a large dataset.
- Fine-tuning:
  - Perform supervised fine-tuning.

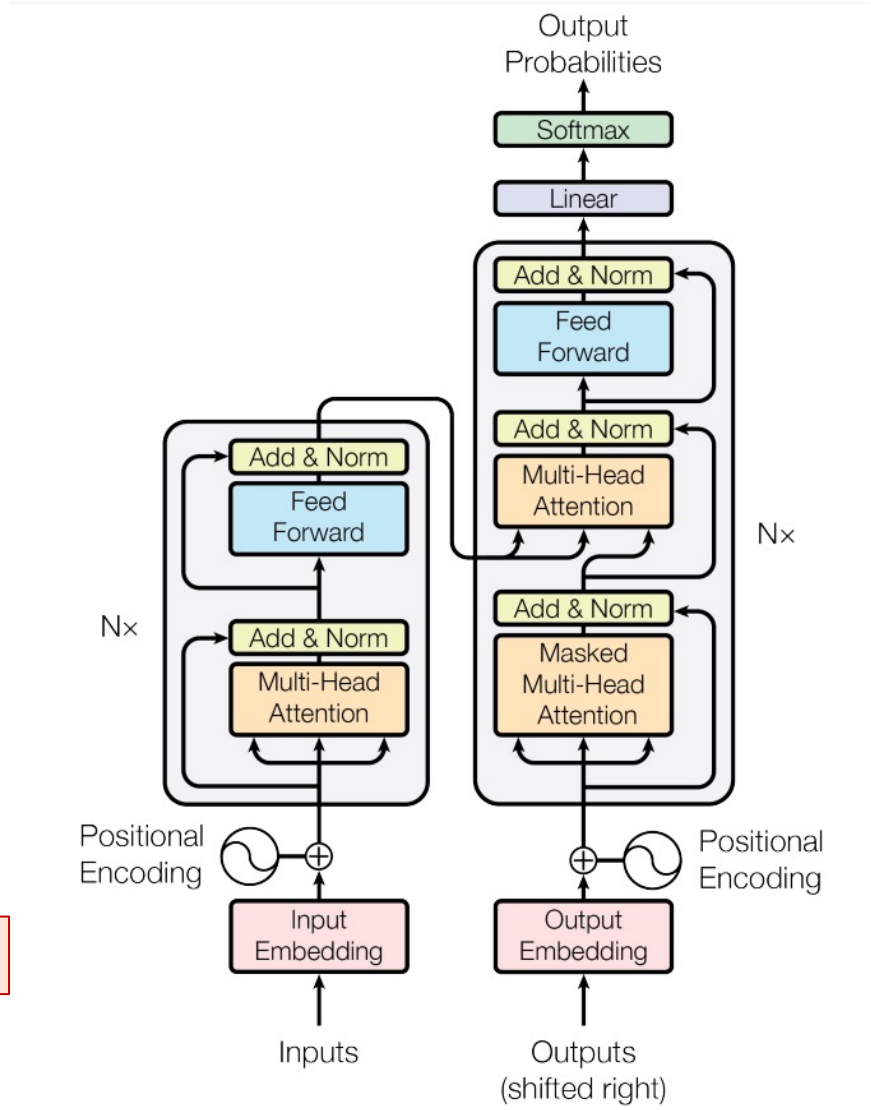


Training Transformers

- Training
  - Pass the input sequence through **encoder**.
  - **Decoder** block generates the output sequence.
  - Use **backpropagation** and a **loss** function like cross-entropy to optimize the model.

The **encoder** can use **all tokens** in the sequence

The **decoder** works **sequentially** and can only pay attention to the **past**



Step 1: Data Collection

- The first step is to gather large amount of data
  - Example: Bilingual dataset.

English Sentence (Source Language)	French Translation (Target Language)
The cat sat on the mat.	Le chat est assis sur le tapis.
She reads a book every night.	Elle lit un livre chaque soir.
The sun sets in the west.	Le soleil se couche à l'ouest.
He loves to play soccer.	Il aime jouer au football.
The sky is clear and blue.	Le ciel est clair et bleu.
They are learning to cook Italian food.	Ils apprennent à cuisiner italien.
The flowers bloom in spring.	Les fleurs fleurissent au printemps.
My brother drives a red car.	Mon frère conduit une voiture rouge.
She sings beautifully.	Elle chante magnifiquement.
We walked along the beach.	Nous avons marché le long de la plage.

Step 2: Tokenization

- Breaking down text into smaller units

Original Sentence: "The cat sat on the mat."

Step 2: Tokenization

- Breaking down text into smaller units

Original Sentence: "The cat sat on the mat."

Word Tokens: ["The", "cat", "sat", "on", "the", "mat"]

Step 2: Tokenization

- Breaking down text into smaller units

Original Sentence: "The cat sat on the mat."

Word Tokens: ["The", "cat", "sat", "on", "the", "mat"]

Subword Tokens: ["Th", "##e", "ca", "##t", "sa", "##t", "on", "th", "##e", "mat"]

Why Use Subword Tokenization

- Motivations:
  - Handling Out-of-Vocabulary (OOV) Words:
    - In word-level tokenization, words not seen during training are treated as unknown
  - Efficient Vocabulary Management:
    - Languages can have a vast number of words.
    - Different forms of a word (like singular, plural, tense variations).
    - Subword tokenization helps in creating a more manageable and smaller vocabulary that can capture this variety more efficiently.

Subword Tokens: ["Th", "##e", "ca", "##t", "sa", "##t", "on", "th", "##e", "mat"]

(Prefix ## means “attach to previous token.”)

Byte Pair Encoding (BPE)

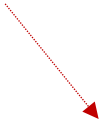
- Start with a large corpus of text
  - Progressively merge the most frequent pairs of bytes or characters.

```
"The cat sat on the mat."
```

Byte Pair Encoding (BPE)

- Start with a large corpus of text
  - Progressively merge the most frequent pairs of bytes or characters.

```
"The cat sat on the mat."
```

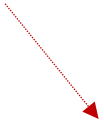


```
["T", "h", "e", " ", "c", "a", "t", " ", "s", "a", "t", "
", "o", "n", " ", "t", "h", "e", " ", "m", "a", "t", "."]
```

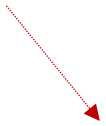
Byte Pair Encoding (BPE)

- Start with a large corpus of text
  - Progressively merge the most frequent pairs of bytes or characters.

```
"The cat sat on the mat."
```



```
["T", "h", "e", " ", "c", "a", "t", " ", "s", "a", "t", "
", "o", "n", " ", "t", "h", "e", " ", "m", "a", "t", "."]
```

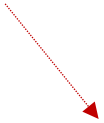


```
["Th”, “##e", "ca”, “##t”, "sa”, “##t", "on", "th”, “##e", "mat"]
```

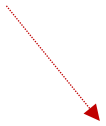
Byte Pair Encoding (BPE)

- Start with a large corpus of text
  - Progressively merge the most frequent pairs of bytes or characters.
  - Used by GPT and BERT.

```
"The cat sat on the mat."
```



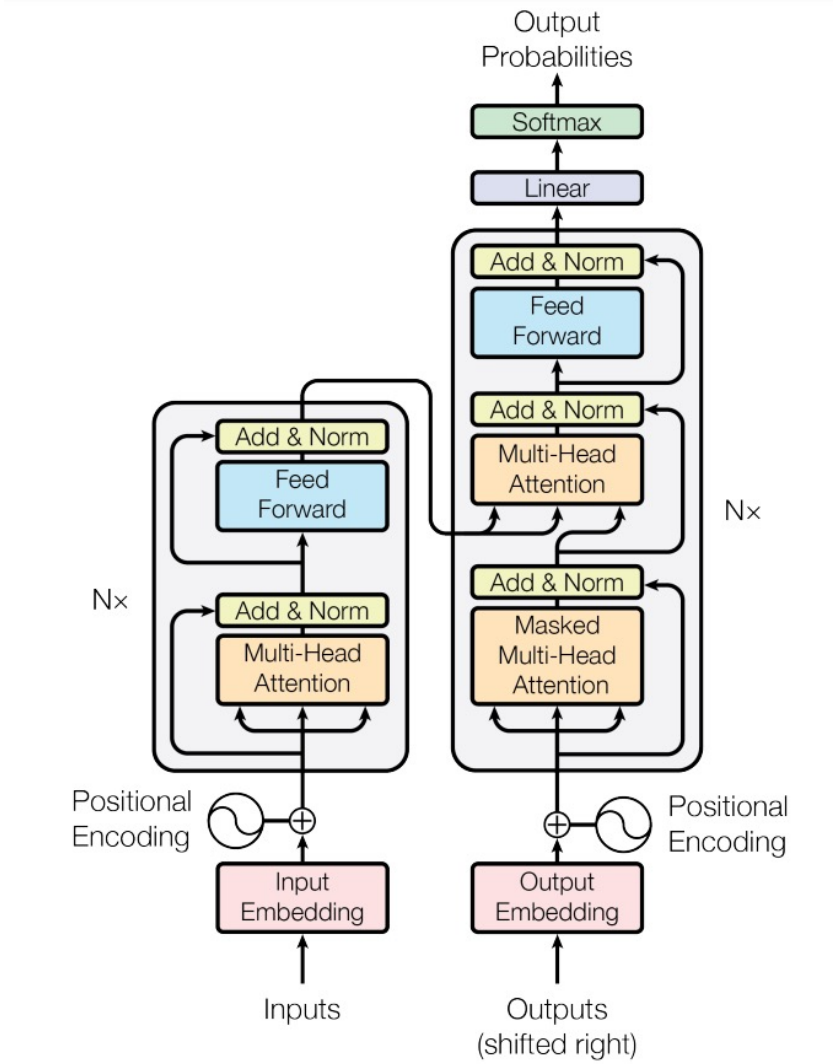
```
["T", "h", "e", " ", "c", "a", "t", " ", "s", "a", "t", "
", "o", "n", " ", "t", "h", "e", " ", "m", "a", "t", "."]
```



```
["Th”, “##e", "ca”, “##t”, "sa”, “##t", "on", "th”, “##e", "mat"]
```

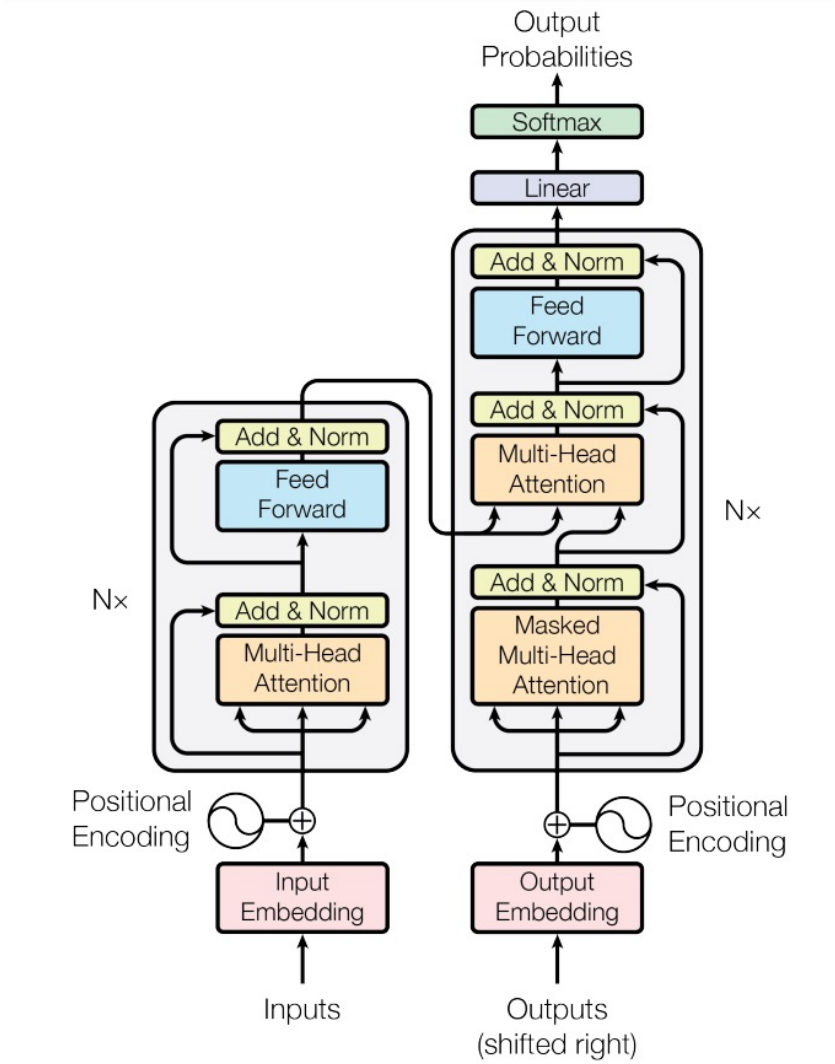
Seq2Seq Training

Step 3: Training



["The", "cat", "sat", "on", "the", "mat"]

Step 3: Training

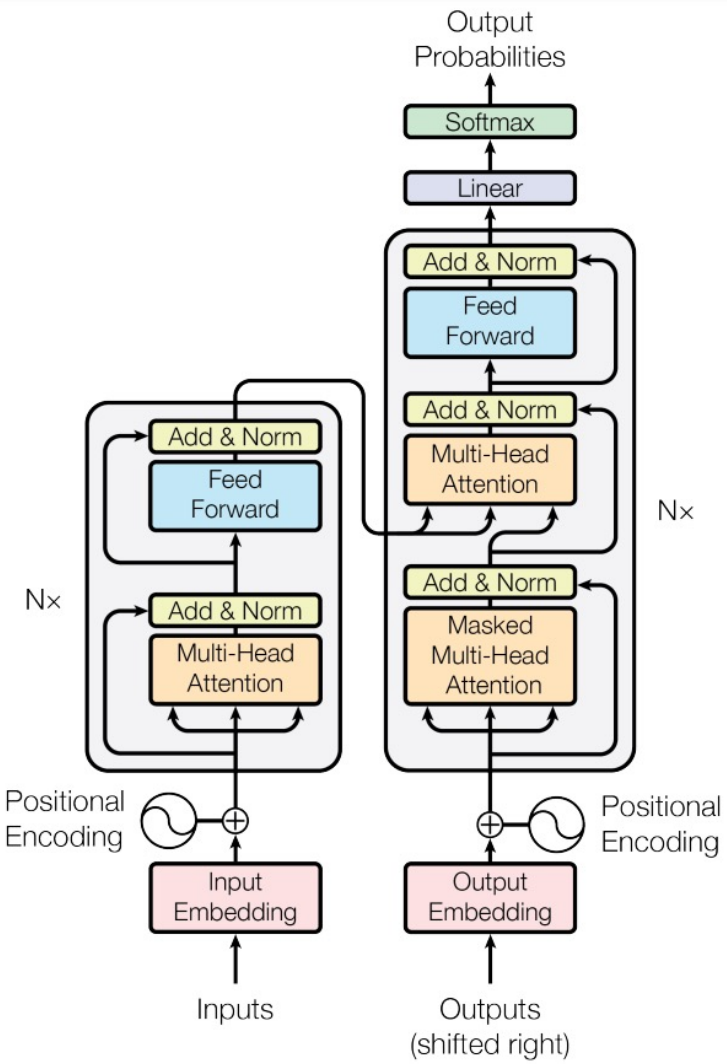


["The", "cat", "sat", "on", "the", "mat"]

[SOS]

Step 3: Training

"Elle"



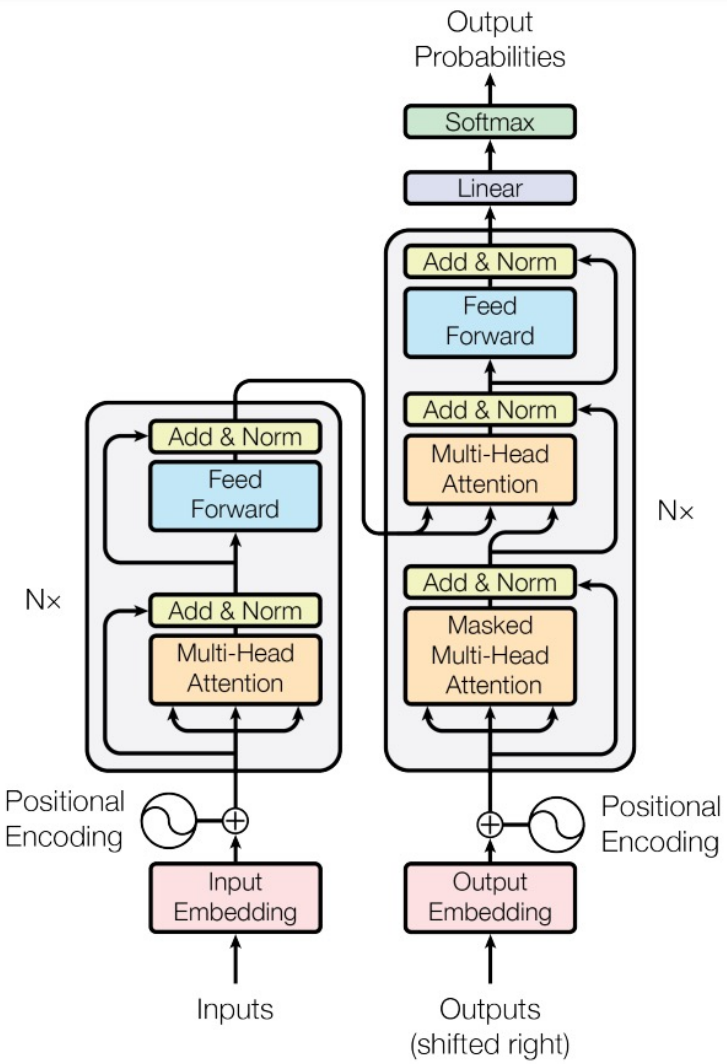
["The", "cat", "sat", "on", "the", "mat"]

[SOS]

Step 3: Training

"Elle"

We should see "Le", but the model generates something else



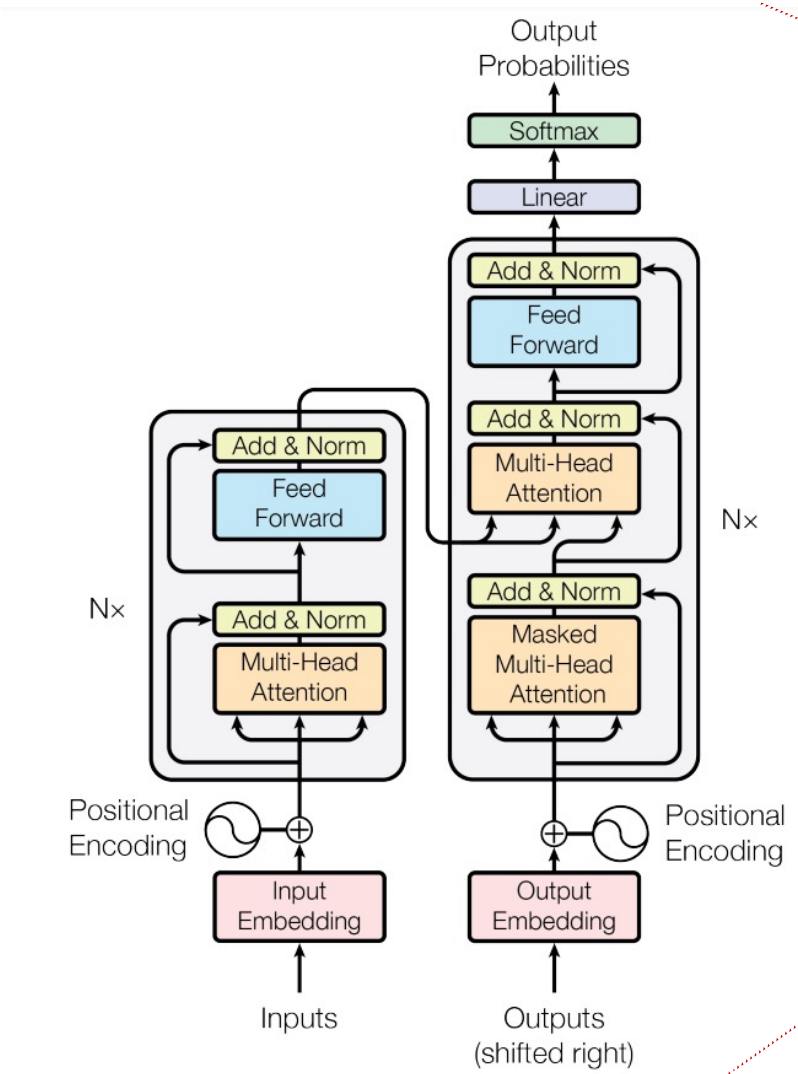
["The", "cat", "sat", "on", "the", "mat"]

[SOS]

Step 3: Training

"Elle"

We should see "Le", but the model generates something else



Teacher Forcing!

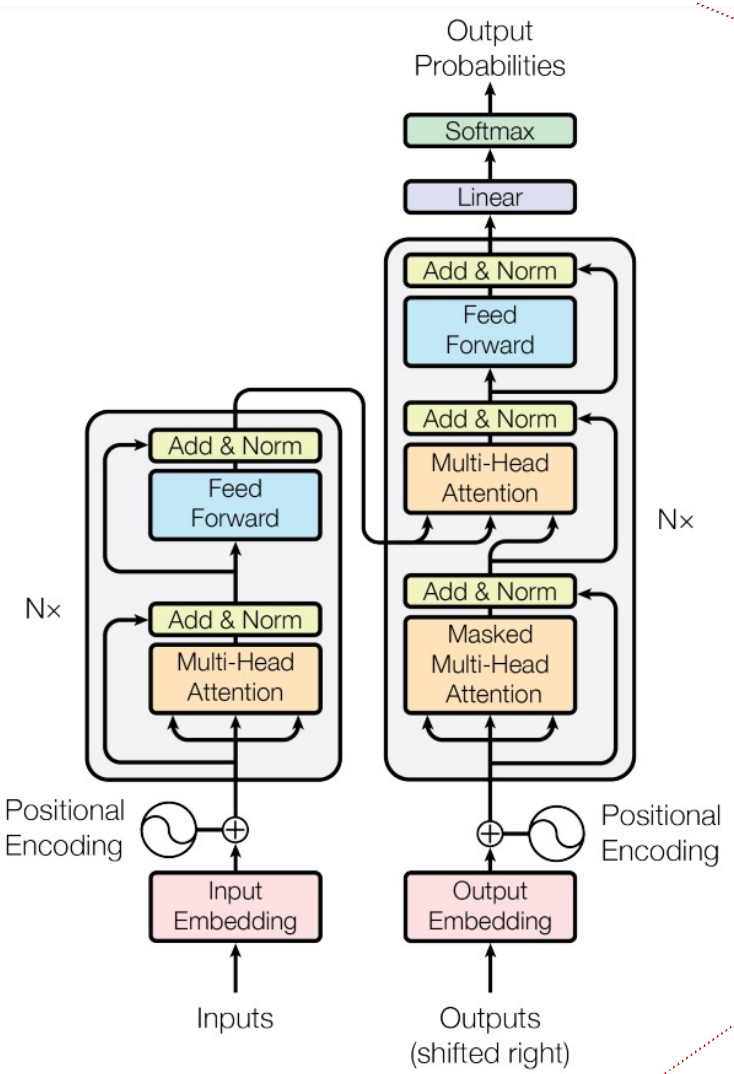
["The", "cat", "sat", "on", "the", "mat"]

[SOS, "Le"]

Step 3: Training

"Elle"

We should see "Le", but the model generates something else



- **Faster Convergence**
- **Accurate Output**

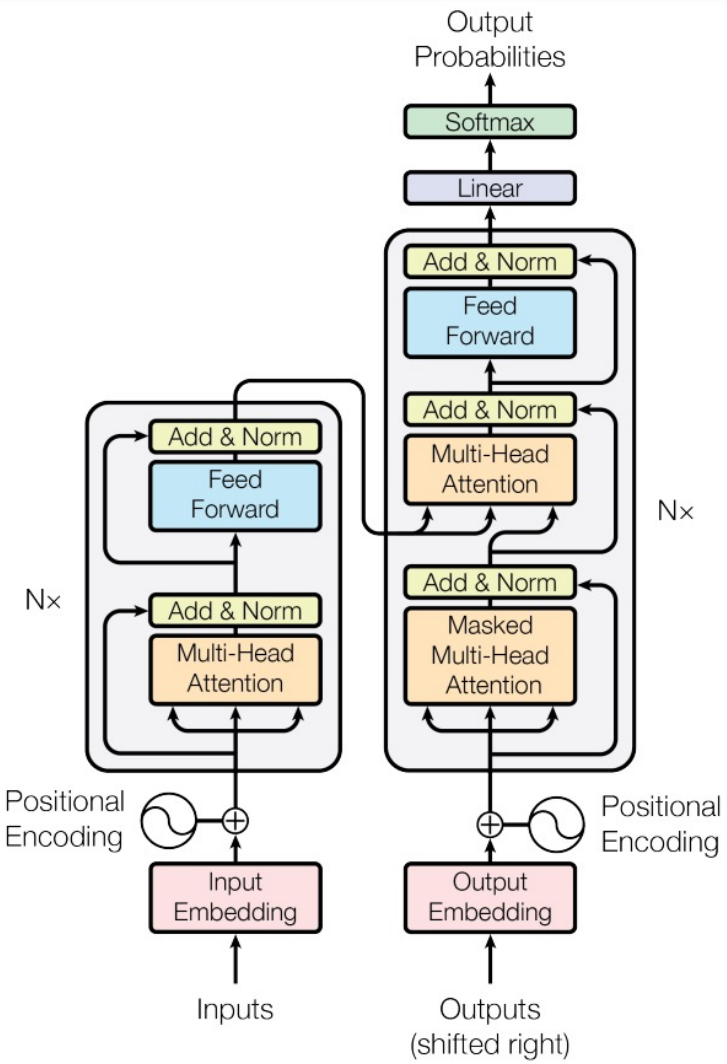
Teacher Forcing!

["The", "cat", "sat", "on", "the", "mat"]

[SOS, "Le"]

Step 3: Training

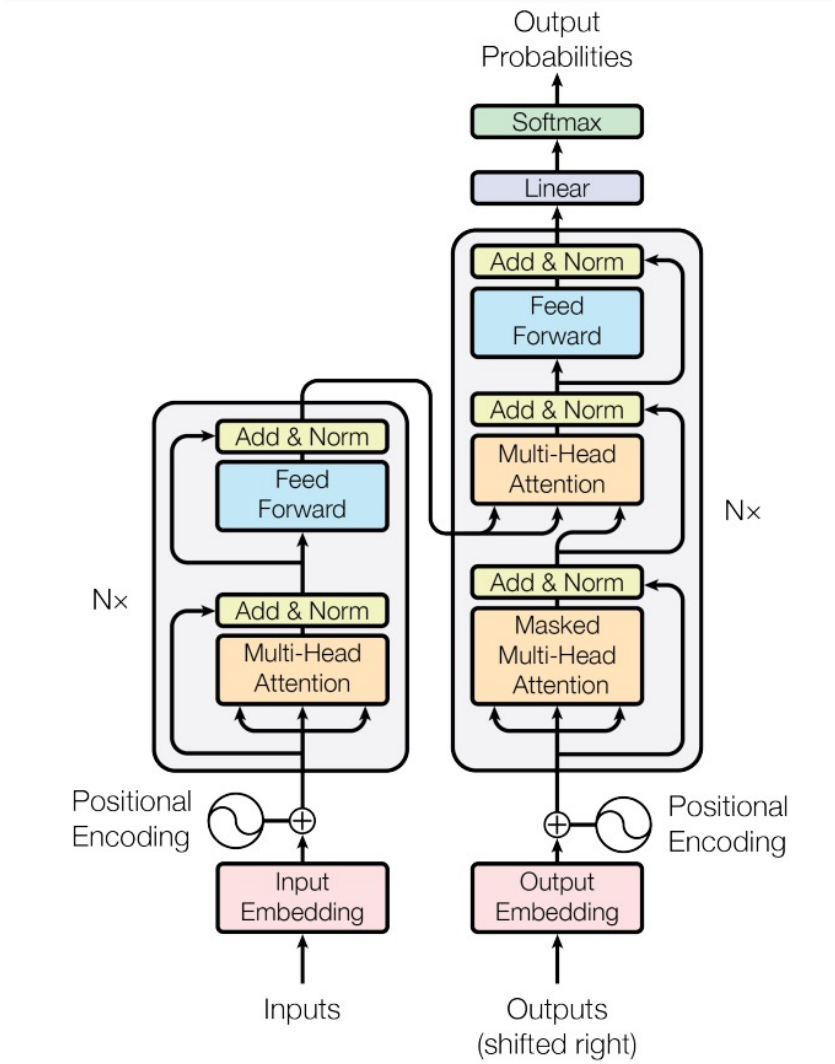
"chat"



["The", "cat", "sat", "on", "the", "mat"]

[SOS, "Le"]

Step 3: Training

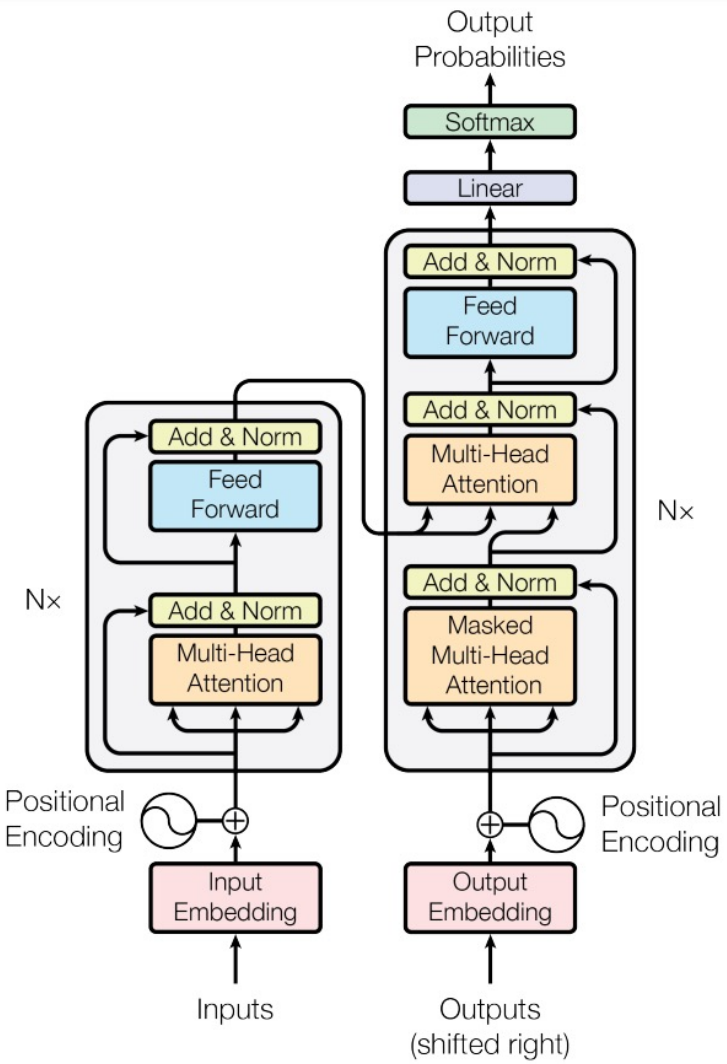


["The", "cat", "sat", "on", "the", "mat"]

[SOS, “Le”, ”chat”]

Step 3: Training

"est"

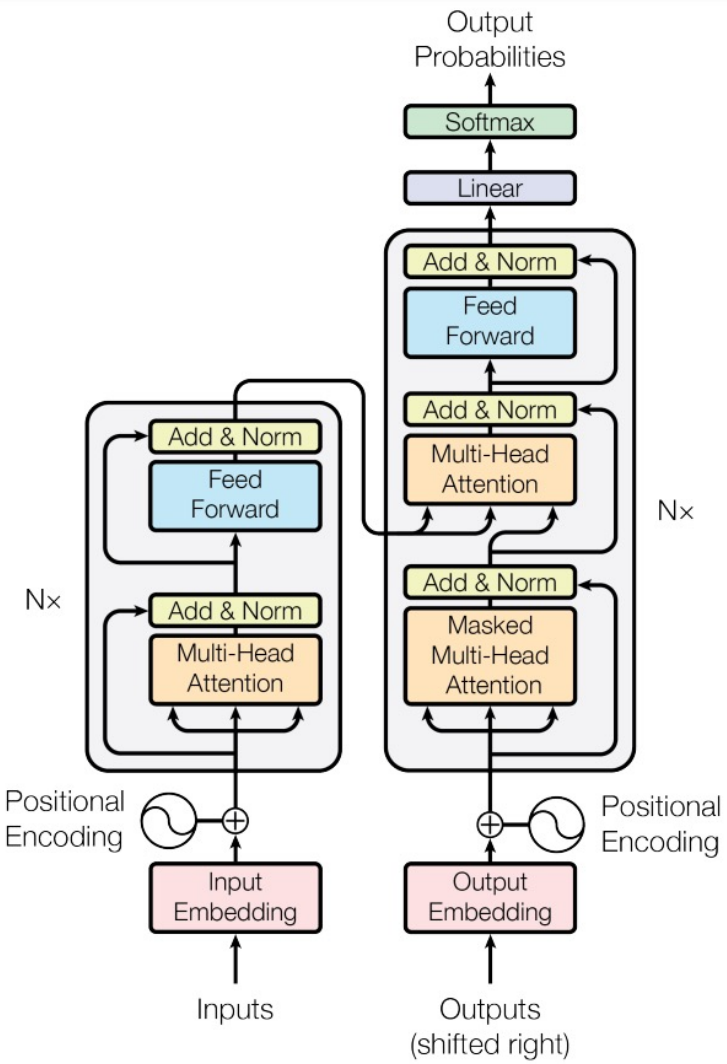


["The", "cat", "sat", "on", "the", "mat"]

[SOS, "Le", "chat"]

Step 3: Training

EOS



…Final sequence

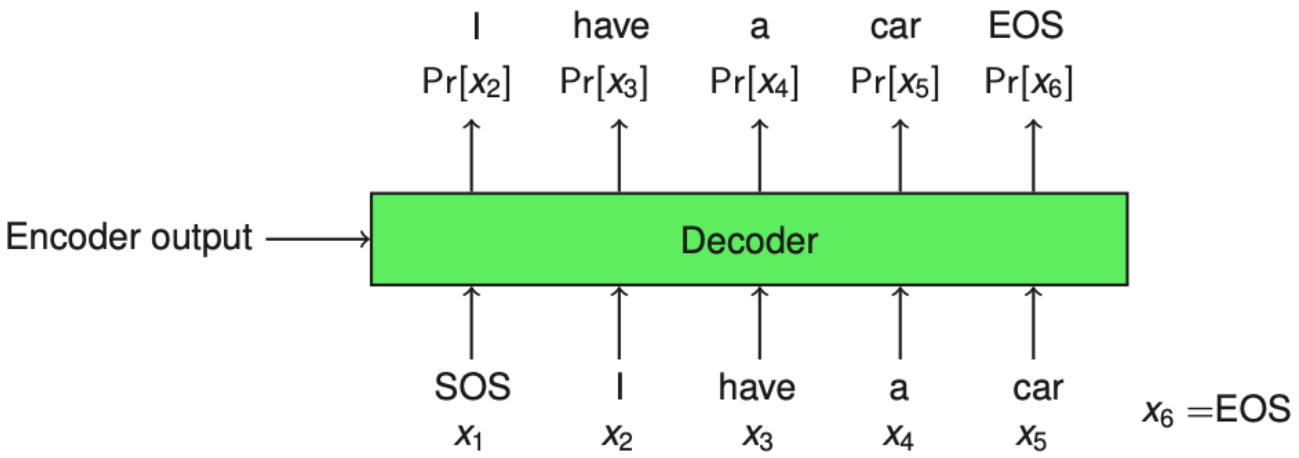
["The", "cat", "sat", "on", "the", "mat"]

[SOS, “Le”, ”chat”, “est”, “assis”, “sur”, “le”, “tapis”]

Masked Attention

Parallelized Predictions During Training

- During **training**
  - The decoder can generate an entire sequence
  - Feed entire target sequence into decoder.

