



BERT & GPT

Pavlos Protopapas

Christopher Montall



Outline

- Transformers Recap
- BERT
- Practical problems in Large Language Models
- GPT
- Hugging Face

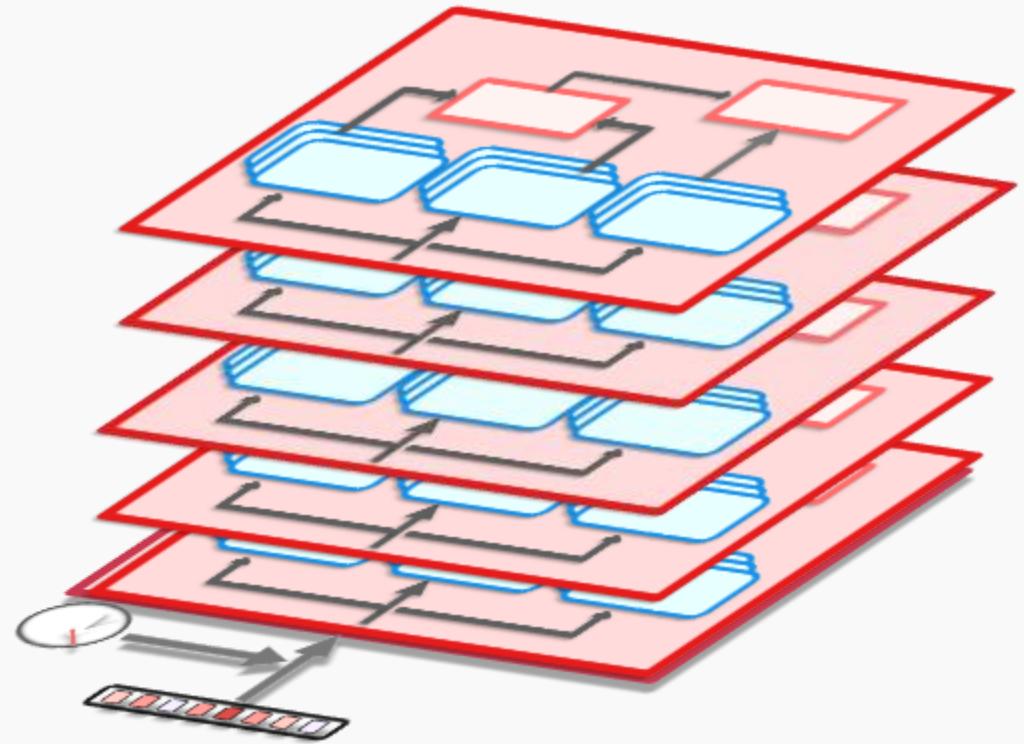
Outline

- Transformers Recap
- BERT
- Practical problems in Large Language Models
- GPT
- Hugging Face

Recap: Encoder

Language Model Wishlist?

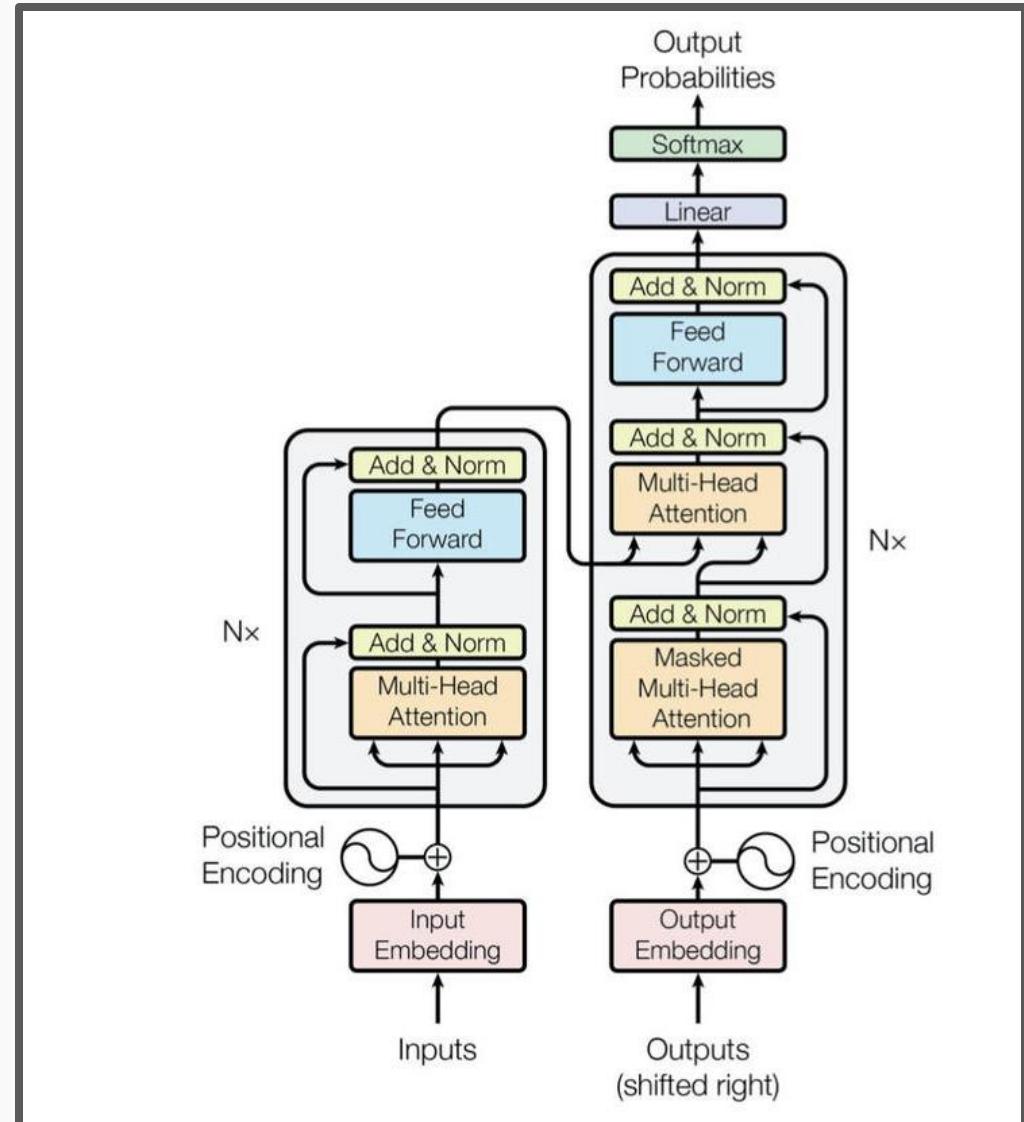
- We want to have **strong contextual relations** between words - **DONE**
- We want words to have **sequential** information - **DONE**
- We need an architecture that can be trained in **parallel** (non-Markovian property) - **DONE**



Recap: Transformers

Transformers were originally devised to help with neural machine translation.

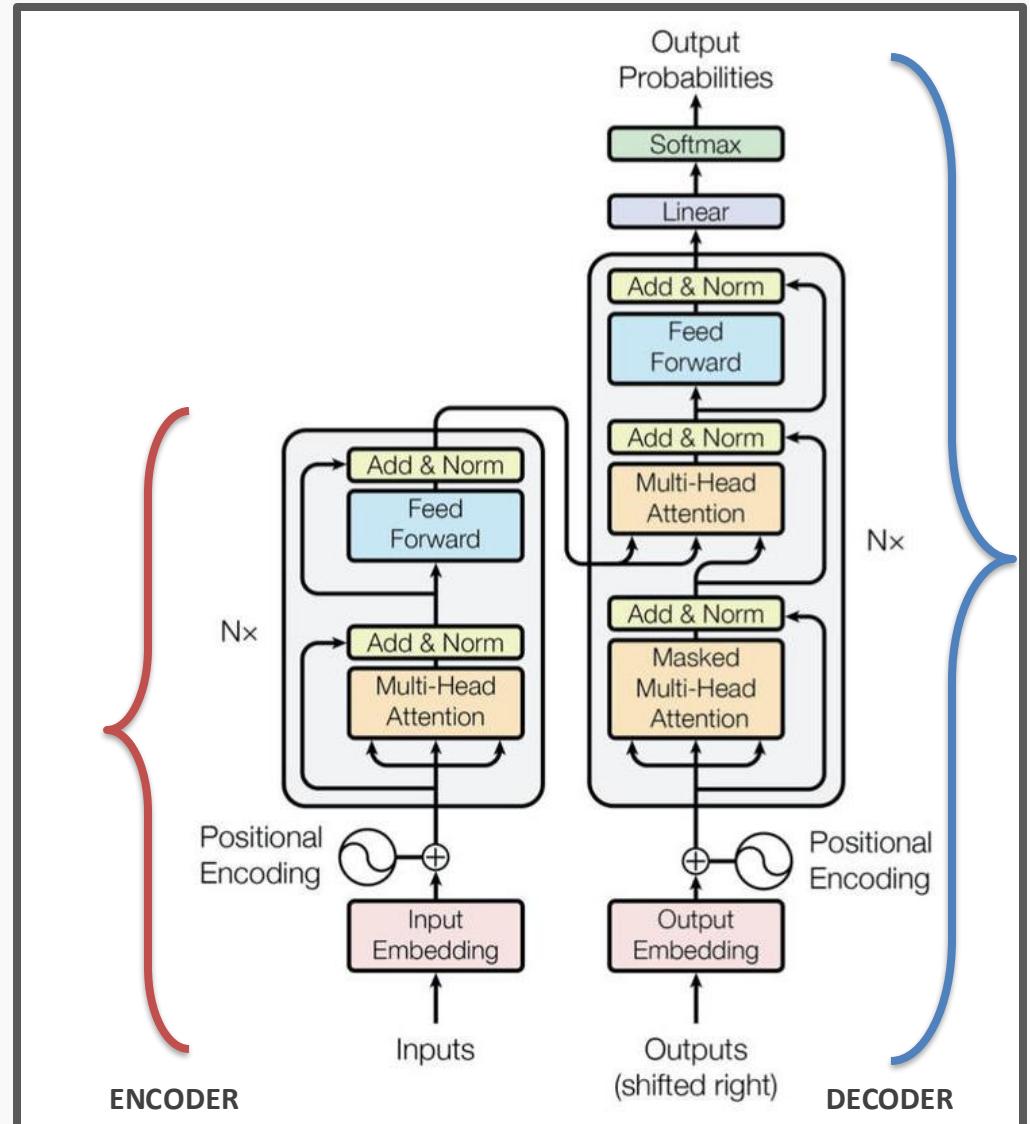
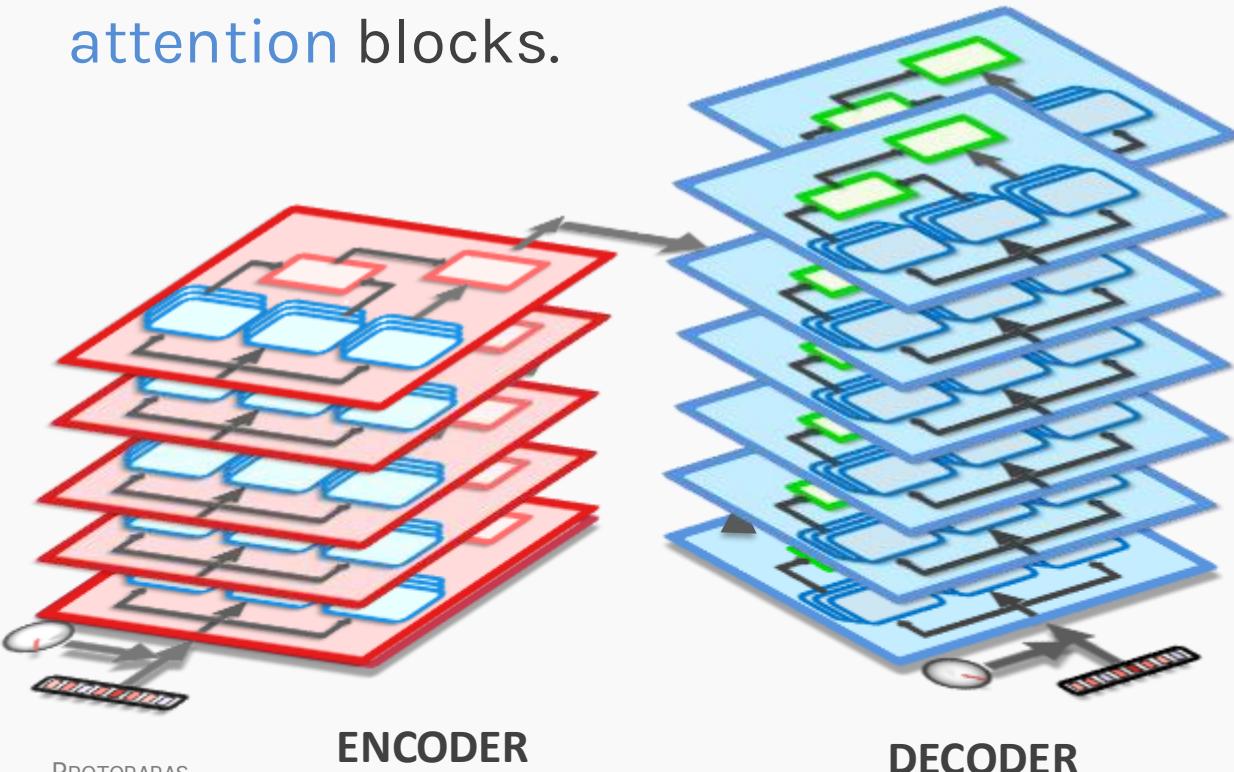
They consist of an encoder-decoder architecture, using stacked multi-head attention blocks.



Recap: Transformers

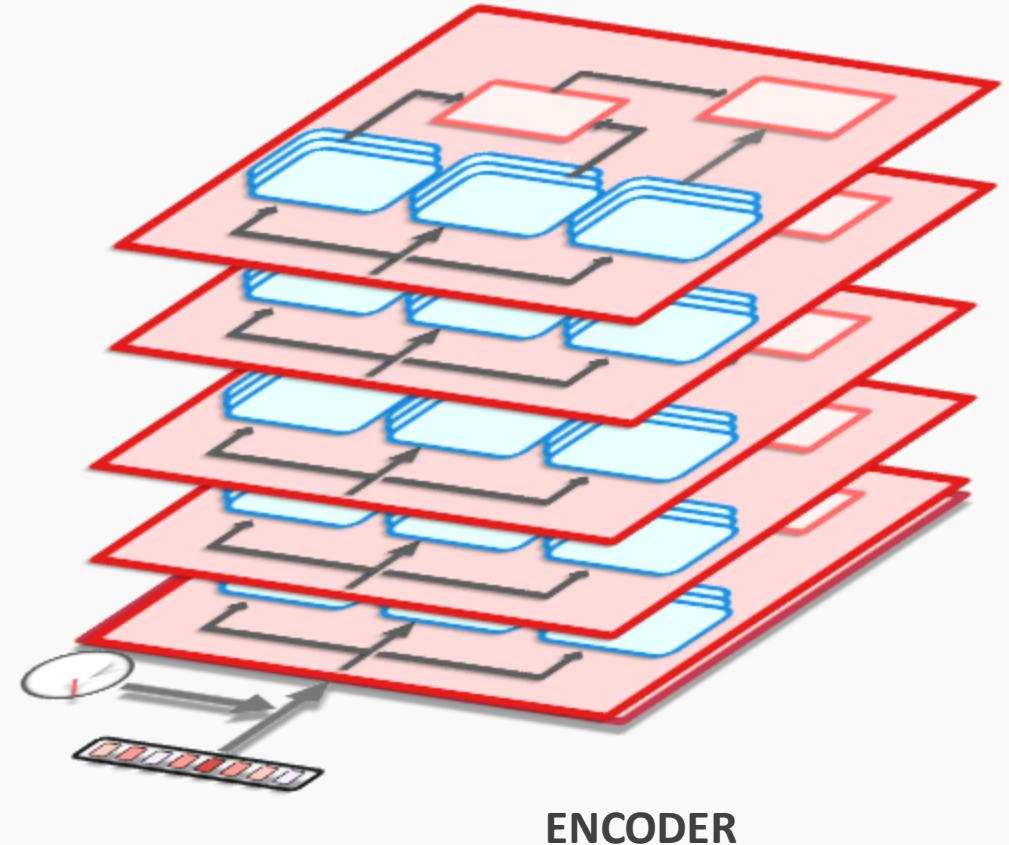
Transformers were originally devised to help with neural machine translation.

They consist of an encoder-decoder architecture, using stacked multi-head attention blocks.



From Transformers to BERT

- Instead of an **encoder-decoder** architecture for machine translation, what if we just use the **encoder** as a language model?
- This led to the new architecture called **Bidirectional Encoder Representations from Transformers**, or more commonly known as BERT



Outline

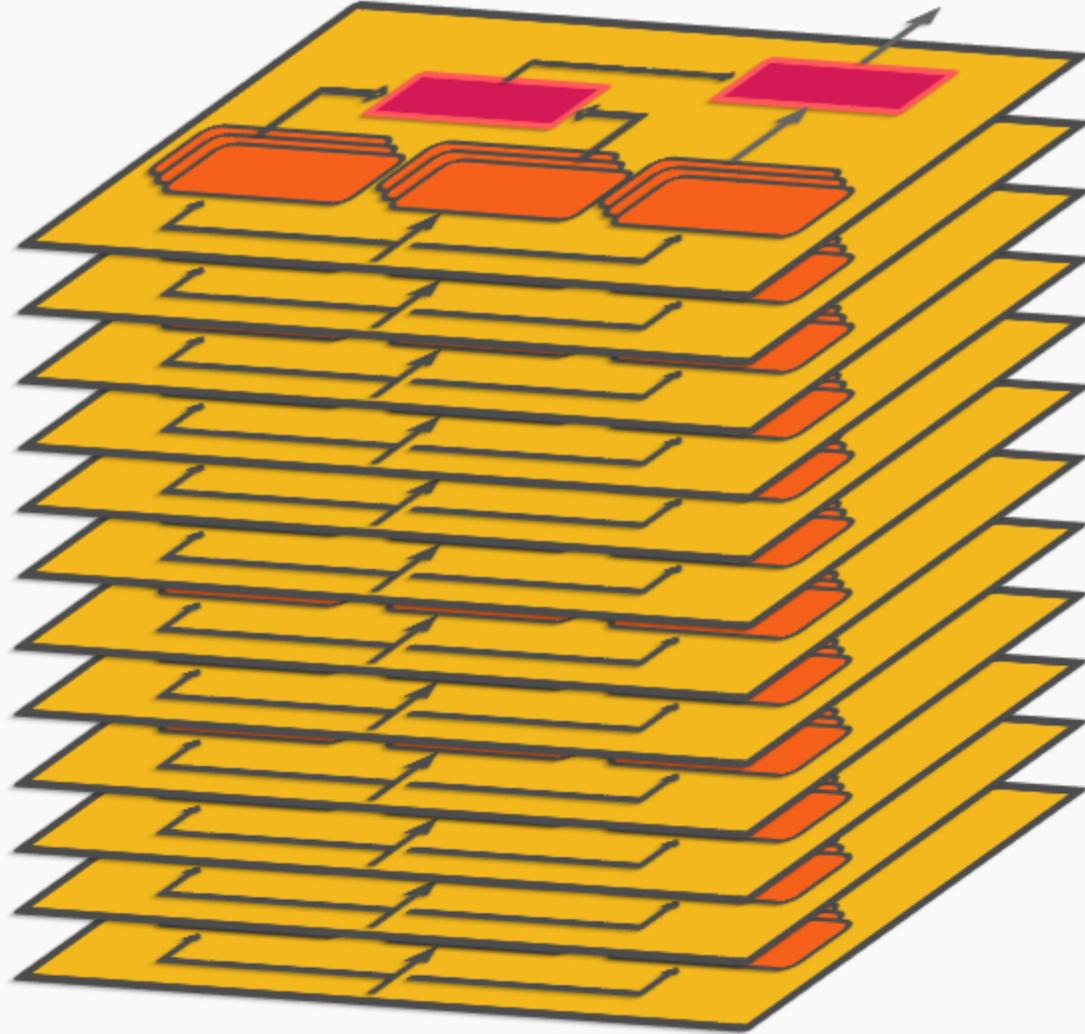
- Transformers Recap
- **BERT**
- Practical problems in Large Language Models
- GPT
- Hugging Face