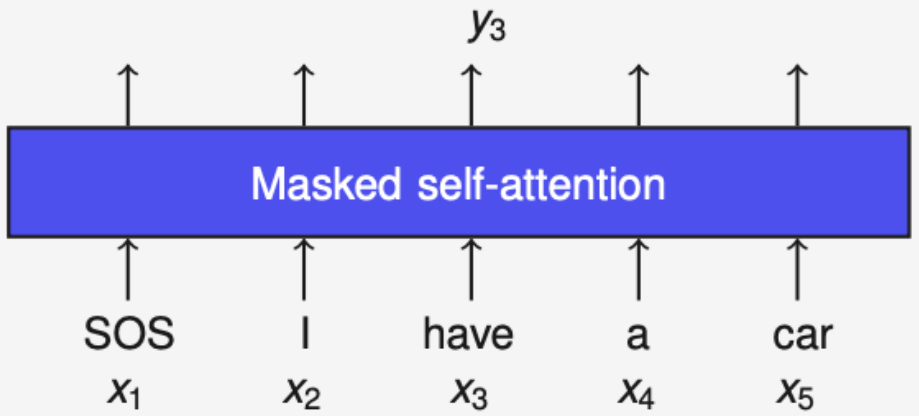


Source: A series of videos on the transformers. Lennart Svensson

Masked Self-Attention

Example: computing y_3

- y_3 should only depend on x_1, x_2, x_3.



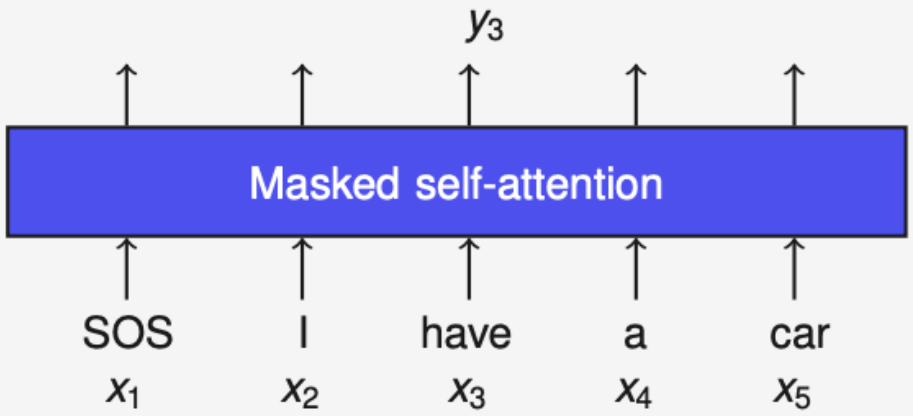
Source: A series of videos on the transformers. Lennart Svensson

Masked Self-Attention

Example: computing y_3

- y_3 should only depend on x_1, x_2, x_3.
- Compute queries, keys and Z:

$$Z_{13} = \frac{k_1^T q_3}{\sqrt{d}}, \ldots, Z_{53} = \frac{k_5^T q_3}{\sqrt{d}}.$$



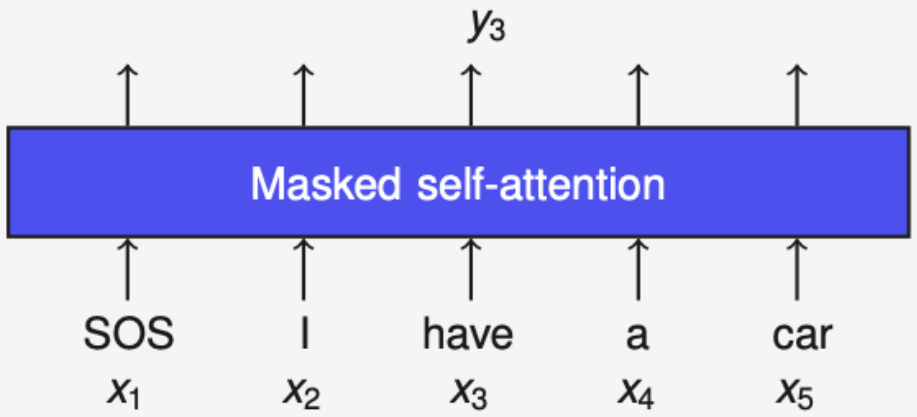
Source: A series of videos on the transformers. Lennart Svensson

Masked Self-Attention

Example: computing y_3

- y_3 should only depend on x_1, x_2, x_3.
- Compute queries, keys and Z:

$$Z_{13} = \frac{k_1^T q_3}{\sqrt{d}}, \ldots, Z_{53} = \frac{k_5^T q_3}{\sqrt{d}}.$$



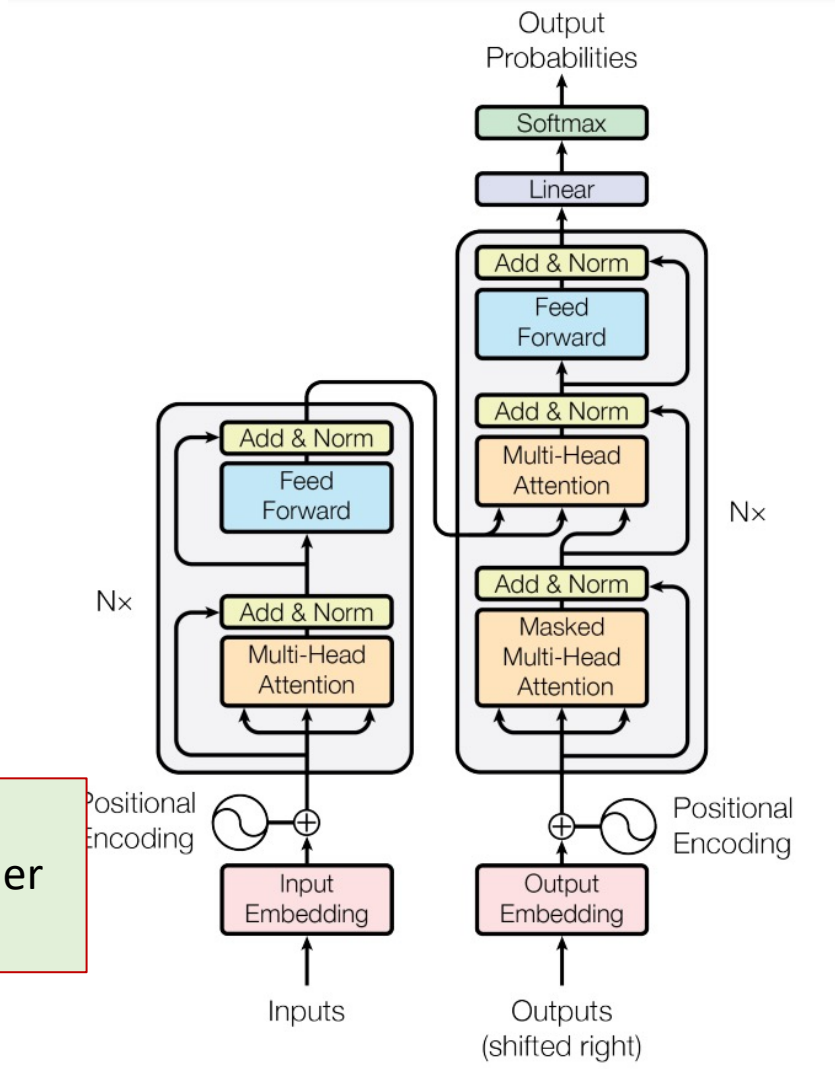
- "Masked" weights:

$$\begin{bmatrix} \tilde{W}_{13} \\ \tilde{W}_{23} \\ \tilde{W}_{33} \\ \tilde{W}_{43} \\ \tilde{W}_{53} \end{bmatrix} = \begin{bmatrix} \exp(Z_{13}) \\ \exp(Z_{23}) \\ \exp(Z_{33}) \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} W_{13} \\ W_{23} \\ W_{33} \\ W_{43} \\ W_{53} \end{bmatrix} = \begin{bmatrix} \text{softmax}(Z_{13}, Z_{23}, Z_{33}) \\ \\ \\ 0 \\ 0 \end{bmatrix} \Rightarrow y_3 = \sum_{i=1}^{5} v_i W_{i3}$$

Source: A series of videos on the transformers. Lennart Svensson

Masked Attention

- Prevents cheating:
  - Masked attention prevents neural networks from attending to future words
- Improves realism:
  - Masked attention can help to improve the realism of generated text by forcing the model to generate text in a sequential manner.
- This allows the model to generate predictions for all tokens in the sequence in one go while respecting the sequential nature of the data.



This parallel processing of the entire sequence, while still respecting the sequential dependency of the tokens, is a key efficiency of the transformer architecture (during training).

Calculating Loss

Loss Function

- **Cross Entropy Loss**:
  - Suppose there are C classes, and the model outputs a probability distribution for a given input:

$$[p_1, p_2, \cdots, p_C]$$

  - True labels: $y_i = 1$ for the correct class and 0 for all other classes.

$$[y_1, y_2, \cdots, y_C]$$

Loss Function

- **Cross Entropy Loss**:
  - Suppose there are C classes, and the model outputs a probability distribution for a given input:

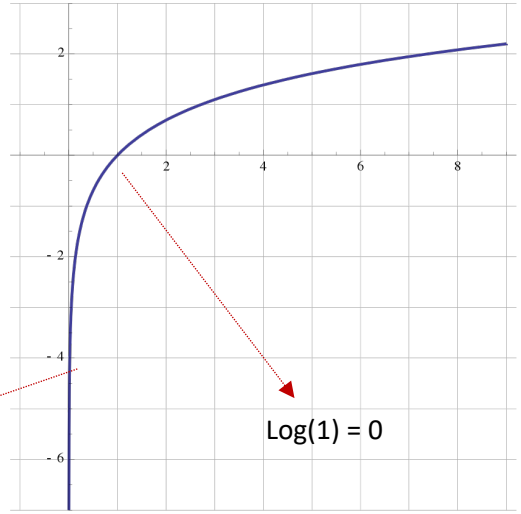
$$[p_1, p_2, \ldots, p_C]$$

  - True labels: $y_i = 1$ for the correct class and 0 for all other classes.

$$[y_1, y_2, \ldots, y_C]$$

  - Cross-Entropy Loss for a Single Example:

$$LOSS = -\sum_{i=1}^{C} y_i \log(p_i)$$



Loss Function

- **Cross Entropy Loss**:
  - Suppose there are C classes, and the model outputs a probability distribution for a given input:

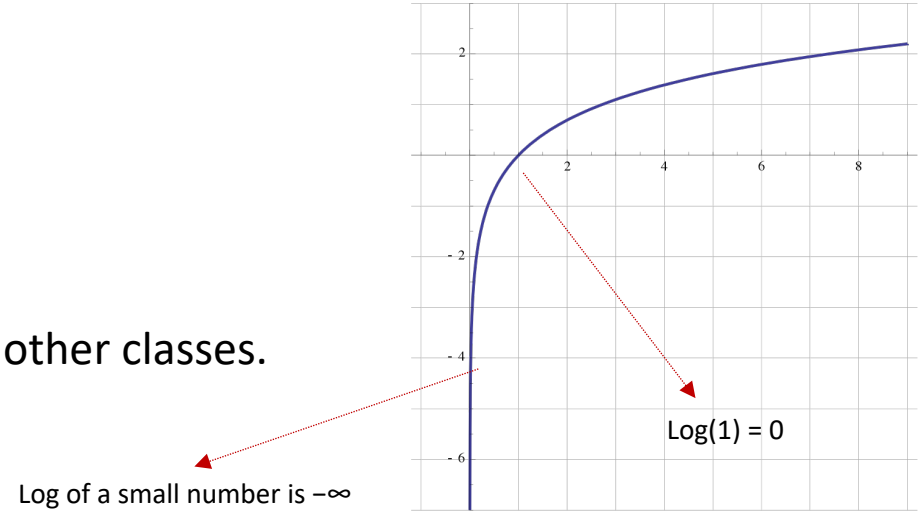
$$[p_1, p_2, \ldots, p_C]$$

  - True labels: $y_i = 1$ for the correct class and 0 for all other classes.

$$[y_1, y_2, \ldots, y_C]$$

  - Cross-Entropy Loss for a Single Example:

$$LOSS = -\sum_{i=1}^{C} y_i \log(p_i)$$



$$y_i \log(p_i)$$

- If $y_i = 1$, the value of p_i should be close to 1 and $\log(p_i)$ should be close to 0
- **High** Confidence, **Wrong** Prediction:
  - For correct y_i, $\log(p_i)$ gets close to $-\infty$
  - High contribution to loss
- Encouraging **Correct High Confidence**:
  - For correct y_i, $\log(p_i)$ gets close to 0
  - Low contribution to loss

Loss Function

- **Cross Entropy Loss**:
  - Suppose there are C classes and the model outputs a probability distribution for a given input:

$$[p_1, p_2, \ldots, p_C]$$

  - True labels: $y_i = 1$ for the correct class and 0 for all other classes.

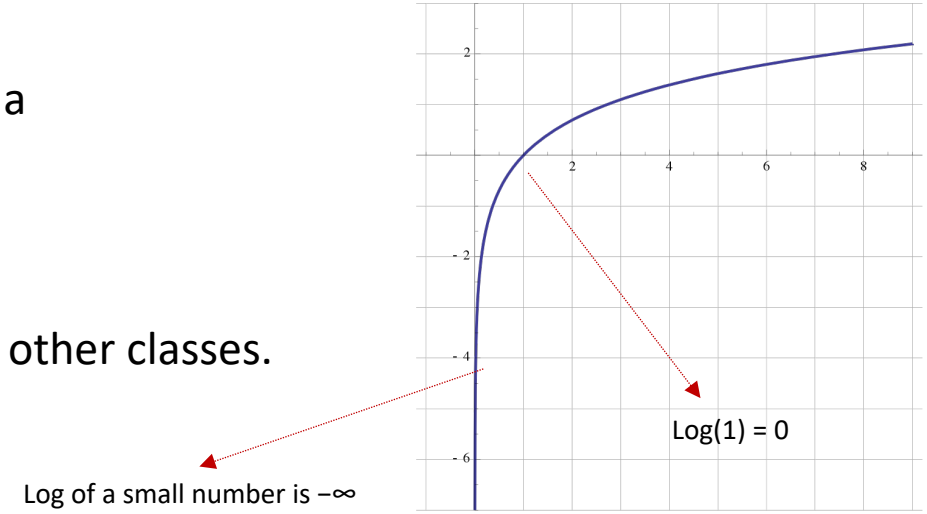
$$[y_1, y_2, \ldots, y_C]$$

  - Cross-Entropy Loss for a Single Example:

$$LOSS = -\sum_{i=1}^{C} y_i \log(p_i)$$

  - Batch Loss Calculation: Add up the loss for each example in the batch

$$-\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{C} y_i \log(p_i)$$



$$y_i \log(p_i)$$

- If $y_i = 1$, the value of p_i should be close to 1 and $\log(p_i)$ should be close to 0
- **High** Confidence, **Wrong** Prediction:
  - For correct y_i, $\log(p_i)$ gets close to $-\infty$
  - High contribution to loss
- Encouraging **Correct High Confidence**:
  - For correct y_i, $\log(p_i)$ gets close to 0
  - Low contribution to loss

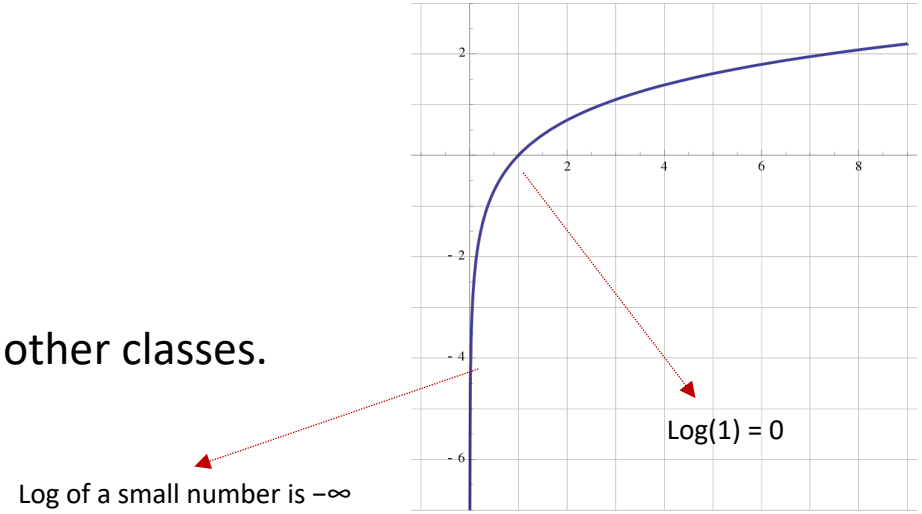
Loss Function

- **Cross Entropy Loss**:
  - Suppose there are C classes, and the model outputs a probability distribution for a given input:

$$[p_1, p_2, \ldots, p_C]$$

  - True labels: $y_i = 1$ for the correct class and 0 for all other classes.

$$[y_1, y_2, \ldots, y_C]$$



  - Cross-Entropy Loss for a Single Example:

$$LOSS = -\sum_{i=1}^{C} y_i \log(p_i)$$

  - Batch Loss Calculation: Add up the loss for each example in the batch

$$-\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{C} y_i \log(p_i)$$

$$y_i \log(p_i)$$

- If $y_i = 1$, the value of p_i should be close to 1 and $\log(p_i)$ should be close to 0
- **High** Confidence, **Wrong** Prediction:
  - For correct y_i, $\log(p_i)$ gets close to $-\infty$
  - High contribution to loss
- Encouraging **Correct High Confidence**:
  - For correct y_i, $\log(p_i)$ gets close to 0
  - Low contribution to loss

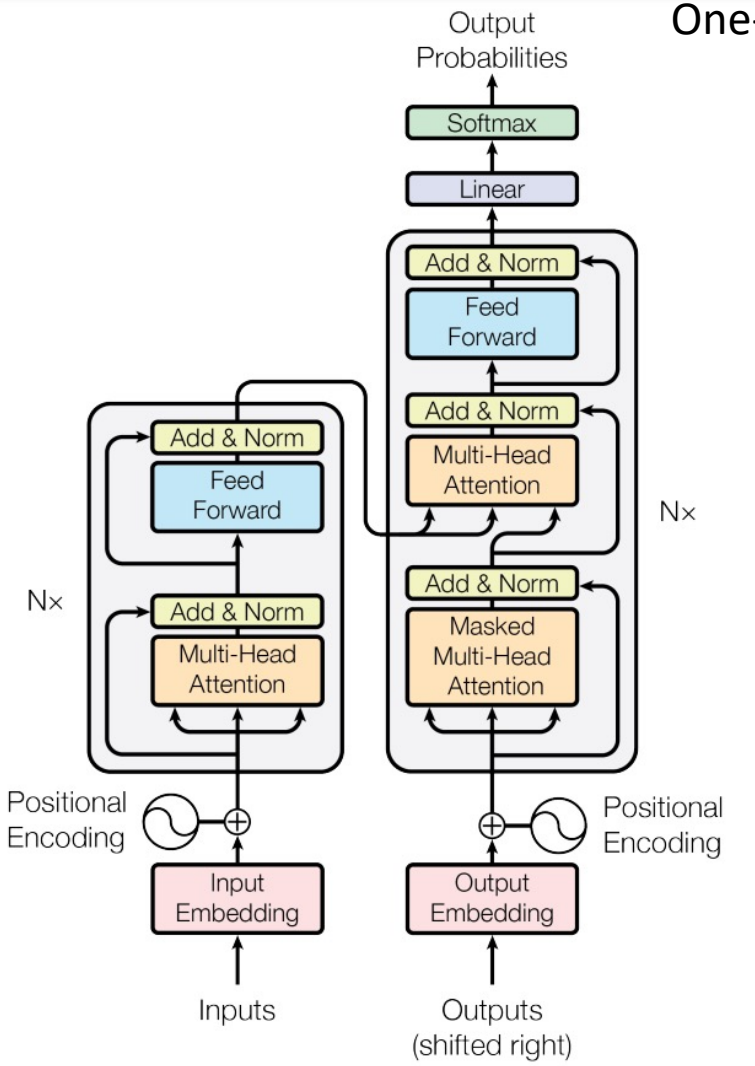
Adjusts parameters not just to predict the correct class, but to do so with high probability

Loss Calculation

$$\text{Loss}_j = \text{CrossEntropy}(\mathbf{y}_j, \mathbf{p}_j)$$

One-Hot Encoding
Of tokens

$$LOSS = -\sum_{i=1}^{C} y_i \log(p_i)$$



Aggregating Loss Over the Sentence

$$-\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{C} y_i \log(p_i)$$

Used for back-propagation

- i: token index in the vocabulary
- j: time step or position in the sequence

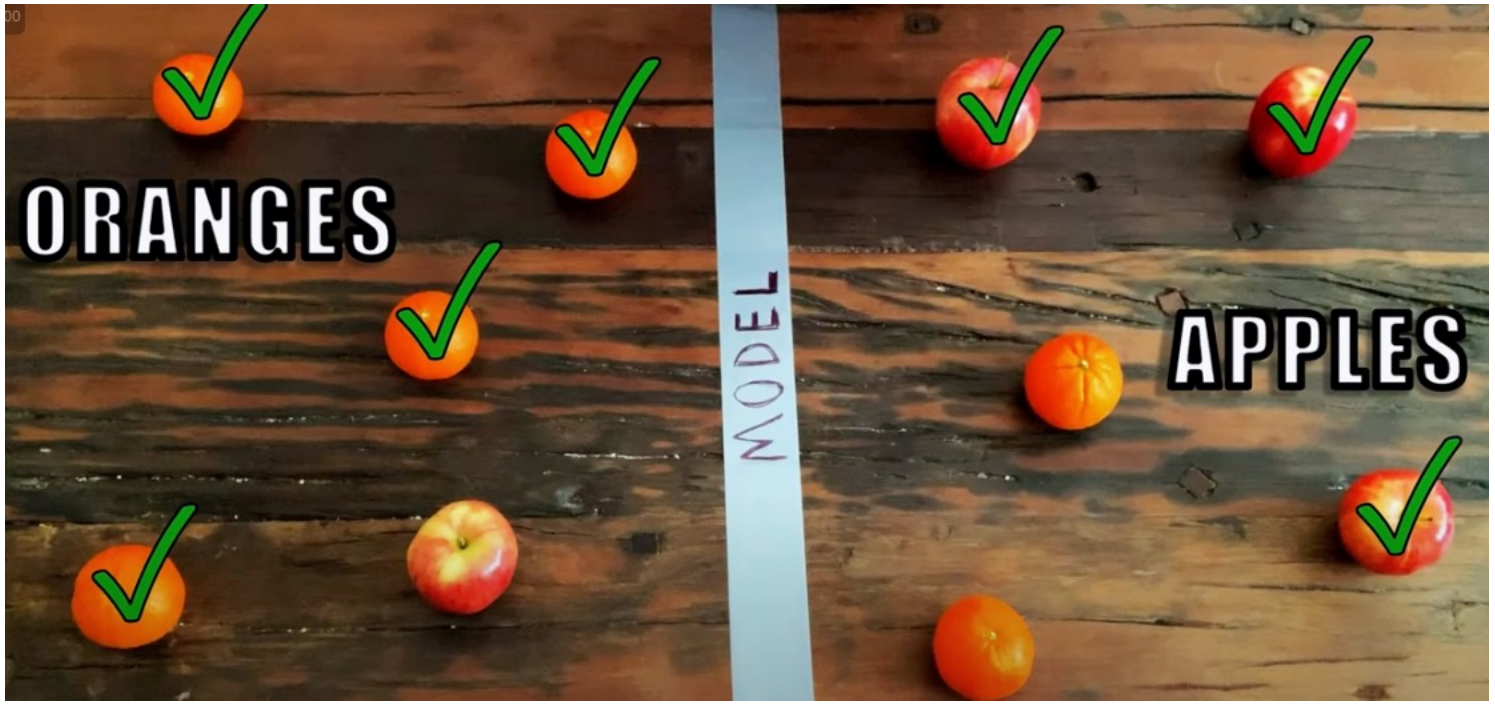
Evaluation

Step 4: Evaluation

- Classification: F1 Score:
  - Widely used metric in the field of machine learning.
  - Particularly for classification tasks.
  - Combines **precision** and **recall** into a single number.
- Language Model: Perplexity
  - Measures how confidently the model predicts the next token. Lower perplexity = better predictions.
  - It's a measure of uncertainty, with lower values indicating better performance.
- Translation / Generation: BLEU Score (Bilingual Evaluation Understudy)
  - Primarily used for machine translation
  - Evaluates the quality of machine-generated text compared to human-generated reference texts.
  - BLEU compares n-gram overlap between model output and reference, e.g.
    - Ref: “The cat is on the mat”
    - Gen: “A cat is on mat” → BLEU penalizes missing “the” & wrong article.

Accuracy

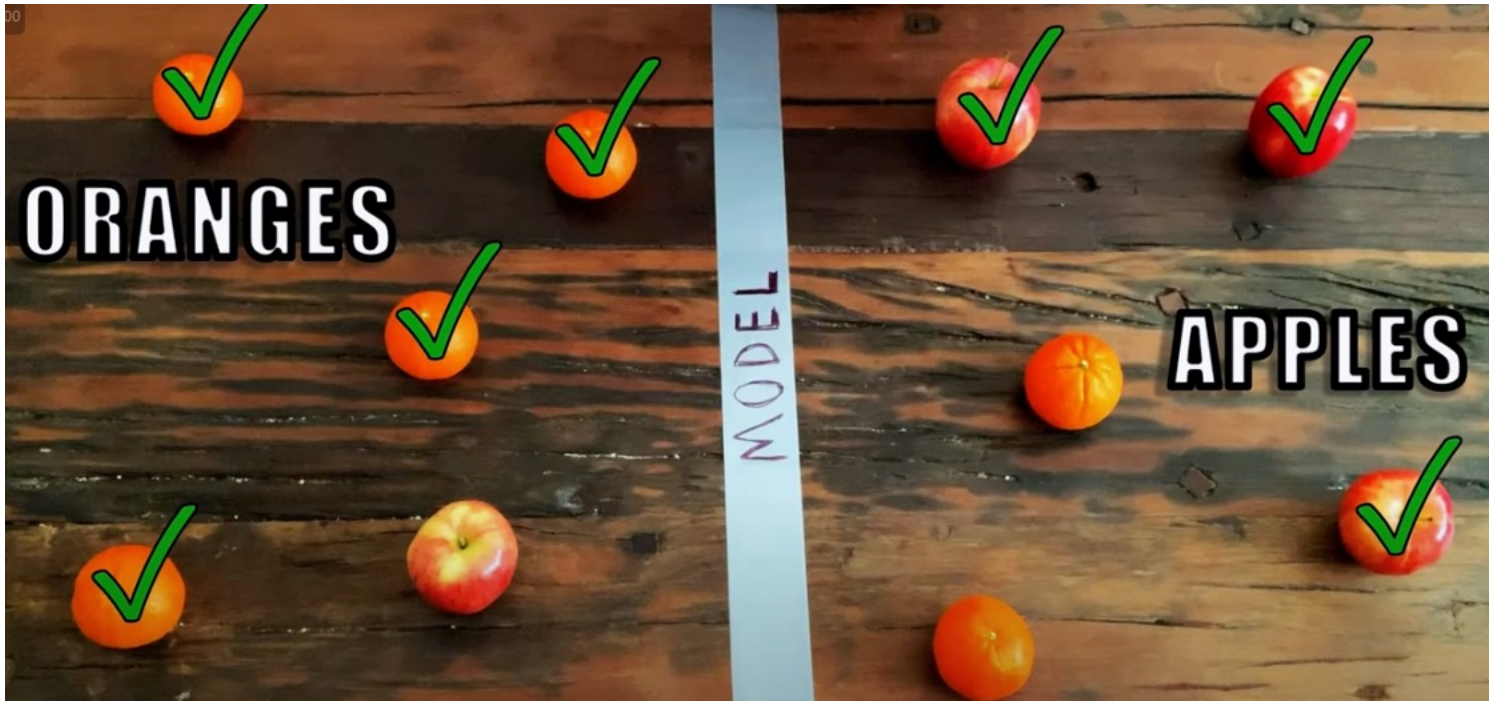
- Accuracy: $\frac{\text{Total correct}}{\text{Total Observations}} = \frac{7}{10} = 70\%$



Source: https://www.youtube.com/watch?v=qWfzIYCvBqo

Accuracy

- Accuracy: $\frac{\text{Total correct}}{\text{Total Observations}} = \frac{7}{10} = 70\%$

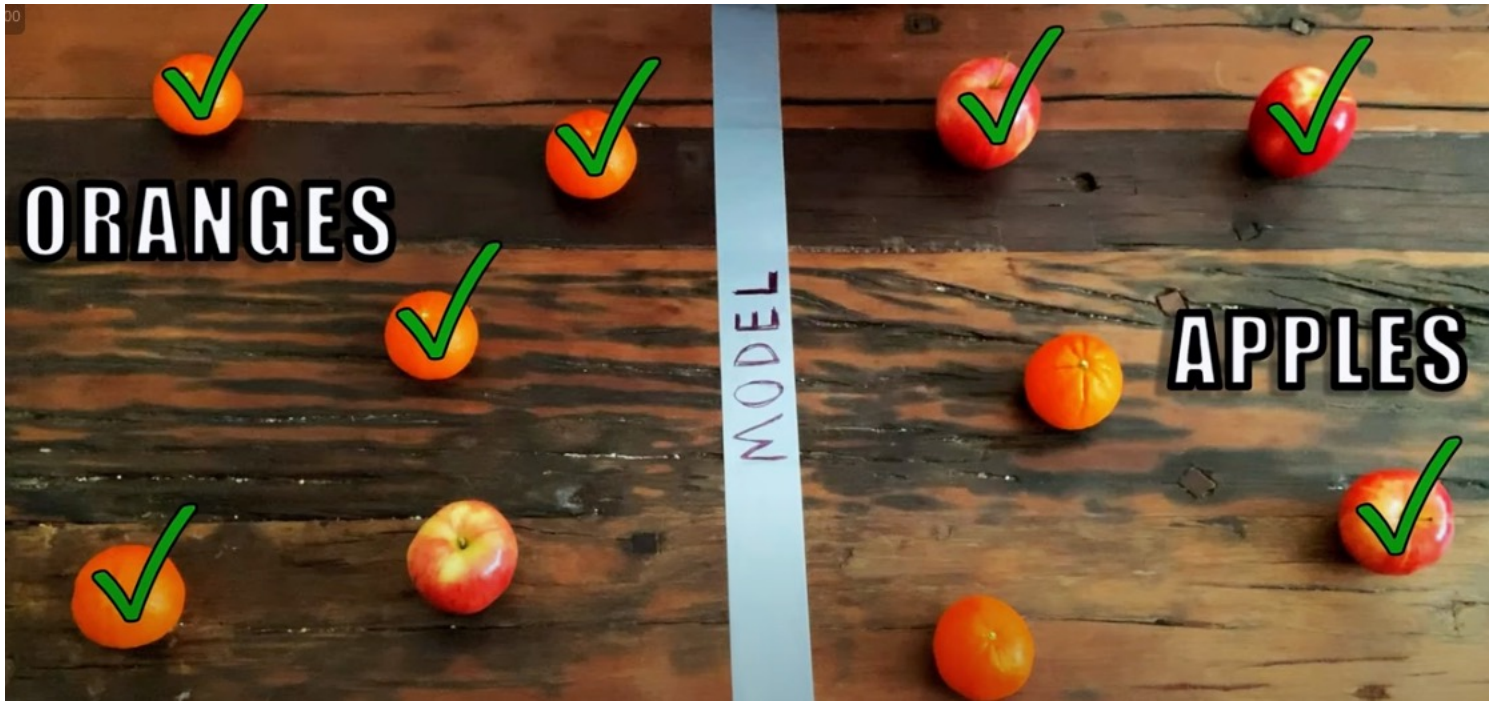


Source: https://www.youtube.com/watch?v=qWfzIYCvBqo

Suppose we have 990 oranges and 10 apples
The model classify ALL of them as oranges. What is the accuracy?

Accuracy

- Accuracy: $\frac{\text{Total correct}}{\text{Total Observations}} = \frac{7}{10} = 70\%$

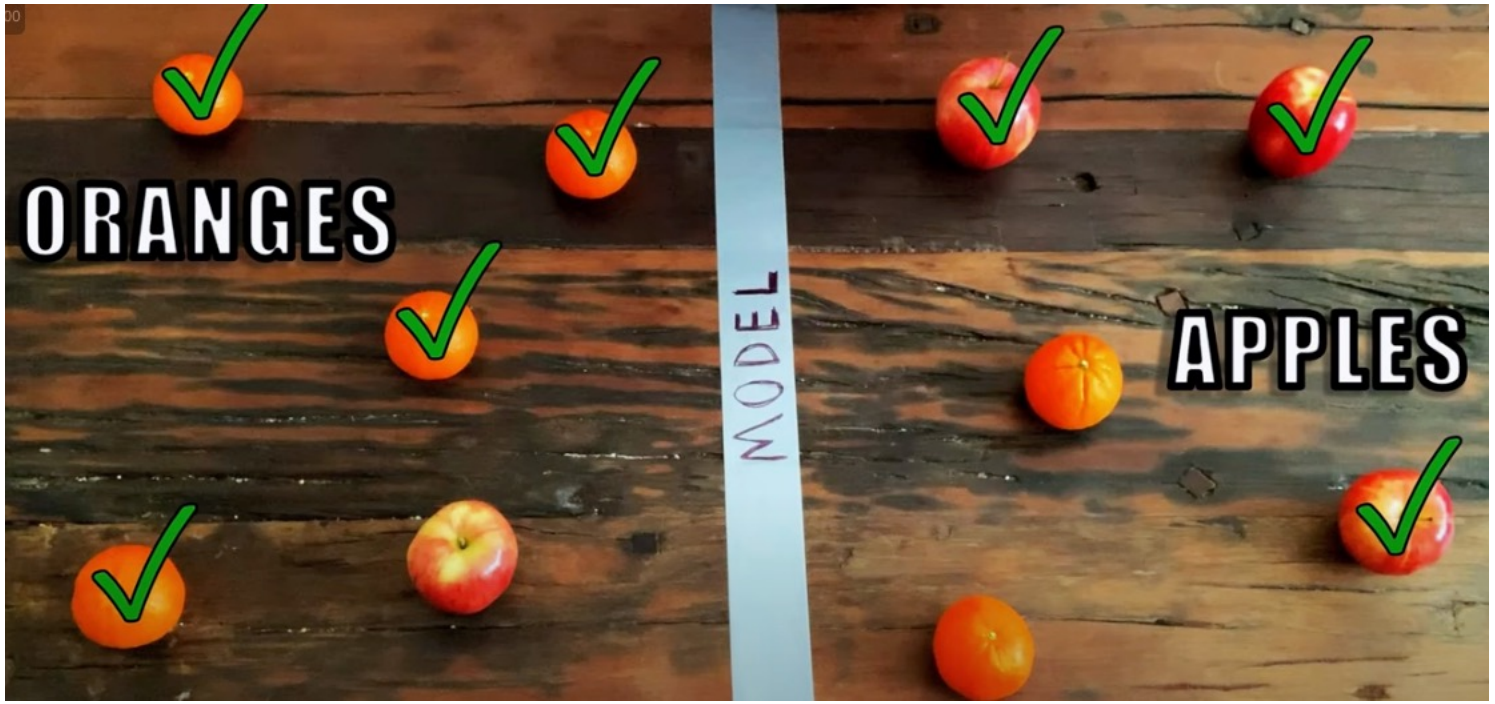


Source: https://www.youtube.com/watch?v=qWfzIYCvBqo

Suppose we have 990 oranges and 10 apples
The model classify ALL of them as oranges. What is the accuracy? Accuracy: $\frac{990}{1000} = 99\%$

Accuracy

- Accuracy: $\frac{\text{Total correct}}{\text{Total Observations}} = \frac{7}{10} = 70\%$



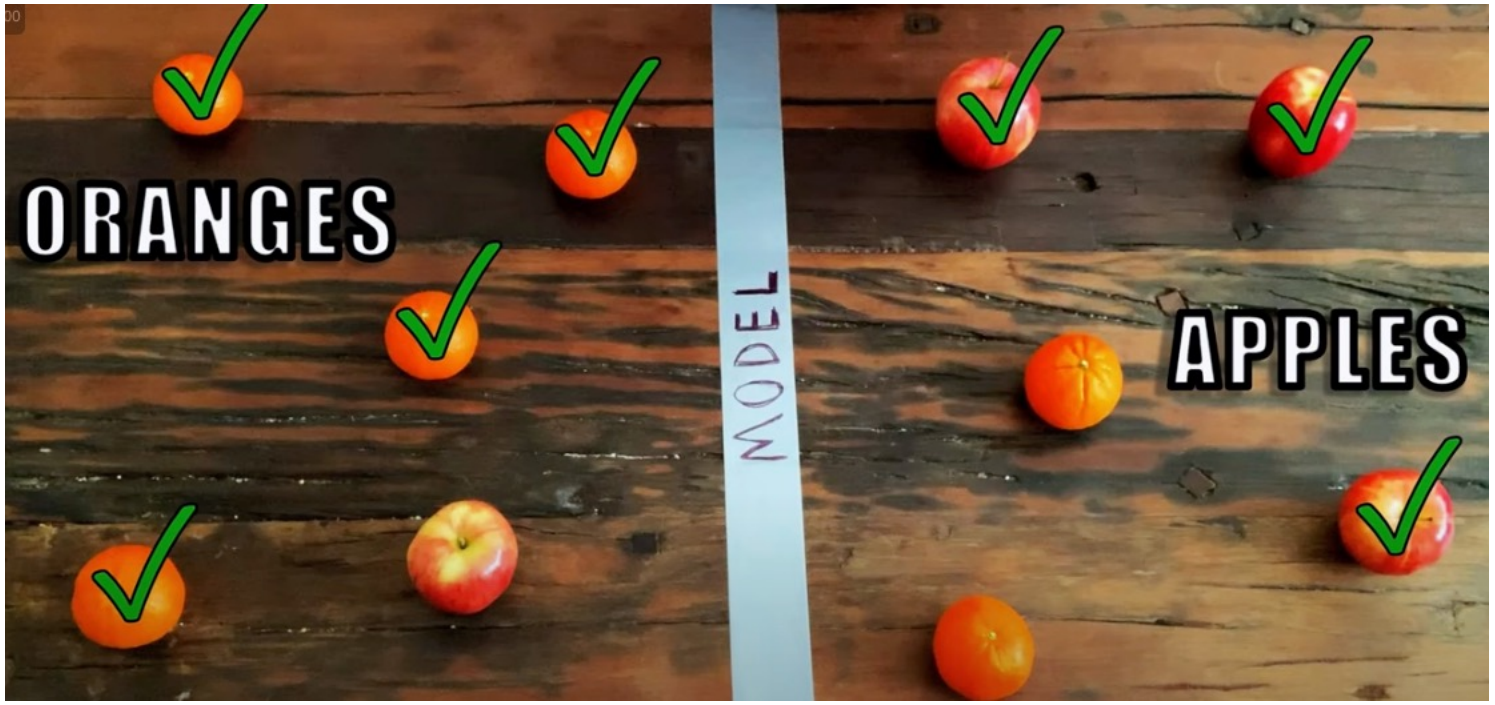
Source: https://www.youtube.com/watch?v=qWfzIYCvBqo

Suppose we have 990 oranges and 10 apples
The model classify ALL of them as oranges. What is the accuracy? Accuracy: $\frac{990}{1000} = 99\%$

Accuracy is not a good metric for imbalanced classes

Precision

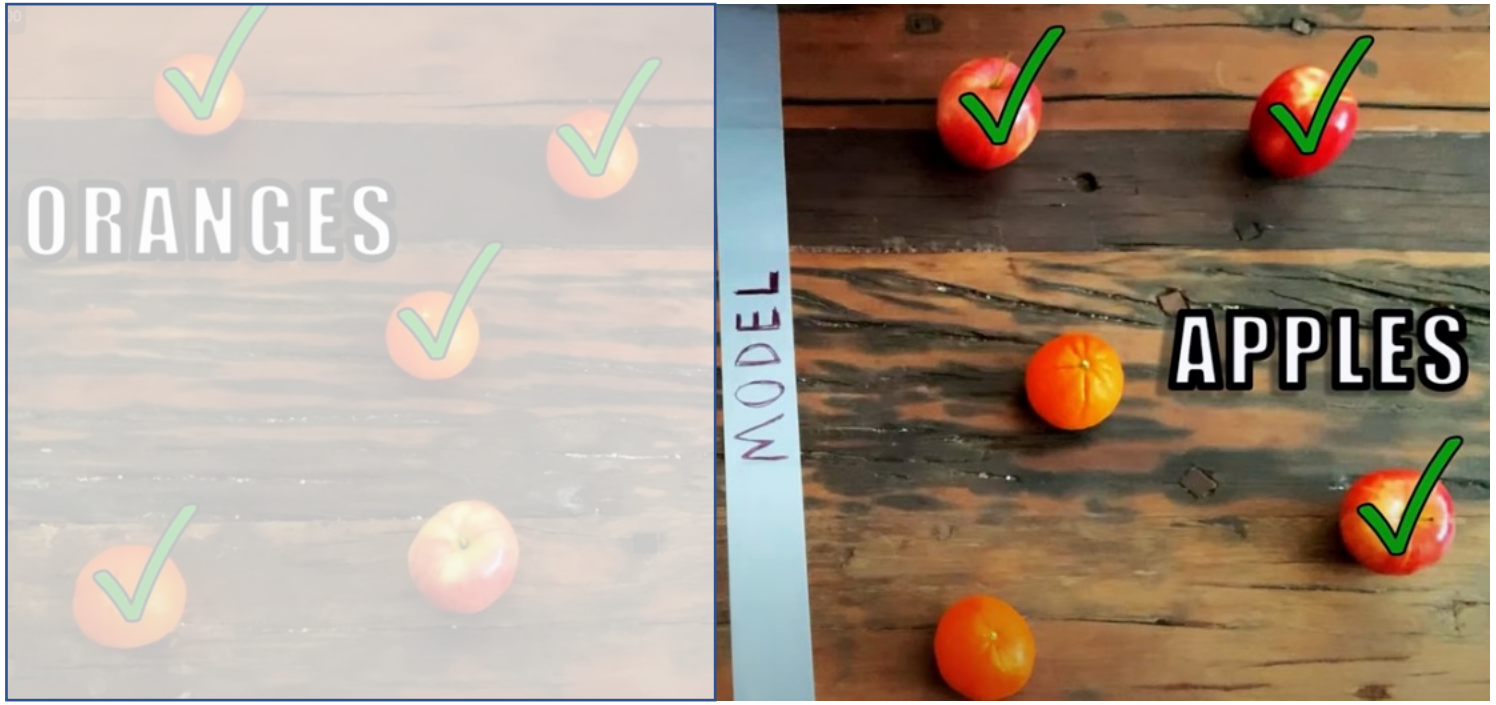
- To calculate the precision of the apple class, we completely forget about the other classes



Source: https://www.youtube.com/watch?v=qWfzIYCvBqo

Precision

- To calculate the precision of the apple class, we completely forget about the other classes

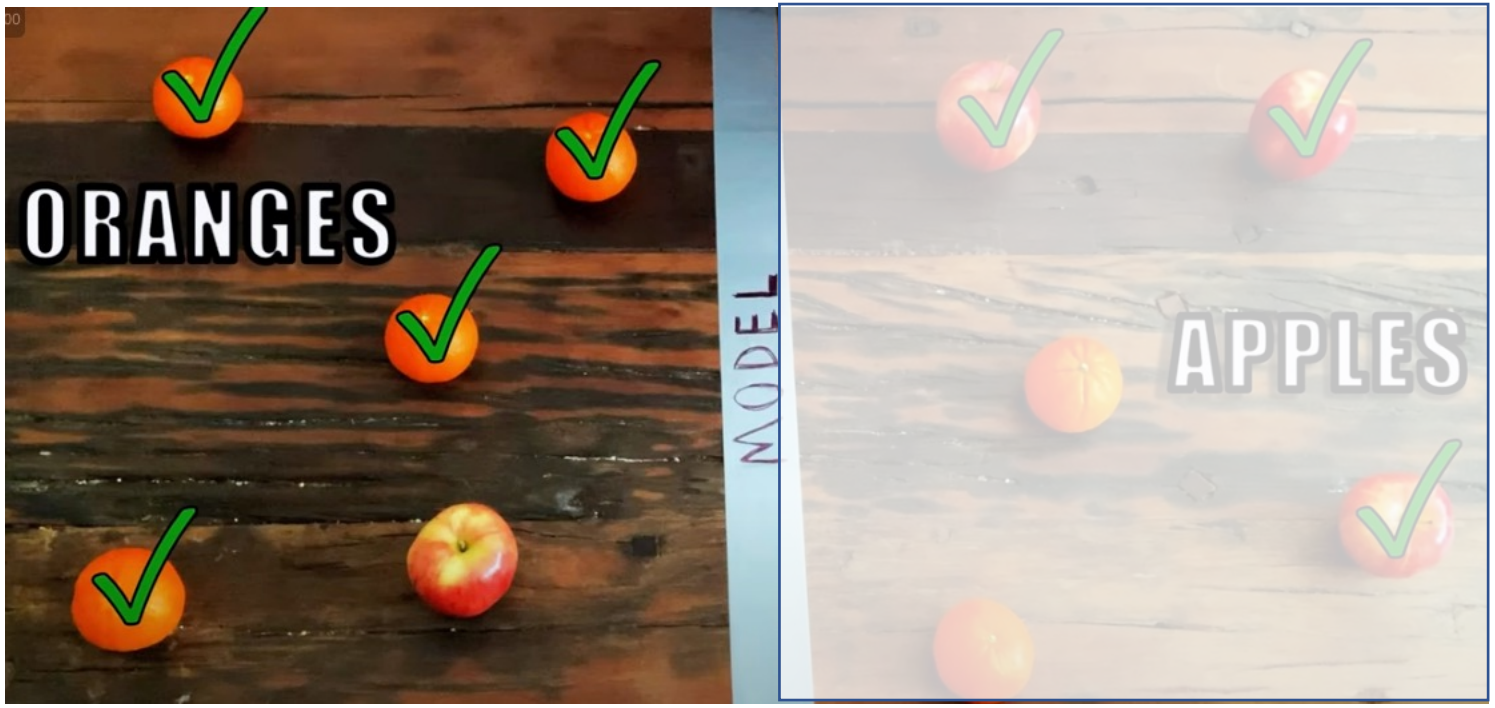


Source: https://www.youtube.com/watch?v=qWfzIYCvBqo

Precision: $\frac{Total\ correct\ apples}{Total\ apple\ side\ observations} = \frac{3}{5} = 60\%$

Precision

- To calculate the precision of the apple class, we completely forget about the other classes

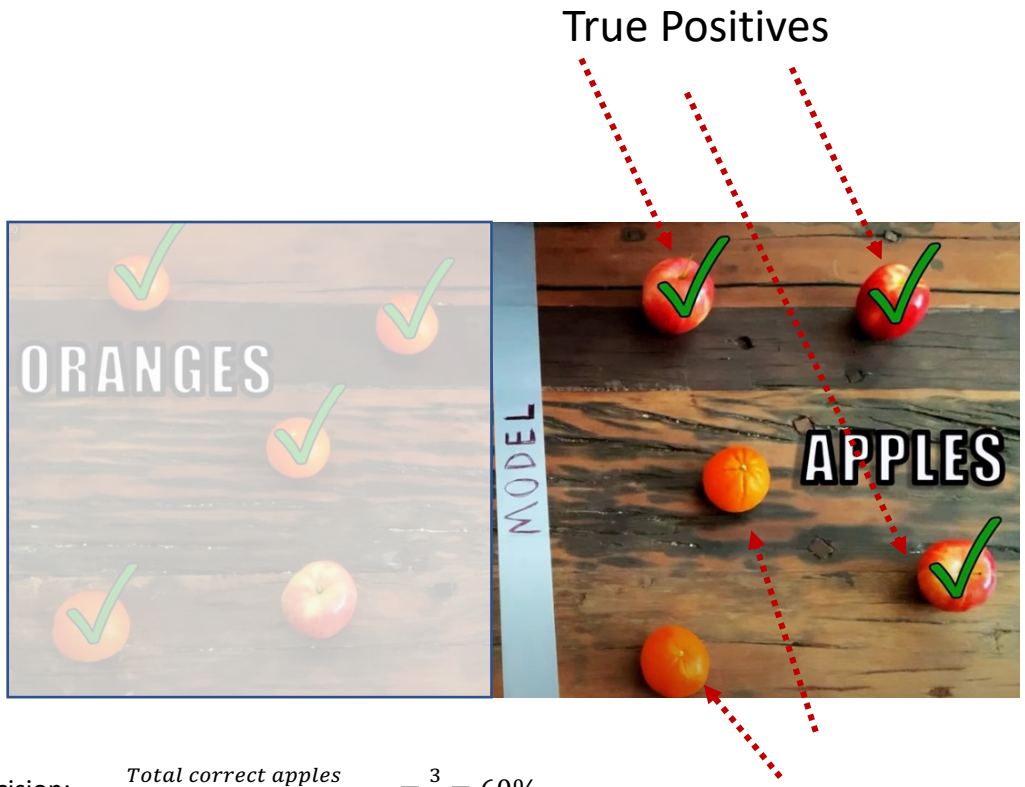


Source: https://www.youtube.com/watch?v=qWfzIYCvBqo

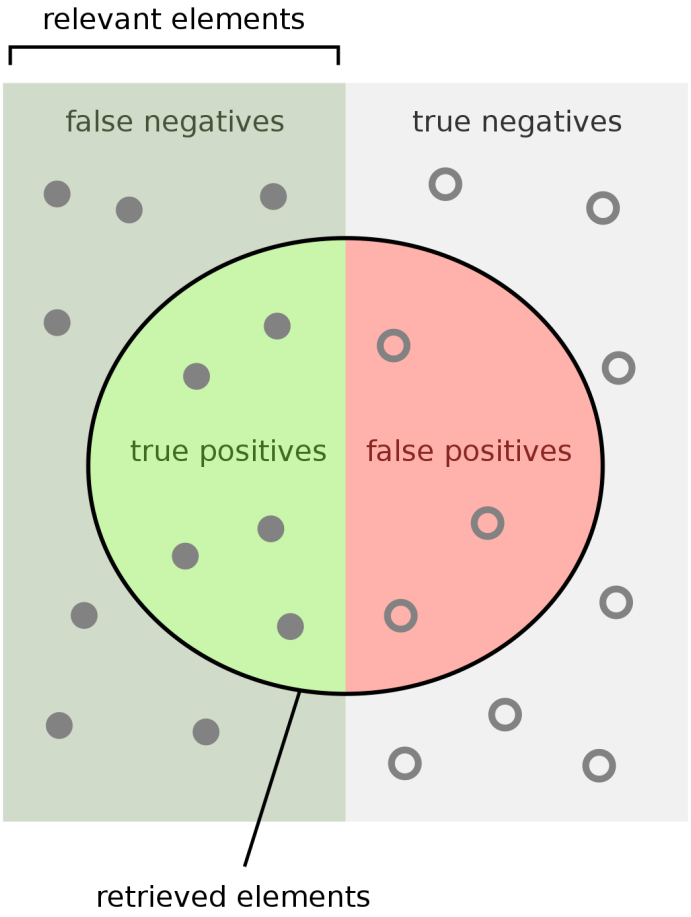
Precision: $\frac{Total\ correct\ orange}{Total\ orange\ side\ observations} = \frac{4}{5} = 80\%$

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Positives(FP)}}$$

Of all the instances **classified as positive**, how many are actually positive



Precision: $\frac{Total\ correct\ apples}{Total\ apple\ side\ observations} = \frac{3}{5} = 60\%$

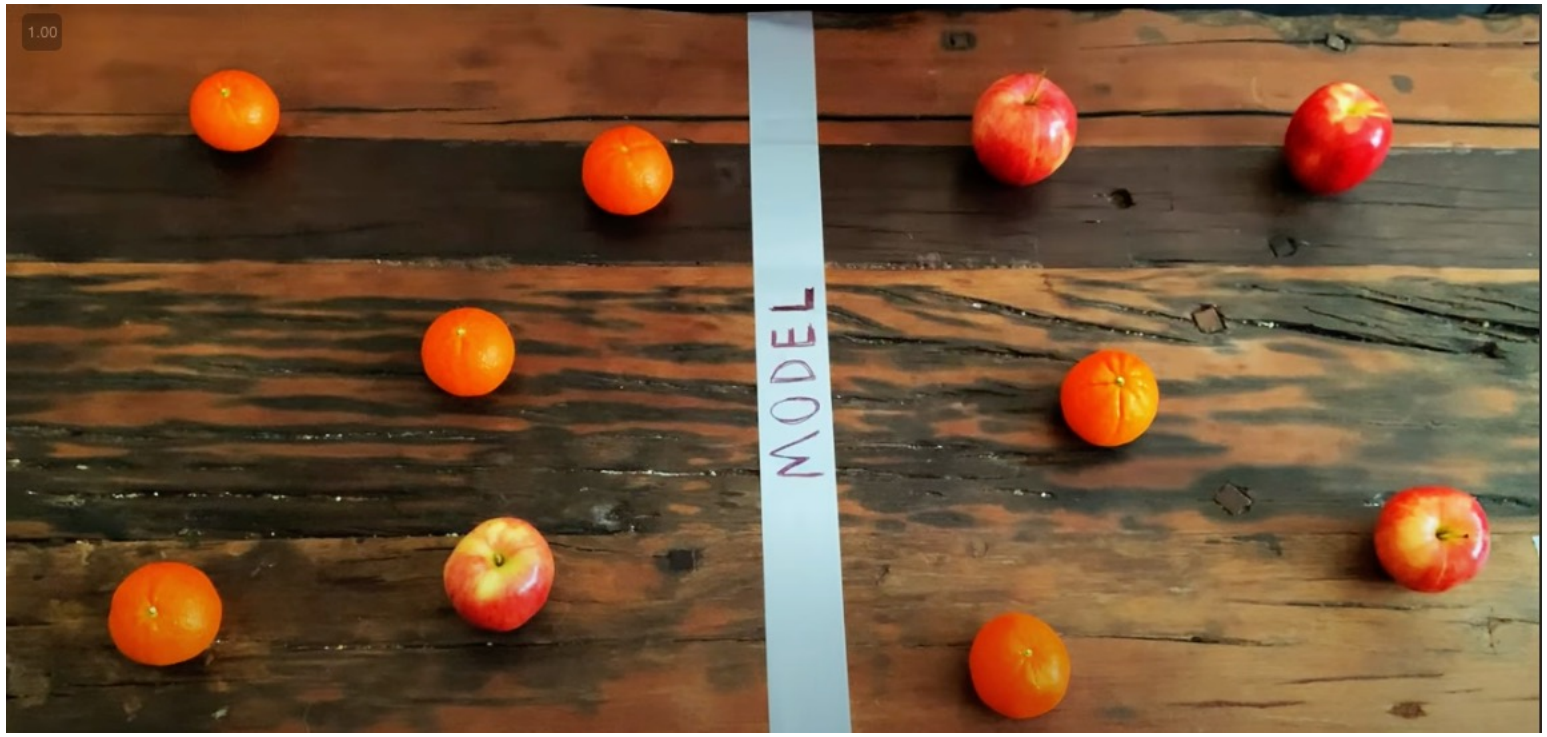


How many retrieved items are relevant?

Precision = $\frac{\text{green}}{\text{green + red}}$

Recall (True Positive Rate)

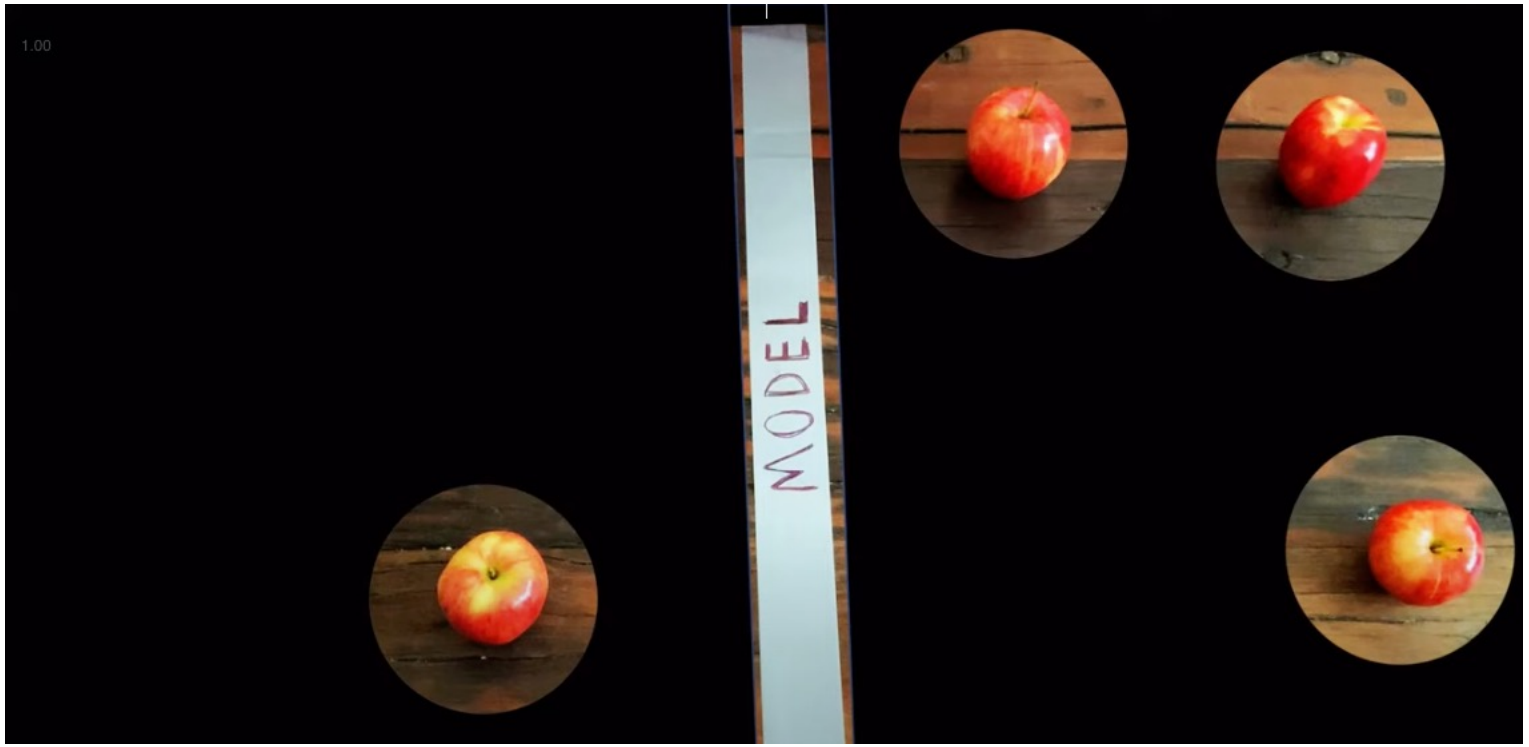
- To calculate the recall of the apple class, we completely forget about the other classes on both sides



Source: https://www.youtube.com/watch?v=qWfzIYCvBqo

Recall (True Positive Rate)

- To calculate the recall of the apple class, we completely forget about the other classes on both sides

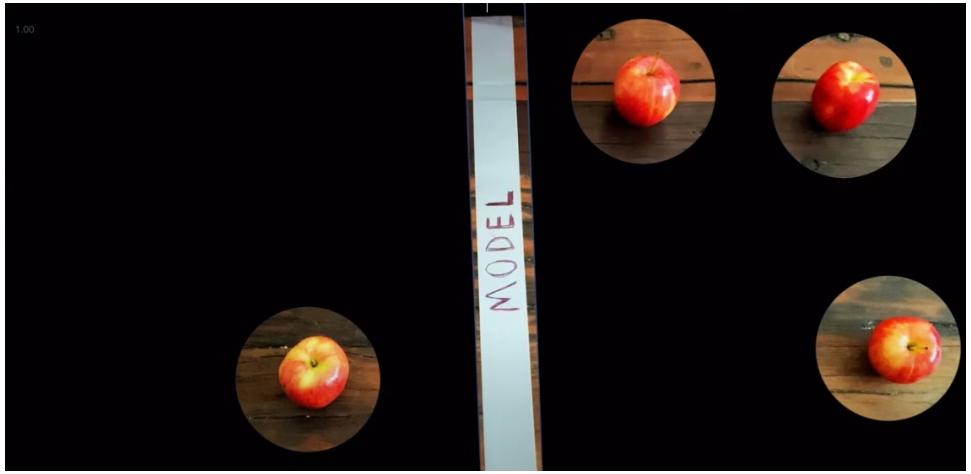


Source: https://www.youtube.com/watch?v=qWfzIYCvBqo

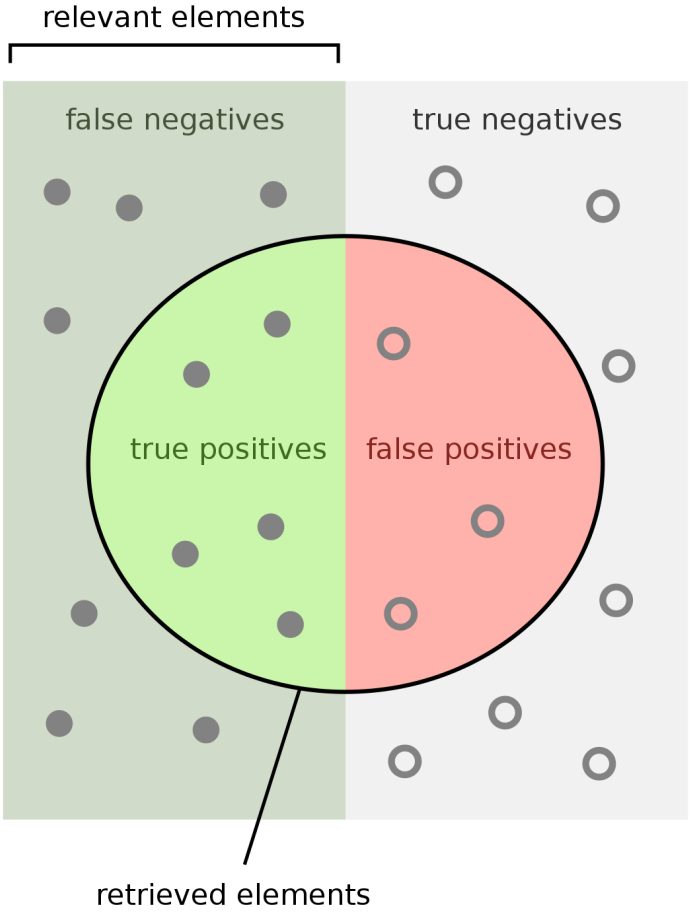
Recall: $\frac{Total\ correct\ apples}{Total\ actual\ apples} = \frac{3}{4} = 75\%$

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Negatives(FN)}}$$

Of all the **actual positives**, how many did the model recall?



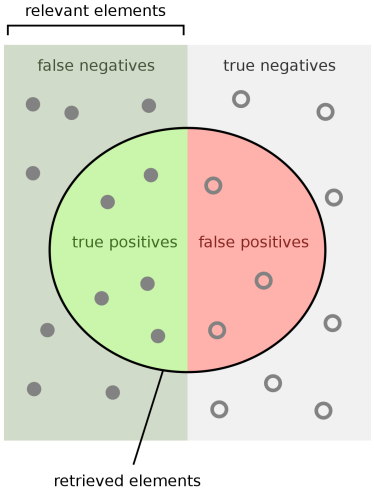
Recall: $\frac{Total\ correct\ apples}{Total\ actual\ apples} = \frac{3}{4} = 75\%$



How many relevant items are retrieved?