Bidirectional Encoder Representations from Transformers (BERT)



BERT

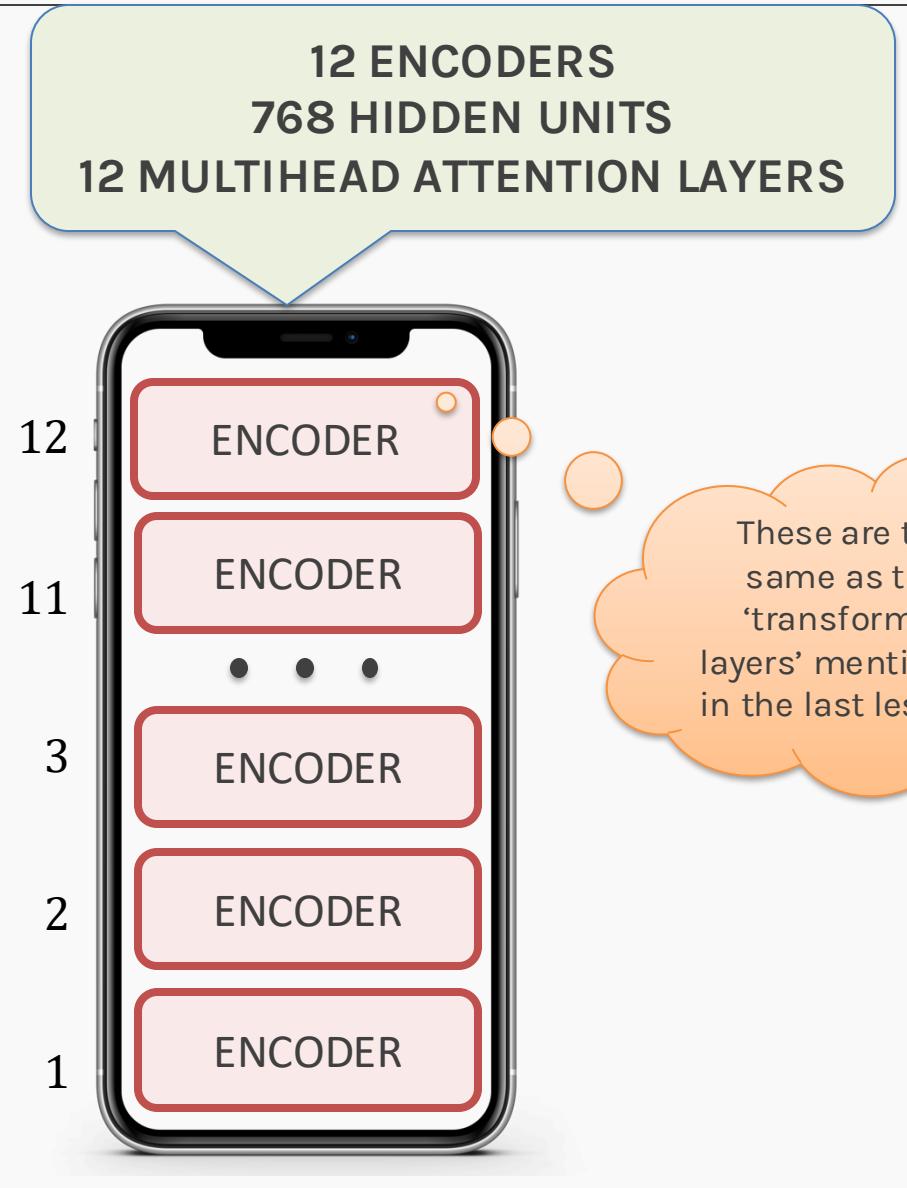
- BERT set new records in language model tasks.
- BERT's code and models are open-source and adaptable.
- BERT is ideal for transfer learning.



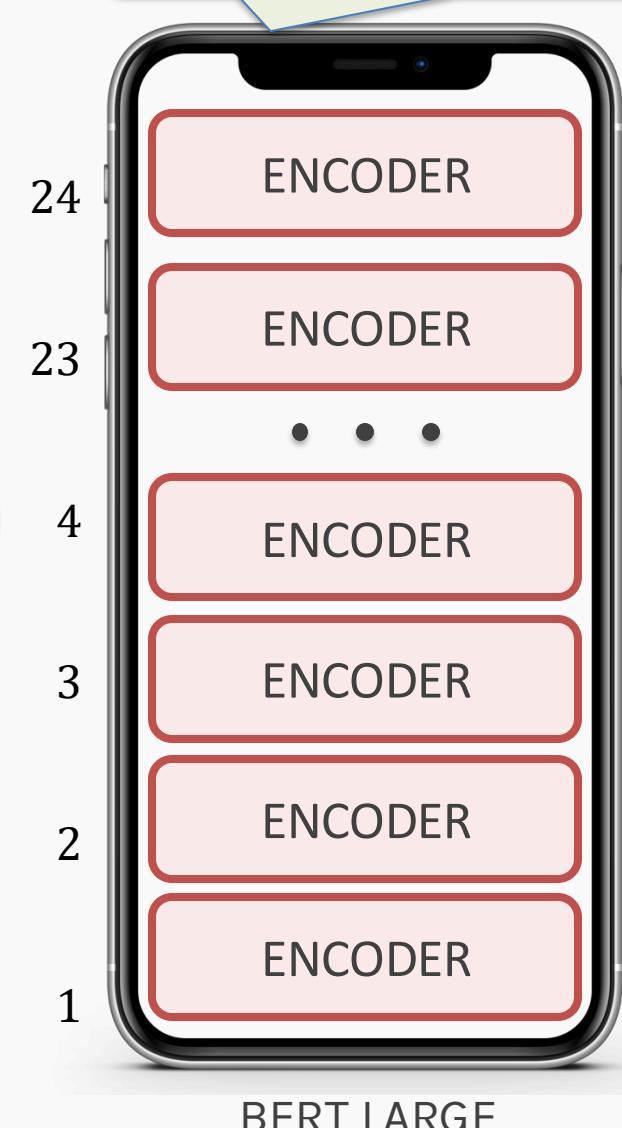
Flavors of BERT



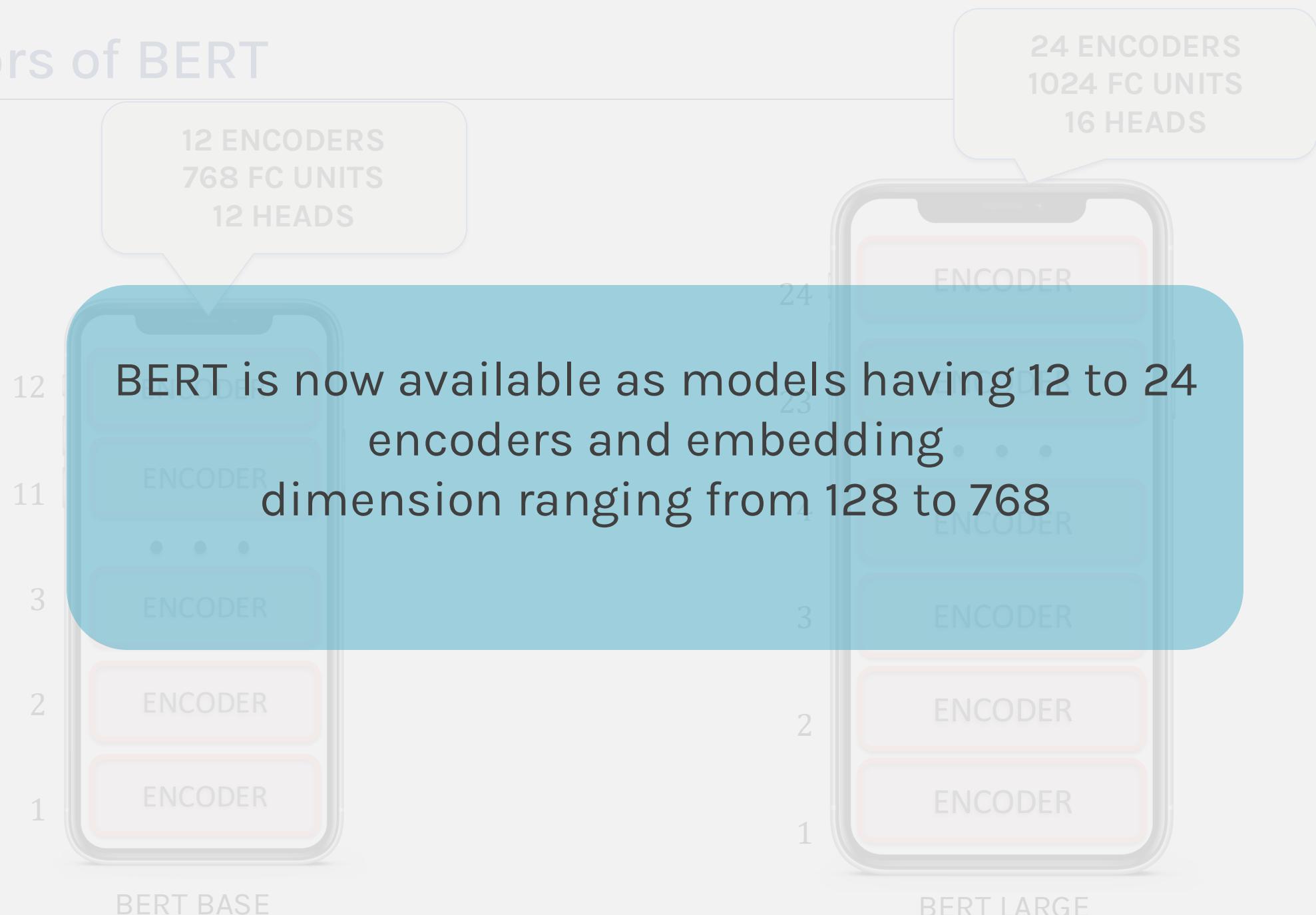
Flavors of BERT



24 ENCODERS
1024 HIDDEN UNITS
16 MULTI HEAD ATTENTION LAYERS

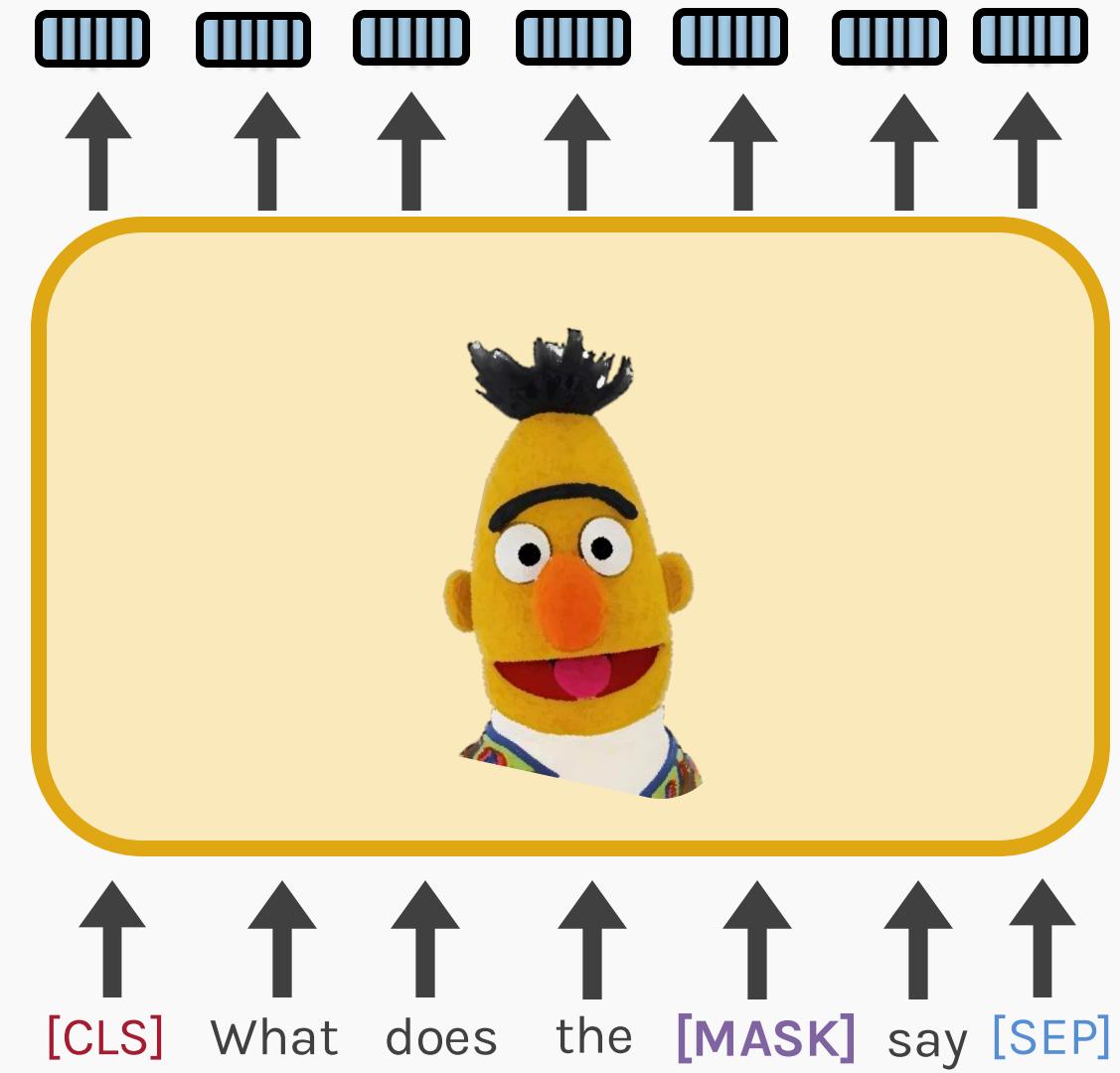


Flavors of BERT



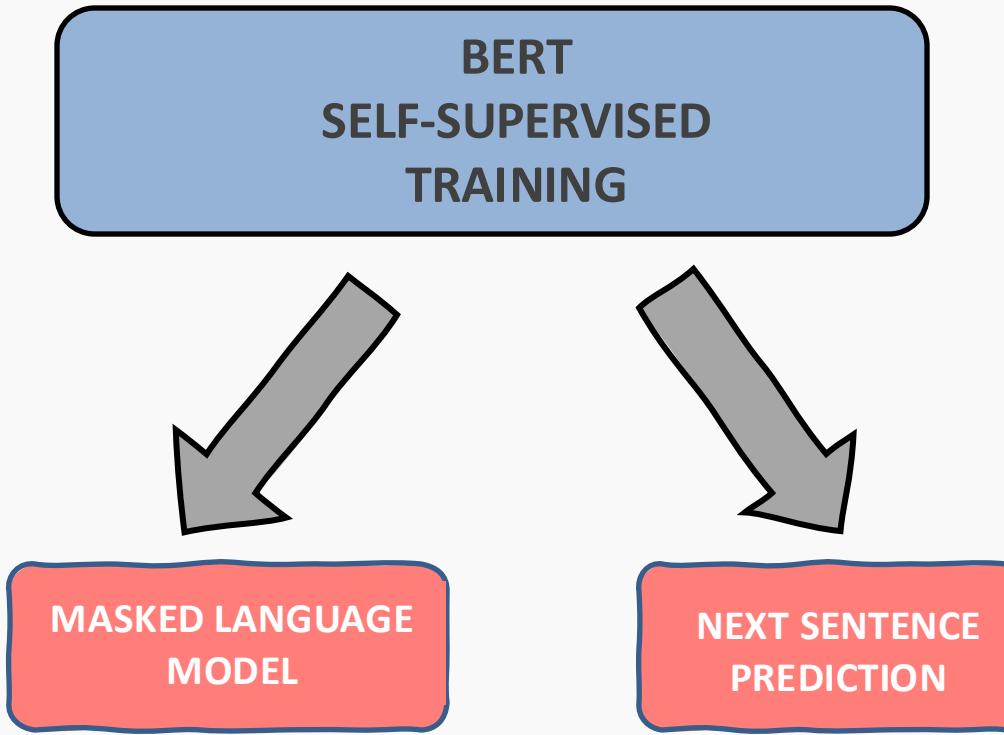
TECHNICAL DETAILS

- Input sequences to BERT must have the following special tokens:
 - [CLS] – Classifier Token
 - [MASK] – Masking token
 - [SEP] – Separator token



How to train a BERT?

How to train a BERT?



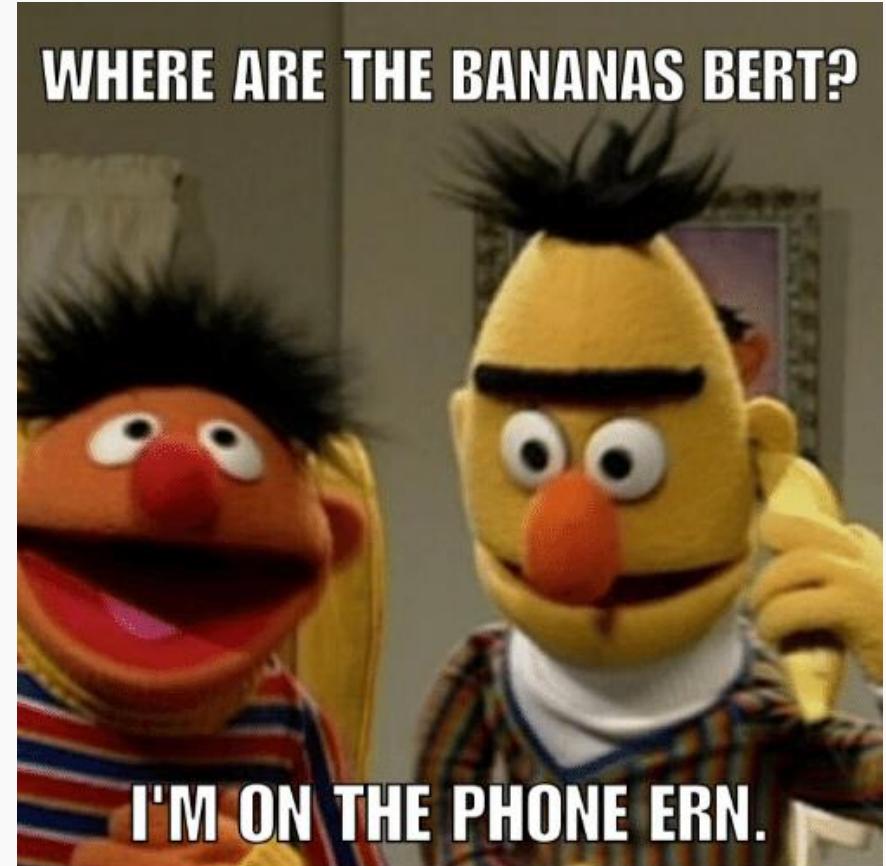
How to train a BERT?

MASKED LANGUAGE
MODEL

Challenges on training BERT as a language model

TRAINING ISSUES?

- If BERT takes the [entire sequence](#) as an input, how can we train it as a [language model](#)?
- How could we classify if two chosen sentences follow each other or are out of place if the output is also a sequence of tokens?



RECAP: Language Model

Amelia was hit by a bus as she was crossing the street

Amelia was hit by a __



So, how do we do this
with self-attention?



This sort of language
modeling is not possible
with self-attention,
because it inputs the
entire sequence at once

Masked Language Model

Amelia was hit by a bus as she was crossing the street

Amelia was hit by a [MASK] as he was ...



Great! We SHALL call this
a masked language model



Let's use my “favorite”
trick, MASKING!