Recall = 

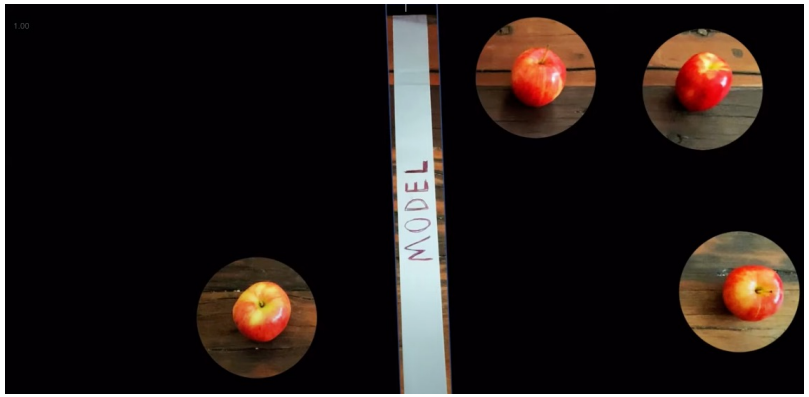
Precision vs Recall



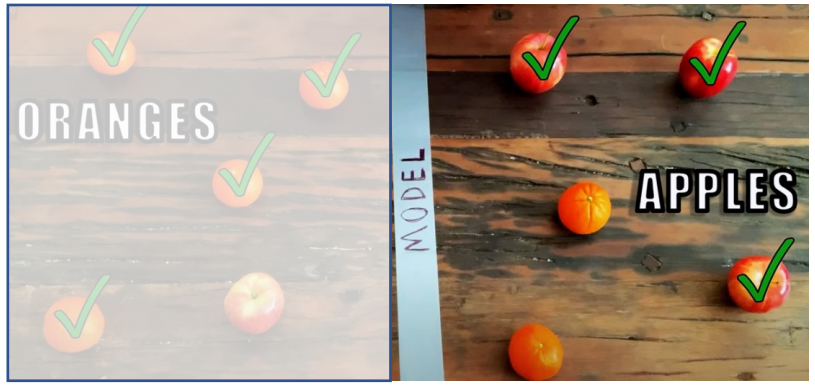
Actual \ Predicted	0	1
0	TN	FP
1	FN	TP

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

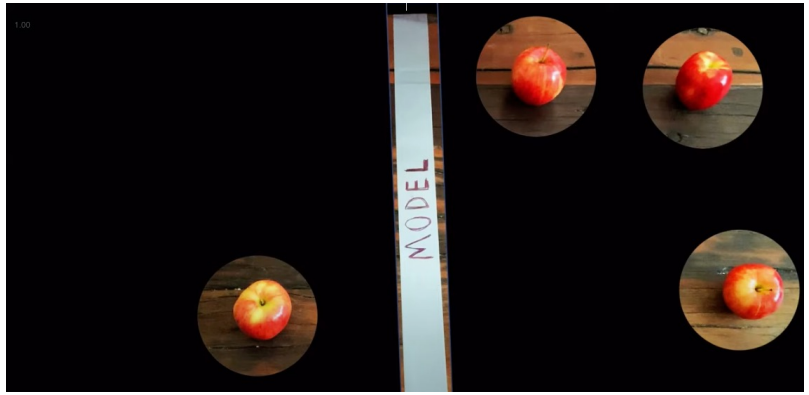


Recall



Precision

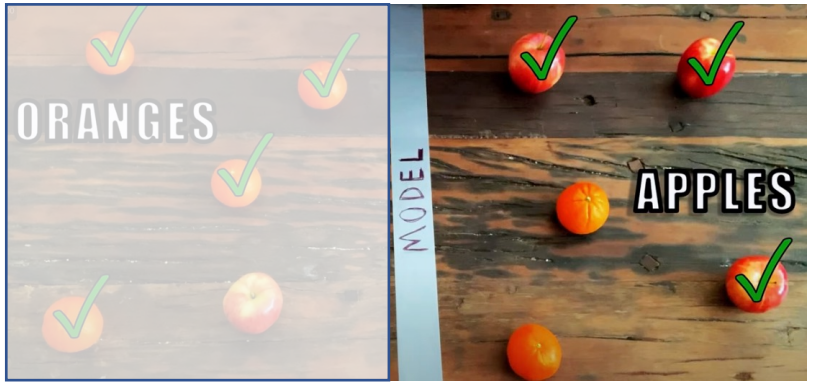
Precision vs Recall



Recall

When the cost of a false **negative** is high

- In fraud detection, missing a fraudulent transaction (low recall) could result in financial loss.
- In a search engine, it's crucial to find most relevant results (high recall) even if some irrelevant ones pop up (low precision).



Precision

When the cost of a false **positive** is high

- Spam Detection: It's better to miss a few real emails (low recall) than be flooded with spam (low precision).
- Medical Diagnostics: when follow-up procedures are costly or invasive, a high precision can reduce the number of unnecessary treatments.

F1 Score Example: Consider a binary classification with to classes of positive and negative

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Positives(FP)}}$$

Of all the instances **classified as positive**, how many are actually positive? (**Accuracy**)

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Negatives(FN)}}$$

Of all the **actual positives**, how many did the model recall?(**Completeness**)

$$\text{F1 Score} = 2\times \frac{\text{Precisioin} \times \text{Recall}}{\text{Precisioin} + \text{ Recall}}$$

F1 Score Example: Consider a binary classification with to classes of positive and negative

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Positives(FP)}}$$

Of all the instances **classified as positive**, how many are actually positive? (**Accuracy**)

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Negatives(FN)}}$$

Of all the **actual positives**, how many did the model recall?(**Completeness**)

$$\text{F1 Score} = 2\times\frac{\text{Precisioin} \times \text{Recall}}{\text{Precisioin} + \text{ Recall}}$$

Suppose we have 990 oranges and 10 apples The model classify ALL of them as oranges. What is the accuracy, precision, recall, and F1?

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{Total} = \frac{0 + 990}{1000} = 0.99 = 99\%$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{0}{0 + 10} = 0$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{0}{0 + 0} \rightarrow \text{undefined} \Rightarrow 0$$

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = 0$$

	Predicted Apple	Predicted Orange
Actual Apple	0 (TP)	10 (FN)
Actual Orange	0 (FP)	990 (TN)

F1 Score Example: Consider a binary classification with to classes of positive and negative

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Positives(FP)}}$$

Of all the instances **classified as positive**, how many are actually positive? (**Accuracy**)

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Negatives(FN)}}$$

Of all the **actual positives**, how many did the model recall?(**Completeness**)

$$\text{F1 Score} = 2\times \frac{\text{Precisioin} \times \text{Recall}}{\text{Precisioin} + \text{ Recall}}$$

Examples:

Scenario	Precision	Recall	F1 Score	Arithmetic Mean
First Scenario	0.95	0.30	0.456	0.625
Second Scenario	0.65	0.60	0.624	0.625

Why not use arithmetic mean?

F1 Score Example: Consider a binary classification with to classes of positive and negative

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Positives(FP)}}$$

Of all the instances **classified as positive**, how many are actually positive? (**Accuracy**)

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Negatives(FN)}}$$

Of all the **actual positives**, how many did the model recall?(**Completeness**)

$$\text{F1 Score} = 2\times \frac{\text{Precisioin} \times \text{Recall}}{\text{Precisioin} + \text{Recall}}$$

Examples:

Scenario	Precision	Recall	F1 Score	Arithmetic Mean
First Scenario	0.95	0.30	0.456	0.625
Second Scenario	0.65	0.60	0.624	0.625

- In the **First Scenario**, despite high precision (0.95), the recall is quite low (0.30).
  - The F1 score (0.46) reflects this imbalance by being much lower than the arithmetic mean (0.625).
- In the **Second Scenario**, precision and recall are more balanced (0.65 and 0.60, respectively).
  - The F1 score (0.63) is very close to the arithmetic mean (0.625).

F1 Score Example: Consider a binary classification with to classes of positive and negative

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Positives(FP)}}$$

Of all the instances **classified as positive**, how many are actually positive? (**Accuracy**)

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives(TP)} + \text{False Negatives(FN)}}$$

Of all the **actual positives**, how many did the model recall?(**Completeness**)

$$\text{F1 Score} = 2\times \frac{\text{Precisioin} \times \text{Recall}}{\text{Precisioin} + \text{ Recall}}$$

For multi-class classification, F1 score can be calculated on each class and averaged on all classes

Perplexity

- Perplexity
  - **Perplexity** measures how well a probability model predicts a sequence.
  - It's a measure of **uncertainty/confidence**:
    - 🔻 **Lower perplexity** = better, more confident model
    - 🔺 **Higher perplexity** = worse, more confused model
  - Common in: **language modeling**, **speech recognition**, **machine translation**

For a text T containing N words $[W_1, W_2, \ldots, W_N]$:

$$PP(T) = P(W_1, W_2, \ldots, W_N)^{-\frac{1}{N}}$$

$$\text{Perplexity}(T) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i \mid w_1, \ldots, w_{i-1})\right)$$

(In practice, we use the log form)

$$PP(T) \in [1, \infty]$$

Good model Bad model

If perplexity is 10, the model is as uncertain as if it's choosing from 10 equally likely options at each step.

Perplexity

- Let’s say we have a toy vocabulary: ["I", "like", "cats", "dogs"].
- **Sequence:**
  - "I like cats"
- Assume your model gives these predicted probabilities at each time step:

Step	Target Token	Predicted Prob. for Target
1	"I"	0.9
2	"like"	0.5
3	"cats"	0.25

$$\text{Loss} = -\frac{1}{3}\left[\log(0.9) + \log(0.5) + \log(0.25)\right]$$

$$= -\frac{1}{3}[-0.105 + (-0.693) + (-1.386)] = \frac{1}{3}(2.184) \approx 0.728$$

$$\text{Perplexity} = e^{0.728} \approx 2.07$$

- Perplexity ≈ 2.07 → the model is "choosing between ~2 options on average"
- If a model is perfect: perplexity → 1
- If random: perplexity → size of vocabulary

BLEU Score

- **BLEU** evaluates the quality of machine-generated text by comparing it to one or more **reference translations**.
- It's based on **n-gram overlap** — how many phrases the model got right.
- Originally proposed for **machine translation**, now also used in **text summarization** and **paraphrasing**.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Formula (Simplified)

- **Reference:**
  - *the cat is on the mat*
- **Candidate Output:**
  - *the cat sat on mat*

- Unigrams matched: **the**, **cat**, **on**, **mat**
- Bigrams matched: **the cat**
- BLEU will **reward matches**, but penalize:
  - Missed bigrams: "is on", "the mat"
  - Shorter length via **brevity penalty**

- p_n = precision for n-grams (e.g., unigram, bigram, etc.)
- w_n = weight for each n-gram (typically uniform)
- **BP** = brevity penalty (punishes too-short translations)

Range:
- 0: Completely unrelated
- 0.7 and higher: high quality

BLEU Score Limitation

- **Insensitive to meaning**: A grammatically correct but wrong-sense output can still score high.
- Doesn't handle **word order flexibility** or **semantic similarity** well.
- Useful for **automatic benchmarking**, but should be combined with **human evaluation**

- **Reference:**
  - *the cat is on the mat*
- **Candidate Output:**
  - *the cat sat on mat*

- Unigrams matched: **the**, **cat**, **on**, **mat**
- Bigrams matched: **the cat**, **on mat**
- BLEU will **reward matches**, but penalize:
  - Missed bigrams: "is on", "the mat"
  - Shorter length via **brevity penalty**

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

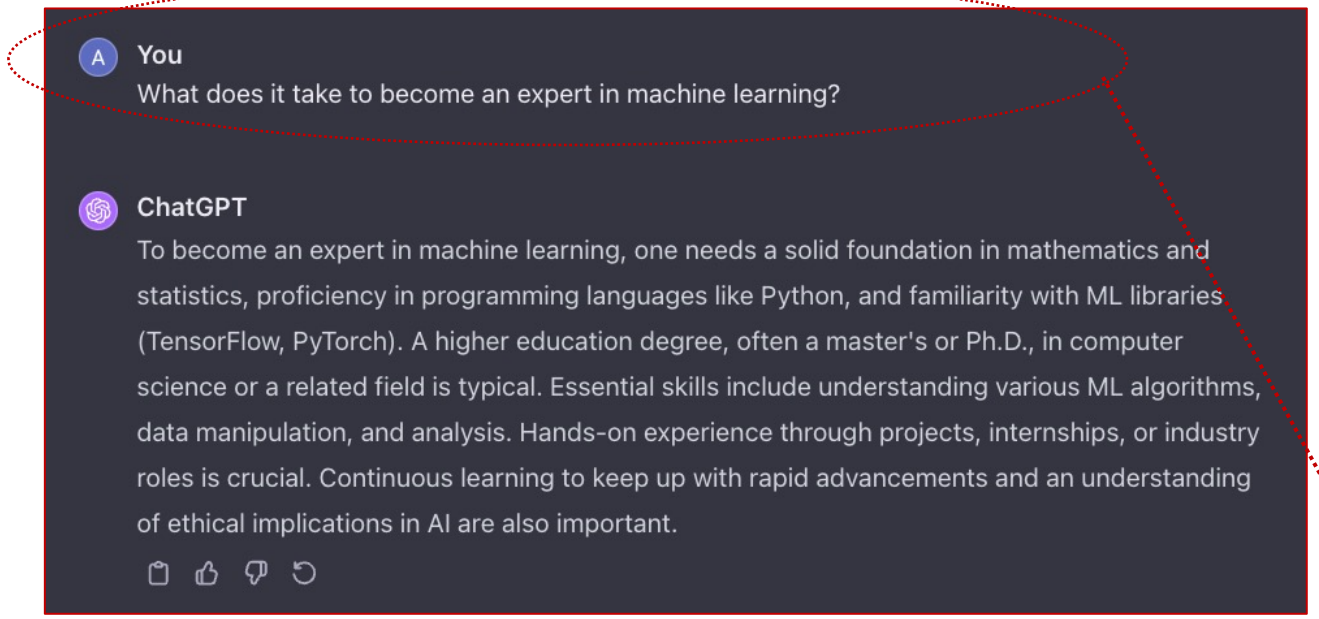
Formula (Simplified)

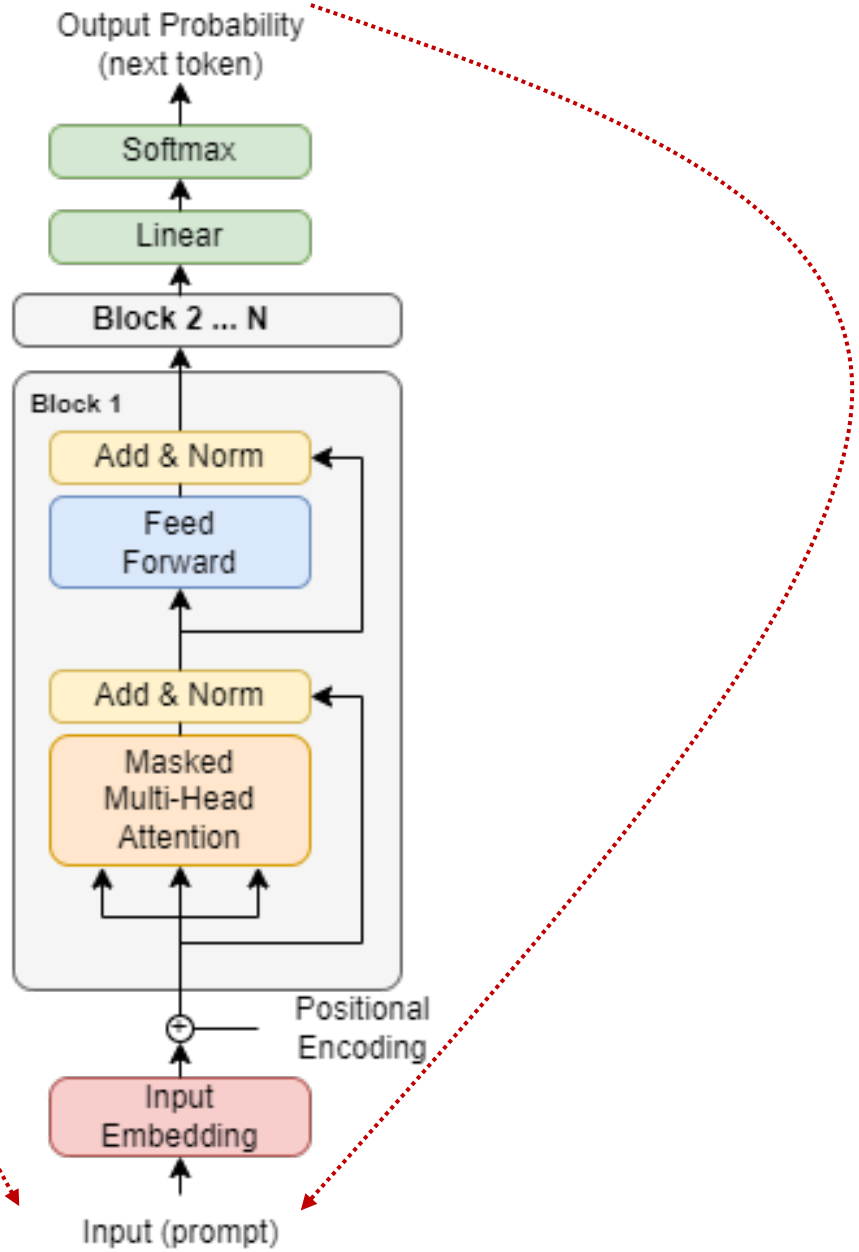
- p_n = precision for n-grams (e.g., unigram, bigram, etc.)
- w_n = weight for each n-gram (typically uniform)
- **BP** = brevity penalty (punishes too-short translations)

Range:
- 0: Completely unrelated
- 0.7 and higher: high quality

Decoder-Only Transformer

Decoder-Only Transformer





Decoding Strategies at Inference Time

Why We Need Decoding Strategies

- **The Problem**
- At each step, a language model outputs a **probability distribution** over the entire vocabulary.
  - We must decide **how to select the next token** from this distribution.
- Example: $P(\text{"cat"}) = 0.35, \quad P(\text{"dog"}) = 0.25, \quad \ldots$

We Need Decoding Strategies Like:

- Greedy decoding
- Top-k / Top-p sampling
- Beam search
- Contrastive / speculative decoding

Without Strategy	Problem
Always pick top-1	Repetitive and dull output
Random sampling	Chaotic or nonsensical text
Deterministic only	Lacks creativity or variation
Too much randomness	Hallucination or incoherence

Temperature in Large Language Models (LLMs)

- **What Is Temperature?**
- Controls **randomness** during text generation.
- Applied to logits before softmax:

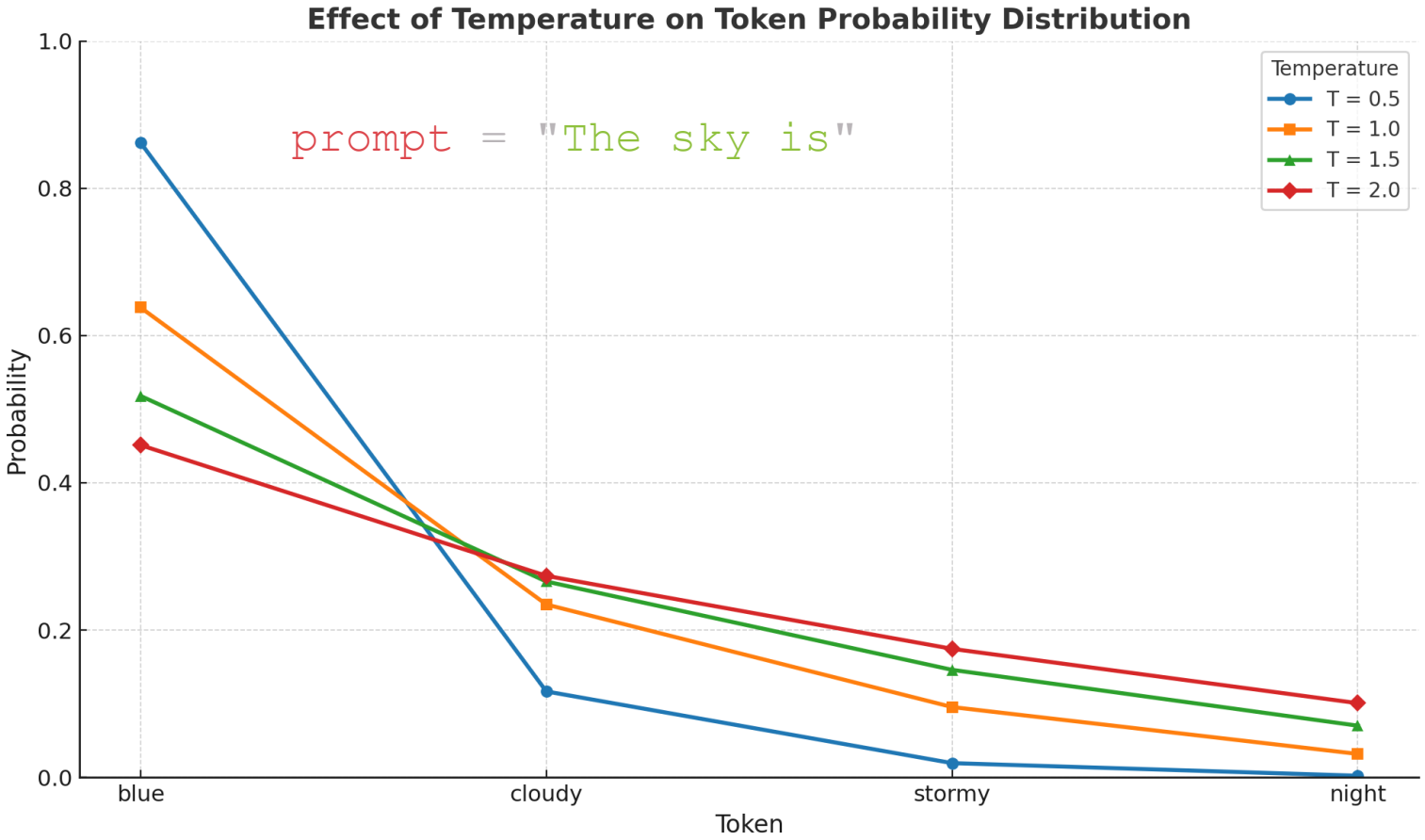
- $T = 1$: normal softmax
- $T < 1$: sharper → more deterministic
- $T > 1$: flatter → more creative/random

$$P(x_i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

Example Prompt:
“The sky is…”

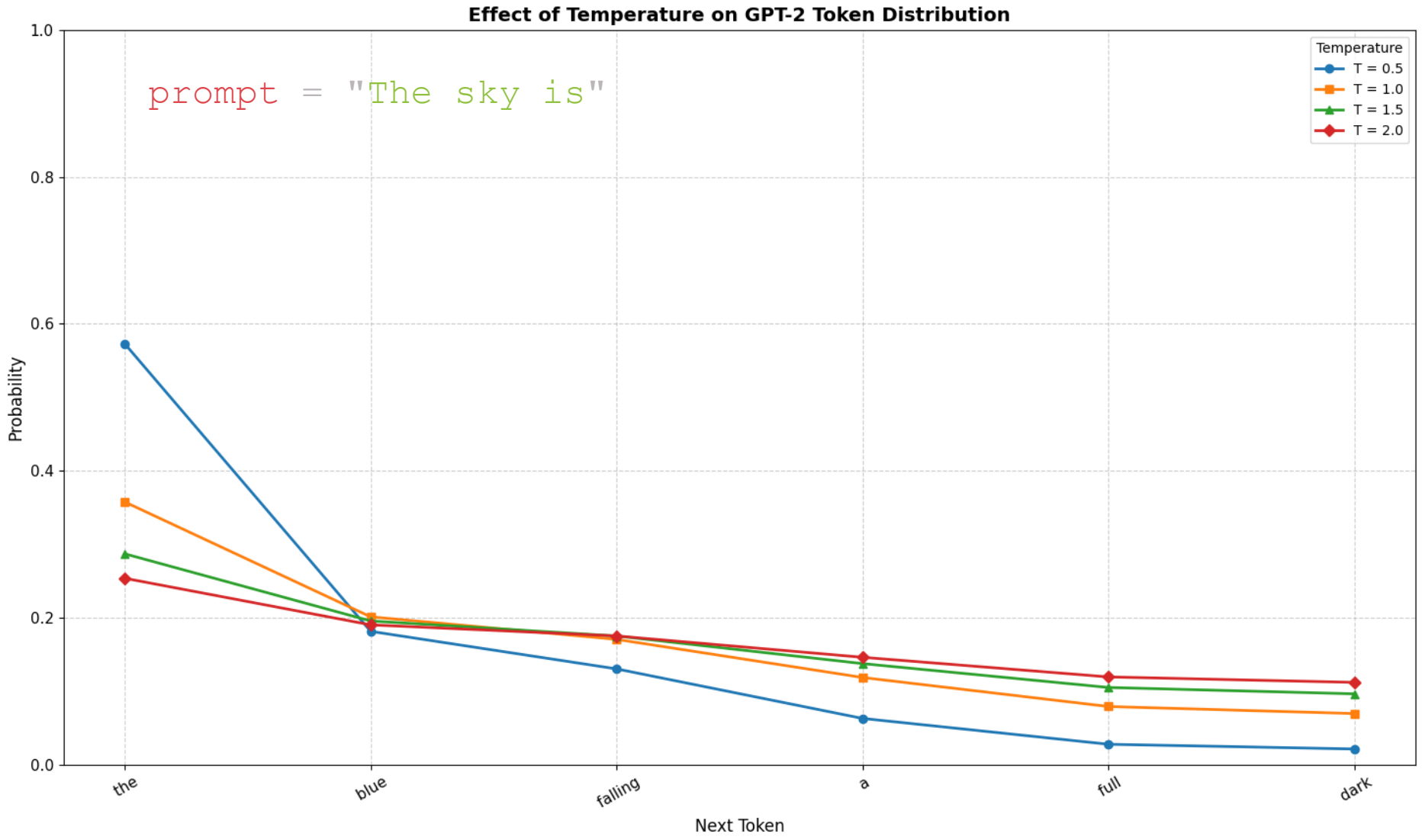
Temperature	Output Example
T=0	"blue."
T=0.7	"clear and blue."
T=1.0	"painted with hues of soft clouds."
T=1.5	"a dream woven from light and vapor."

Temperature in Large Language Models (LLMs)



At lower temperatures (e.g., 0.5), the distribution is sharp—favoring the highest-probability token strongly. As temperature increases (e.g., to 1.5 or 2.0), the distribution flattens, allowing more diverse and creative token choices, but with less certainty.

Temperature in Large Language Models (LLMs)



Effects of changing temperature on a real model, GPT-2

Overview of Sampling-Based Decoding

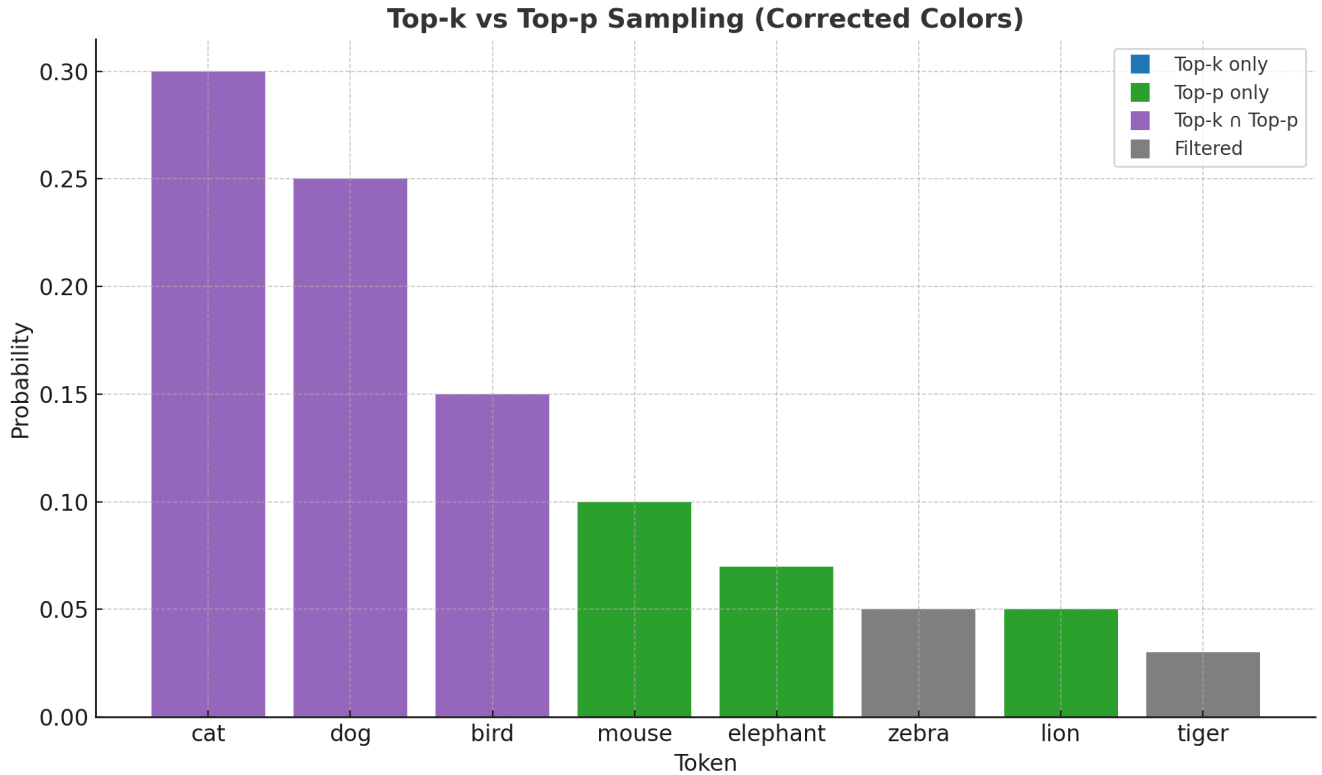
- **What is Sampling?**
  - Instead of choosing the top-1 token (greedy), **sampling** selects from a **probability distribution**.
  - Allows **diverse**, **creative**, and **non-repetitive** outputs.
- **Why Use It?**
  - Greedy decoding can be **boring and repetitive**
  - Sampling introduces **controlled randomness**

Top-k Sampling

- **How It Works:**
  - Keep only the **top k tokens**
  - Renormalize their probabilities
  - Sample from the reduced distribution
- **Example (k=3):**
  - *"cat": 0.4, "dog": 0.3, "bird": 0.2, "elephant": 0.1*
  - Keep: "cat", "dog", "bird"
- **Pros:** Simple
- **Cons:** Fixed size, not adaptive

Top-p (Nucleus) Sampling

- **How It Works:**
  - Sort tokens by probability
  - Select the **smallest set** whose cumulative prob ≥ p
  - Sample from this **adaptive subset**
- **Example (p = 0.9):**
  - Tokens: **"cat": 0.4**, **"dog": 0.3**, **"bird": 0.15**, **"elephant": 0.05**
  - Stop at "elephant" → total = 0.9
  - Sample from all 4
- **Pros:** Adaptive (the number of candidate tokens changes depending on the shape of the probability distribution.)
- **Cons:** Slightly more complex



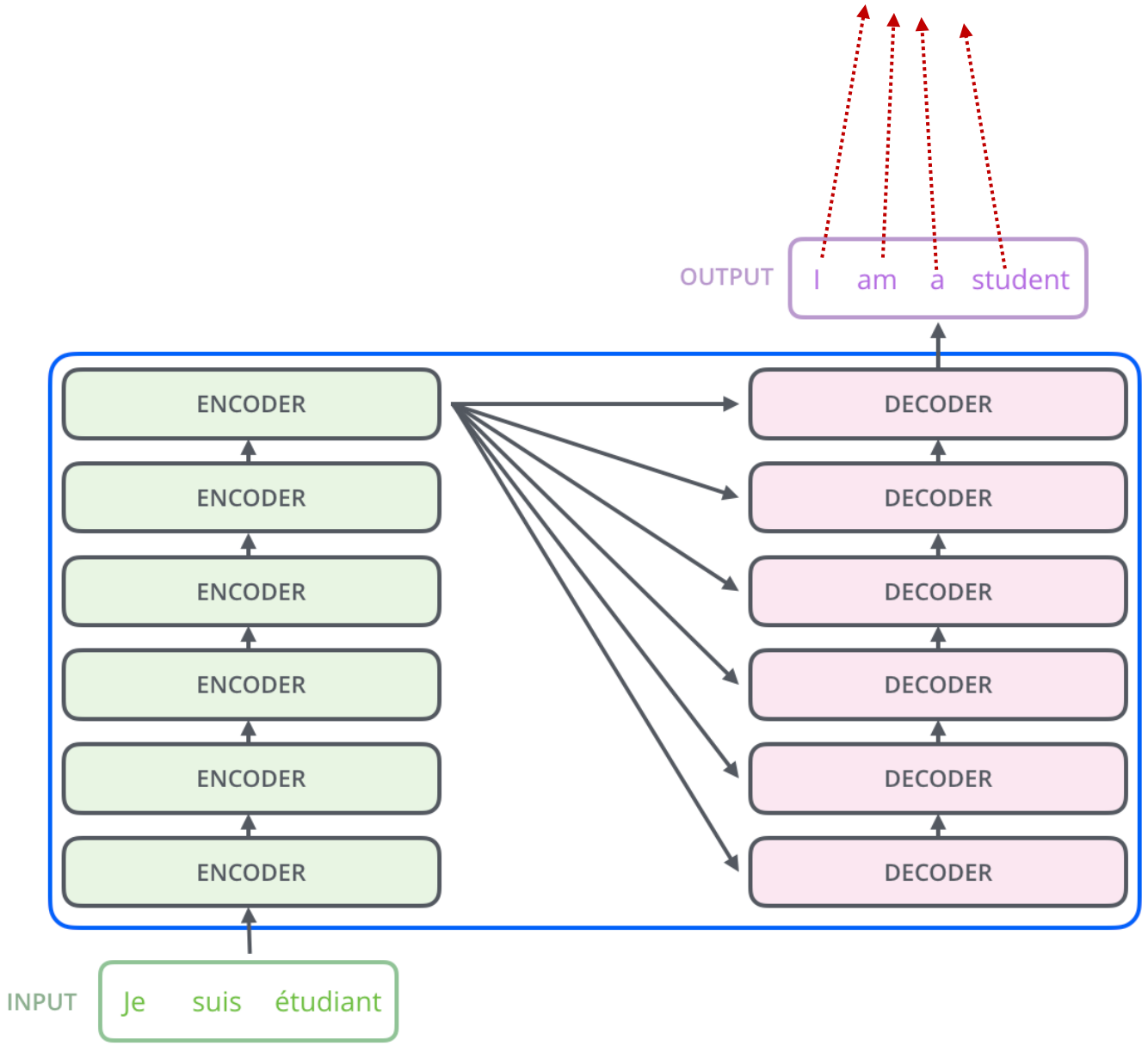
Token probabilities for a simulated language model output.