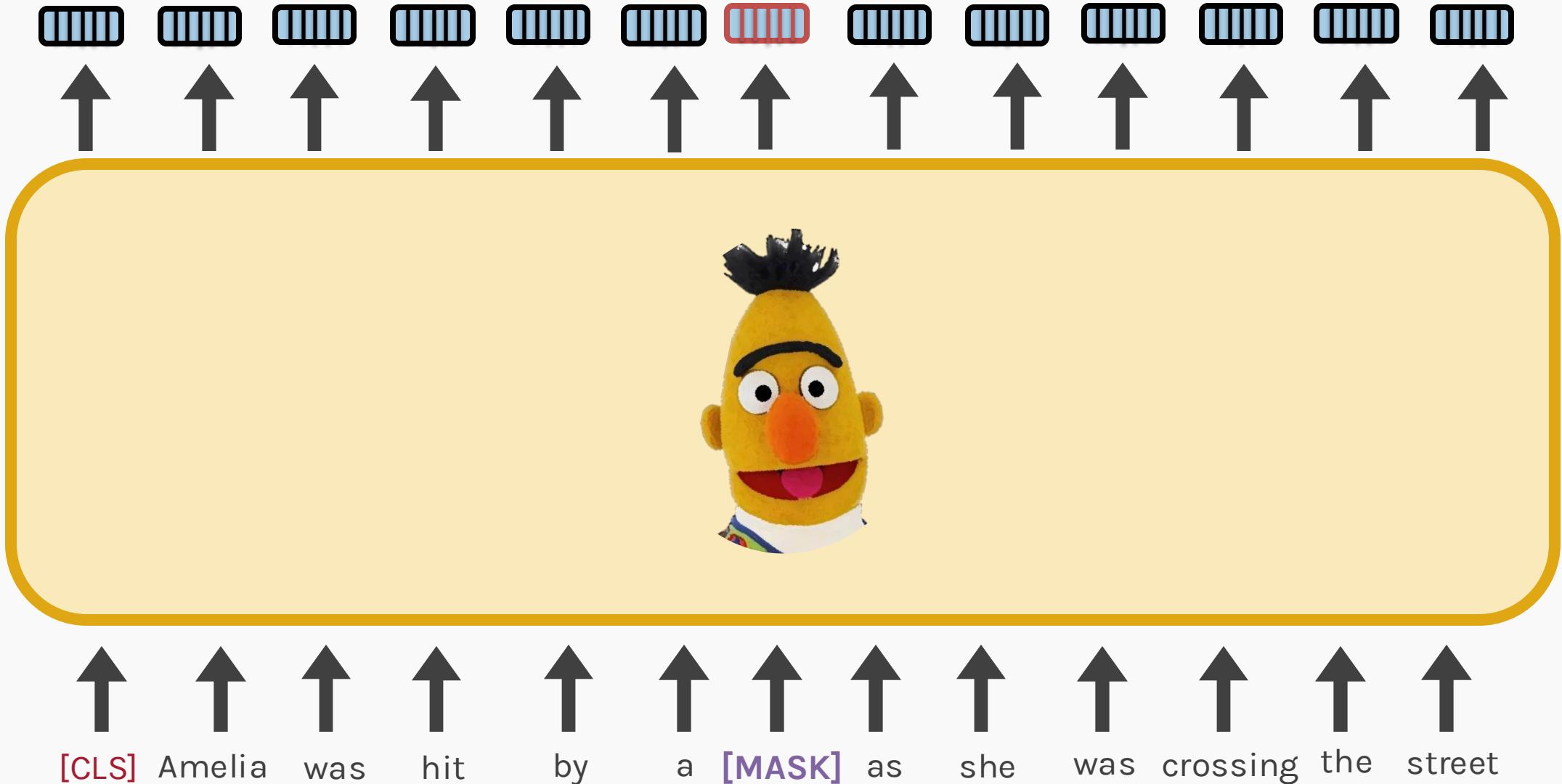
Amelia was hit by a **bus** as she was crossing the street

[CLS] Amelia was hit by a **bus** as she was crossing the street

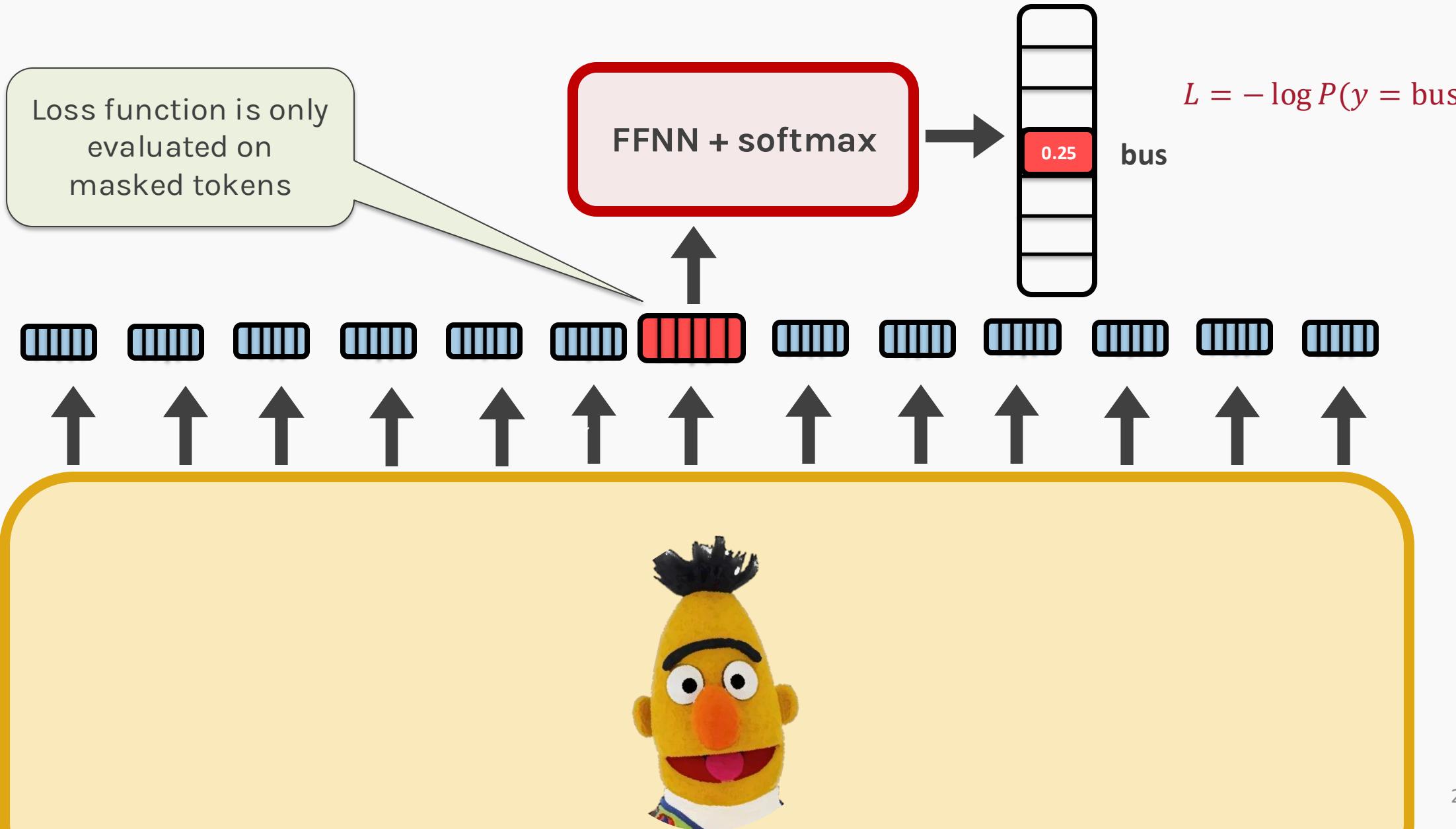
Amelia was hit by a **bus** as she was crossing the street

[CLS] Amelia was hit by a [MASK] as she was crossing the street

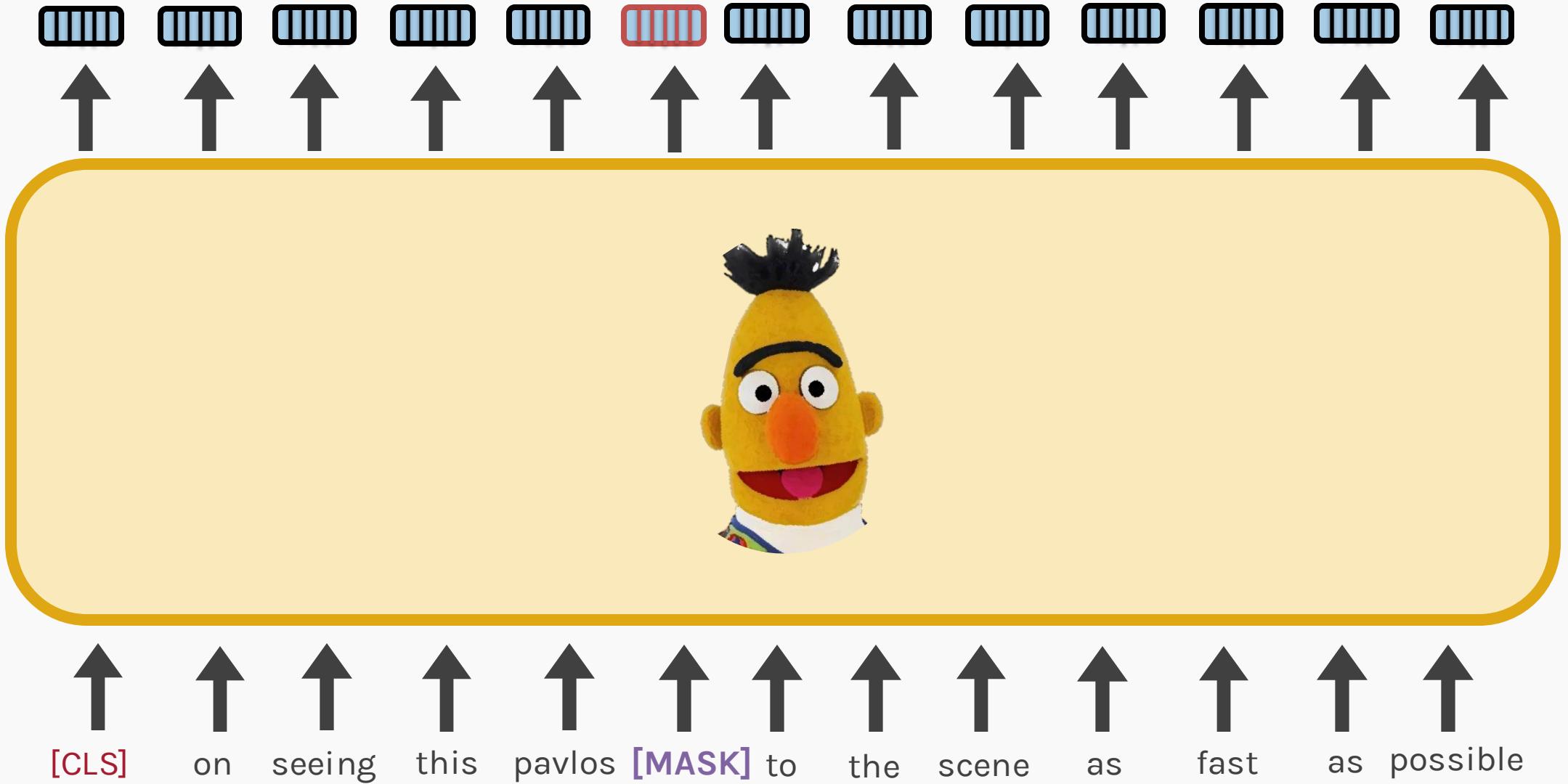
Amelia was hit by a **bus** as she was crossing the street



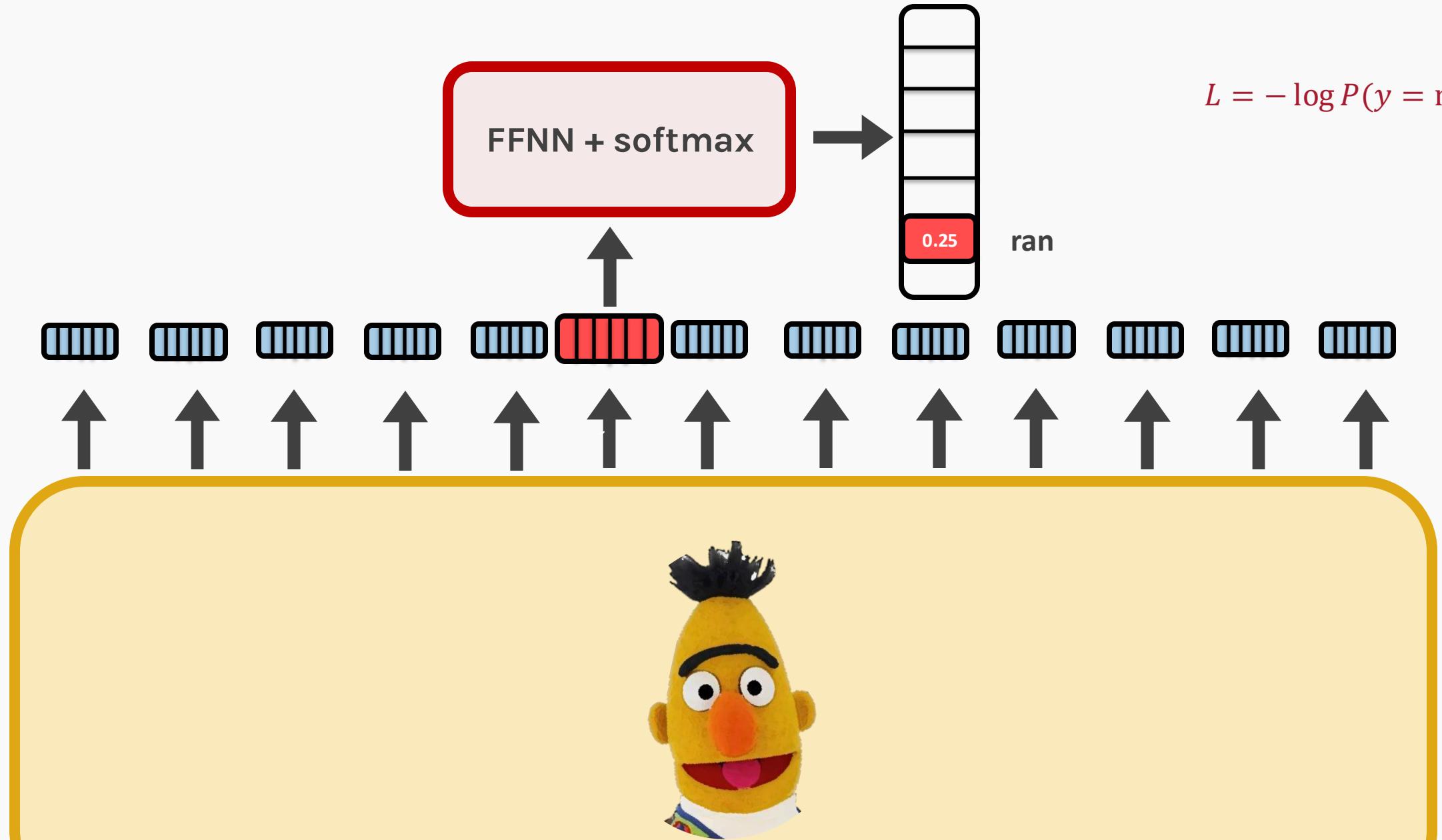
Amelia was hit by a **bus** as she was crossing the street



On seeing this, Pavlos **ran** to the scene as fast as possible



On seeing this, Pavlos **ran** to the scene as fast as possible

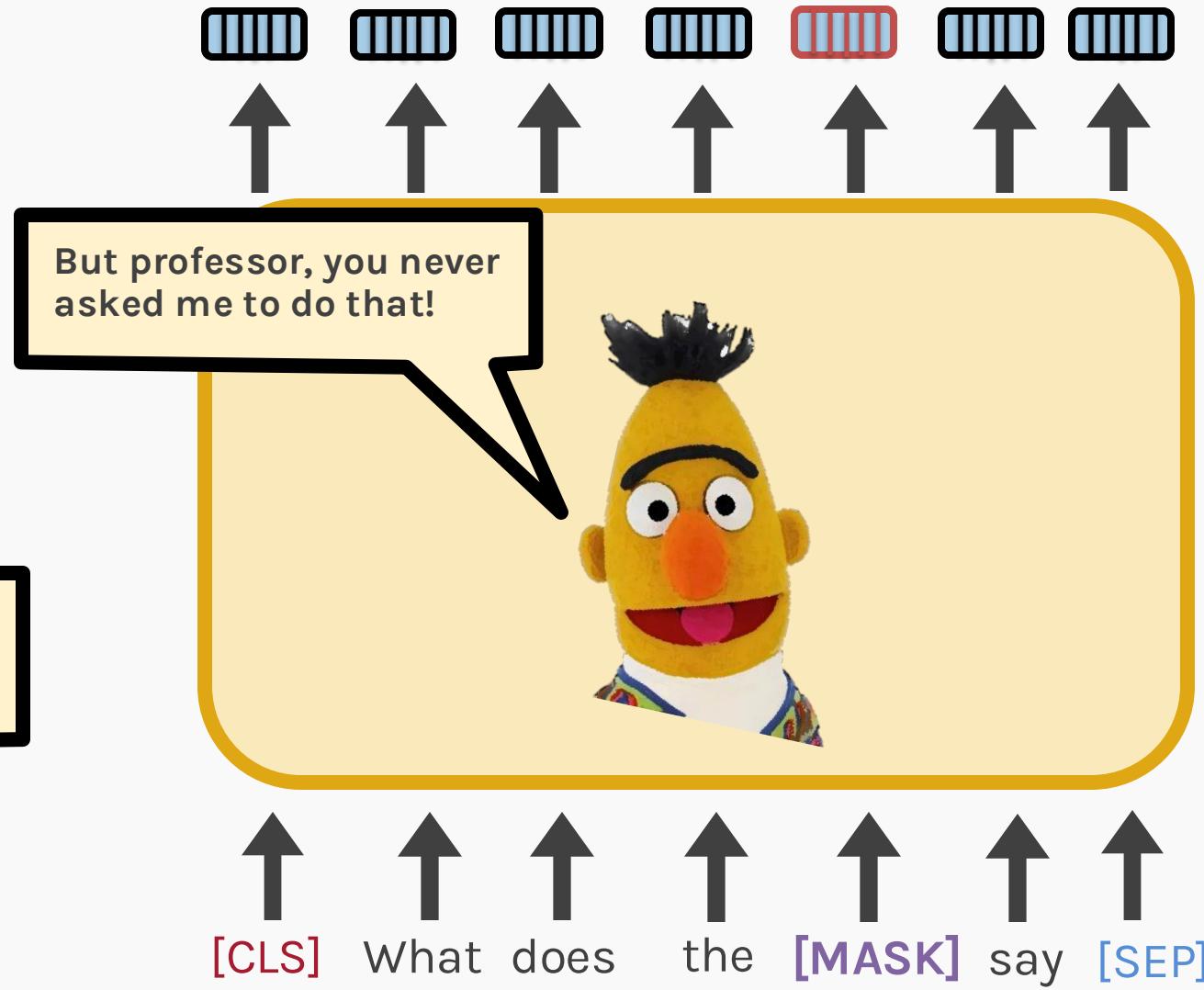


MASKING ISSUES?

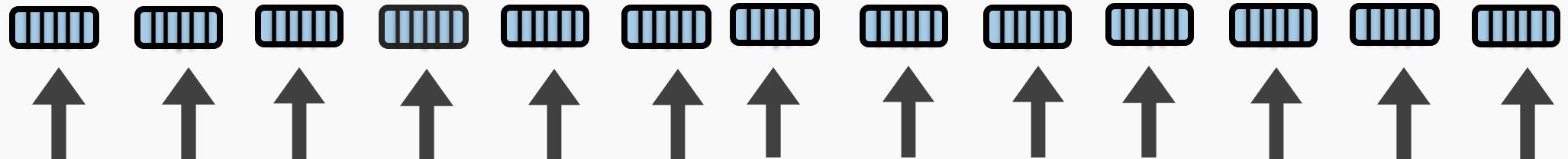
- The model learns how to correctly predict the masked words but completely ignores other words.



Hey BERT, why aren't
you predicting other
words correctly?

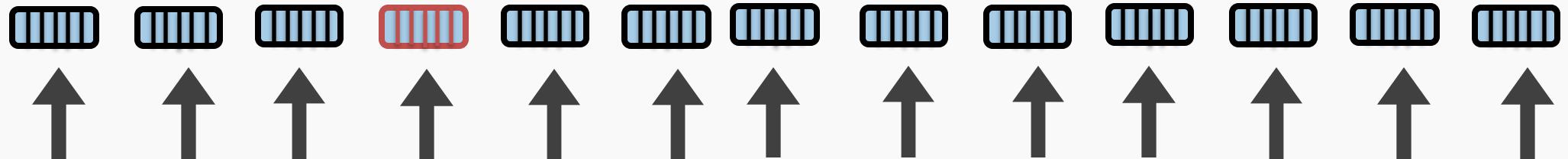


Amelia was **hit** by a bus as she was crossing the street



[CLS] Amelia was hit by a [MASK] as she was crossing the street

Amelia was **hit** by a bus as she was crossing the street

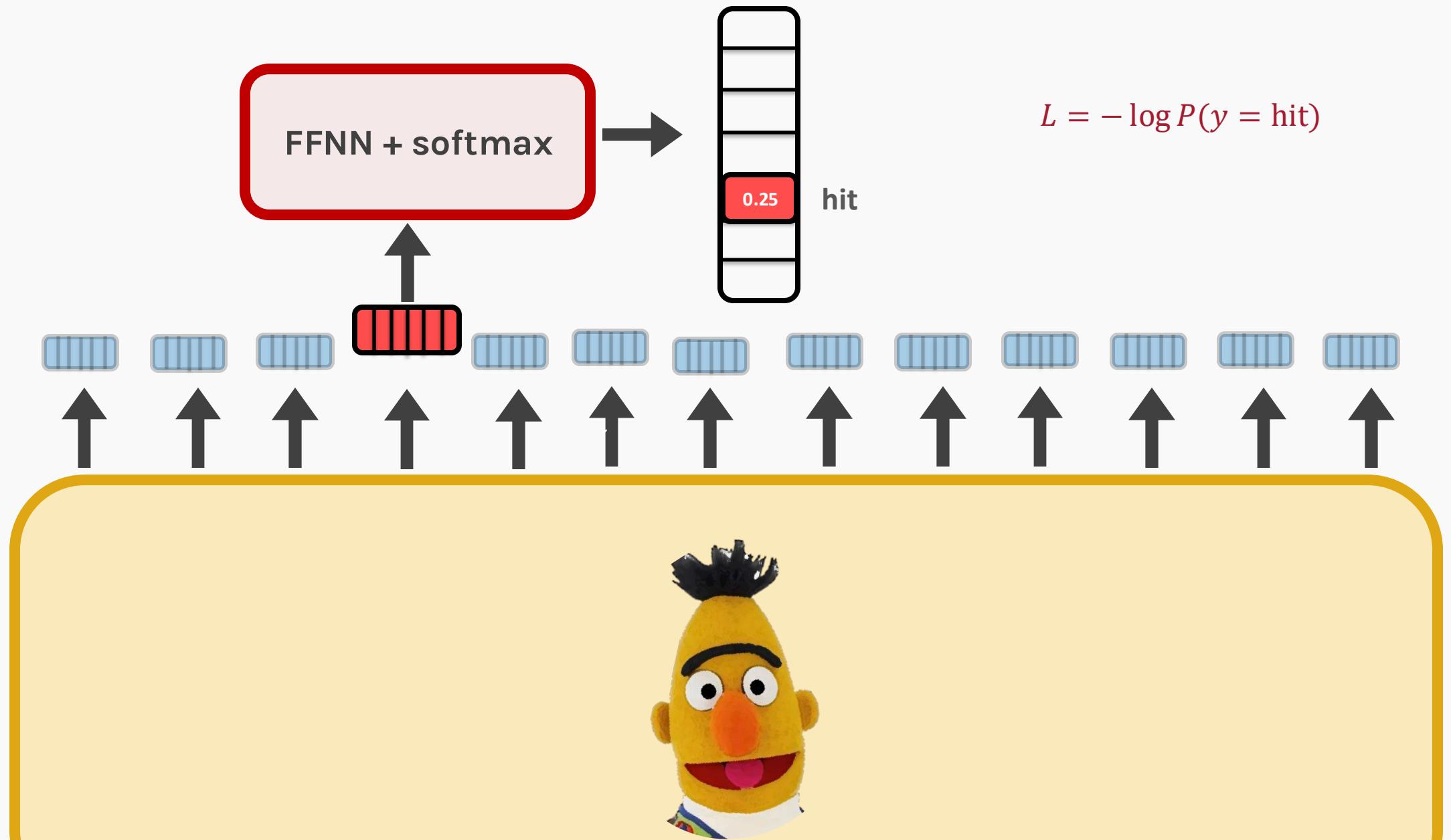


Occasionally, we randomly pick an *unmasked word* and ask the network to predict it



[CLS] Amelia was **hit** by a [MASK] as she was crossing the street

Amelia was hit by a bus as she was crossing the street

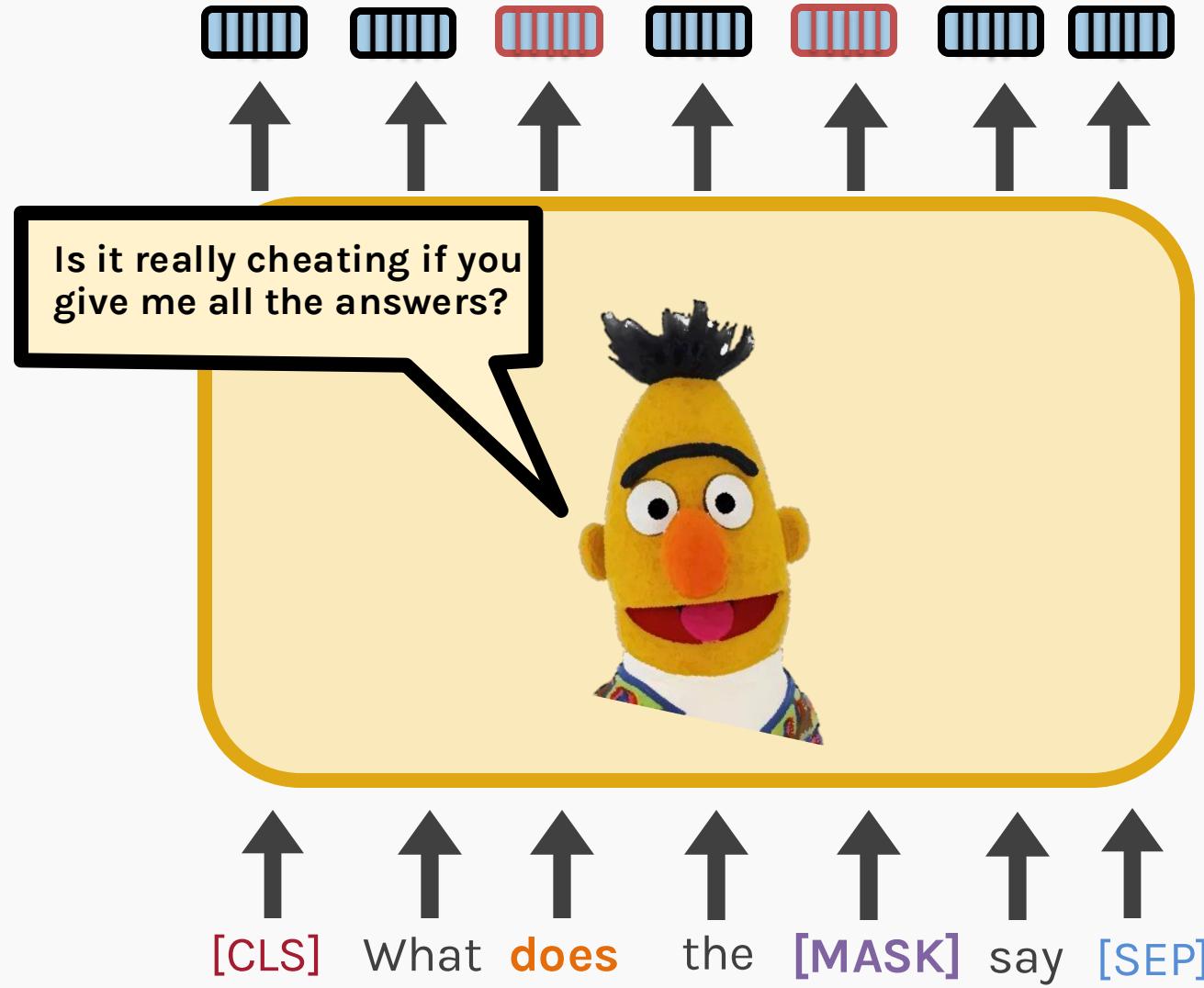


BERT

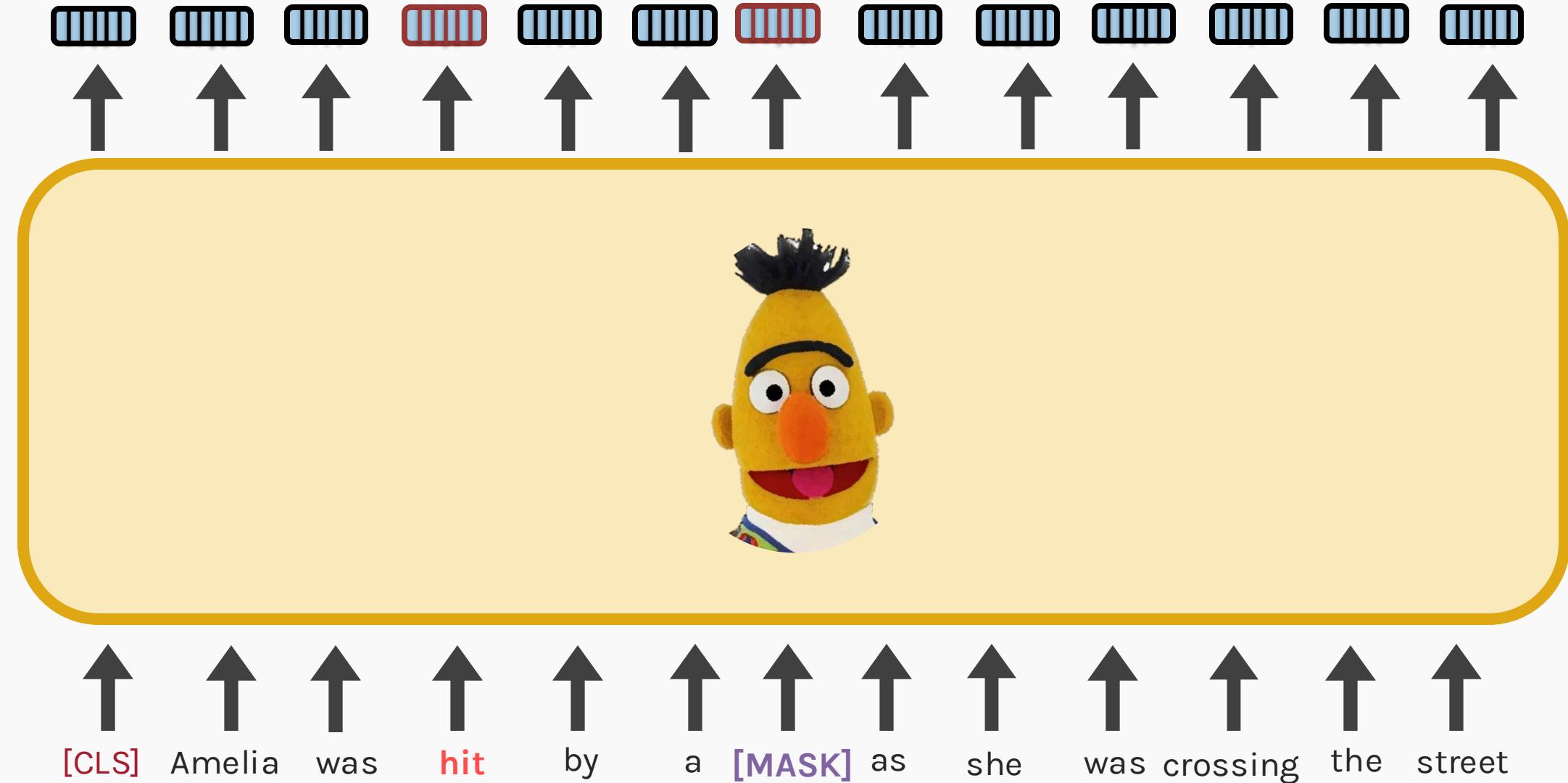
ISSUES?

- The model could use an identity mapping to return **unmasked** words.

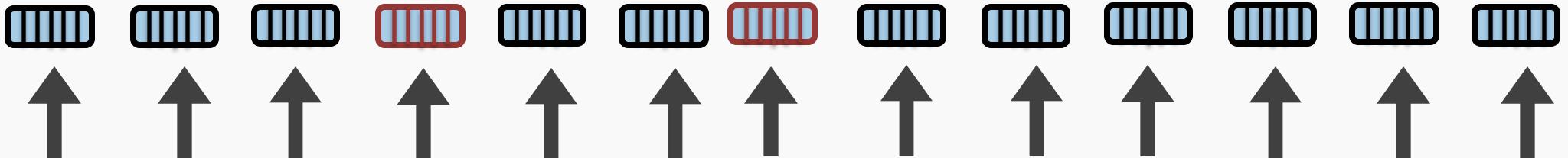
Hey BERT, why are you cheating on the test?



Amelia was **hit** by a bus as she was crossing the street



Amelia was **hit** by a bus as she was crossing the street

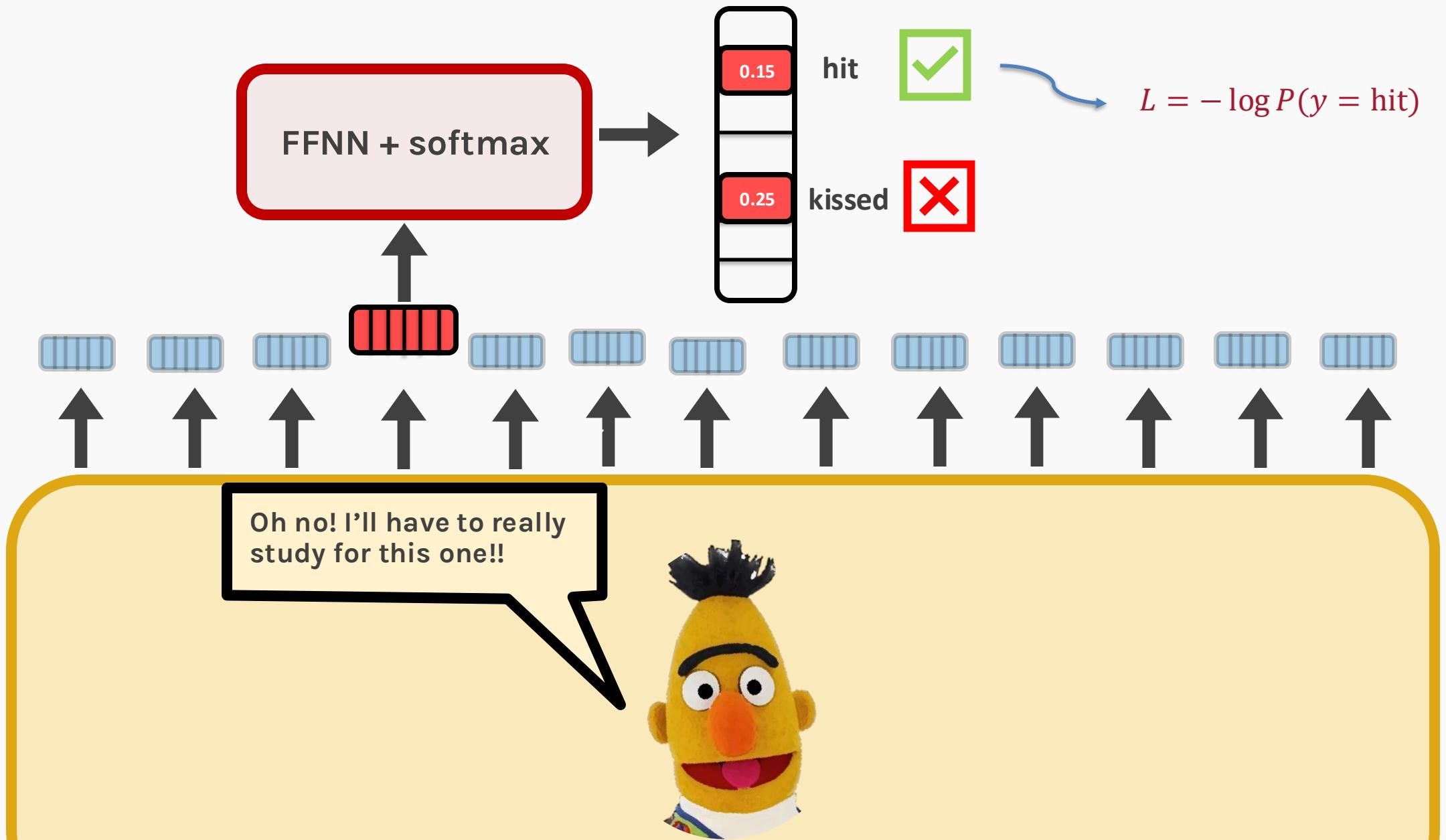


For every selected unmasked word, we can either give the correct input, or a **random input**



[CLS] Amelia was **kissed** by a [MASK] as she was crossing the street

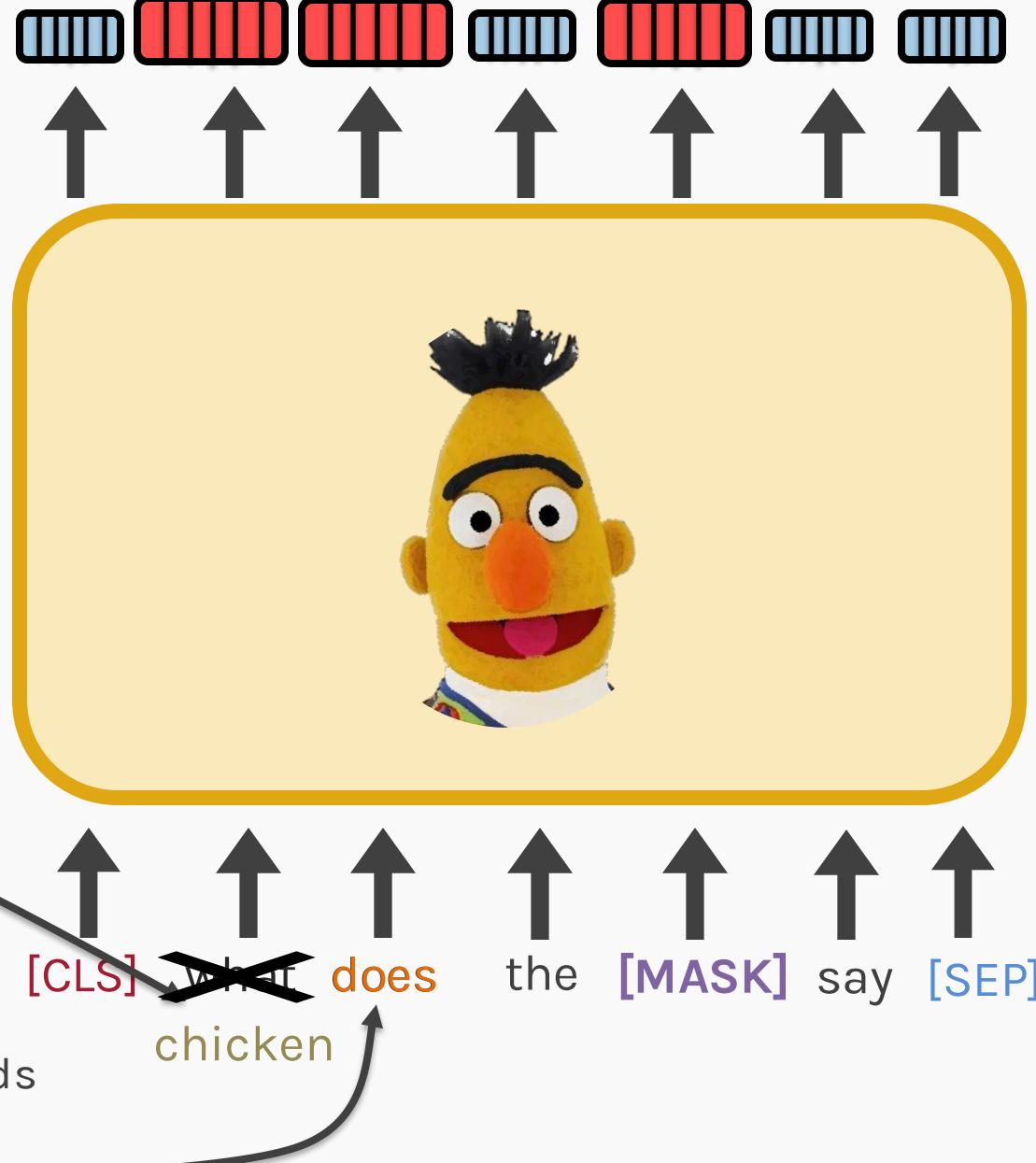
Amelia was hit by a bus as she was crossing the street



BERT

MASKING DETAILS

- BERT's language modeling task selects 15% of the input tokens and asks the model to predict the selected words.
- 80% of the selected tokens are masked, 10% are the same words and 10% are randomly replaced words.