- **Top-k sampling** (blue/purple) selects the top 3 most likely tokens.
- **Top-p sampling** (green/purple) selects the smallest set of tokens whose cumulative probability exceeds 90%.
- Overlapping tokens (purple) are chosen by both methods.
- Remaining tokens (gray) are ignored in both strategies.

This highlights how **top-k is size-constrained**, while **top-p is probability-constrained and adaptive**.

Beam Search

The token with the highest probability?



Beam Search

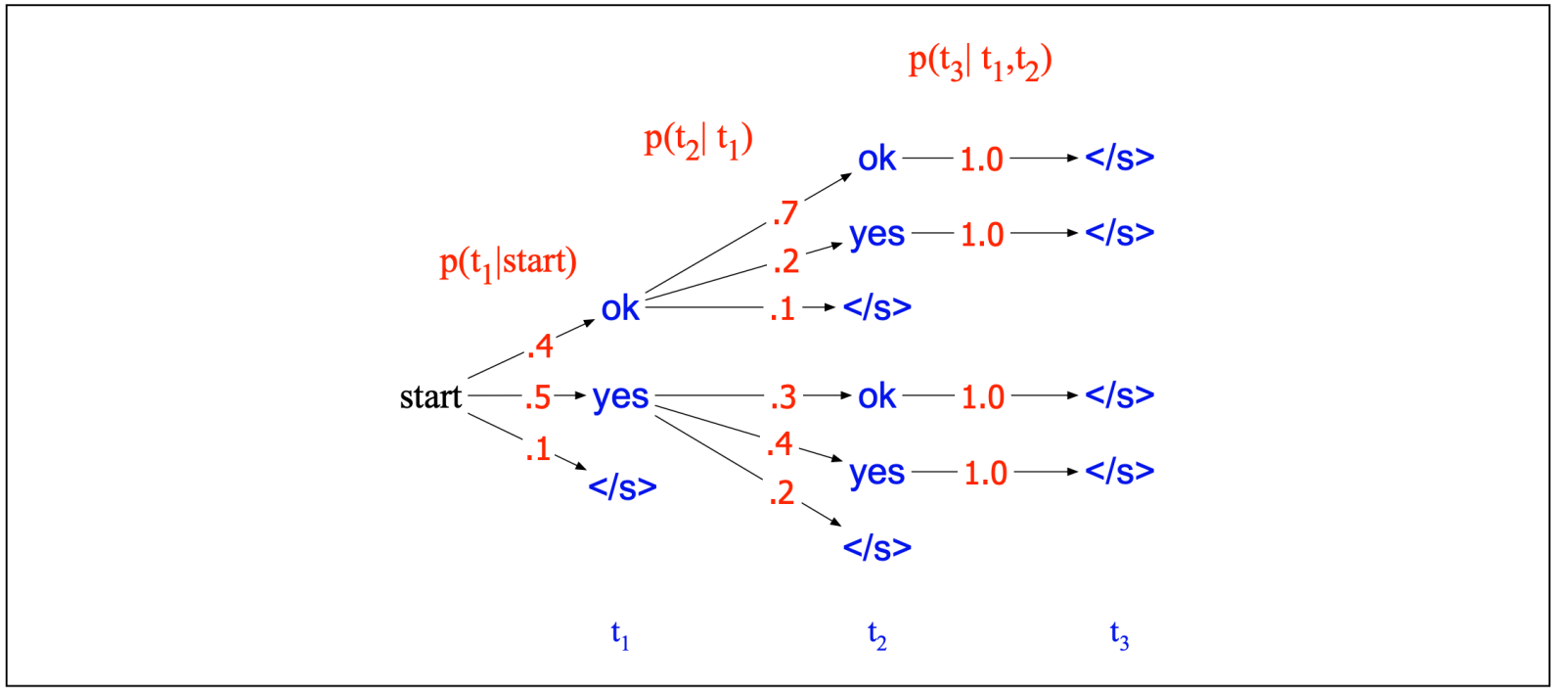


Figure 10.8 A search tree for generating the target string $T = t_1, t_2, \ldots$ from the vocabulary $V = \{\text{yes}, \text{ok}, \text{<s>}\}$, showing the probability of generating each token from that state. Greedy search would choose *yes* at the first time step followed by *yes*, instead of the globally most probable sequence *ok ok*.

Source: Speech and Language Processing (3rd ed. draft)
Dan Jurafsky and James H. Martin

Beam Search

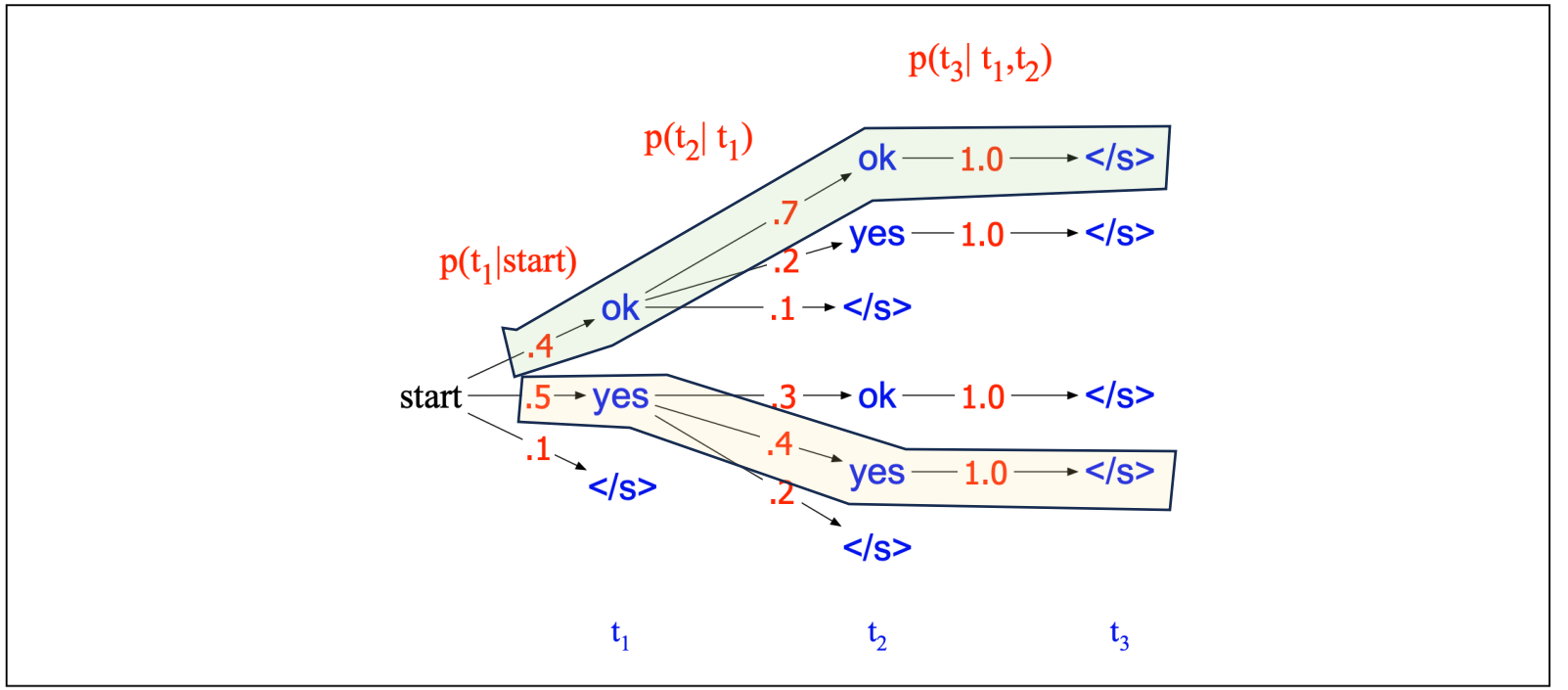
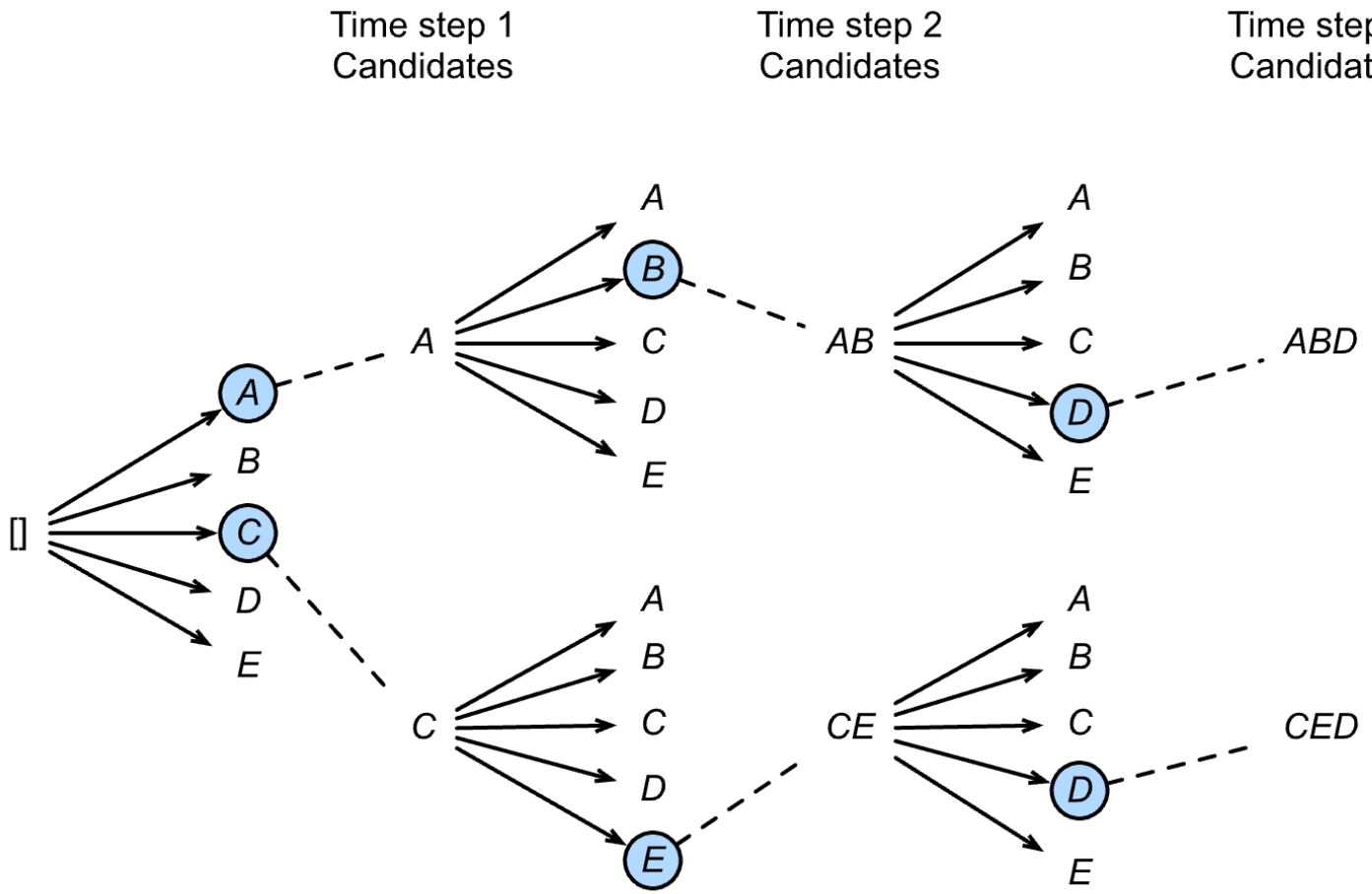


Figure 10.8 A search tree for generating the target string $T = t_1, t_2, \ldots$ from the vocabulary $V = \{\text{yes}, \text{ok}, \text{<s>}\}$, showing the probability of generating each token from that state. Greedy search would choose *yes* at the first time step followed by *yes*, instead of the globally most probable sequence *ok ok*.

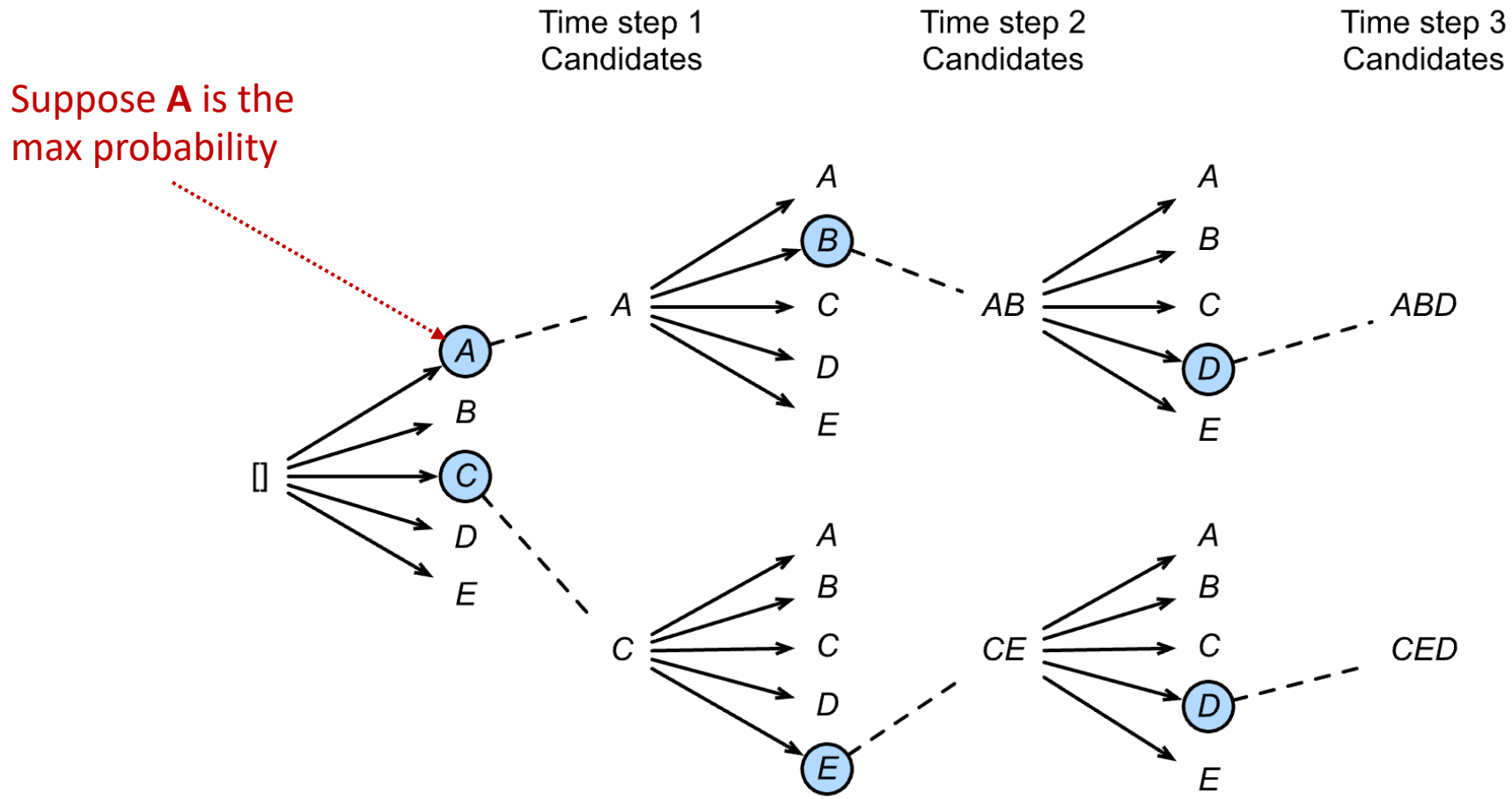
Source: Speech and Language Processing (3rd ed. draft)
Dan Jurafsky and James H. Martin

Beam Search

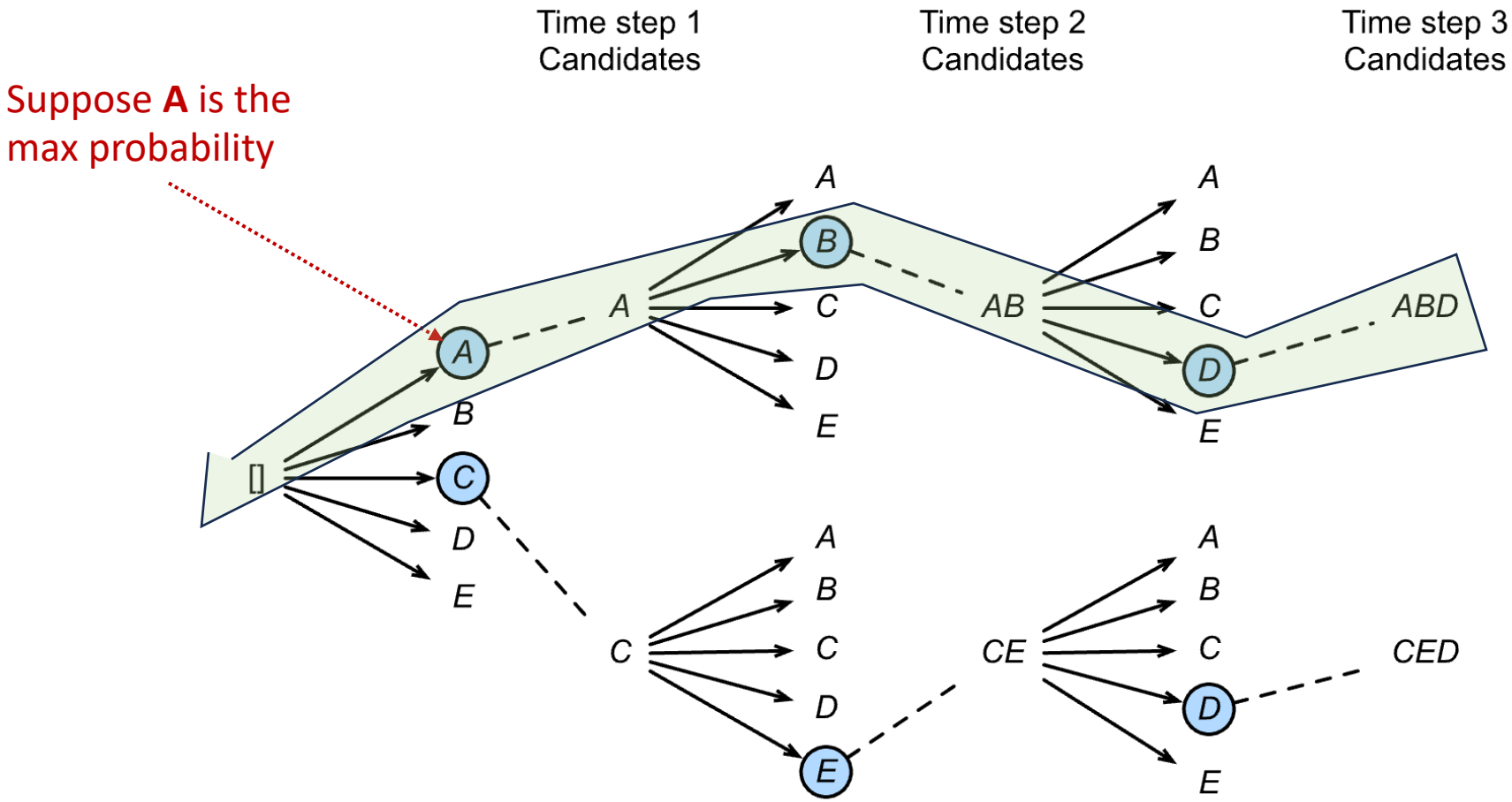


Source: https://d2l.ai/chapter_recurrent-modern/beam-search.html

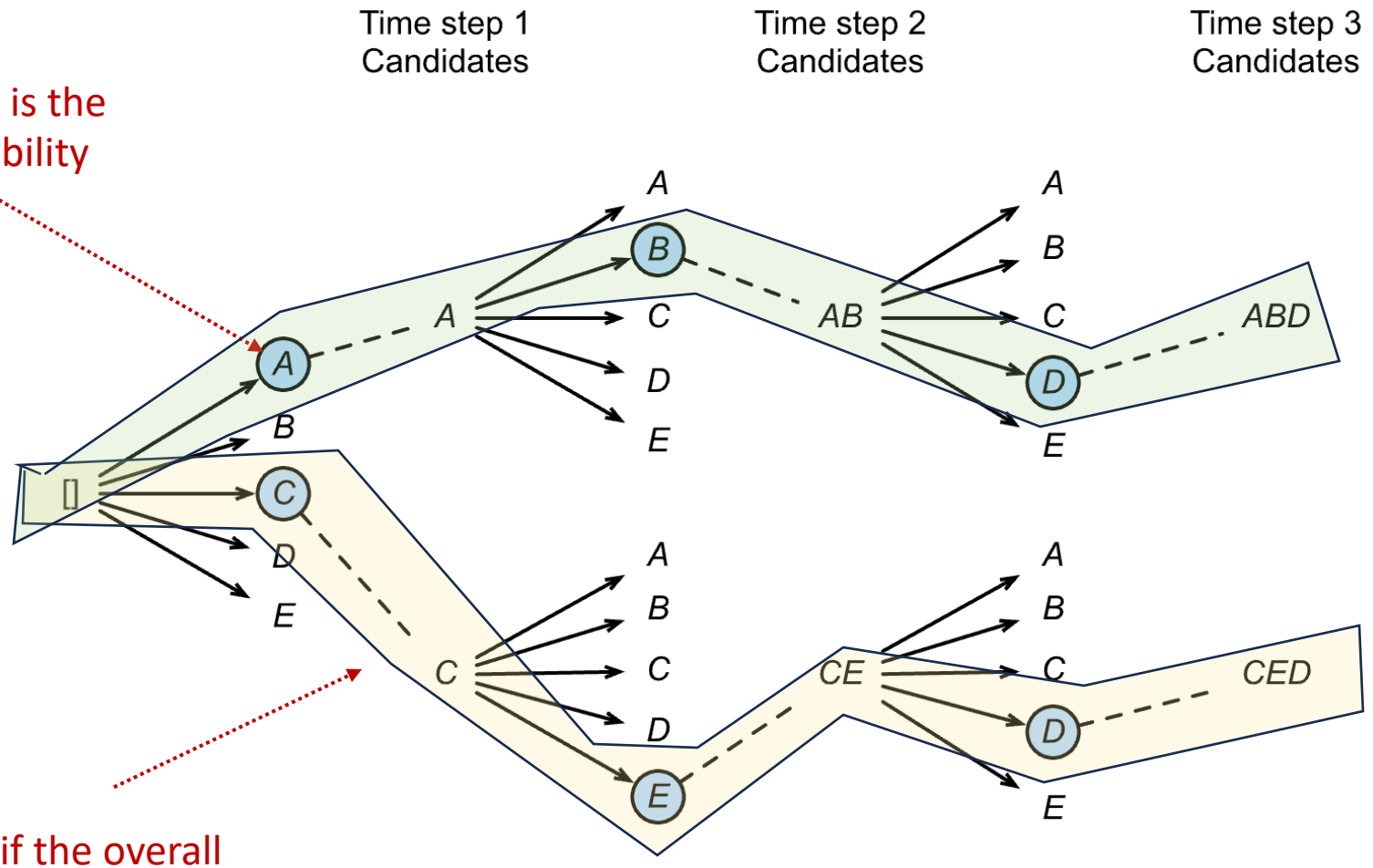
Beam Search


Time step 1
Candidates
Time step 2
Candidates
Time step 3
Candidates
Suppose **A** is the
max probability
[]
A
B
C
D
E
A
B
C
D
E
AB
ABD
C
CE
CED

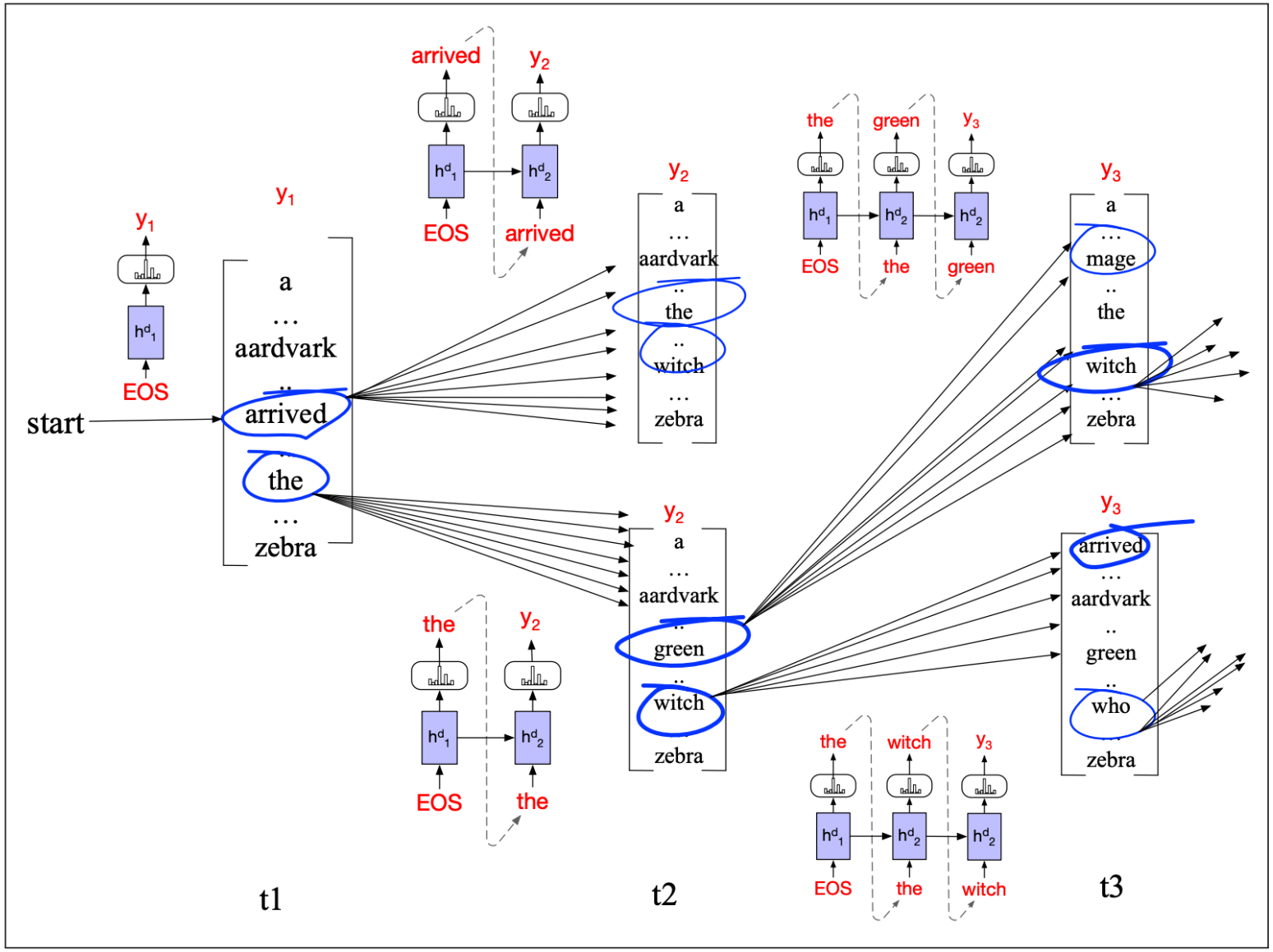
Beam Search


Time step 1
Candidates
Time step 2
Candidates
Time step 3
Candidates
Suppose **A** is the
max probability
[]
A
B
C
D
E
A
B
C
D
E
AB
ABD
C
CE
CED

Beam Search



Greedy approach cannot find the max overall probability



Beam search decoding with a beam width of k = 2. At each time step, we choose the k best hypotheses, compute the V possible extensions of each hypothesis, score the resulting k ∗V possible hypotheses and choose the best k to continue.

Source: Speech and Language Processing (3rd ed. draft) Dan Jurafsky and James H. Martin

Remember the Chain Rule...

- **Conditional Probabilities Defined**:
  - Given: $\mathrm{P}(\mathrm{B}|\mathrm{A}) = \frac{\mathrm{P}(\mathrm{A},\mathrm{B})}{\mathrm{P}(\mathrm{A})}$
  - Or: $\mathrm{P}(\mathrm{A}, B) = \mathrm{P}(\mathrm{A})\mathrm{P}(\mathrm{B}|\mathrm{A})$
- **With More Variables**:
  - $\mathrm{P}(\mathrm{A}, B, C, D) = \mathrm{P}(\mathrm{A})\mathrm{P}(\mathrm{B}|\mathrm{A})\mathrm{P}(C|A, B)\mathrm{P}(D|\mathrm{A},\mathrm{B},\mathrm{C})$
- **General Chain Rule Formula**:
  - $\mathrm{P}(\mathrm{X}_1,\mathrm{X}_2,\mathrm{X}_3,\ldots,\mathrm{X}_n) = \mathrm{P}(\mathrm{X}_1)\mathrm{P}(\mathrm{X}_2|\mathrm{X}_1)\mathrm{P}(\mathrm{X}_3|\mathrm{X}_1,\mathrm{X}_2)\ldots\mathrm{P}(\mathrm{X}_n|\mathrm{X}_1,\ldots,\mathrm{X}_{n-1})$

Remember the Chain Rule...

- **Objective**:
  - Use the Chain Rule to determine the joint probability of words in a sentence.
- **General Formula**:
  - Given a sentence: $P(w_1\ w_2\ ...\ w_n)$
  - Calculation: $P(w_1\ w_2\ \ldots\ w_n) = \prod_{i=1}^{n} P(w_i \mid w_1\ w_2\ \ldots\ w_{i-1})$
- **Example Calculation**:
  - Sentence: "The cat sat on the mat"
  - Probability Breakdown:

    P(The) *
    P(cat | The) *
    P(sat | The cat) *
    P(on | The cat sat) *
    P(the | The cat sat on) *
    P(mat | The cat sat on the)

$$P(X_1,X_2,X_3,\ldots,X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)\ldots P(X_n|X_1,\ldots,X_{n-1})$$

Score Definition for Beam Search

- y: the output sequence
- x: the context from the encoder

$$
\begin{aligned}
score(y) &= \log P(y|x) \\
&= \log \left(P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)...P(y_t|y_1,...,y_{t-1},x)\right) \\
&= \sum_{i=1}^{t} \log P(y_i|y_1,...,y_{i-1},x) \qquad (10.22)
\end{aligned}
$$

Question: Why do we use log?

Score Definition for Beam Search

- y: the output sequence
- x: the context from the encoder

$$
\begin{aligned}
score(y) &= \log P(y|x) \\
&= \log \left(P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)...P(y_t|y_1,...,y_{t-1},x)\right) \\
&= \sum_{i=1}^{t} \log P(y_i|y_1,...,y_{i-1},x) \qquad (10.22)
\end{aligned}
$$

Question: Why do we use log?

With log, we add values instead of multiplying probabilities, which might cause floating point inaccuracies

Beam Search Algorithm

Beam Search (k = beam width)

1. **Start** with initial token (e.g., <BOS>).
2. **Expand** all possible next tokens, keep top k sequences with highest log-probability.
3. **Repeat**: for each partial sequence, expand next token, keep top k combined sequences.
4. **Stop** when all sequences end with <EOS> or max length reached.
5. **Return** the sequence with highest total score.