NEXT SENTENCE PREDICTION

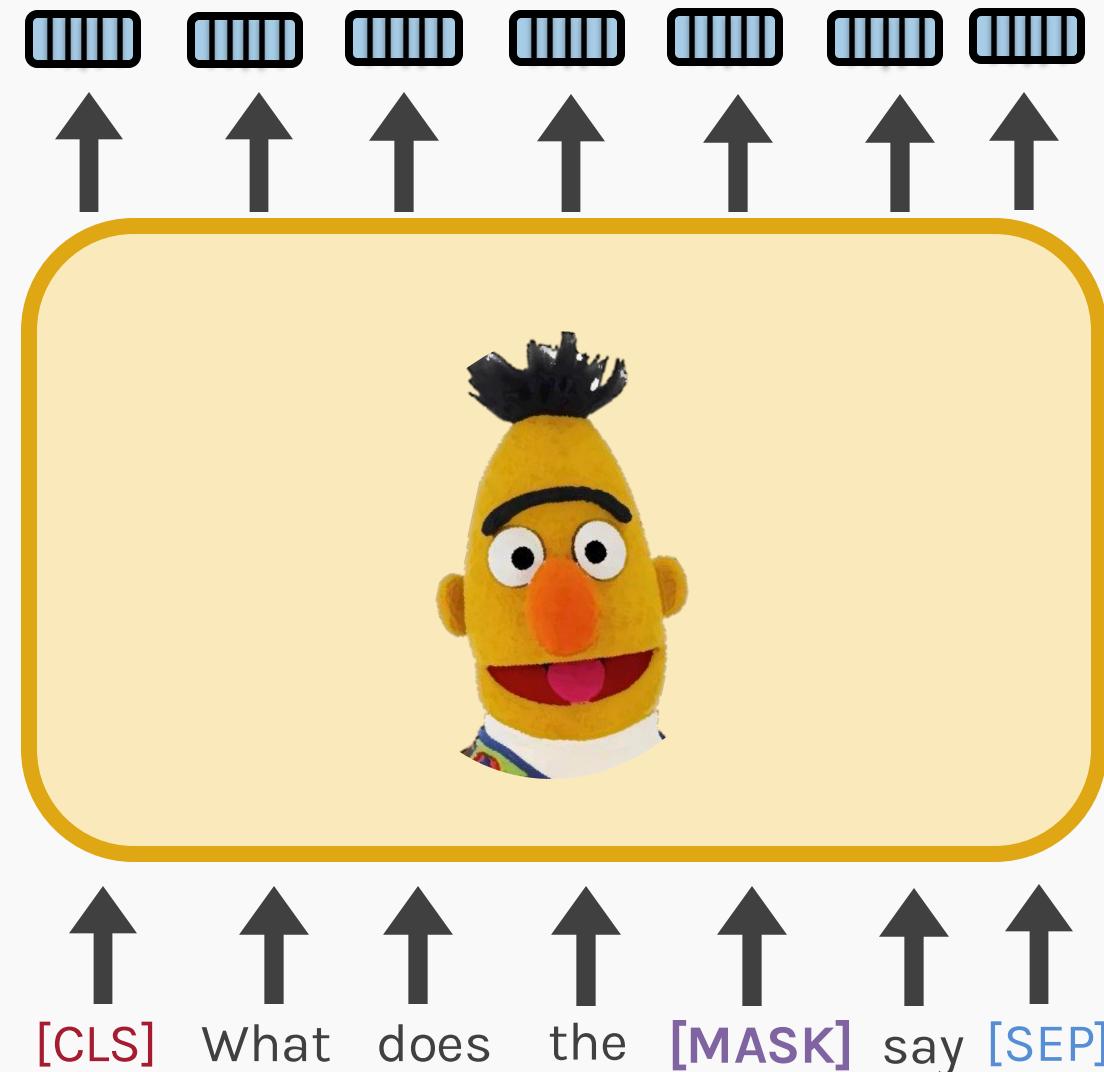
Two sentence task

TECHNICAL DETAILS

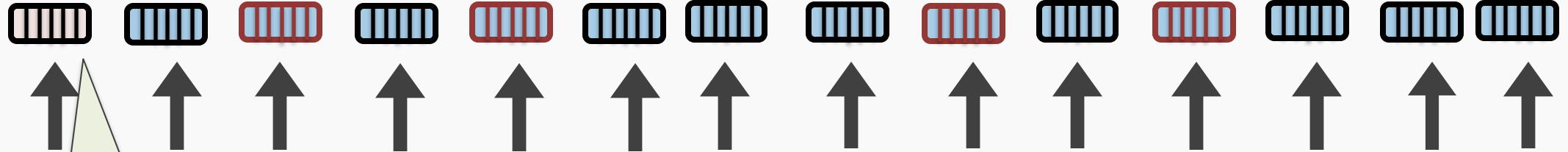
- To make BERT better at handling relationships between multiple sentences, the training process includes an additional task:

Given two sentences (A and B), is B likely to be the sentence that follows A, or not?

The individual sentences are separated by the [SEP] tag mentioned before.



This is the **last** class. The professor will **miss** you all



To assess if the two sentences follow each other, we use the embedding of the [CLS] tag from the start of the sentence

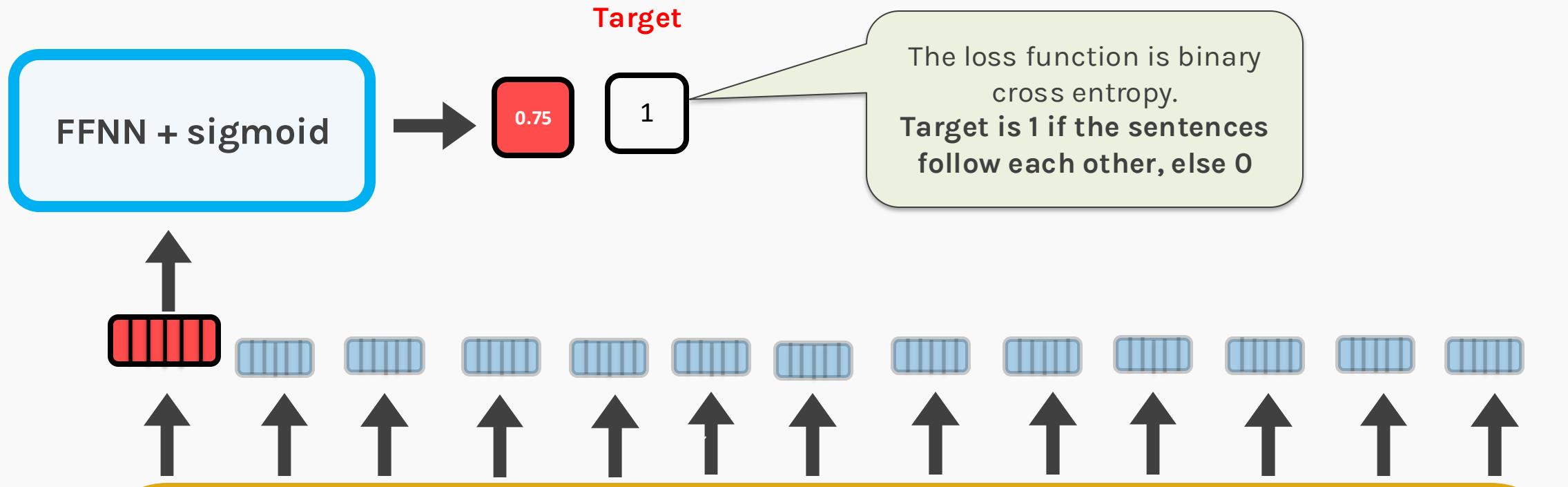


[CLS] This is the [MASK] class [SEP] The professor will [MASK] you all [SEP]

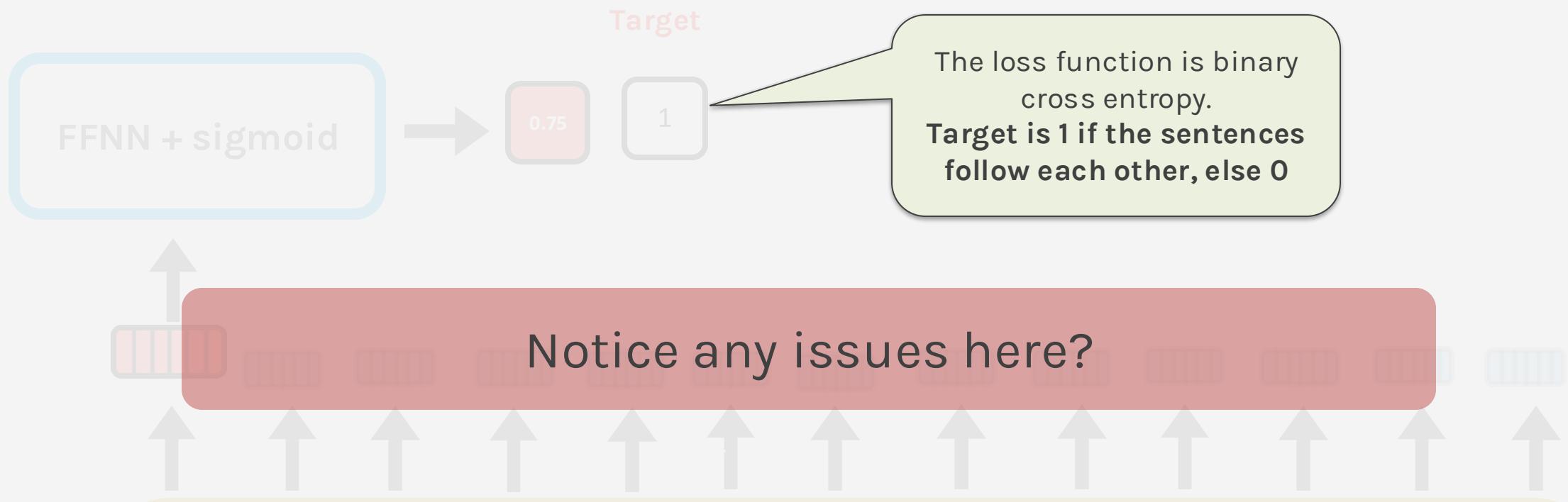
SENTENCE A

SENTENCE B

This is the **last** class. The professor will **miss you all**



This is the last class. The professor will miss you all



The texts used for training have continuous sentences that follow each other. Therefore, all of the examples for training would be labelled 1!

This is the last class. The professor will miss you all



Let us first take a quick class poll!

Class Poll - Negative Sampling



Do you think these two sentences belong together?

Tennis | requires | speed | and | agility.

Wimbledon | is | the | ultimate | tournament

Class Poll - Negative Sampling

CORRECT!

Tennis | requires | speed | and | agility.

Wimbledon | is | the | ultimate | tournament

Class Poll - Negative Sampling



How about now?

Tennis requires speed and agility.

Brown butter is the secret ingredient

Class Poll - Negative Sampling

CORRECT AGAIN, Humans!

Brown | butter | is | the | secret | ingredient

Tennis | requires | speed | and | agility.

Wimbledon | is | the | ultimate | tournament

Negative Sampling

However, this understanding is not intuitive for a language model.

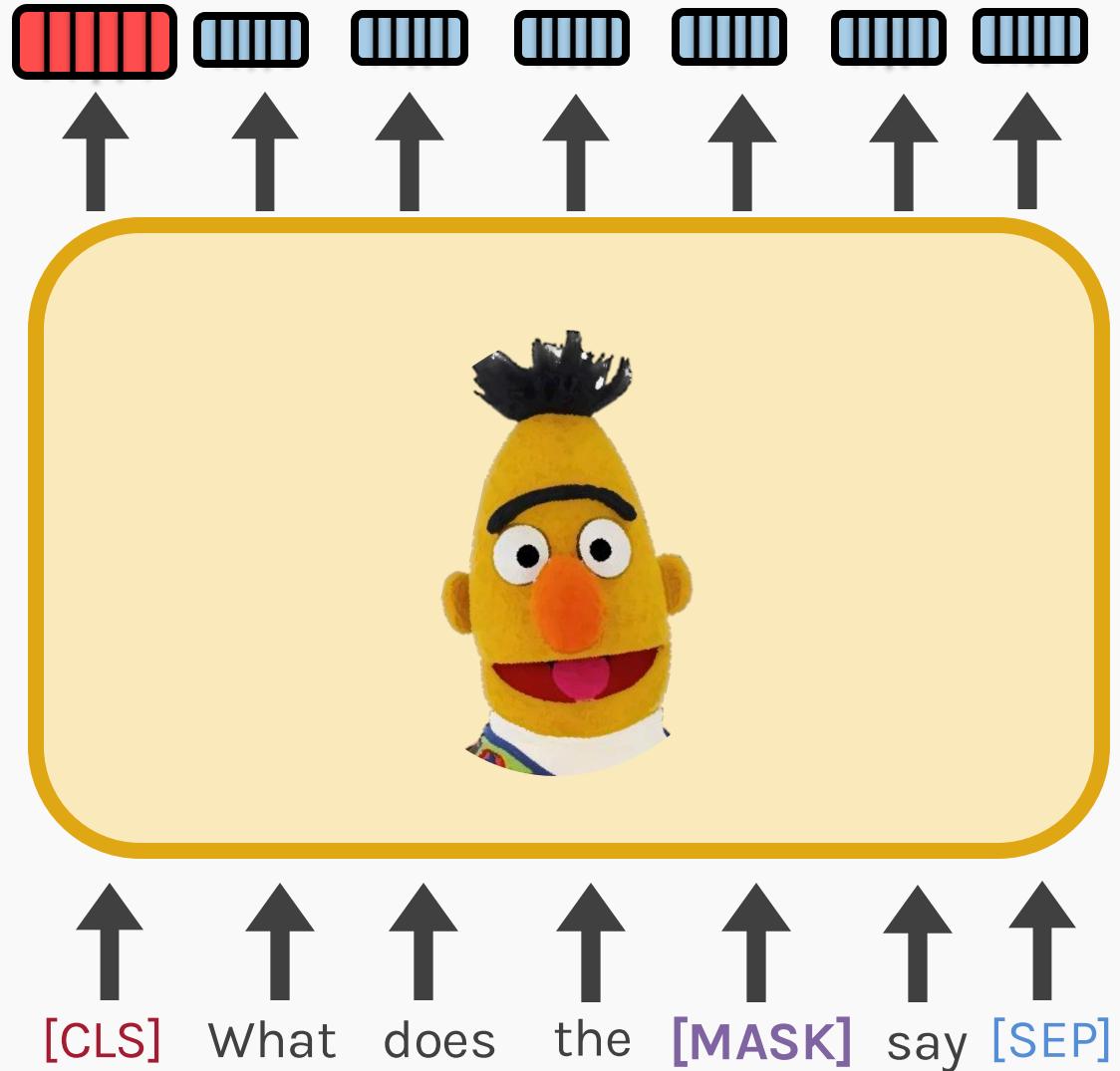
- We need to create ‘synthetic’ negative samples of sentences that don’t belong together to ‘teach’ the model.
- We can use sample sentences from widely different sources and train a simple classifier.

Sentence A	Sentence B	Target
Tennis requires speed and agility.	Wimbledon is the ultimate tournament.	1
Tennis requires speed and agility.	Brown butter is the secret ingredient.	0
Brown butter is the secret ingredient	Grand Slam titles define tennis legends.	0
Brown butter is the secret ingredient.	It adds a rich, nutty flavor.	1

How to train BERT?

CLIFF NOTES

- Using a large corpus (e.g. Wikipedia), we *pre-train* a BERT model for two tasks:
 - *Masked word* prediction
 - *Next sentence* prediction
- The goal is minimizing the combined loss function.
- The selected ([MASK]/true/random) tags are used for the language model task, and the [CLS] and [SEP] tag helps to train the next sentence prediction task.



Outline

- Transformers Recap
- BERT
- **Practical problems in Large Language Models**
- GPT
- Hugging Face

BERT ISSUES?

- The vanilla BERT (base) has **109,482,240** trainable parameters.
- An extremely **large** dataset is required to train such model sufficiently.
- The sheer size makes it extremely **slow** to **train**.
- The **fine-tuning** - we will explain later - requires lots of **tweaks** and **experimentation** (for e.g. you must use the *same* tokenizer used during pre-training).

Hey! I never said I was perfect!

Vanilla BERT cannot be used for natural text generation
(GPT can be used instead)



How to *use* BERT?

How to use a BERT?

There are three main steps in using BERT for a given NLP task

Pre-training



Fine-tuning



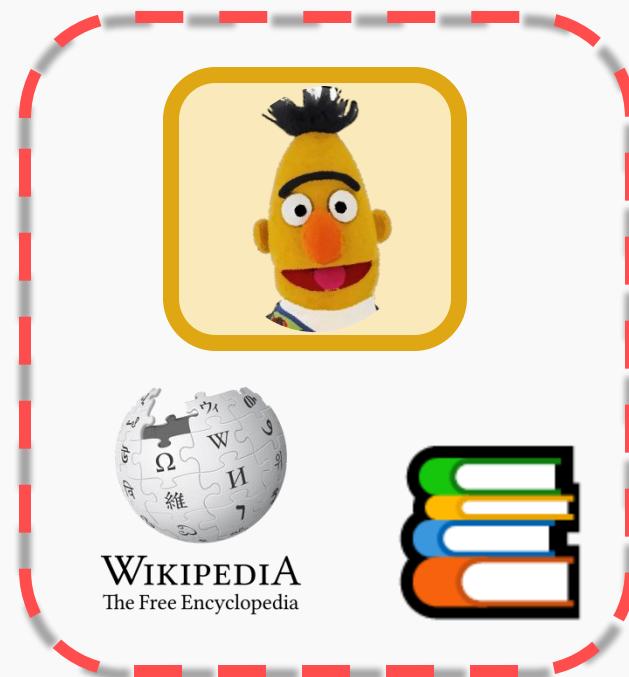
Language tasks



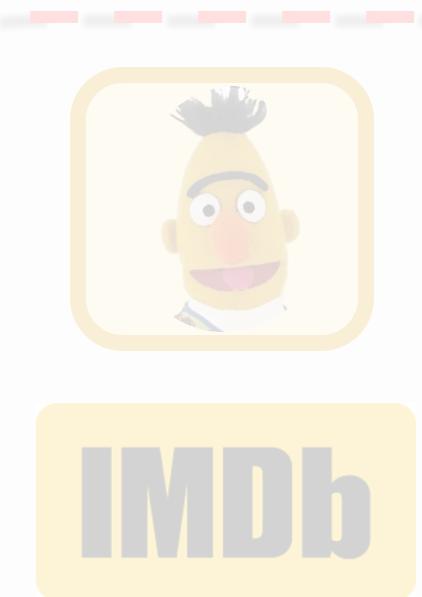
How to use a BERT?

There are three main steps in using BERT for a given NLP task

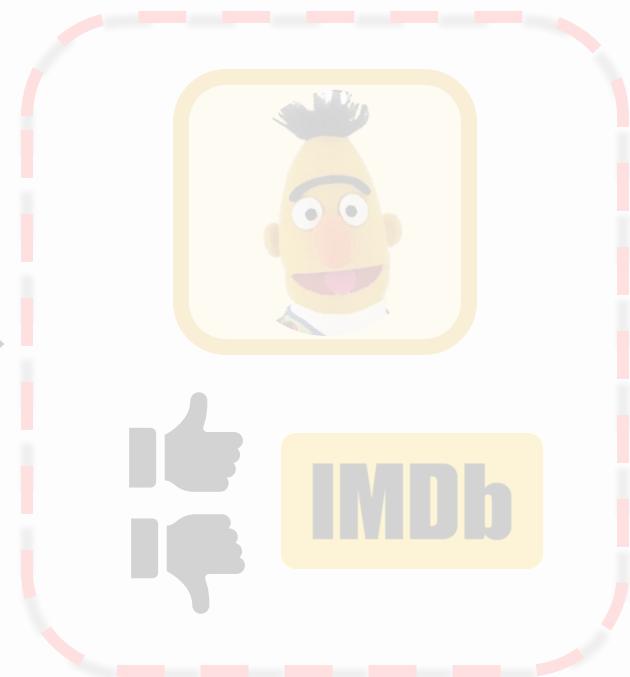
Pre-training



Fine-tuning



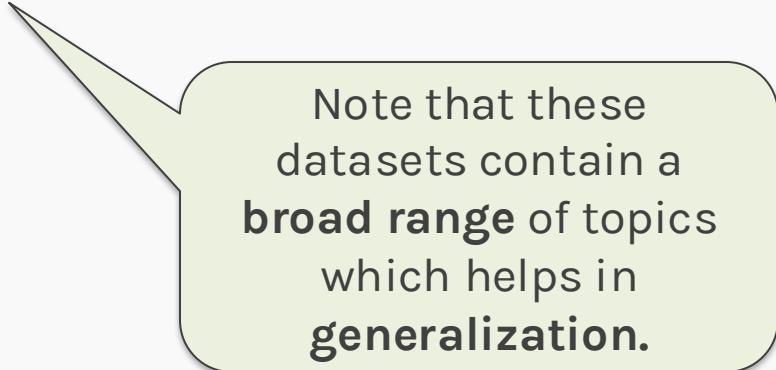
Language tasks



Pre-training

We train a Language Model on a large dataset!

For example, BERT Large was pretrained on BooksCorpus which contains **800 million** words and Wikipedia which contains **2.5 billion** words.



Note that these datasets contain a **broad range** of topics which helps in **generalization**.

Pre-training issues

Large dataset

Uncompressed, the total amount of data is roughly **40 TB**.

That amount of storage is not unattainable but will imply an investment to store that much data.

Computational Complexity

BERT was trained for 40 epochs on 16 TPUs.

This pre-training takes around **4 days** to complete and impacts the **environment!**

Assuming you don't have that hardware or resources at your disposal, the next solution is to use cloud resources. At a 12 USD an hour, for todays cost it would cost around **18000 USD**.

Pre-training issues

Large dataset

Uncompressed, the total amount of data is roughly **40 TB**.

That amount of storage is

Computational Complexity

BERT was trained for 40 epochs on 16 TPUs.

This pre-training takes around **4**

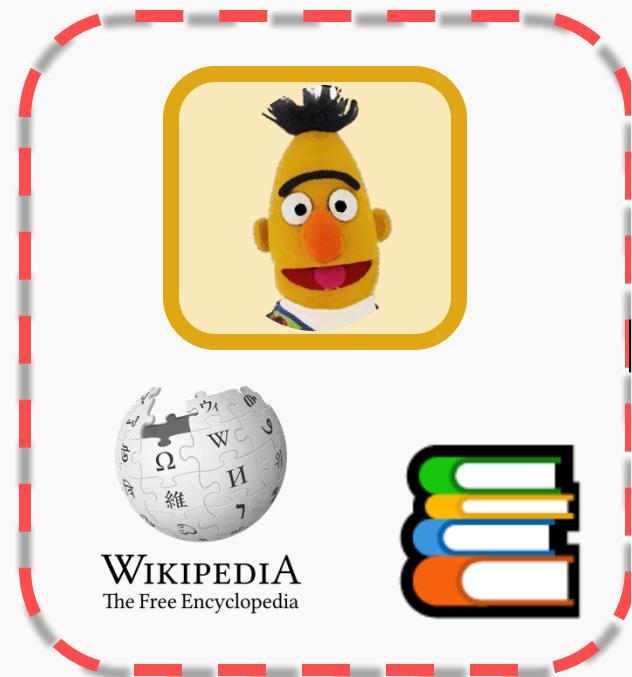
**Let someone else do it for us
(transfer learning!)**

cloud resources. At a 12 USD an hour, for todays cost it would cost around **18000 USD**.

How to use a BERT?

There are three main steps in using BERT for a given NLP task

Pre-training



Fine-tuning



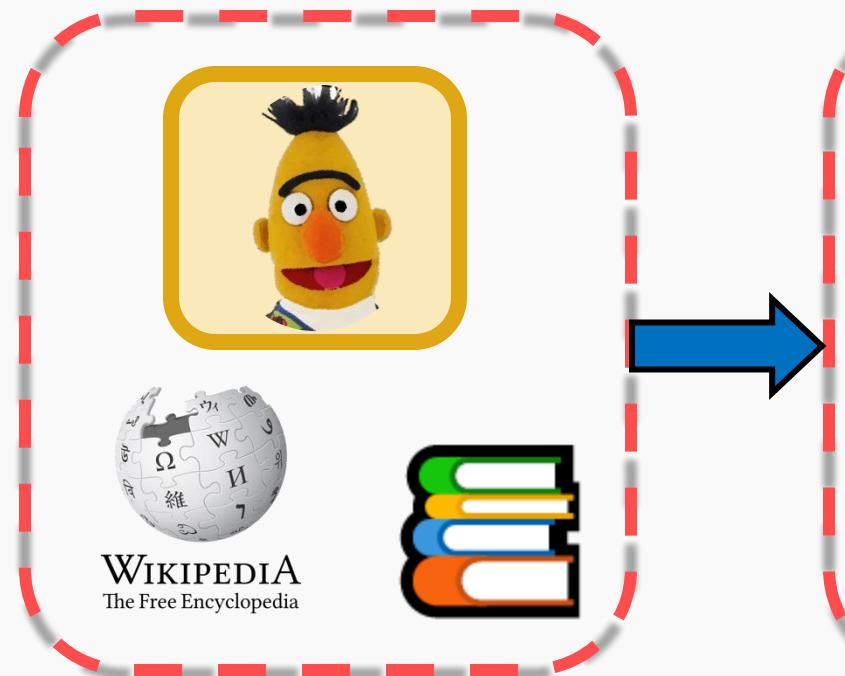
Language tasks



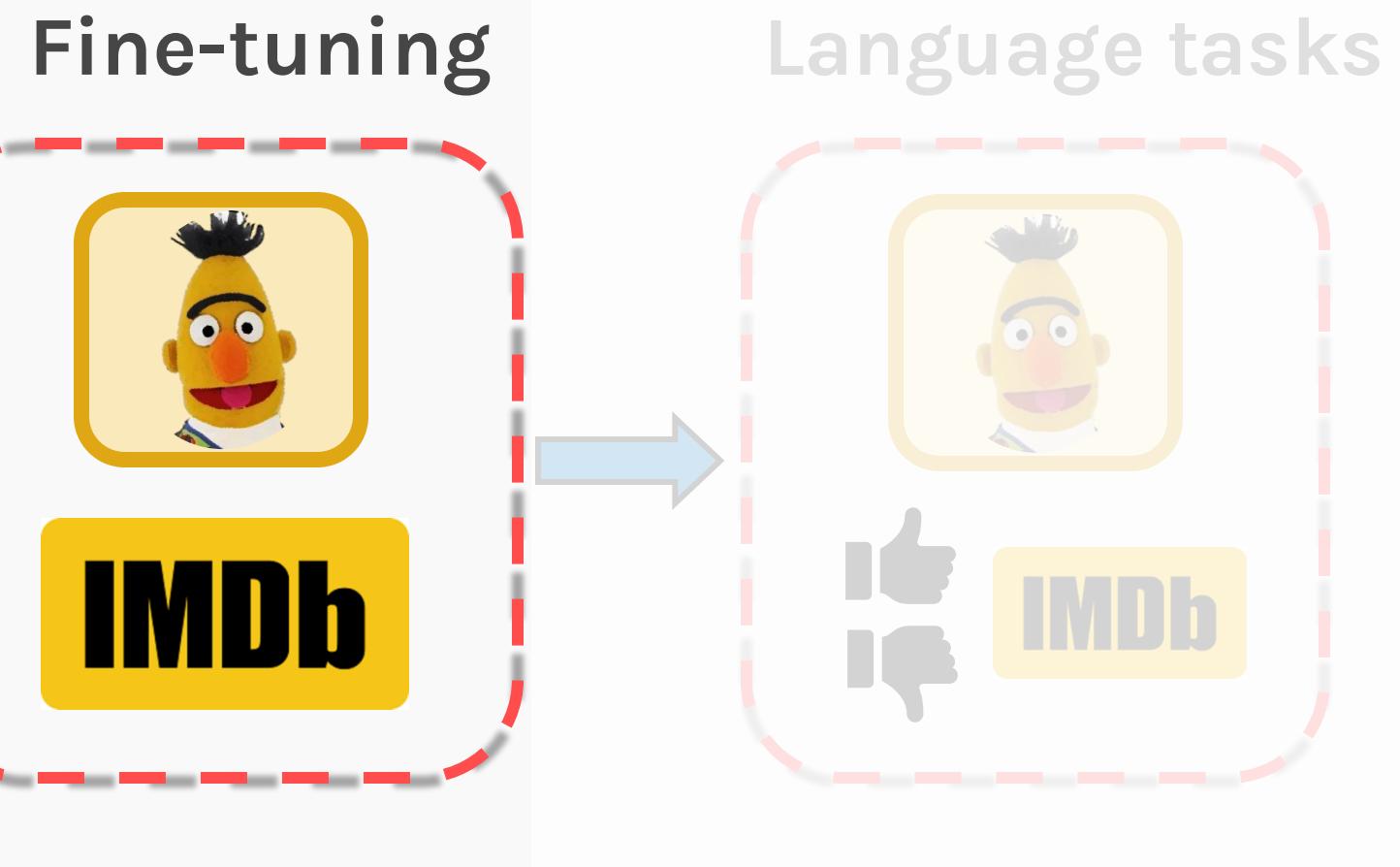
How to use a BERT?

There are three main steps in using BERT for a given NLP task

Pre-training



Fine-tuning



Why fine-tune?

What if our requirement is domain specific, like scientific journals or legal contracts?

By fine-tuning the language model on in-domain data, you can boost the performance of many downstream tasks.



Fine-tuning is the process of providing a model with labelled examples to update its weight and improve performance on a specific task!

How to fine-tune?

We do **Masked Language Modelling** to fine-tune BERT.

This is similar to what we do when we pretrain the model. The only difference **is** that we load pretrained weights trained on a very large corpus before fine-tuning it to a domain specific dataset.

For a movie dataset:

This is a great [MASK]. →

This is a great deal.

This is a great success.

This is a great adventure.

This is a great idea.

This is a great [MASK]. →

This is a great movie.

This is a great film.

This is a great story.

This is a great character.

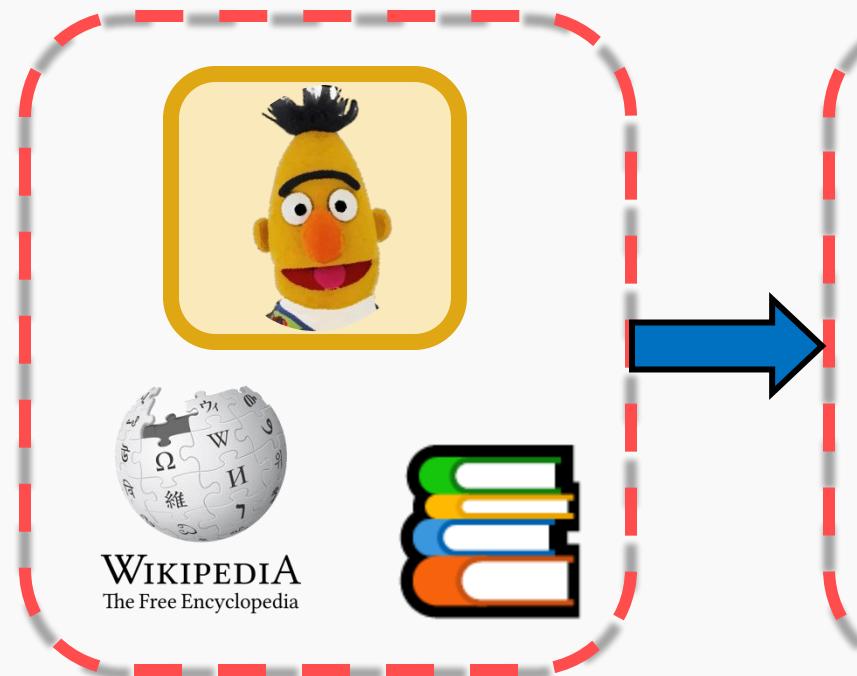
Before Fine Tuning

After Fine Tuning

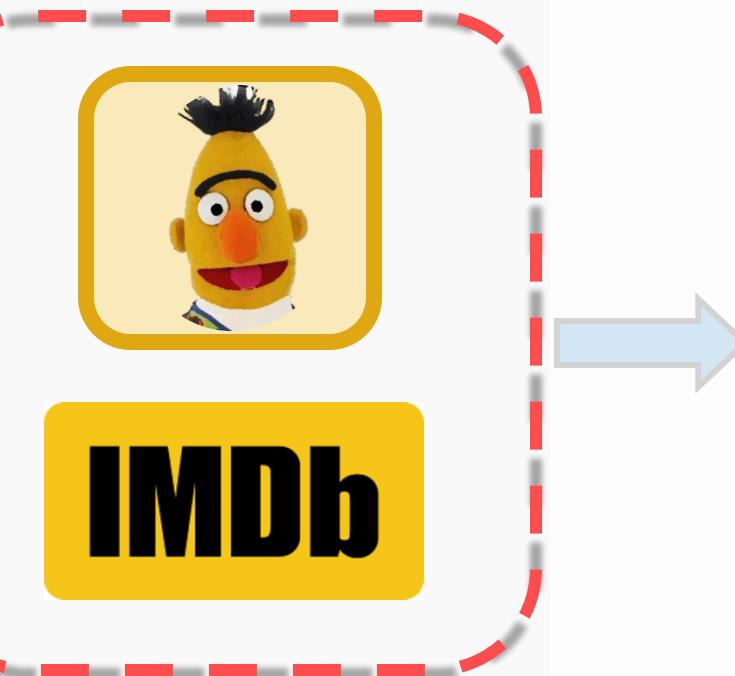
How to use a BERT?

There are three main steps in using BERT for a given NLP task

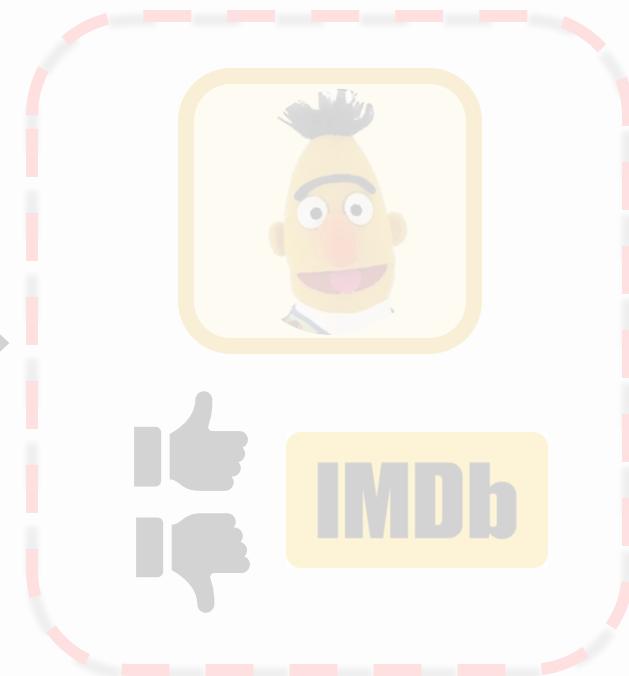
Pre-training



Fine-tuning



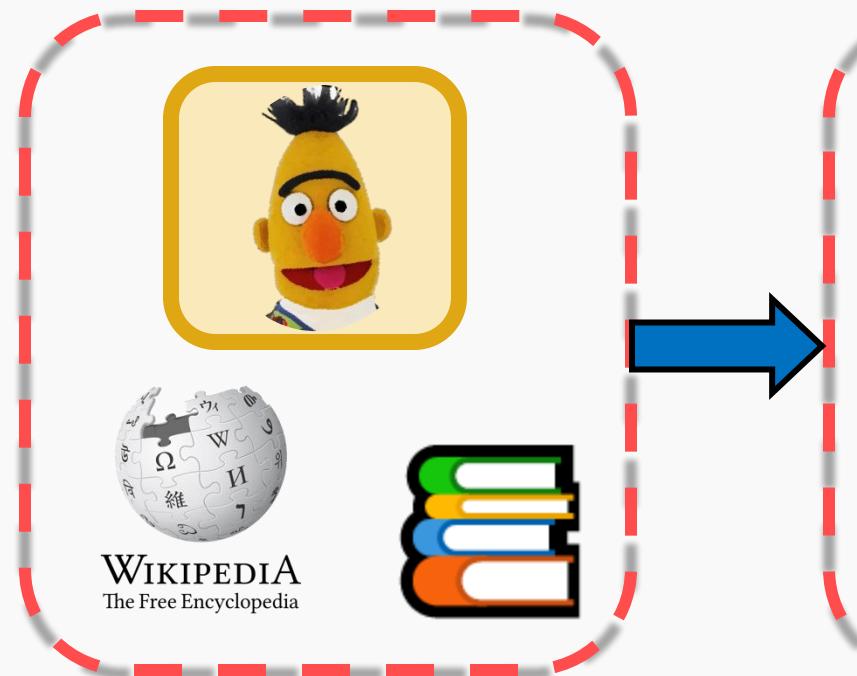
Language tasks



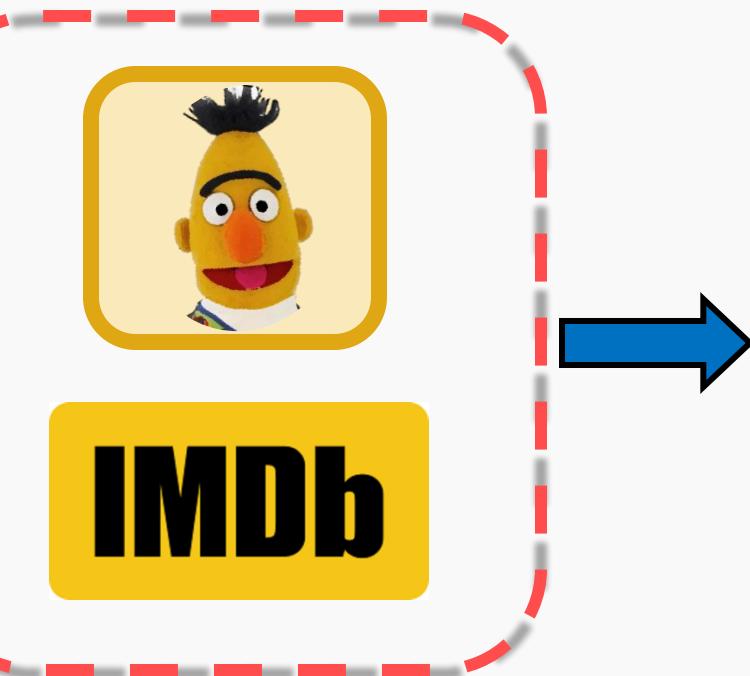
How to use a BERT?