Greedy = beam size 1

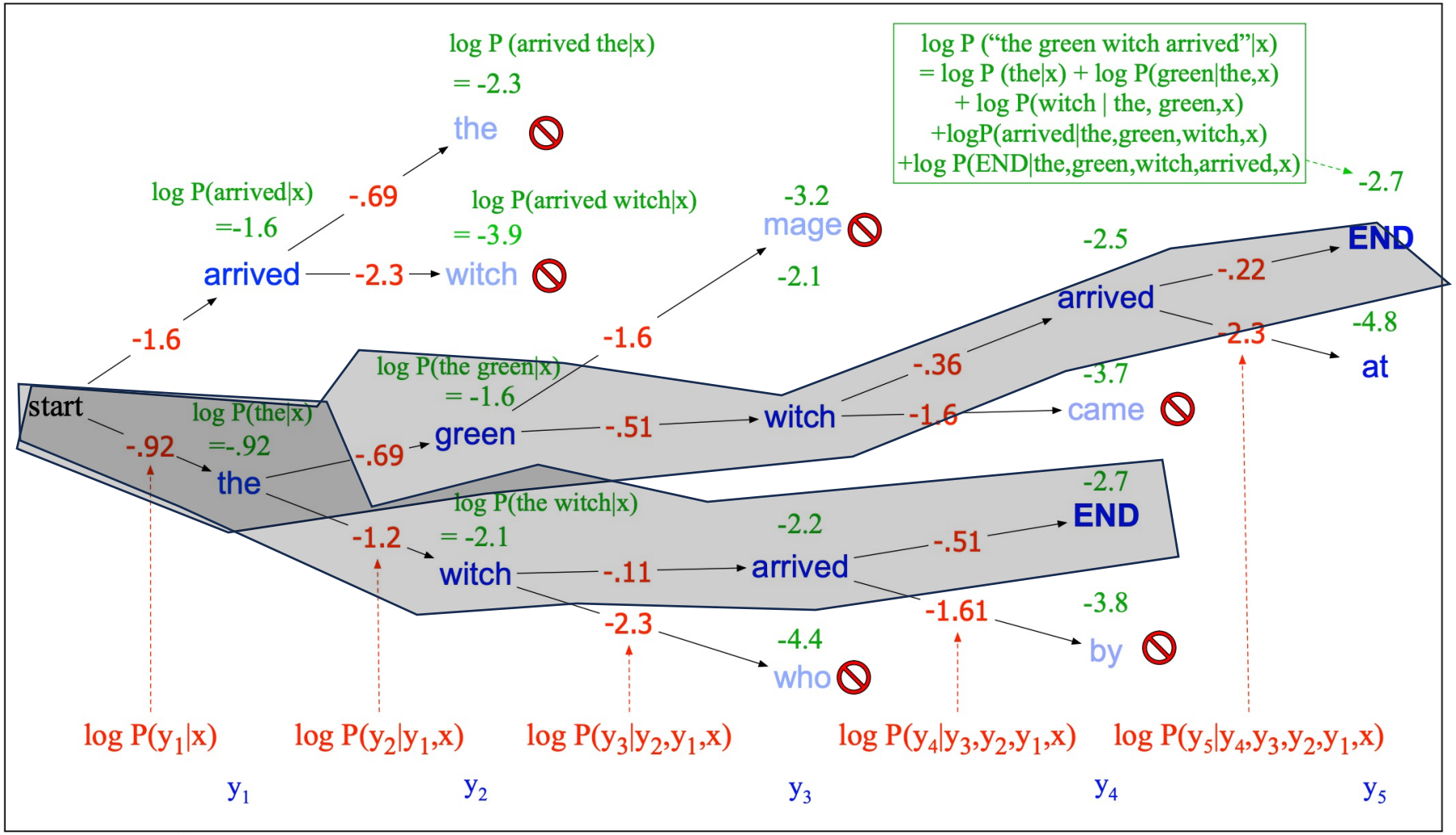


Figure 10.10 Scoring for beam search decoding with a beam width of $k = 2$. We maintain the log probability of each hypothesis in the beam by incrementally adding the logprob of generating each next token. Only the top k paths are extended to the next step.

Length normalization

- y: the output sequence
- x: the context from the encoder
- T: number of words

$$
\begin{aligned}
score(y) &= \log P(y|x) \\
&= \log\left(P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)...P(y_t|y_1,...,y_{t-1},x)\right) \\
&= \sum_{i=1}^{t} \log P(y_i|y_1,...,y_{i-1},x) \qquad (10.22)
\end{aligned}
$$

Length normalization

- y: the output sequence
- x: the context from the encoder
- T: number of words

$$
\begin{aligned}
score(y) &= \log P(y|x) \\
&= \log\left(P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)...P(y_t|y_1,...,y_{t-1},x)\right) \\
&= \sum_{i=1}^{t} \log P(y_i|y_1,...,y_{i-1},x) \qquad (10.22)
\end{aligned}
$$

$$
score(y) = -\log P(y|x) = \frac{1}{T}\sum_{i=1}^{t} -\log P(y_i|y_1,...,y_{i-1},x)
$$

Length normalization

- y: the output sequence
- x: the context from the encoder
- T: number of words

$$
\begin{aligned}
score(y) &= \log P(y|x) \\
&= \log\left(P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)...P(y_t|y_1,...,y_{t-1},x)\right) \\
&= \sum_{i=1}^{t} \log P(y_i|y_1,...,y_{i-1},x) \qquad (10.22)
\end{aligned}
$$

$$
score(y) = -\log P(y|x) = \frac{1}{T}\sum_{i=1}^{t} -\log P(y_i|y_1,...,y_{i-1},x)
$$

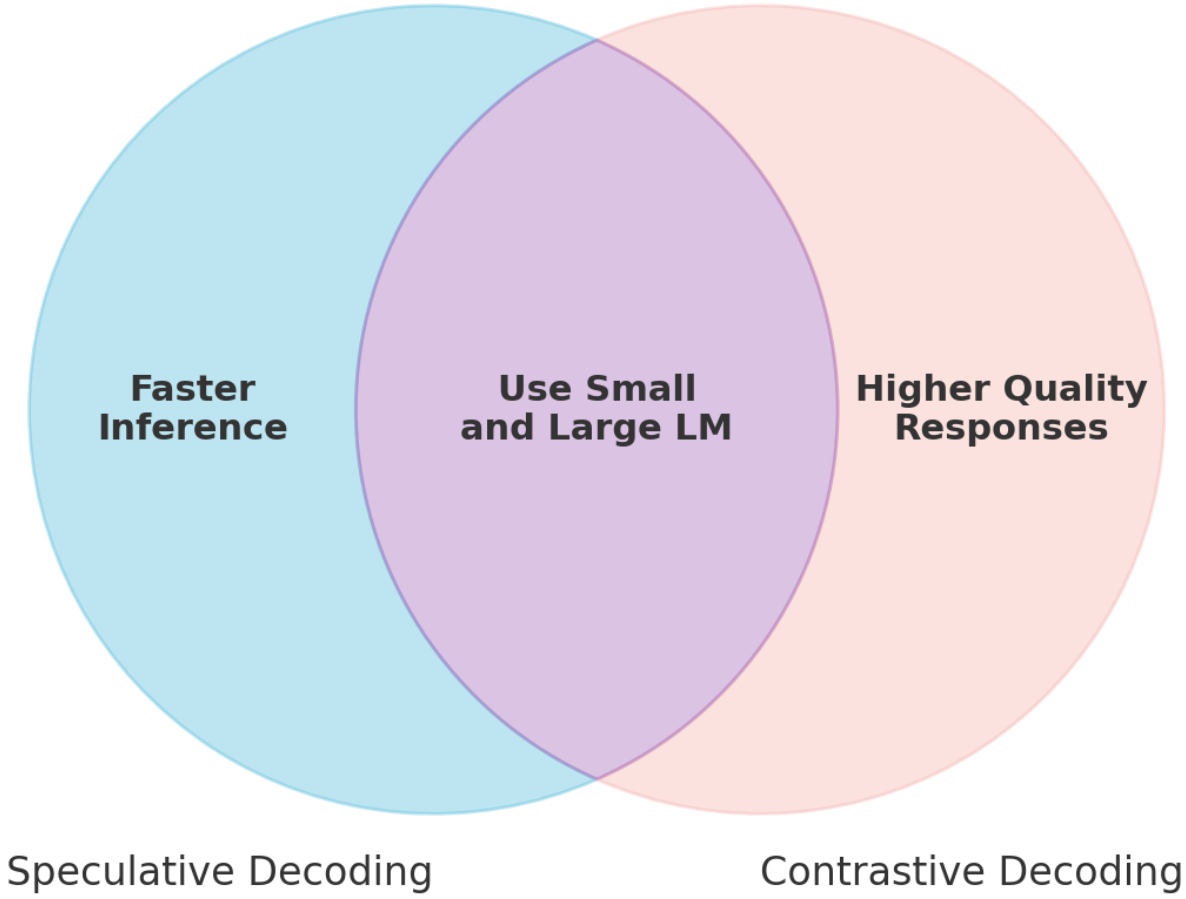
Question: Why divide by T?

Dividing by T normalizes the score in beam search to prevent bias toward shorter sequences, ensuring sequences of different lengths can be compared fairly.

Search Strategy	Pros	Cons	Runtime Complexity	Memory Complexity
Exhaustive Search	- Guarantees the optimal solution by exploring all possibilities. - Simple to understand and implement.	- Computationally very expensive. - Time-consuming and impractical for large search spaces.	Exponential (e.g., O(2^n) for binary decisions)	Exponential (e.g., O(2^n) for binary decisions)
Greedy Search	- Efficient in time and memory as it makes the best local choice. - Quick to find a solution for some problems. - Easy to implement.	- Often finds suboptimal solutions due to lack of global context. - Can get stuck in local optima and doesn't backtrack.	Potentially O(d) but varies with the problem where d is the depth of the solution space	O(1) or O(d), depending on the need to store history where d is the depth of the solution space
Beam Search	- More efficient than exhaustive search. - Balances between greedy and exhaustive by considering a subset of best options. - Can find good solutions without searching the entire space.	- No guarantee of finding the optimal solution. - The choice of beam width can significantly affect performance. - Can miss the global optimum if it's not within the beam.	O(b*d) where b is the beam width and d is the depth of the solution space	O(b*d) where b is the beam width and d is the depth of the solution space

Speculative and Contrastive Decoding

Speculative vs Contrastive Decoding

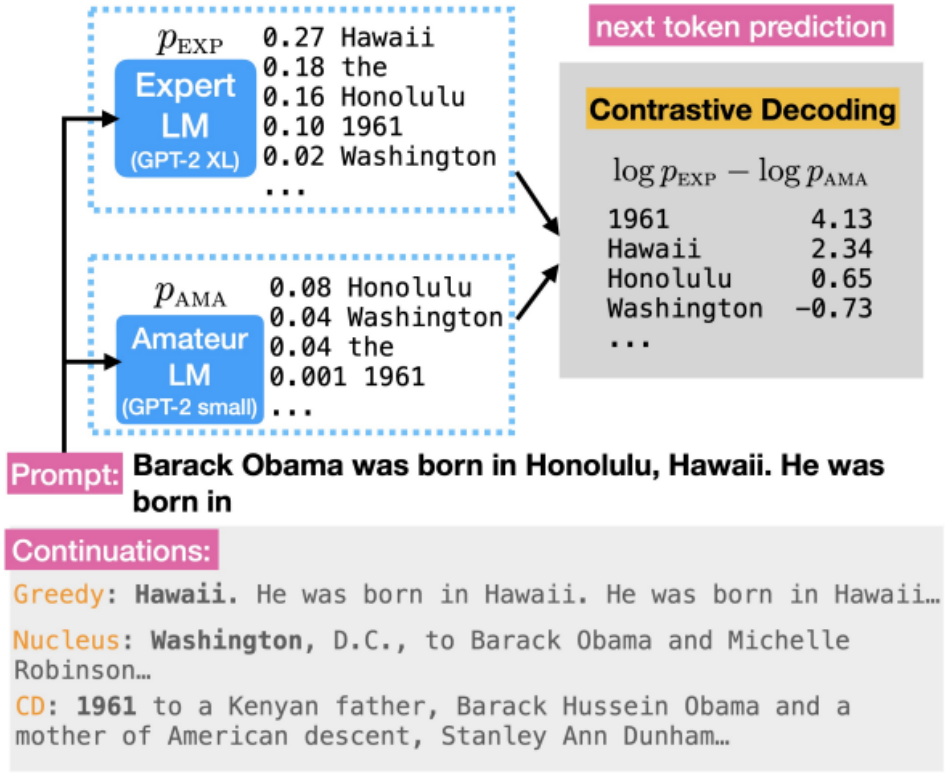


Note: this decoding is used during inference

Contrastive Decoding

- During decoding, we **compare (contrast)** the predictions of the two models to select tokens that are:
  - **High-likelihood under the target model, and**
  - **Not overly favored by the base model**
- This ensures that:
  - The target model doesn't generate generic or overconfident outputs.
  - You filter out "bland" completions the base model might suggest.

q (amateur model) is the **base model (which is fast)**.
p (expert model) is the **stronger/final model** (e.g., after RLHF or fine-tuning).



CD exploits the contrasts between expert and amateur LM of different sizes by choosing tokens that maximize their log-likelihood difference.
CD produces high-quality text that amplifies the good expert behavior and diminishes the undesired amateur behavior.

https://arxiv.org/abs/2210.15097

Contrastive & Speculative Decoding

- **Motivation**
  - Traditional decoding (greedy, beam, sampling) often struggles with:
    - Hallucinations
    - Repetition
    - Slow inference in large models
- **Contrastive Decoding**
  - Balances relevance and fluency
- Selects tokens that are:
  - The token has a **high probability** according to the base model's output distribution
  - But also **informative** or diverse compared to the context

Analogy
It's like asking:
"Which token makes sense **according to a fluent speaker**, but also adds something new or non-redundant?"

Contrastive Decoding

Tunable parameter

$$\text{Score} = \text{Relevance} - \lambda \cdot \text{Degeneration Penalty}$$

Comes from amateur (**or base**) model
This discourages repetitive or boring text — the penalty increases if:
•A token is already frequent in the context
•The token brings little **novelty** or **semantic diversity**

$$\text{Penalty} \propto \text{similarity}(x_i, \text{context window})$$

Comes from the **expert model**
High relevance → token is grammatically correct, fits well

q (amateur model) is the **base model (which is fast)**.
p (expert model) is the **stronger/final model** (e.g., after RLHF or fine-tuning).

Choose the next token that is:
•Likely under a fluent base model
•But **not too repetitive**
•Balancing **safe fluency** vs. **interesting generation**
Combined, it reduces repetition without sacrificing quality

Example: Contrastive Decoding in Action

Prompt:
"The cat sat on the"

Token	Relevance (log P from base LM)	Degeneration Penalty (similarity)	λ	Final Score
mat	-0.2	0.1	2	-0.2 - 2×0.1 = **-0.4**
roof	-0.5	0.0	2	-0.5 - 2×0.0 = **-0.5**
cat	-0.1	0.5 (repetitive!)	2	-0.1 - 2×0.5 = **-1.1**

- **"mat"** is highly relevant (fits grammatically), slight repetition penalty
- **"roof"** is less likely, but not repetitive
- **"cat"** is highly likely under base LM, but repetitive — so gets heavily penalized

Another Example

- A similar formulation:
  - We favor tokens with high probability under the target model but not overly favored by the base model
  - First Discard all tokens that do not have sufficiently high probability under the base model

$$\text{Score}(x) = \log p_{\text{target}}(x) - \alpha \log p_{\text{base}}(x)$$

We **first restrict** the candidate set by:

$$x \in \{x' \mid p_{\text{base}}(x') \geq \beta \cdot \max_{x''} p_{\text{base}}(x'')\}$$

Where:

- $\beta \in [0, 1]$ is a threshold (like 0.5 or 0.6)
- This ensures we're only considering tokens that the **base model (expert)** thinks are at least "good enough"
- Then we score and select from that filtered list

q (amateur model) is the **base model (which is fast)**.
p (expert model) is the **stronger/final model** (e.g., after RLHF or fine-tuning).

Why Base Model Contributes the Penalty?

- Because the **base model** is often:
  - **Smaller**
  - **Less trained**
  - **More prone to generic, boring, or repetitive completions**
- **Contrastive Decoding Idea:**
  - If the **base model is very confident** in a token,
  - But the **expert isn't much more confident**, then the token is probably:
    - Generic (e.g., "I don't know", "Yes.", etc.)
    - **Degenerate** (repetitive, dull, vague)
  - So, we **penalize** tokens the **base model overpredicts**.

p = **expert model** (larger, higher-quality)
q = **base model** (smaller, faster)

$$\log p(x) - \log q(x) = \log\left(\frac{p(x)}{q(x)}\right)$$

So:

- If $p \gg q$ → positive → **expert strongly prefers it**
- If $p \approx q$ → near 0 → **expert agrees with base**
- If $p \ll q$ → negative → **base overpredicts**, expert disagrees → **penalize**

negative log contrast = signal of degeneration (low-quality or repetitive output)

overpredict: base model assigns a high probability to a token — it's very "confident" — **but it shouldn't be** because the token is generic, repetitive, or unhelpful

Another Example

p (expert model)	
1961	0.10
Hawaii	0.32
the	0.18
Honolulu	0.16
Washington	0.02

q (amateur model)	
1961	0.001
Hawaii	0.07
the	0.04
Honolulu	0.04
Washington	0.01

=

log(p) - log(q)	
1961	4.61
Hawaii	1.52
the	1.50
Honolulu	1.39
Washington	0.69

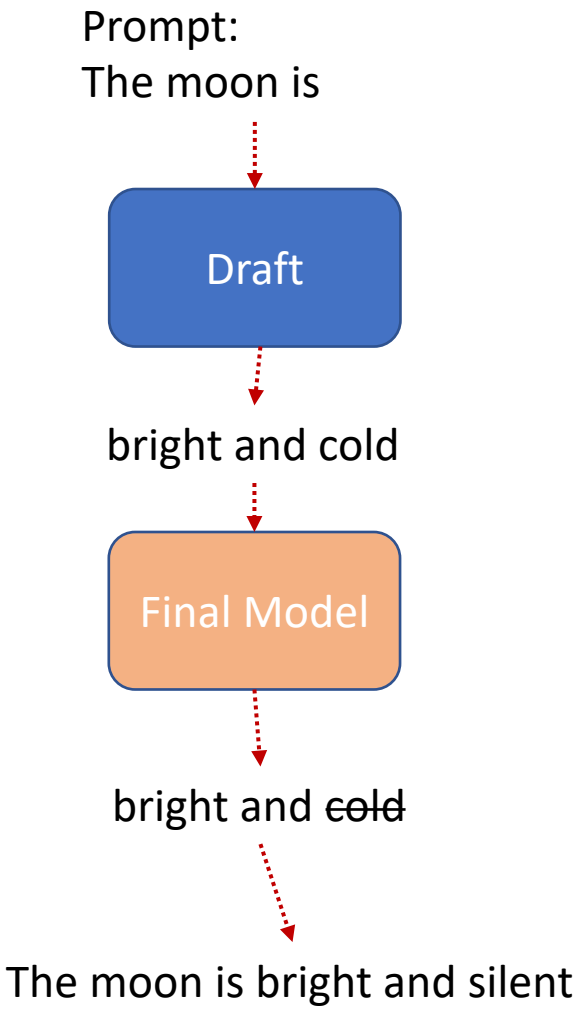
[Start] Barack Obama was born in Honolulu, Hawaii. He was born in **1961.**

q (amateur model) is the **base model**.
p (expert model) is the **stronger/final model** (e.g., after RLHF or fine-tuning).

Source: https://towardsdatascience.com/combining-large-and-small-llms-for-inference-time-and-quality-boosts-1779b6b5100b/

Speculative Decoding

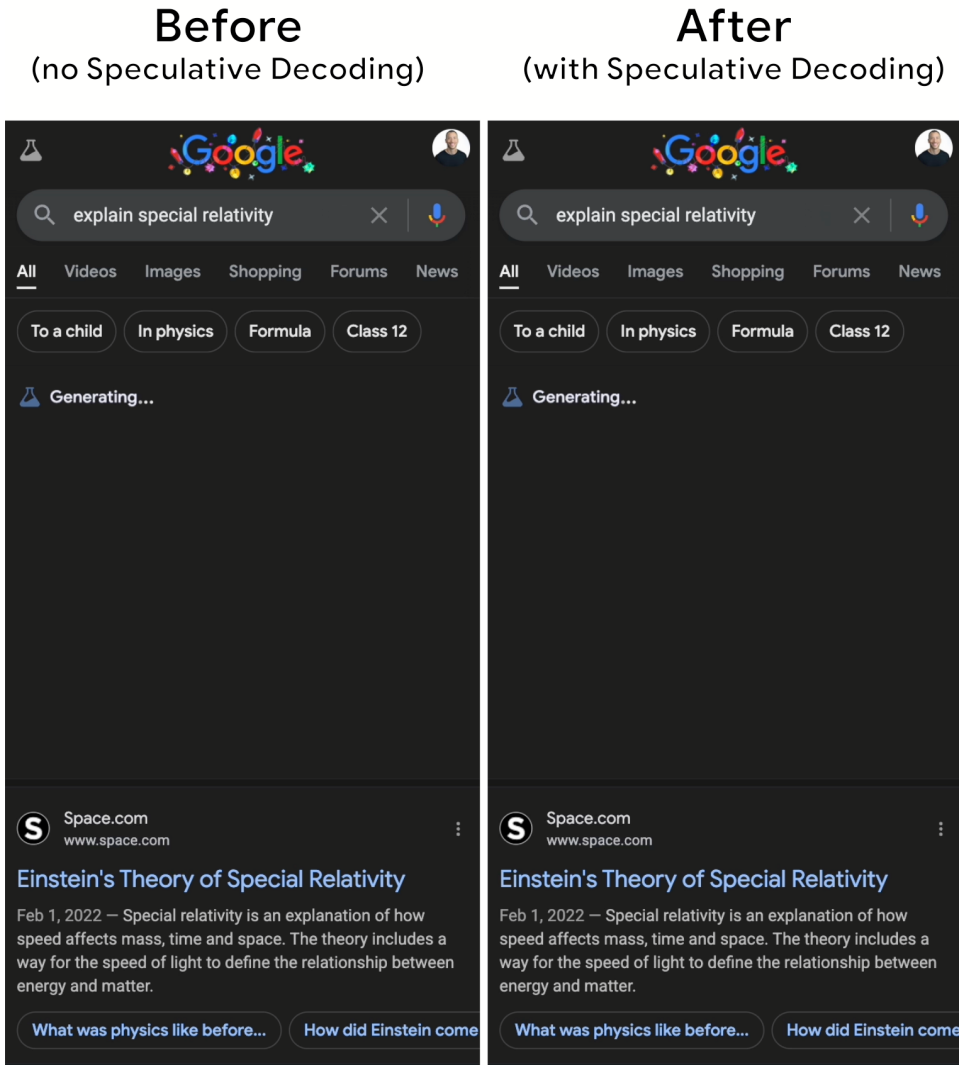
- Speed up inference in large language models
  - by combining a **fast draft model** with a **powerful final model**
- **Prompt**:
  "The moon is"
  - **Draft Model (small)** generates multiple tokens:
    → *["bright", "and", "cold"]*
  - **Final Model (large)** verifies all tokens **in parallel**:
  - ✅ "bright" → accepted
  - ✅ "and" → accepted
  - ❌ "cold" → rejected
  - **Final Model resumes decoding** from the rejected token



Analogy:
Like letting an intern draft your sentences —
the expert only steps in to tweak or fix what's off.

Speculative Decoding

- Introduced by Google (2022) to **speed up inference** in LLMs
  - Uses a **small draft model** to predict tokens
  - Large model **verifies** predictions in parallel
- **Benefits**
  - **2x–3x speedup** in real-world tasks (e.g., translation, summarization)
  - No compromise in **output quality**



https://research.google/blog/looking-back-at-speculative-decoding/

Workflow

Terms

q: Logits from **amateur model**

p: Logits from **expert model**

1.Draft Generation

1. Amateur model generates **n tokens** in parallel

2.Verification

1. Expert model evaluates all n tokens in a batch

3.Acceptance Rules (per token x):

1. ✅ Accept if: $p(x)>q(x)$
2. ❌ Reject with probability: $1-p(x)/q(x)$

4. On Rejection:

1. Use expert model to sample next token
2. Reset both caches and repeat the n-token draft + check

[Start] Barack Obama was born on Hawaii

[Start] Barack Obama was born on ~~Hawaii~~ August

[Start] Barack Obama was born on August 12th , 1964 ~~before~~ in

[Start] Barack Obama was born on August 12th , 1964 in Honolulu , Hawaii .

[Start] Barack Obama was born on August 12th , 1964 in Honolulu , Hawaii .

Here, the blue signifies accepted tokens, and red/green signify tokens rejected and then sampled from the expert or adjusted distribution.

Speedup

- Shifting work from the **slow expert model** to the **fast amateur model**, while keeping quality high.
  - **Standard Generation (Slow)**
    - Expert model generates **one token at a time**
    - Each step needs a full forward pass: n tokens → n expert passes
  - **Speculative Decoding (Fast)**
    1. **Amateur model** generates n tokens in one pass
    2. **Expert model** checks all n tokens in **one batch**
    3. If no rejection → you get n tokens at the cost of **1 expert pass** → Up to **n×** speedup!

Speculative Decoding

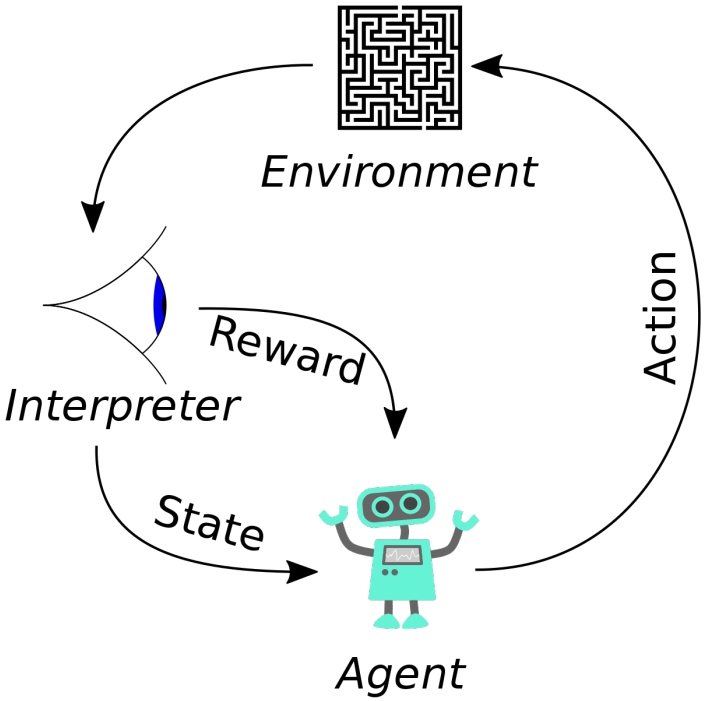
An animation, demonstrating the speculative decoding algorithm in comparison to standard decoding. The text is generated by a large GPT-like transformer decoder. In the case of speculative decoding, a much smaller model is used as the guessing mechanism. The accepted guesses are shown in green and the rejected suggestions in red.

Step 5: Fine Tuning

- Domain-Specific Training:
  - Fine-tuned on specific types of dialogues, such as customer service, casual conversation, or domain-specific knowledge.
- Reinforcement Learning:
  - Model is further refined using reinforcement learning based on user interactions and feedback.

Reinforcement Learning

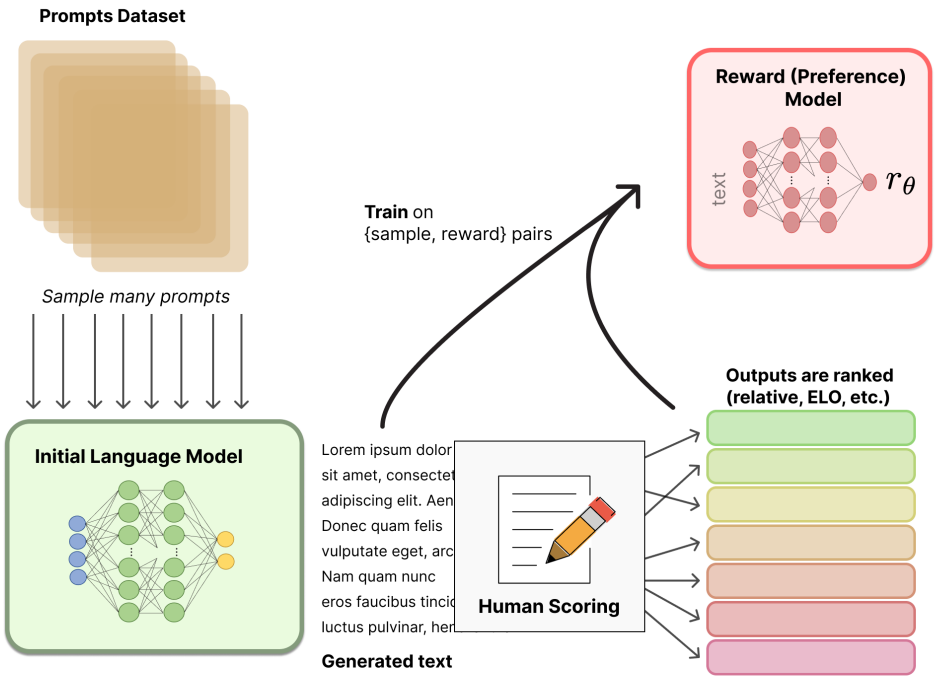
- Reinforcement Learning:
  - An **agent** takes **actions** in an **environment**, which is interpreted into a **reward** and a representation of the **state**, which are fed back into the agent.
    - Agent:
      - The agent is the transformer model itself.
    - Environment:
      - The environment includes the context of the conversation and the interactions with the user.
    - Reward:
      - Rewards are assigned based on the effectiveness of the chatbot's responses.
        - Satisfaction, appropriateness of the response, length of the conversation, etc.
        - Negative rewards: Generating irrelevant, repetitive, or inappropriate responses.



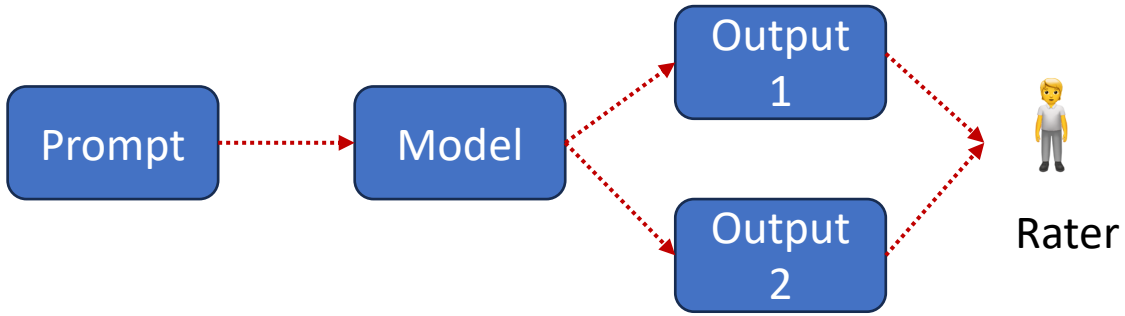
Source: Wikipedia

Reinforcement Learning with Human Feedback (**RLHF**)

- **Motivation**:
  - Human feedback has high quality, but it's time consuming to obtain.
  - Let's train a model as a proxy instead!
- **Prompts Dataset**
  - A dataset of prompts is used to generate multiple pieces of text.
- **Initial Language Model**
  - The LM generates text based on the sampled prompts.
- **Human Scoring**:
  - The generated texts are scored by humans.



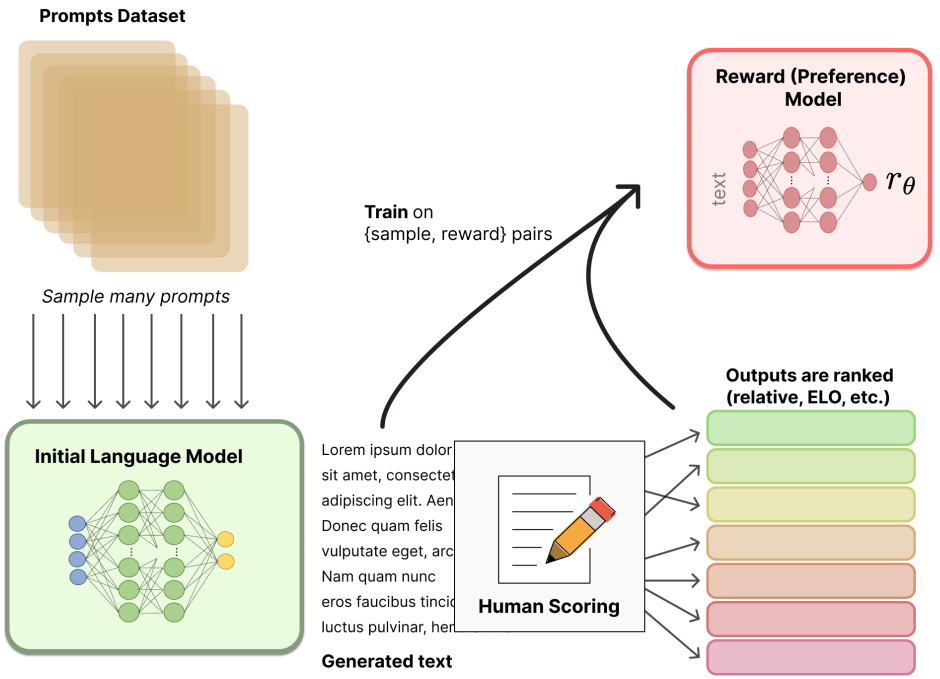
Source: Hugging Face



Human feedback is often preference ranking (e.g., which of two responses is better)

Reinforcement Learning with Human Feedback (**RLHF**)

- **Motivation**:
  - Human feedback has high quality, but it's time consuming to obtain.
  - Let's train a model as a proxy instead!
- **Prompts Dataset**
  - A dataset of prompts is used to generate multiple pieces of text.
- **Initial Language Model**
  - The LM generates text based on the sampled prompts.
- **Human Scoring**:
  - The generated texts are scored by humans.