There are three main steps in using BERT for a given NLP task

Pre-training



Fine-tuning



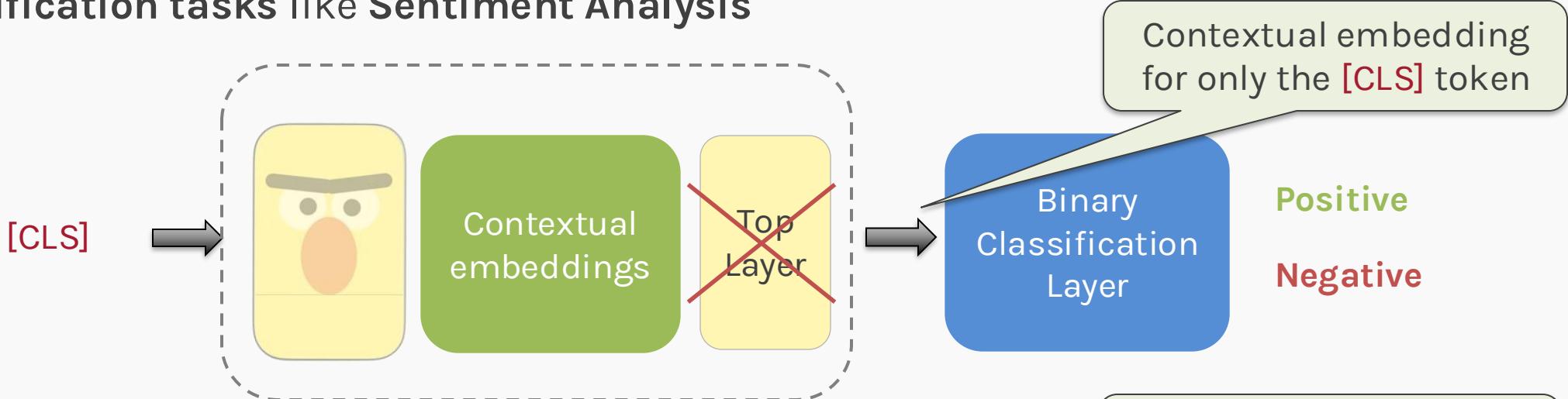
Language tasks



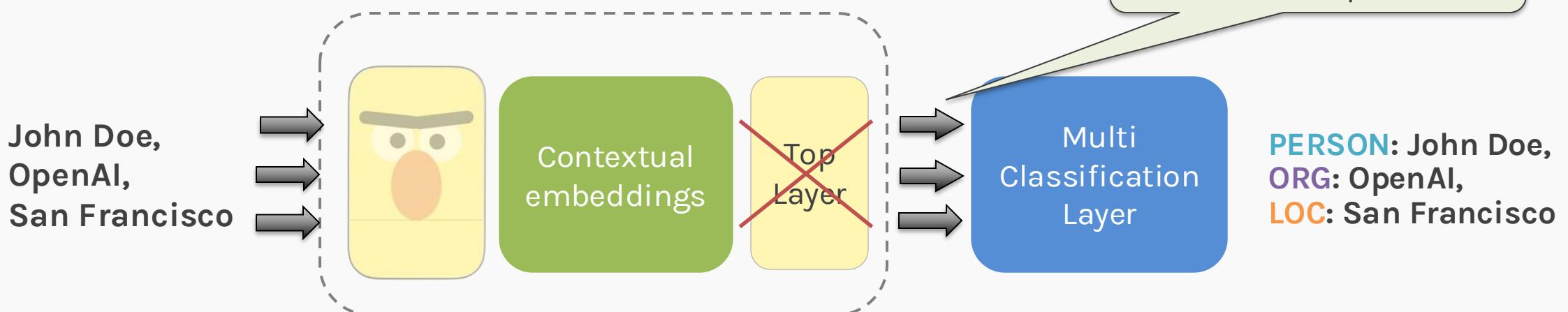
How to use BERT for NLP Tasks?

BERT can be used for a wide variety of language tasks, for example:

(A) Classification tasks like Sentiment Analysis

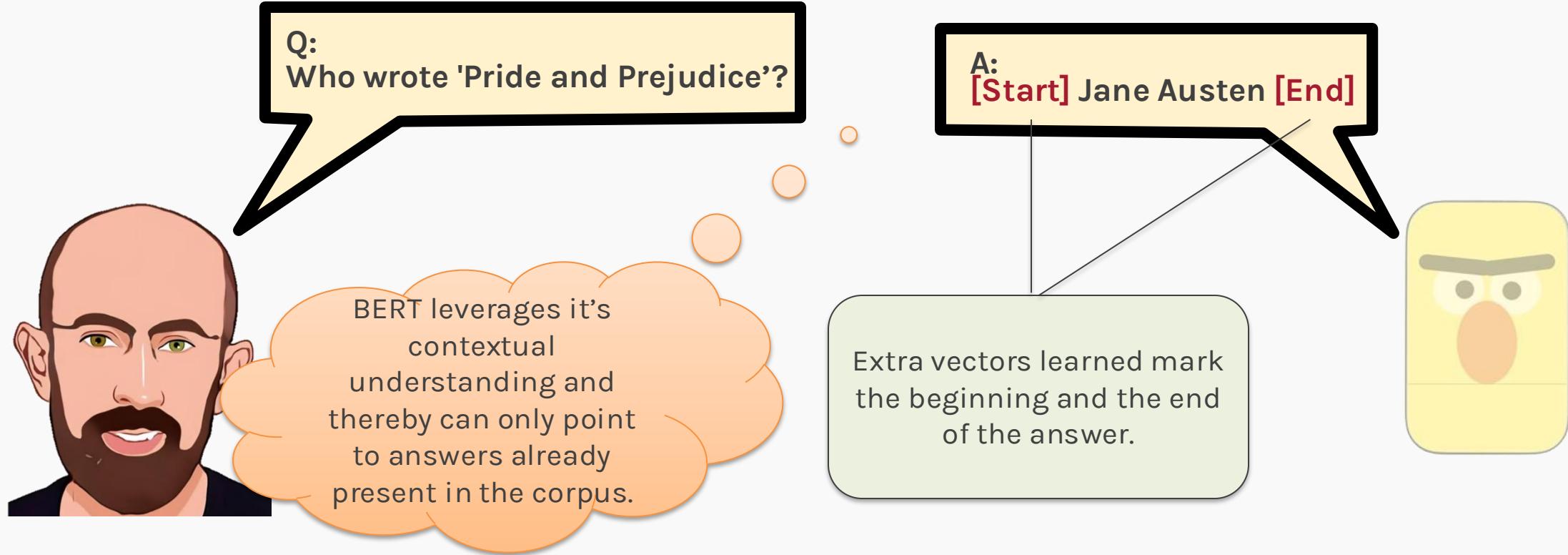


Named Entity Recognition (NER)



How to use BERT for NLP Tasks?

(B) Question Answering tasks



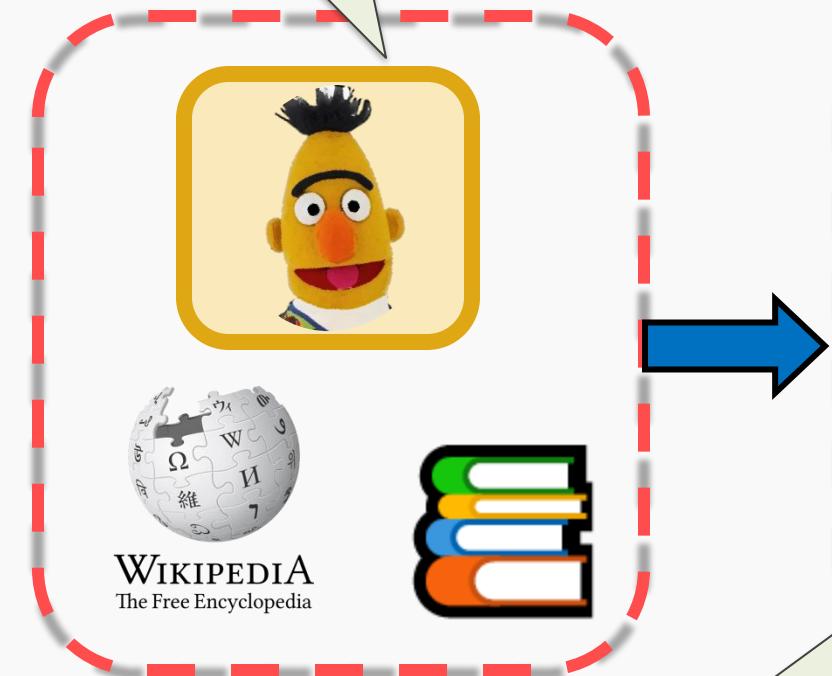
(C) various other tasks like language translation, semantic text similarity, search relevance, spell check and grammar check, etc.

How to use a BERT?

You will (almost)
never train a BERT
from scratch

There are three main steps in using BERT for a given NLP task

Pre-training



Fine-tuning



Language tasks



We don't use the
labels in the fine-
tuning step

The final step is the
supervised learning
using labels

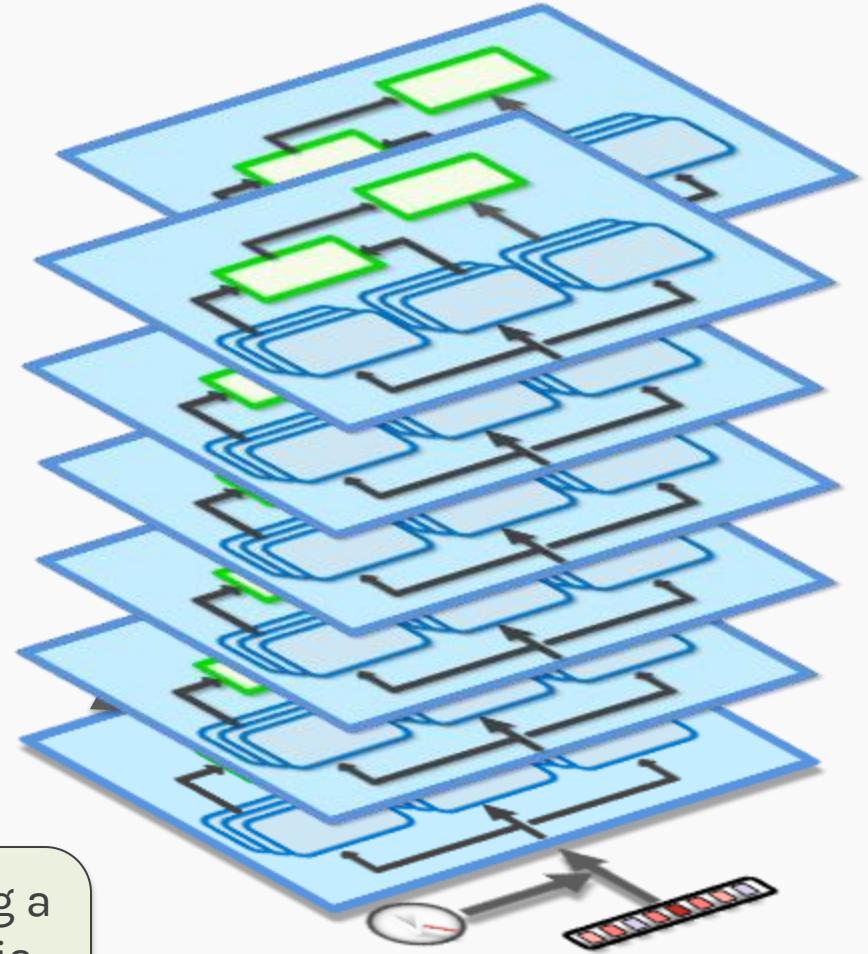
Outline

- Transformers Recap
- BERT
- Practical problems in Large Language Models
- GPT
- Hugging Face

From Transformers to GPT

- We saw how we can make use of the encoder blocks of the Transformers in BERT. Now, what if we use just the decoder blocks?
- Voilà, The **Generative Pre-trained Transformer!** Or GPT in short.

Note that we are simply using a decoder like architecture. It is not essentially a decoder anymore!



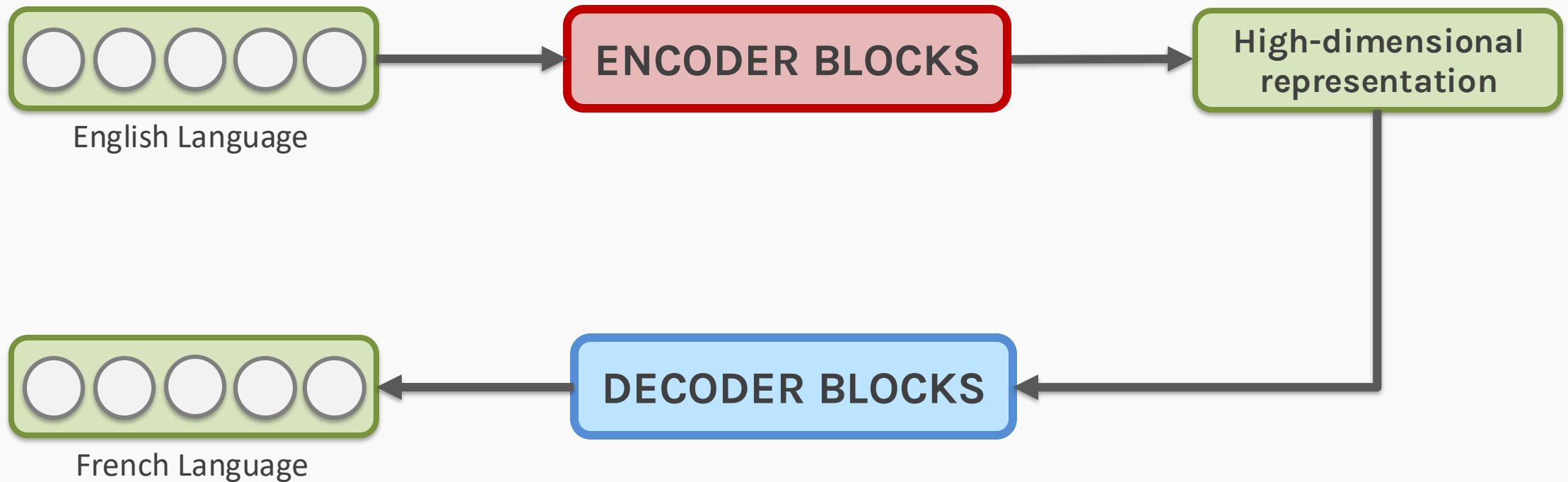
BERT v/s GPT: Key Difference

- Apart from the fact that GPT uses decoder like blocks of the transformer while BERT uses encoder like blocks, there is another key difference between the two models.
- BERT is auto-encoding, while GPT is auto-regressive.

But what do auto-encoding and auto-regressive mean?

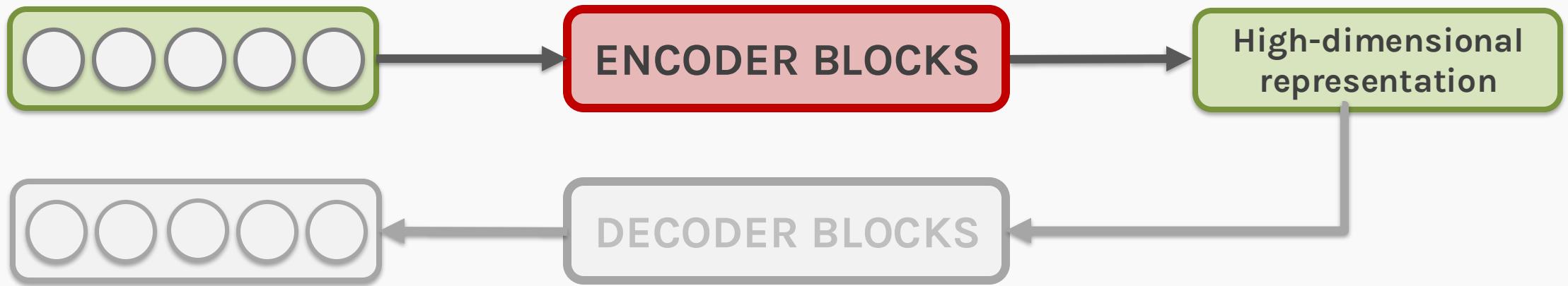
BERT v/s GPT: Key Difference

- Before we understand what those two terms mean, let's recap what the original transformer was tasked to do.



Auto-encoding Model

- Now if we take just the **encoder blocks** and train that part of the network separately, we have an **auto-encoding** model.



- The aim of an auto-encoding model is to learn a representation for a set of data. One way to do this can be to train the model on a reconstruction task.

Auto-encoding Model/BERT

- Now if we take just the encoder blocks and train that part of the network separately, we have an auto-encoding model.



- The aim of an auto-encoding model is to learn a representation for a set of data. One way to do this can be to train the model on a reconstruction task.
- For e.g. BERT

Auto-regressive Model

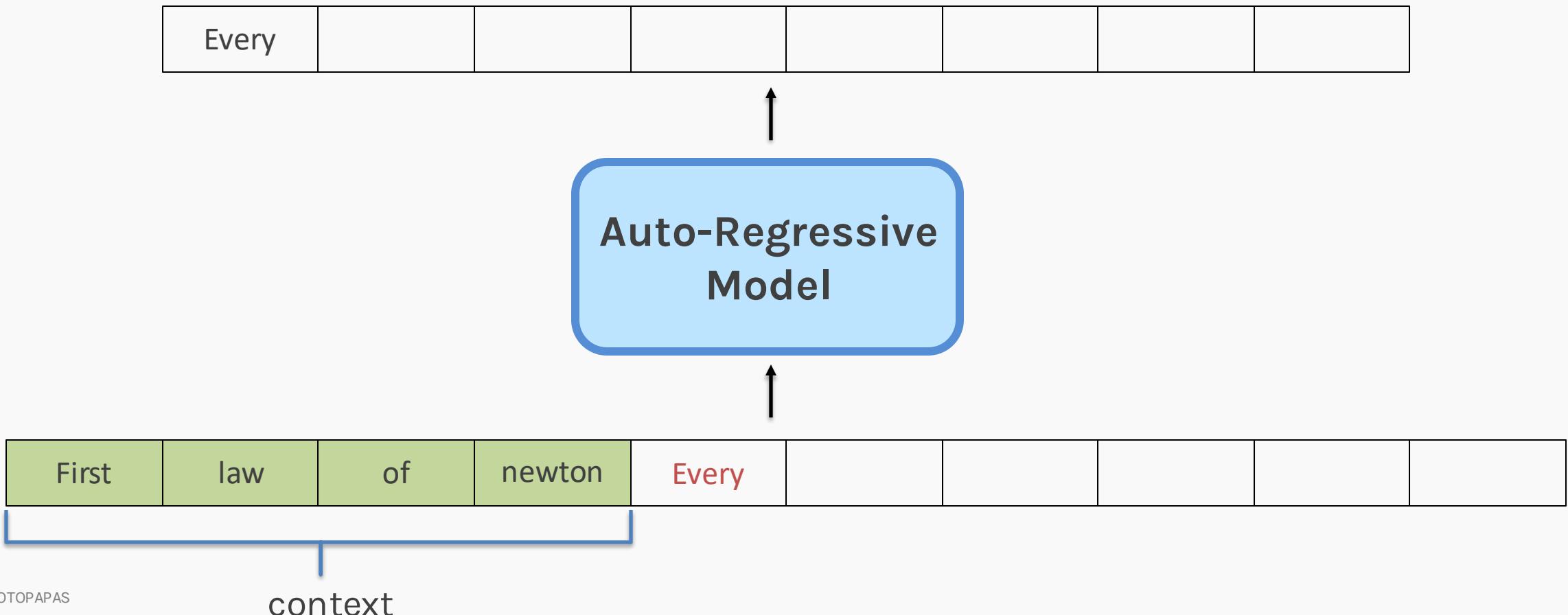
- Now if we take just the **decoder blocks** and train that part of the network separately, we have an **auto-regressive** model.



- A model is said to be auto-regressive if it predicts future values based on past values.