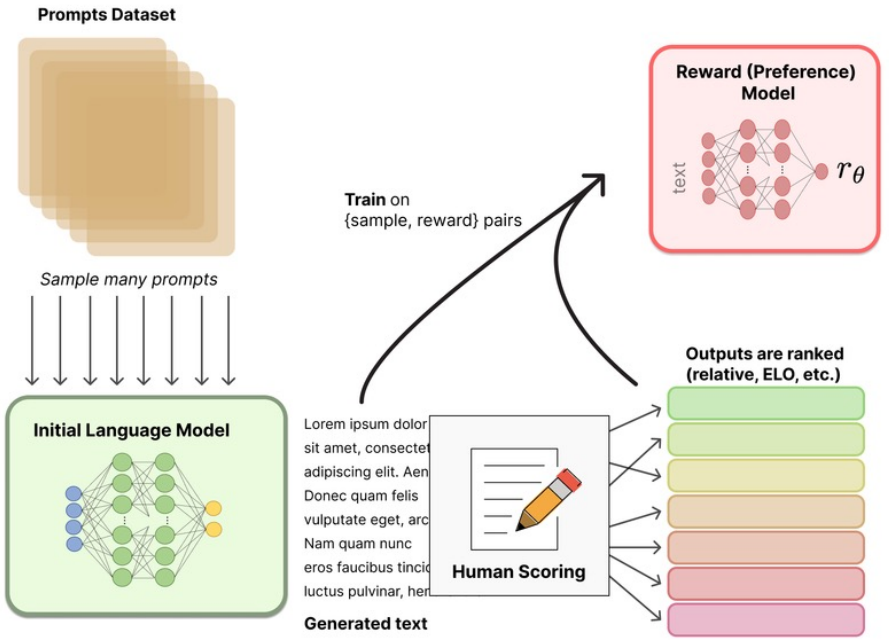


Source: Hugging Face

Prompt	Response	Reward
What is the weather like in London today?	It is currently raining in London.	1
	I'm not sure what the weather is like in London today.	0

Step 1: Training the Reward Model in RLHF

- **Goal:**
  - Train a **Reward Model (RM)** r_θ to predict how well a model-generated response aligns with human preferences.



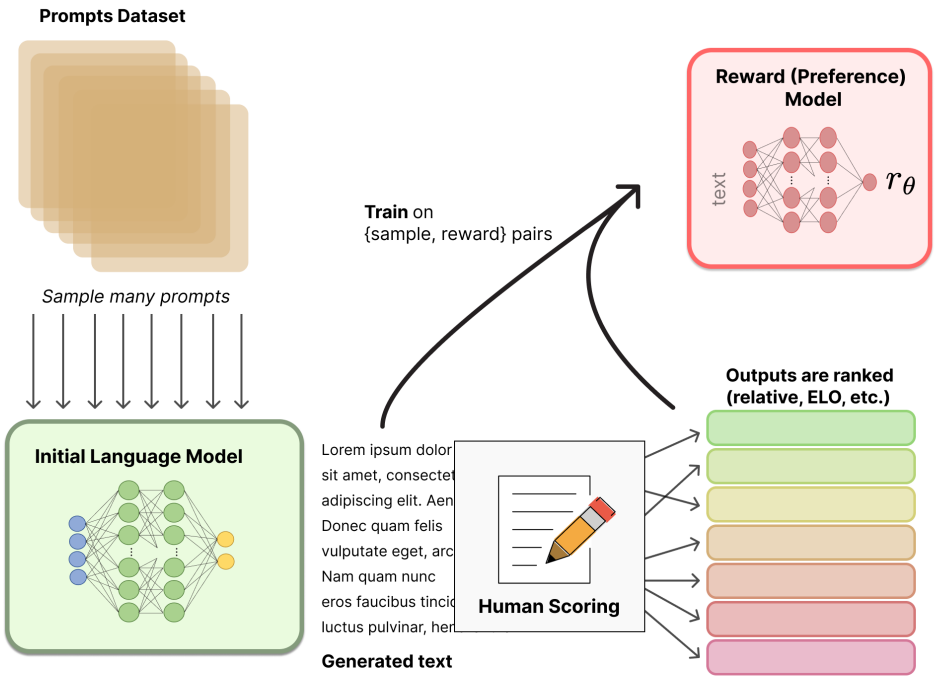
Source: Hugging Face

$$r_\theta(\text{prompt}, \text{response}) \rightarrow \text{reward}$$

Response A > Response B” → label: A=1.0, B=0.2

Step 1: Training the Reward Model in RLHF

- **Goal:**
  - Train a **Reward Model (RM)** r_θ to predict how well a model-generated response aligns with human preferences.
- **Workflow**
  - **Sample Prompts**
    - Select prompts from a dataset
    - Generate multiple responses using the **initial language model**



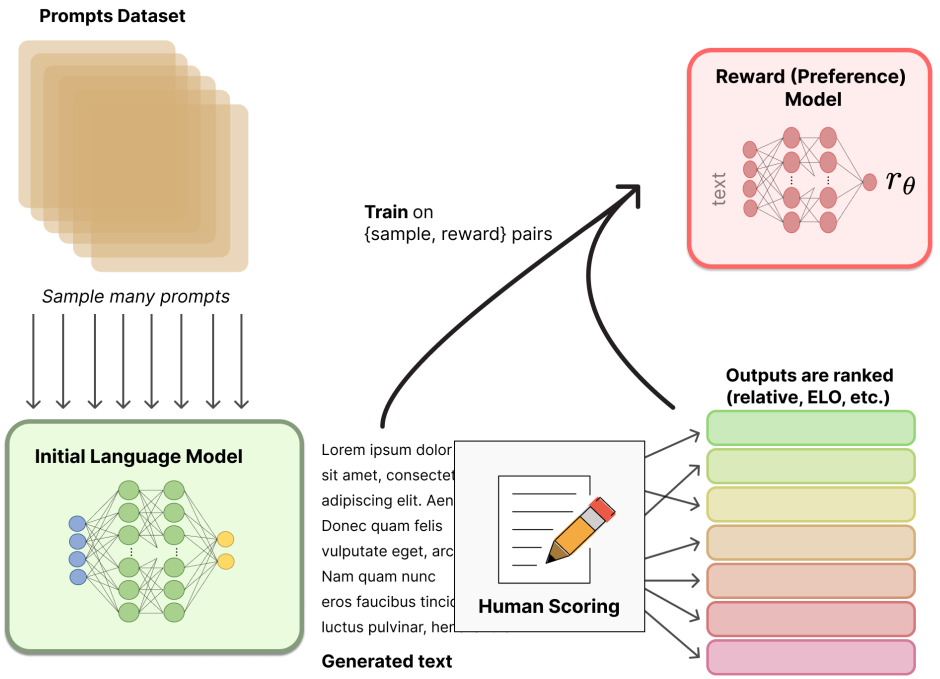
Source: Hugging Face

$$r_\theta(\text{prompt}, \text{response}) \rightarrow \text{reward}$$

Response A > Response B" → label: A=1.0, B=0.2

Step 1: Training the Reward Model in RLHF

- **Goal:**
  - Train a **Reward Model (RM)** r_θ to predict how well a model-generated response aligns with human preferences.
- **Workflow**
  - **Sample Prompts**
    - Select prompts from a dataset
    - Generate multiple responses using the **initial language model**
  - **Collect Human Feedback**
    - Human annotators **rank responses** (e.g., preferred > less preferred)
    - Ratings are used as training labels



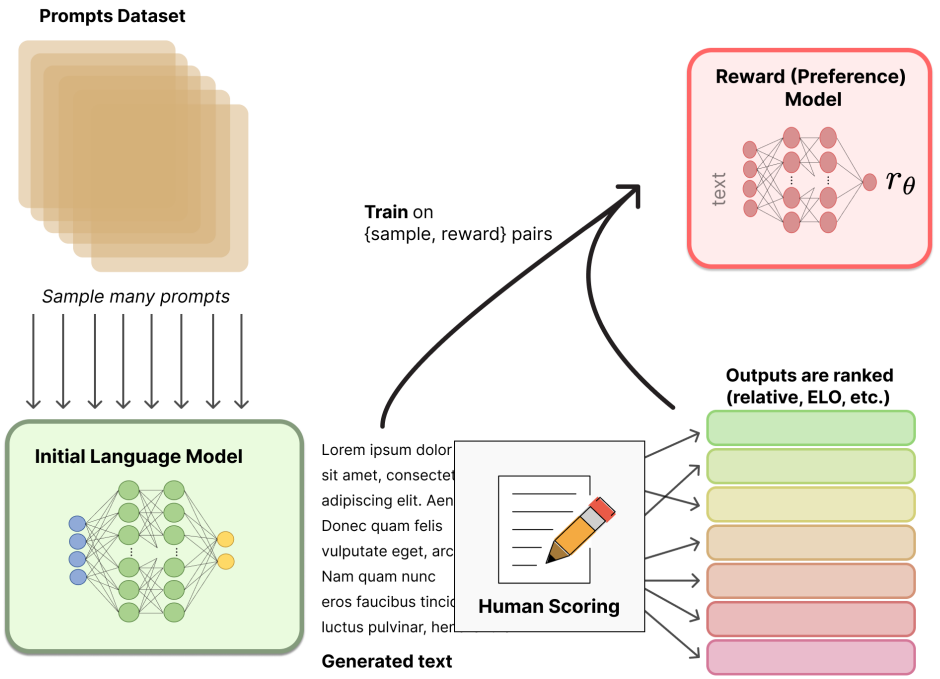
Source: Hugging Face

$$r_\theta(\text{prompt}, \text{response}) \rightarrow \text{reward}$$

Response A > Response B" → label: A=1.0, B=0.2

Step 1: Training the Reward Model in RLHF

- **Goal:**
  - Train a **Reward Model (RM)** r_θ to predict how well a model-generated response aligns with human preferences.
- **Workflow**
  - **Sample Prompts**
    - Select prompts from a dataset
    - Generate multiple responses using the **initial language model**
  - **Collect Human Feedback**
    - Human annotators **rank responses** (e.g., preferred > less preferred)
    - Ratings are used as training labels
  - **Train Reward Model**
    - Input: (prompt, response)
    - Target: scalar reward (based on ranking)
    - The RM learns to **score new outputs** based on learned preferences



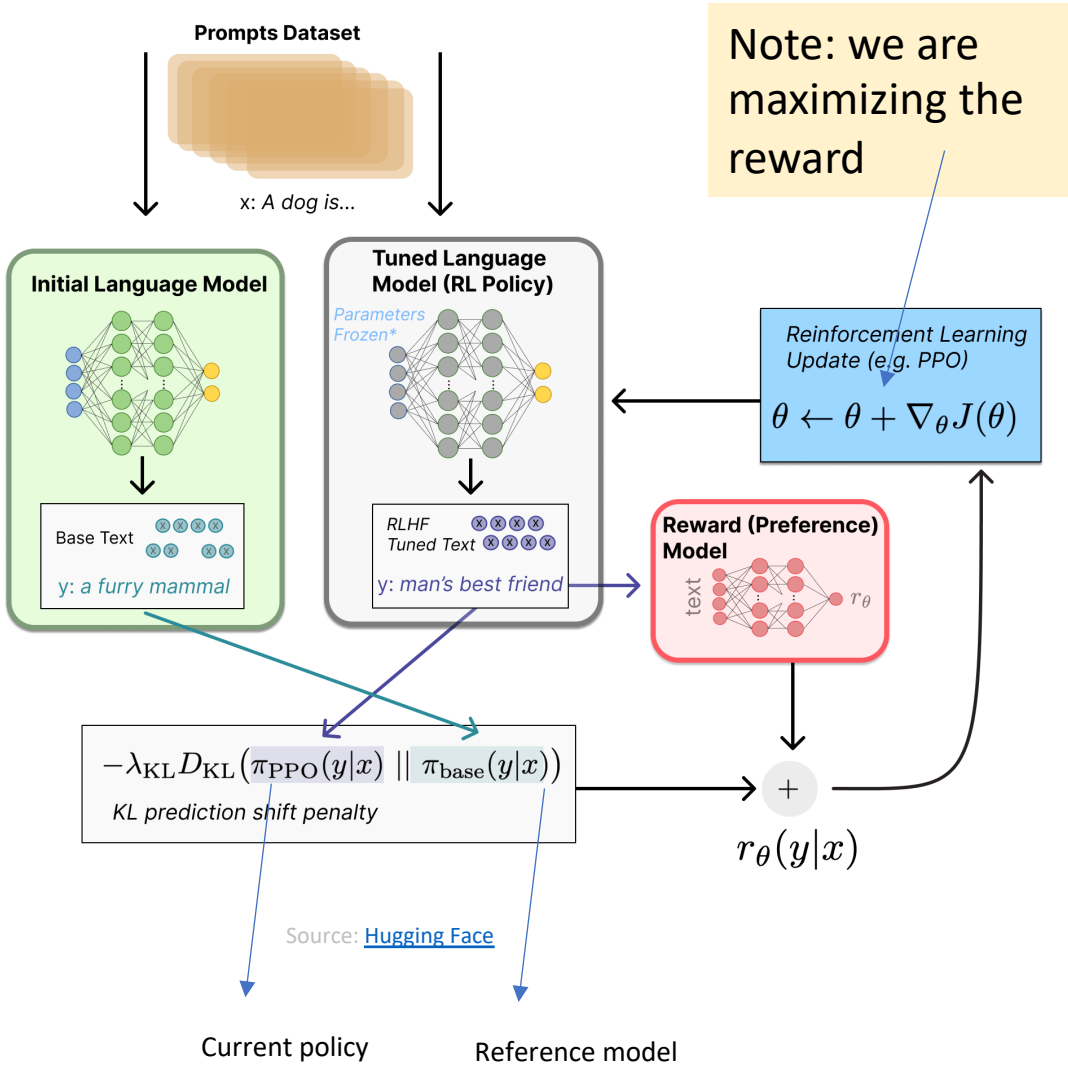
Source: Hugging Face

$$r_\theta(\text{prompt}, \text{response}) \rightarrow \text{reward}$$

Response A > Response B" → label: A=1.0, B=0.2

Step 2: Fine-Tuning with PPO in RLHF

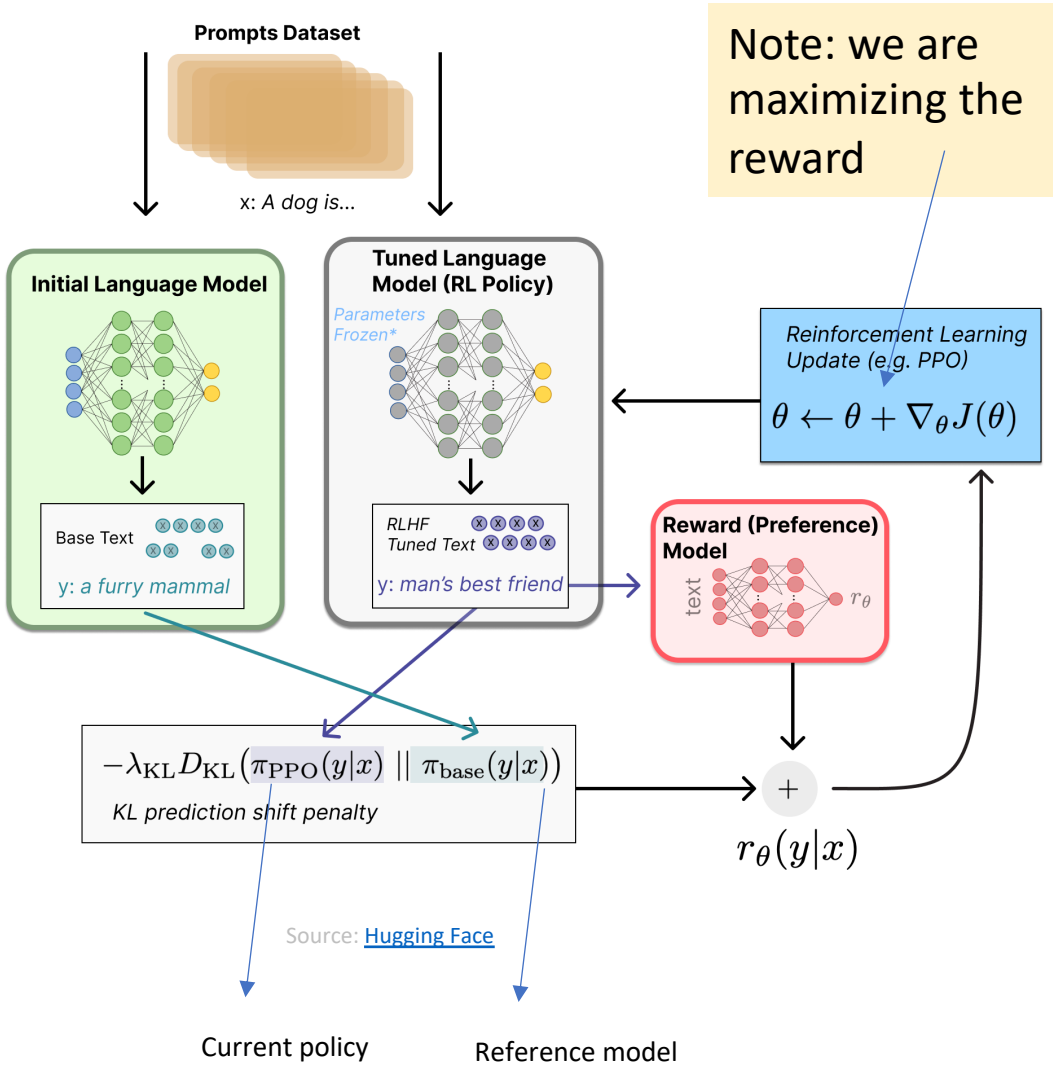
- **Prompt:**
  - x: “A dog is…”



$$J(\theta) = \mathbb{E}_{x,y\sim\pi_\theta}\left[r_\theta(x,y) - \lambda \cdot D_{KL}(\pi_\theta(y|x) \,\|\, \pi_{\text{base}}(y|x))\right]$$

Step 2: Fine-Tuning with PPO in RLHF

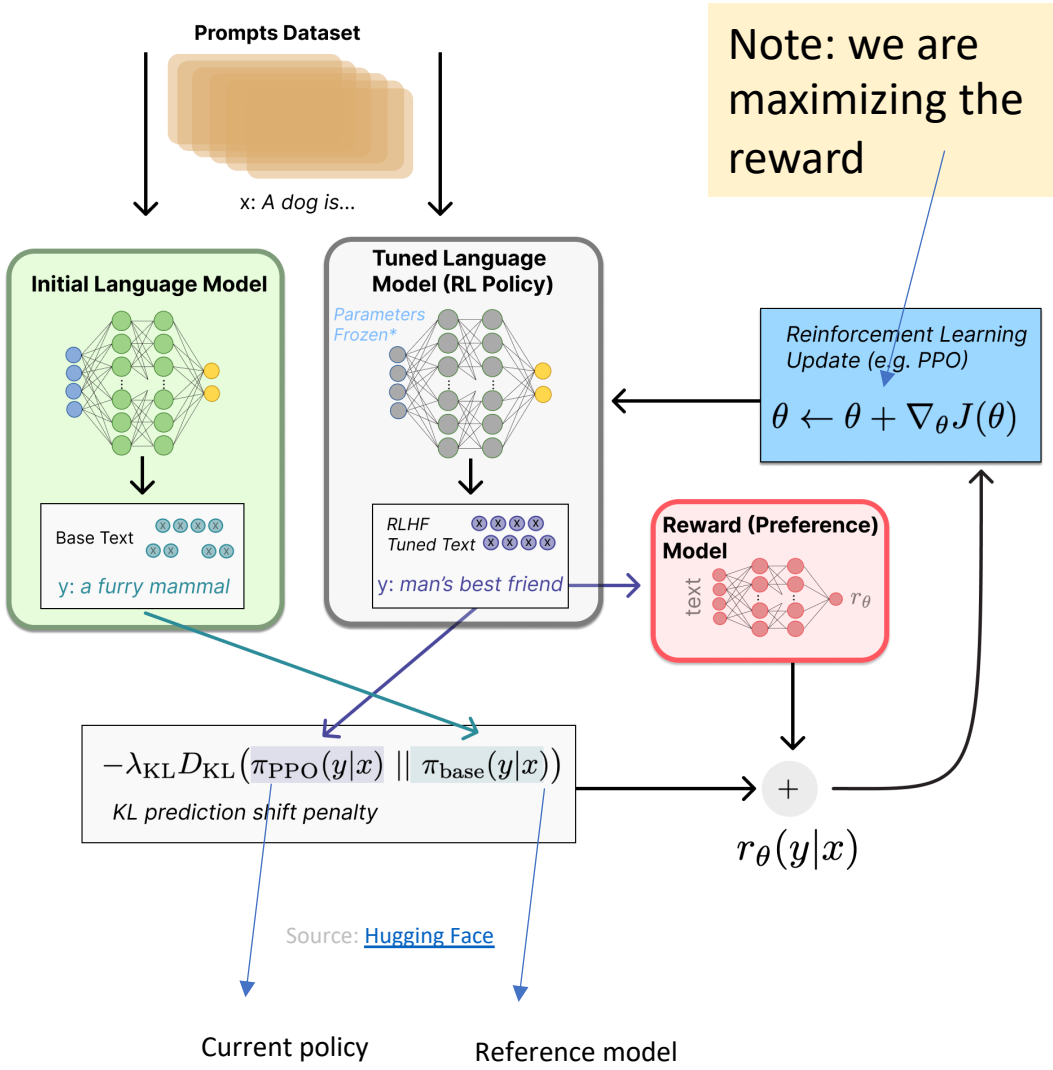
- **Prompt:**
  - x: “A dog is…”
- **Initial Language Model**:
  - The initial model generates a base text output
    - (e.g., y = "a furry mammal") in response to the prompt.



$$J(\theta) = \mathbb{E}_{x,y\sim\pi_\theta}\left[r_\theta(x,y) - \lambda \cdot D_{KL}(\pi_\theta(y|x) \,\|\, \pi_{\text{base}}(y|x))\right]$$

Step 2: Fine-Tuning with PPO in RLHF

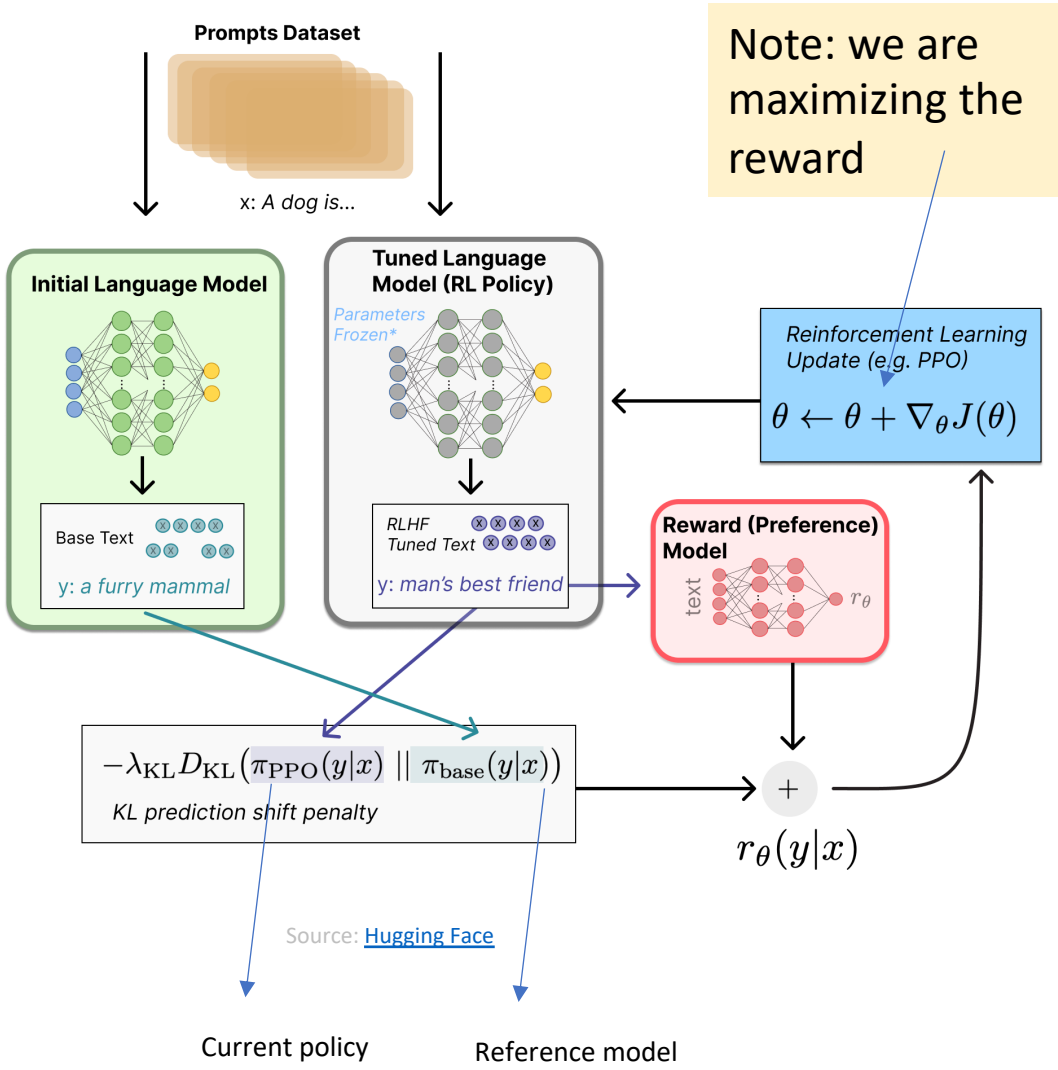
- **Prompt:**
  - x: “A dog is…”
- **Initial Language Model**:
  - The initial model generates a base text output
    - (e.g., y = "a furry mammal") in response to the prompt.
- **Tuned Language Model (RL Policy)**:
  - The model is then fine-tuned with some parameters possibly frozen producing a more refined output
  - (e.g., y = "man's best friend").



$$J(\theta) = \mathbb{E}_{x,y\sim\pi_\theta}\left[r_\theta(x,y) - \lambda \cdot D_{KL}(\pi_\theta(y|x) \,\|\, \pi_{\text{base}}(y|x))\right]$$

Step 2: Fine-Tuning with PPO in RLHF

- **Prompt:**
  - x: “A dog is…”
- **Initial Language Model**:
  - The initial model generates a base text output
    - (e.g., y = "a furry mammal") in response to the prompt.
- **Tuned Language Model (RL Policy)**:
  - The model is then fine-tuned with some parameters possibly frozen producing a more refined output
  - (e.g., y = "man's best friend").
- **Reward (Preference) Model**:
  - Assigns a reward score $r_\theta(y|x)$, indicating how well it aligns with human preferences.

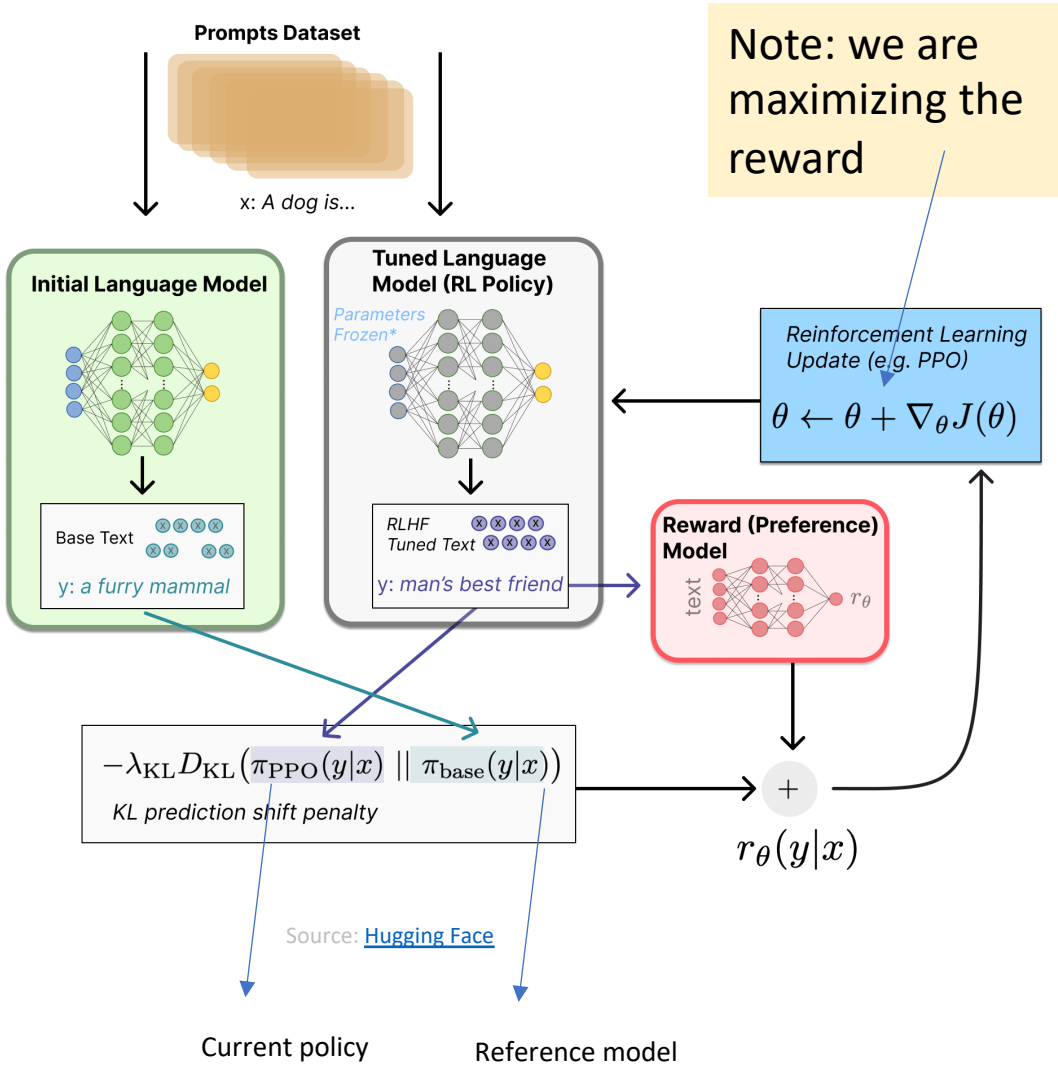


Note: we are maximizing the reward

$$J(\theta) = \mathbb{E}_{x,y\sim\pi_\theta}\left[r_\theta(x,y) - \lambda \cdot D_{KL}(\pi_\theta(y|x) \,||\, \pi_{\text{base}}(y|x))\right]$$

Step 2: Fine-Tuning with PPO in RLHF

- **Prompt:**
  - x: “A dog is…”
- **Initial Language Model**:
  - The initial model generates a base text output
    - (e.g., y = "a furry mammal") in response to the prompt.
- **Tuned Language Model (RL Policy)**:
  - The model is then fine-tuned with some parameters possibly frozen producing a more refined output
  - (e.g., y = "man's best friend").
- **Reward (Preference) Model**:
  - Assigns a reward score $r_\theta(y|x)$, indicating how well it aligns with human preferences.
- **Reinforcement Learning Update (e.g., PPO)**:
  - The parameters θ of the language model are updated by taking gradients of the reward score.

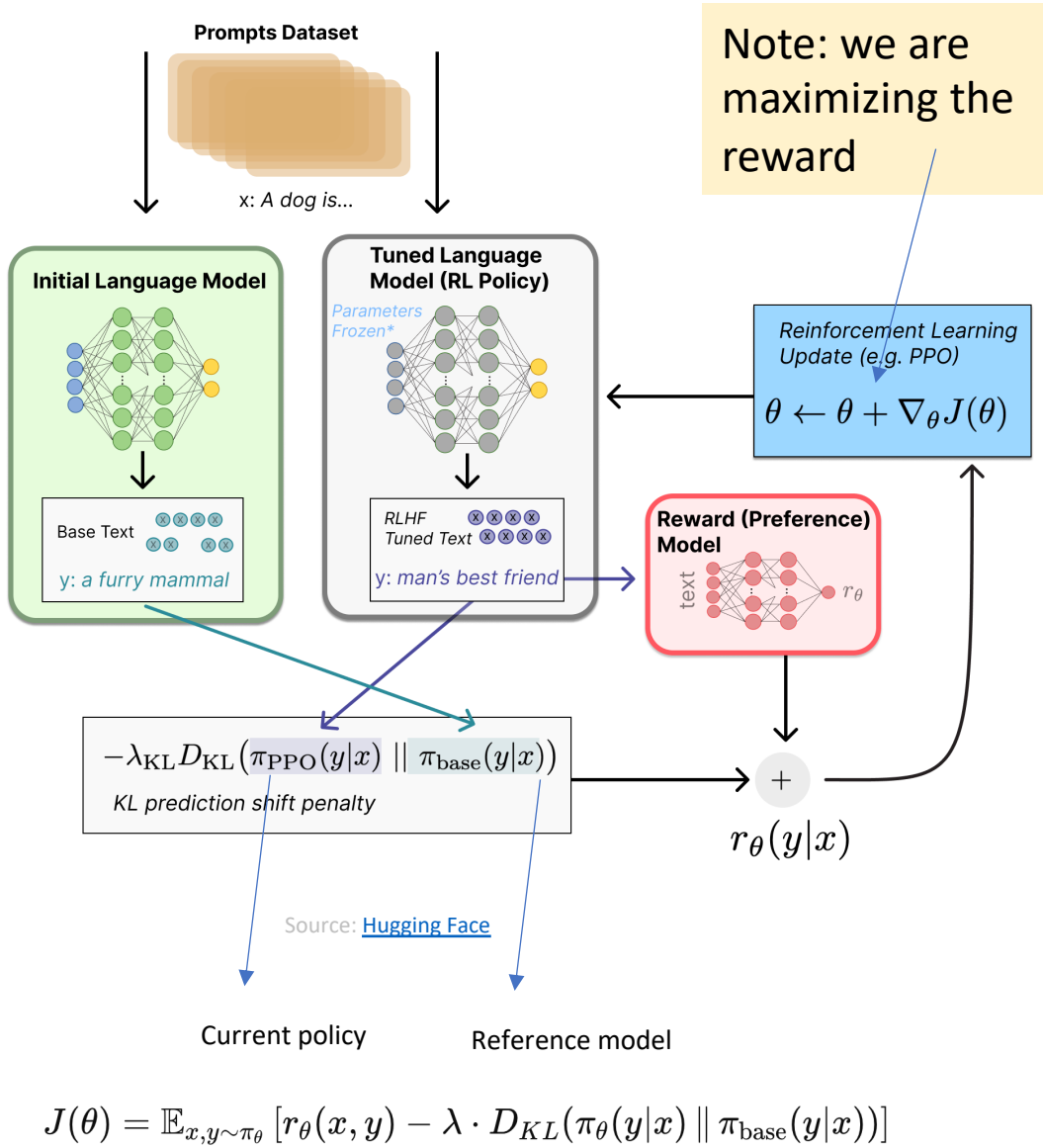


Note: we are maximizing the reward

$$J(\theta) = \mathbb{E}_{x,y\sim\pi_\theta}\left[r_\theta(x,y) - \lambda \cdot D_{KL}(\pi_\theta(y|x) \,\|\, \pi_{\text{base}}(y|x))\right]$$

Step 2: Fine-Tuning with PPO in RLHF

- **Prompt:**
  - x: “A dog is…”
- **Initial Language Model**:
  - The initial model generates a base text output
    - (e.g., y = "a furry mammal") in response to the prompt.
- **Tuned Language Model (RL Policy)**:
  - The model is then fine-tuned with some parameters possibly frozen producing a more refined output
  - (e.g., y = "man's best friend").
- **Reward (Preference) Model**:
  - Assigns a reward score $r_\theta(y|x)$, indicating how well it aligns with human preferences.
- **Reinforcement Learning Update (e.g., PPO)**:
  - The parameters θ of the language model are updated by taking gradients of the reward score.
- **KL Prediction Shift Penalty**:
  - There's a penalty for shifts in prediction (the KL divergence) between the fine-tuned model's outputs and the base model's outputs
  - Ensure the text doesn't drift too far from the original model's language understanding.



Note: we are maximizing the reward

$$J(\theta) = \mathbb{E}_{x,y\sim\pi_\theta}\left[r_\theta(x,y) - \lambda \cdot D_{KL}(\pi_\theta(y|x) \,\|\, \pi_{\text{base}}(y|x))\right]$$

Advanced Considerations in RLHF

- **Considerations**
  - *Reward Hacking*: The model might exploit quirks in the reward model.
    - Model learns to "game" the reward.
    - Human oversight or iterative tuning is important
  - *Labeler Bias*: Human feedback can introduce subjective bias.
  - *Reward Overfitting*: Model may over-optimize on imperfect rewards.
    - If overused, the policy may overfit to the quirks of the reward model instead of generalizing to true human preferences.
- **Iterative Improvement**
  - Use human feedback *after PPO* to retrain the reward model.
  - Enables continuous improvement (e.g., InstructGPT, Constitutional AI).

Summary

- **RLHF = Reinforcement Learning with Human Feedback**
  Aligns models with human preferences.
  - **Step 1: Supervised Fine-Tuning (SFT)**
    Train model on human-written examples.
  - **Step 2: Train Reward Model**
    Learn to score responses based on human rankings.
  - **Step 3: Reinforcement Learning (e.g., PPO)**
    Fine-tune the model to maximize reward while staying close to the base model.
- **Goal:**
  Produce outputs that are helpful, safe, and aligned with human values.

In theory, RLHF can be done without a reward model by using humans to directly score outputs during training. This gives more accurate feedback but is costly and slow.

But this would be **Much more expensive** and requires **humans-in-the-loop for every training step.**

Reinforcement Learning from AI Feedback (RLAIF)

- **Overview:**
  - RLAIF replaces human feedback with AI-generated feedback, guided by a **constitution** (set of ethical principles).
  - Goal:
    - Align AI behavior with human values at **scale**, improving efficiency and consistency.
- **Benefits Over RLHF:**
  - **Scalable** – less dependence on human annotators.
  - **Consistent** – feedback aligned with fixed ethical rules.
  - **Faster** – more efficient training and deploymen.

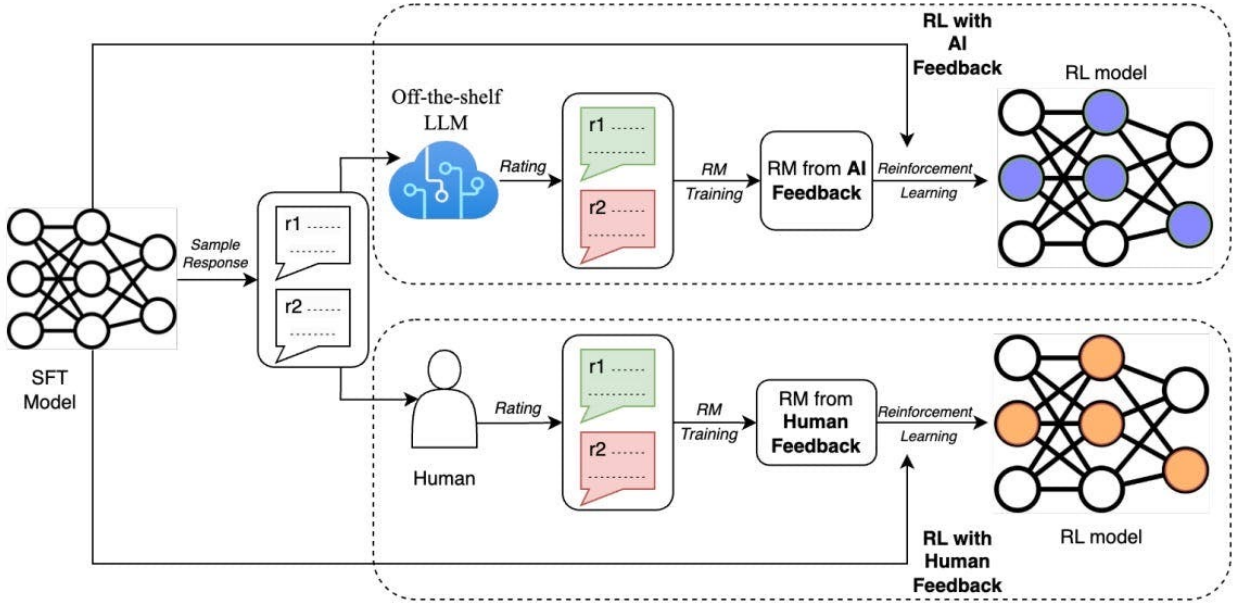


Figure 2: A diagram depicting RLAIF (top) vs. RLHF (bottom)

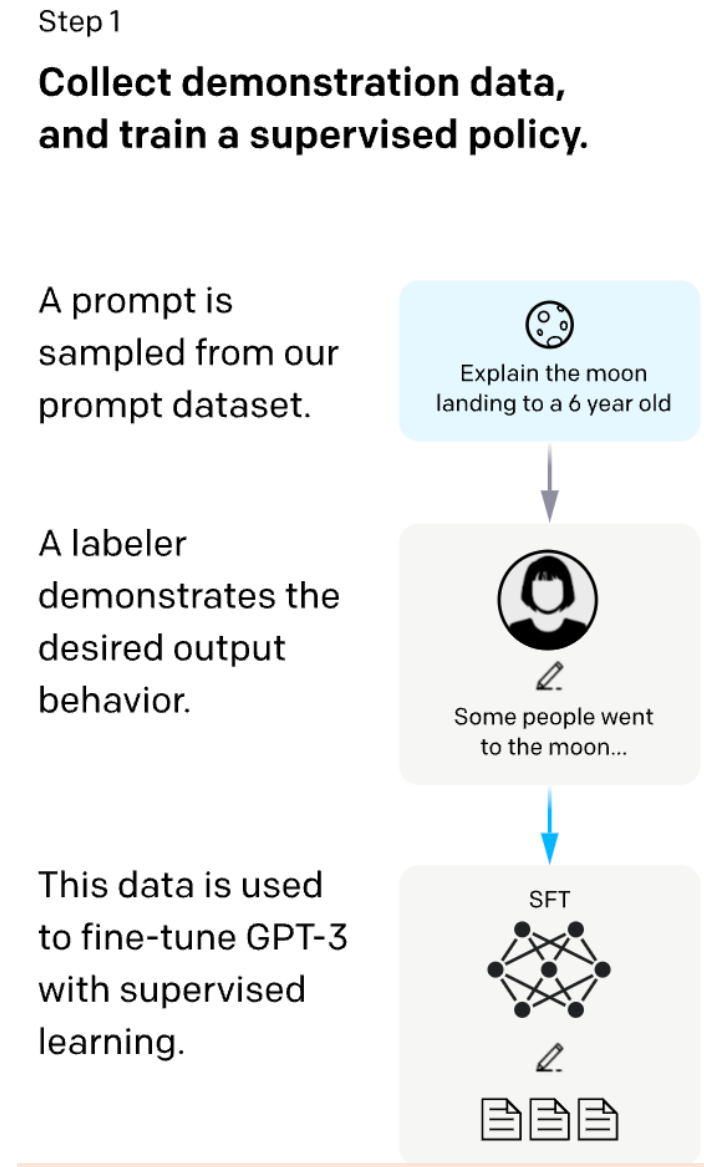
https://encord.com/blog/reinforecement-learning-from-ai-feedback-what-is-rlaif/

- A separate AI model evaluates responses based on the constitution, ranking them accordingly
- The **reward model** is created using an **off-the-shelf LLM**
- Reinforcement learning (like PPO) needs **many fast reward evaluations** to optimize the policy.
- Calling the full LLM every step would be **too slow and costly**.

Open-AI

Step 1: Supervised Fine-Tuning (SFT)

- **Goal:** Teach the model to imitate high-quality outputs.
  - A **prompt** is sampled (e.g., "Explain the moon landing to a 6-year-old").
  - A **human labeler** writes a good response.
  - These (prompt, response) pairs are used to train the model using **supervised learning** → this becomes the **SFT model**.

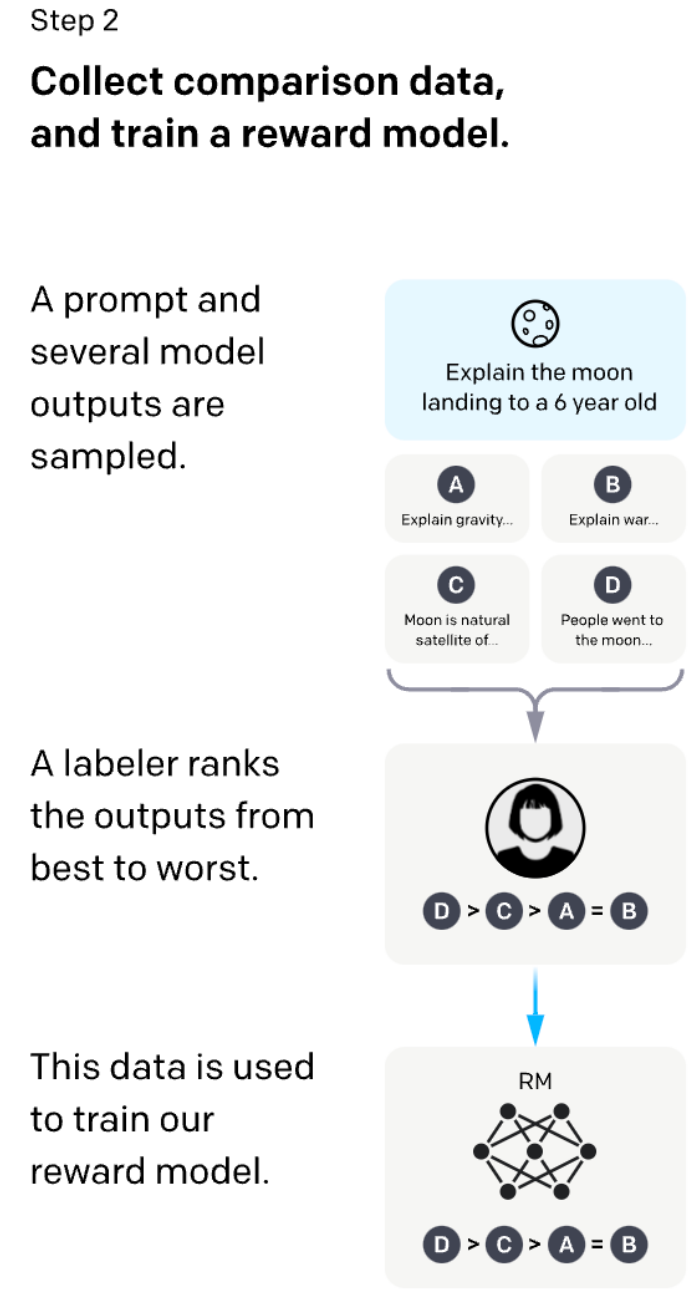


Mimic ideal responses

https://openai.com/index/chatgpt/

Step 2: Train a Reward Model

- **Goal:** Learn what makes one response better than another.
  - For a new prompt, **multiple model responses** are generated.
  - A **labeler ranks** the responses (e.g., D > C > A = B).
  - These rankings train a **reward model (RM)** to estimate human preference.
    - Input: (prompt, response)
    - Output: scalar reward

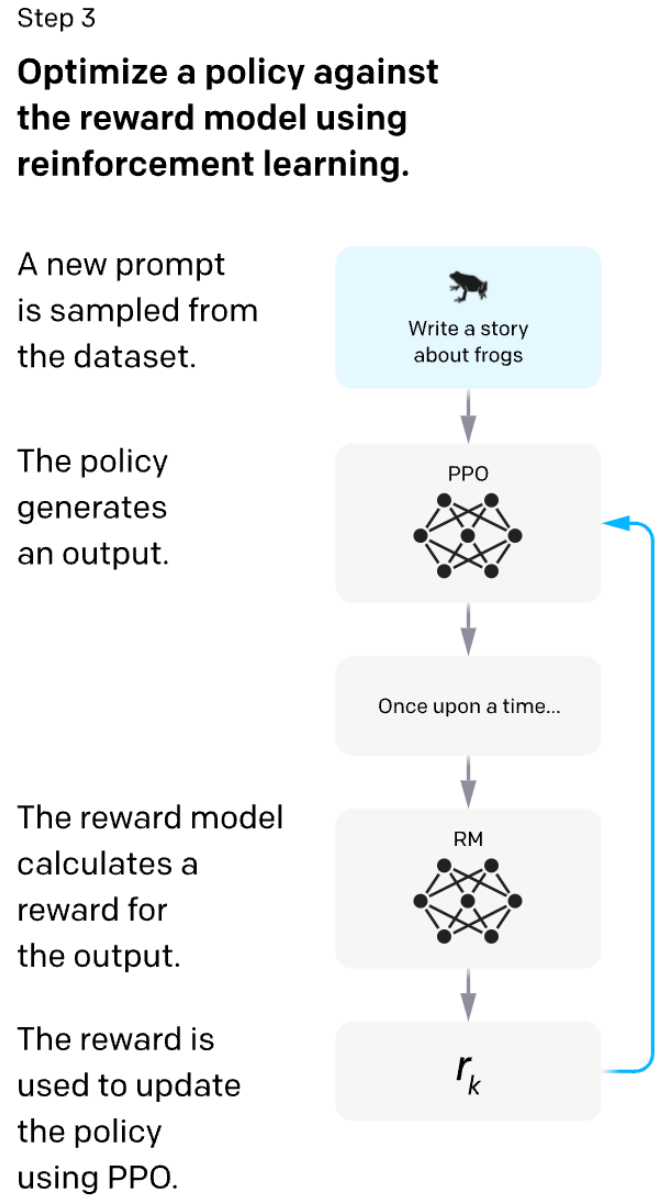


https://openai.com/index/chatgpt/

Learn preferences

Step 3: Reinforcement Learning with PPO

- **Goal:** Optimize the model to maximize the reward model's score.
  - The current policy (model) generates an output for a new prompt.
  - The **reward model scores** the output.
  - Using **PPO (Proximal Policy Optimization)**, the model is updated to improve future outputs while staying close to its original behavior.



https://openai.com/index/chatgpt/

Improve policy

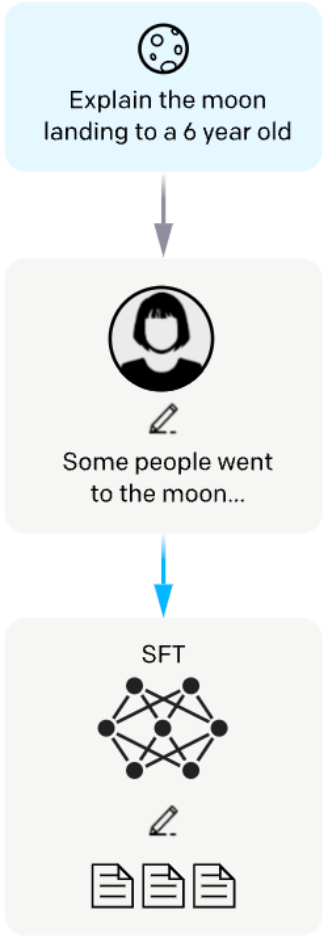
Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



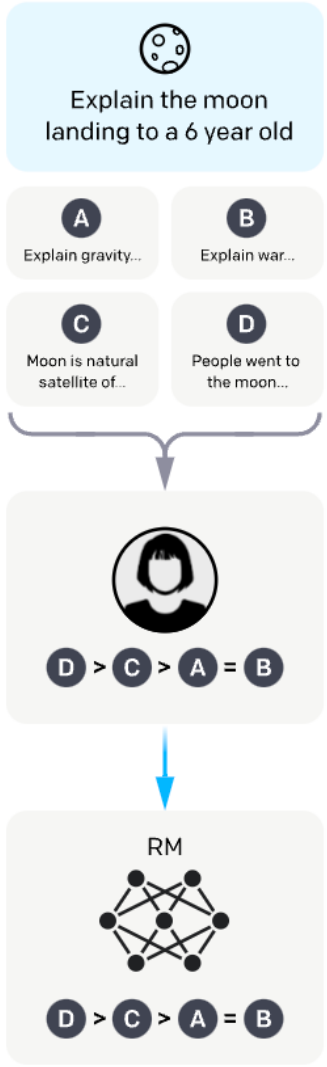
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

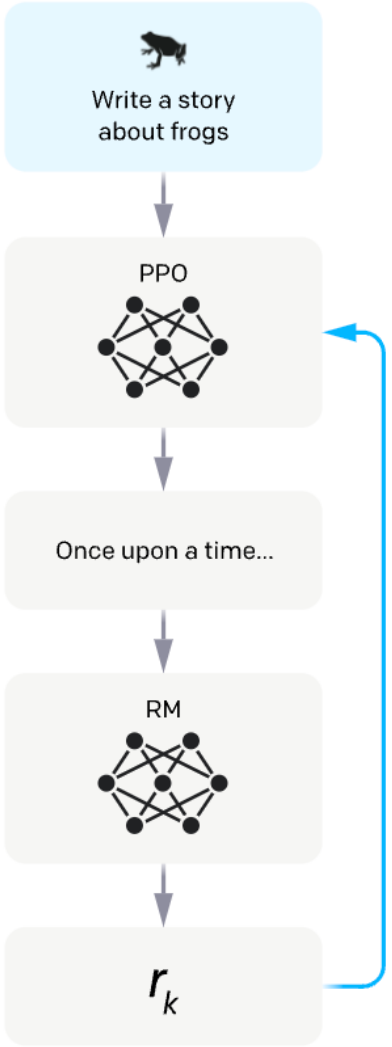
Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy

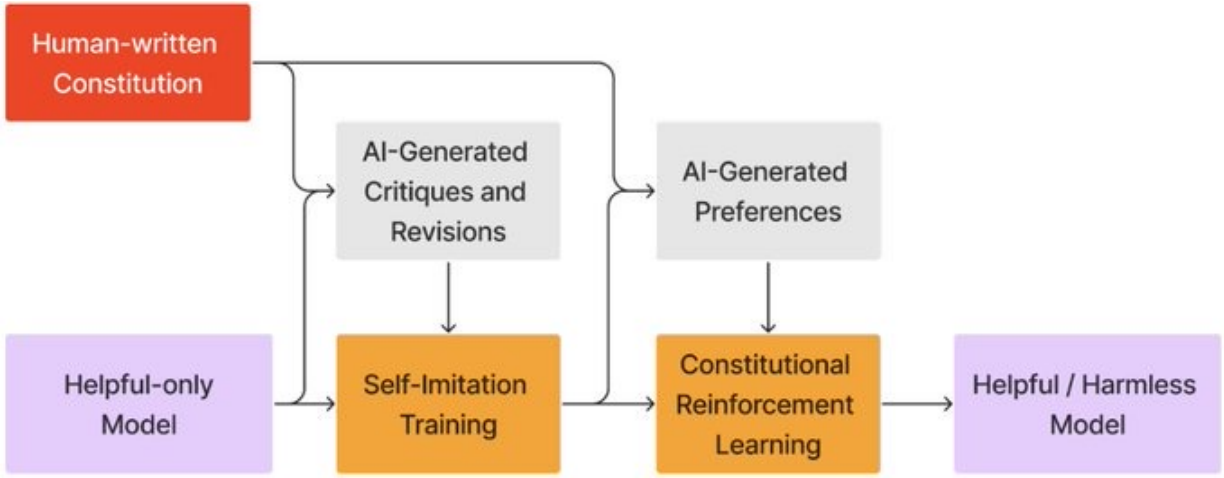


https://openai.com/index/chatgpt/

Constitutional AI (Anthropic)

Step 1 – Constitution and Helpful-Only Model

- **Inputs:**
  - A **human-written constitution** defines safety and helpfulness principles .
  - A **helpful-only model** is trained using supervised learning on helpful responses.
- **Purpose:**
  - Build a base model that is helpful, but not yet aligned with safety or harmlessness constraints.



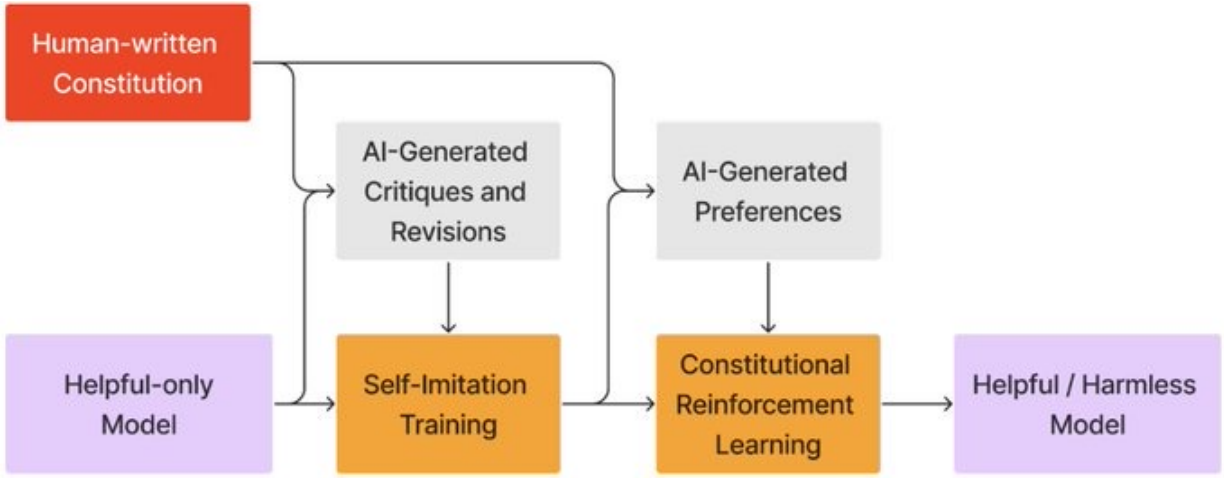
https://x.com/AnthropicAI/status/1603791173772275712

- The model uses a human-written constitution to critique, revise, and rank its own outputs—enabling self-training and reinforcement without human labelers.
- Helpful-only model is trained based on **human demonstrations** of helpful behavior, But it's not yet trained to avoid harm or follow safety principles.

Example Rules (Constitution)
•Avoid toxic or biased content
•Be concise, helpful, and safe
•Uphold privacy and factual integrity

Step 2 – Self-Critique and Self-Imitation

- **AI-Generated Critiques and Revisions**
  - The helpful-only model generates an output to a prompt.
  - It then **critiques and revises** its own response using the constitution.
- **Self-Imitation Training**
  - The revised output is treated as the new **ground truth**.
  - The model is **fine-tuned to imitate its own improved outputs**.



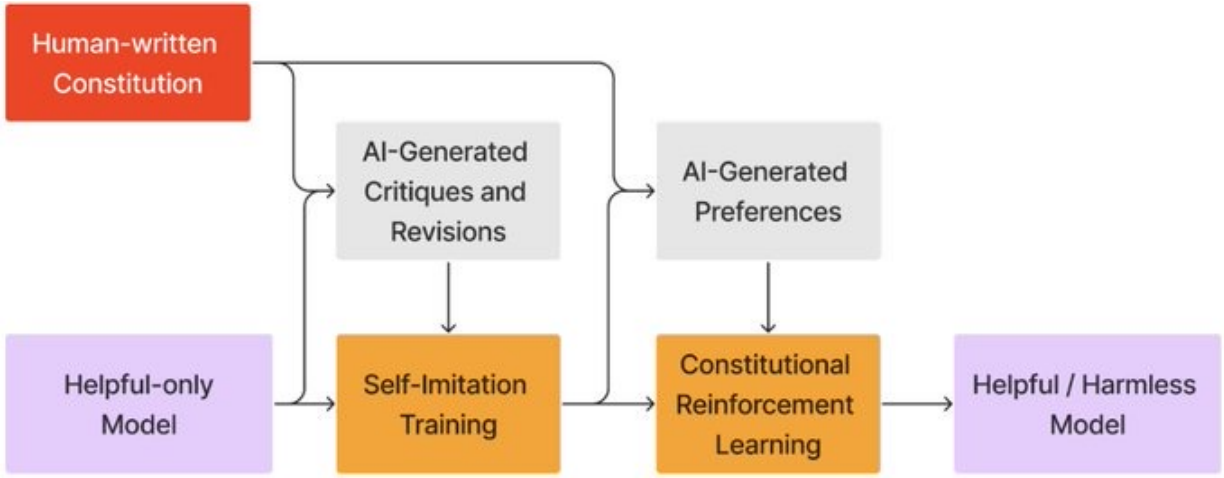
https://x.com/AnthropicAI/status/1603791173772275712

- The model uses a human-written constitution to critique, revise, and rank its own outputs—enabling self-training and reinforcement without human labelers.
- Helpful-only model is trained based on **human demonstrations** of helpful behavior, But it's not yet trained to avoid harm or follow safety principles.

Example Rules (Constitution)
•Avoid toxic or biased content
•Be concise, helpful, and safe
•Uphold privacy and factual integrity

Step 3 – Generating Preferences

- **AI-Generated Preferences**
  - For a single prompt, the model creates **multiple responses**.
  - It ranks these outputs based on how well they follow the constitution.
- **Why this matters:**
  - Replaces human feedback with **automated preference data** based on consistent rules.



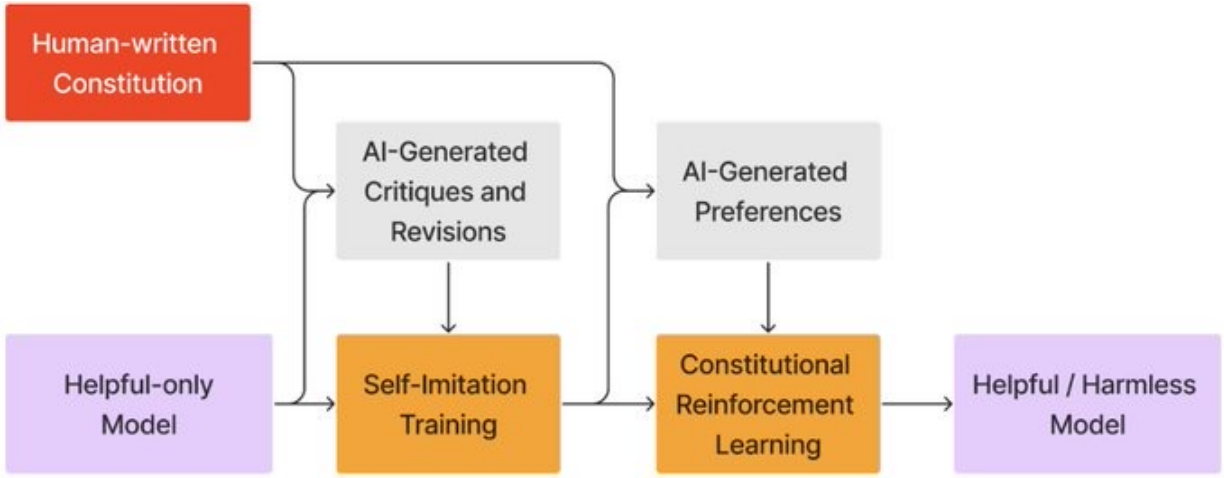
https://x.com/AnthropicAI/status/1603791173772275712

- The model uses a human-written constitution to critique, revise, and rank its own outputs—enabling self-training and reinforcement without human labelers.
- Helpful-only model is trained based on **human demonstrations** of helpful behavior, But it's not yet trained to avoid harm or follow safety principles.

Example Rules (Constitution)
•Avoid toxic or biased content
•Be concise, helpful, and safe
•Uphold privacy and factual integrity

Step 4 – Constitutional Reinforcement Learning

- **Fine-Tuning with PPO**
  - The model is now optimized using **RL (e.g., PPO)**.
  - Reward = How well the output satisfies constitutional principles (based on rankings).
- **Outcome:**
  - Final model is both **helpful and harmless**, trained entirely without human preference comparisons.



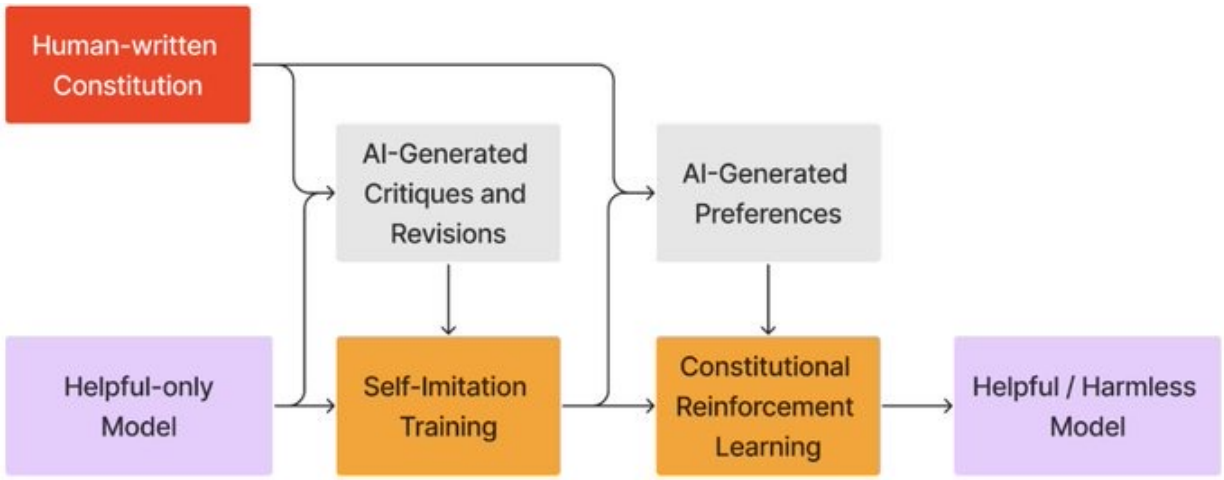
https://x.com/AnthropicAI/status/1603791173772275712

- The model uses a human-written constitution to critique, revise, and rank its own outputs—enabling self-training and reinforcement without human labelers.
- Helpful-only model is trained based on **human demonstrations** of helpful behavior, But it's not yet trained to avoid harm or follow safety principles.

Example Rules (Constitution)
- Avoid toxic or biased content
- Be concise, helpful, and safe
- Uphold privacy and factual integrity

Constitutional AI (Anthropic)

- **What is Constitutional AI?**
  - Alternative to RLHF
  - Replaces human feedback with **principle-based self-critiques**
  - Model learns to revise outputs using written rules (the "constitution")
- **Steps:**
  1. **Supervised Fine-Tuning**
     1. Train model on high-quality human-written examples
  2. **Model Self-Critique**
     1. Model compares or critiques its own outputs
     2. Based on a set of principles
  3. **Reinforcement Learning**
     1. Model is fine-tuned using RL
     2. Reward = how well output follows the constitution
- **Benefits**
  - Scales better than human-labeled RLHF
  - Transparent: rules are explicit and auditable
  - Consistent: less noisy than human rankings
- **Challenges**
  - Rule writing is difficult and subjective
  - Needs safeguards against rule exploitation



https://x.com/AnthropicAI/status/1603791173772275712

- The model uses a human-written constitution to critique, revise, and rank its own outputs—enabling self-training and reinforcement without human labelers.
- Helpful-only model is trained based on **human demonstrations** of helpful behavior, But it's not yet trained to avoid harm or follow safety principles.

Example Rules (Constitution)

•Avoid toxic or biased content

•Be concise, helpful, and safe

•Uphold privacy and factual integrity