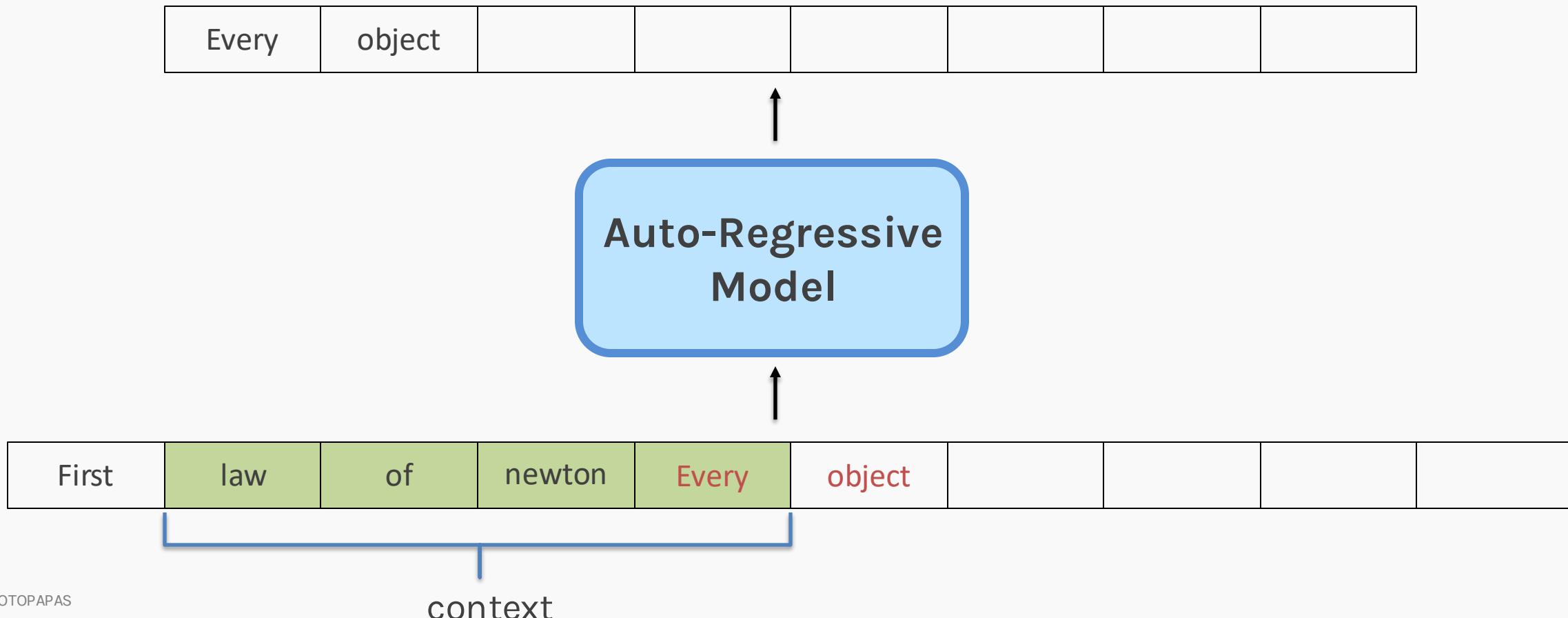
Auto-regressive Model

- Unidirectional processing and a fixed length sliding window are characteristic of a general auto-regressive model



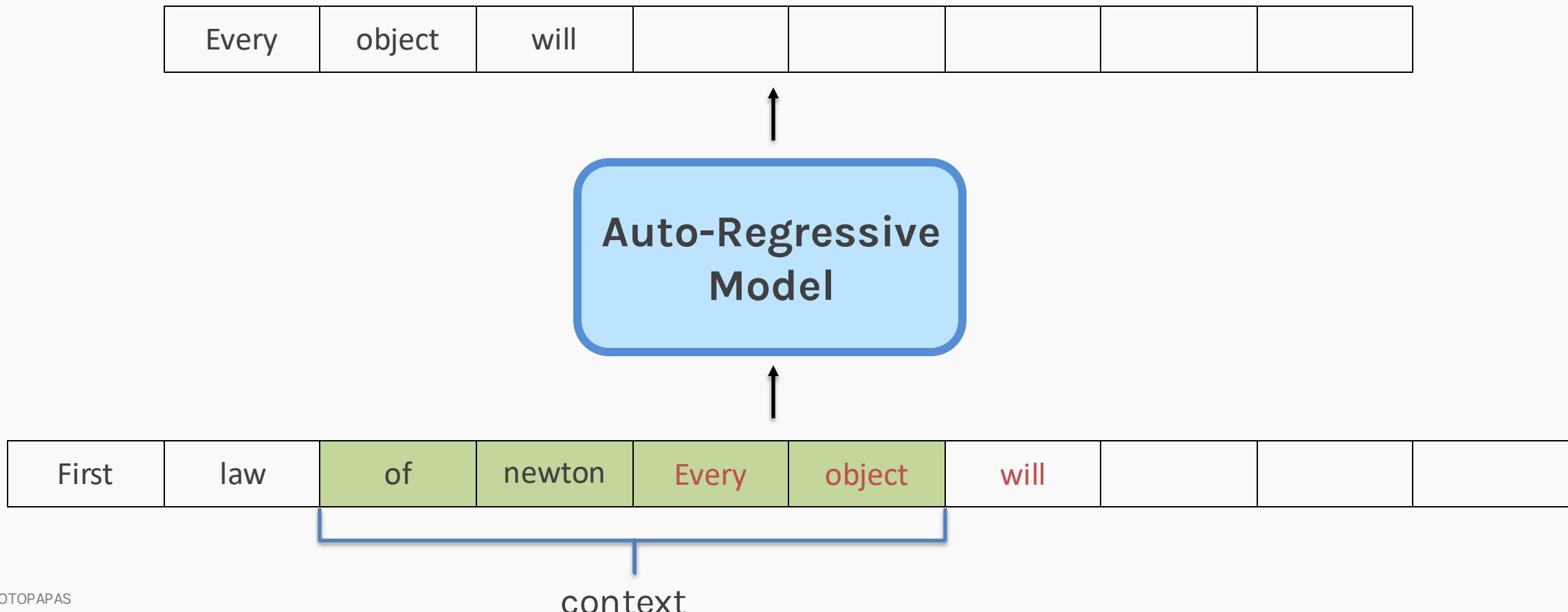
Auto-regressive Model

- Unidirectional processing and a fixed length sliding window are characteristic of a general auto-regressive model



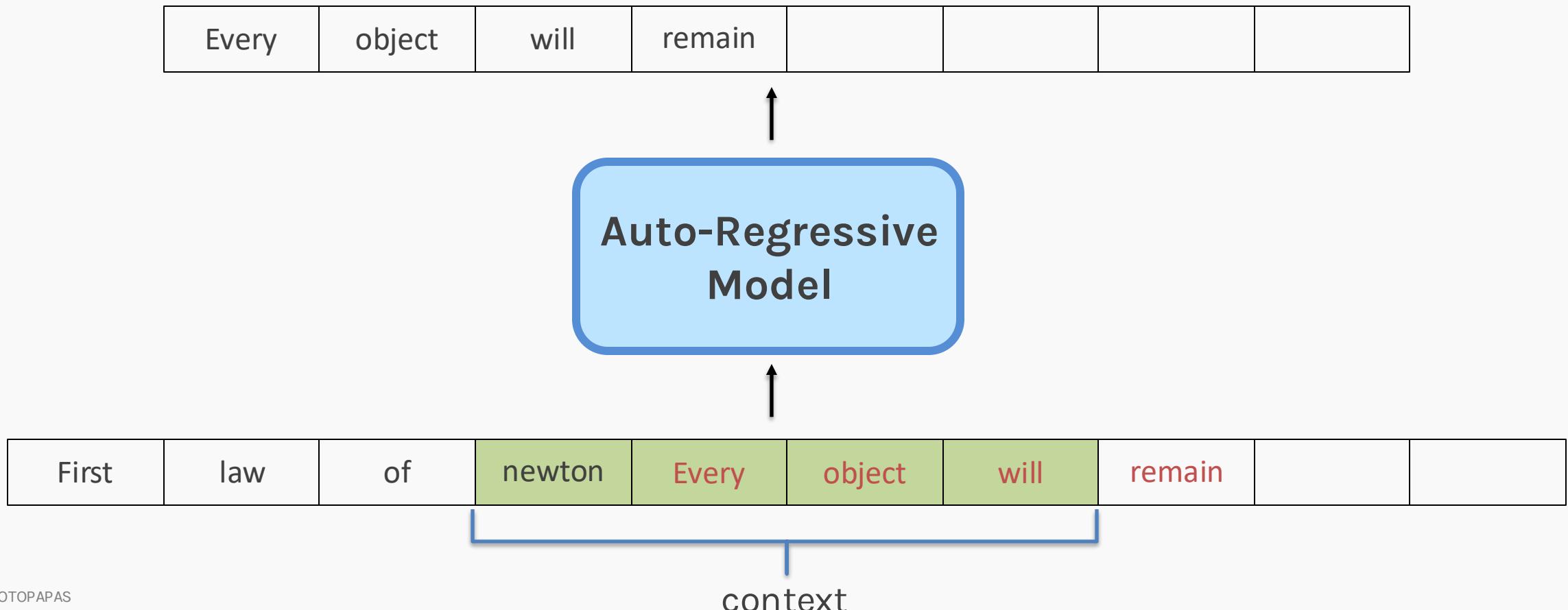
Auto-regressive Model

- Unidirectional processing and a fixed length sliding window are characteristic of a general auto-regressive model



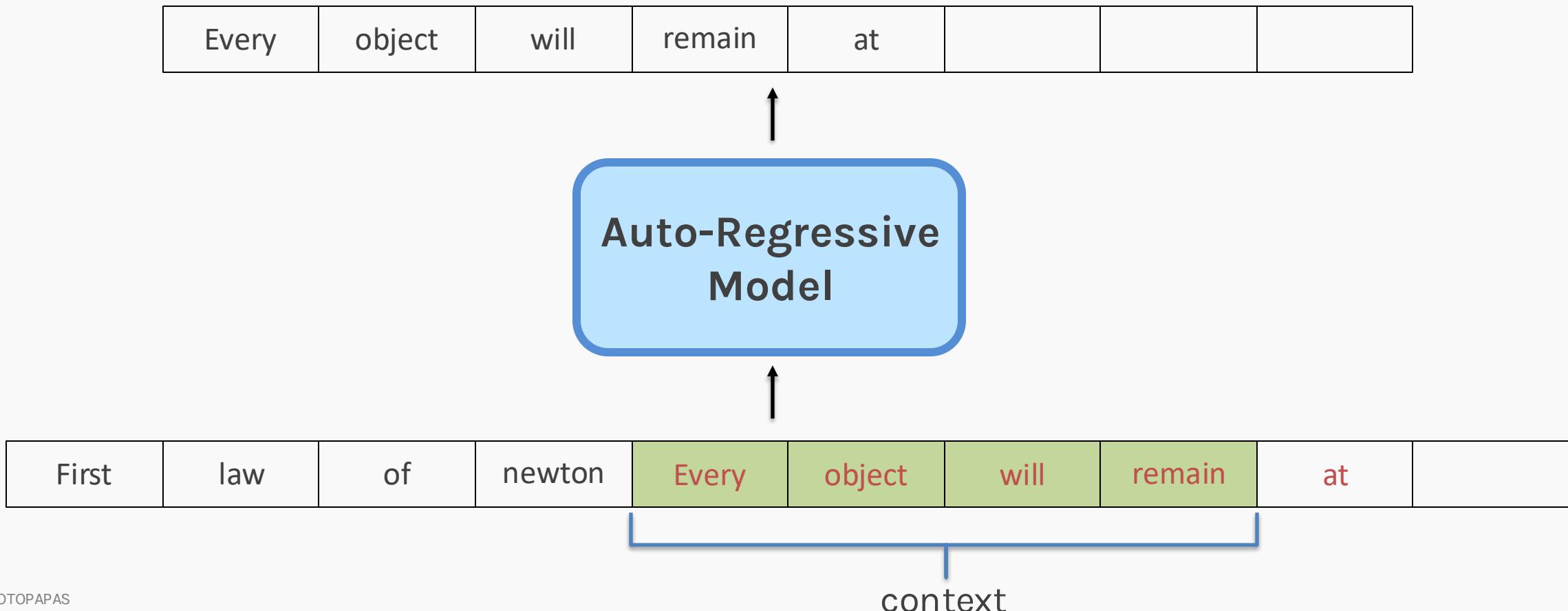
Auto-regressive Model

- Unidirectional processing and a fixed length sliding window are characteristic of a general auto-regressive model



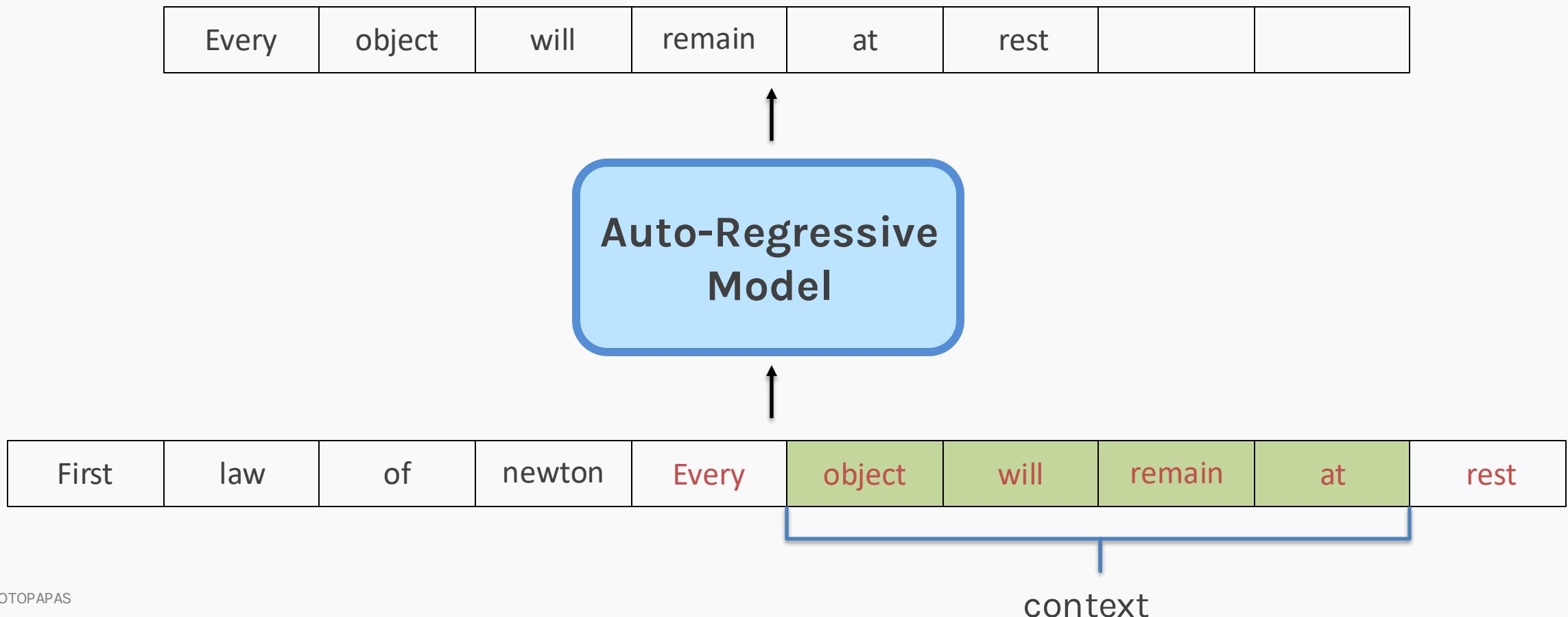
Auto-regressive Model

- Unidirectional processing and a fixed length sliding window are characteristic of a general auto-regressive model



Auto-regressive Model

- Unidirectional processing and a fixed length sliding window are characteristic of a general auto-regressive model



- GPT belongs to the auto-regressive family of models.

How do we train a GPT model?

- GPT belongs to the auto-regressive family of models.

How do we train a GPT model?



Since every next prediction is dependent on the previous one, don't we train this sequentially?



No, that will be very inefficient considering the increasing size of the transformer models

- GPT belongs to the auto-regressive family of models.

How do we train a GPT model?

Then how do we train
this?

It's MAGIC! Let me
show you.



GPT - Training

- The goal when training an auto-regressive model is to not let the model have a **sneak peak** into the **future** tokens when training.

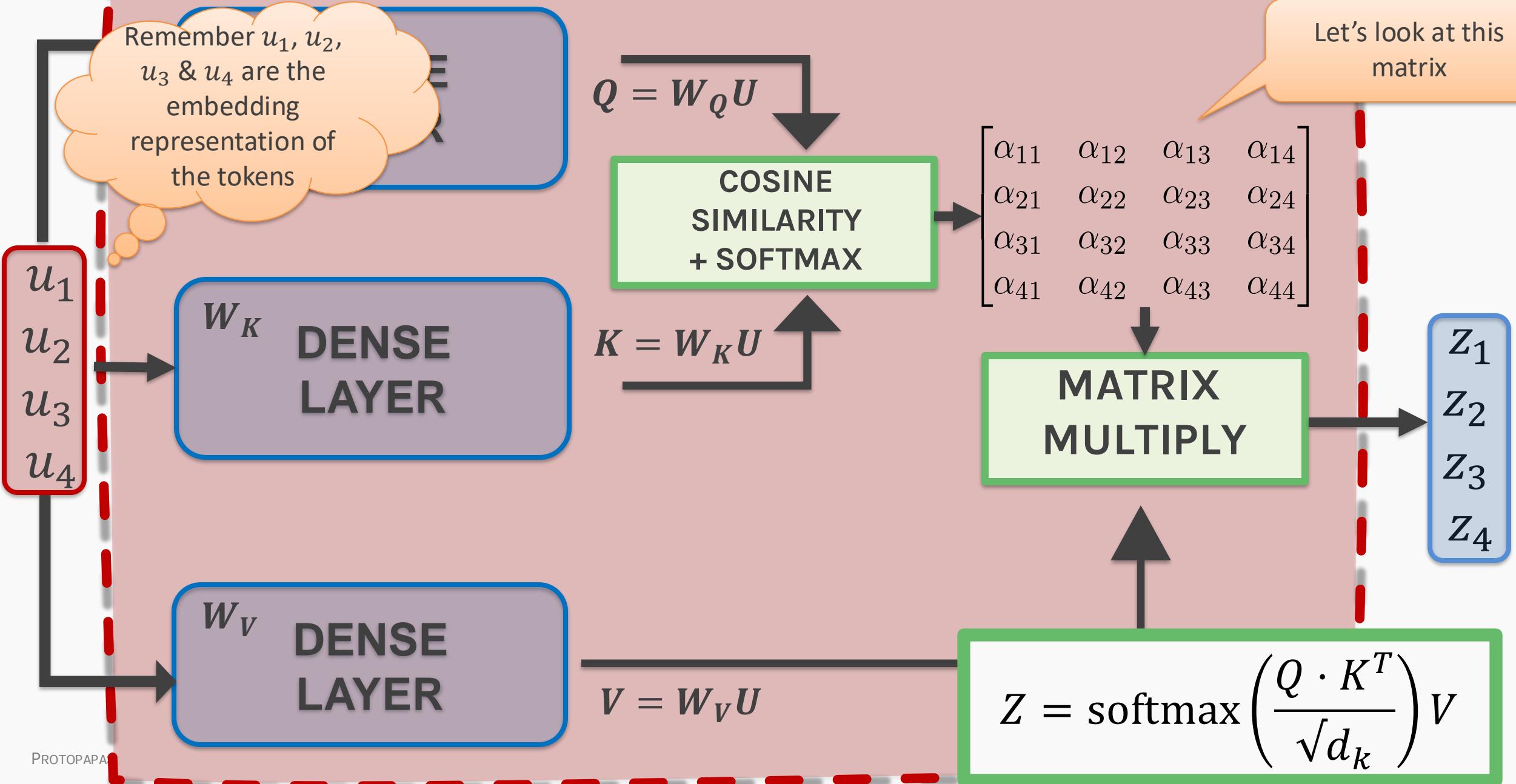


But, how can we do this if we don't train sequentially?

We modify the self-attention calculation.



ATTENTION BLOCK - Recap



Attention - Recap

Remember, α_{12} is the similarity score for the **first** transformed token with the **second** transformed token

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix}$$

Attention - Recap



But, doesn't this mean that model is looking at the future here if it is calculating the similarity score between **first** and the **third** transformed token?

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix}$$

α_{13} is the similarity score for the **first** transformed token with the **third** transformed token

Exactly! How about we mask the problematic scores?



Masked Self-Attention



$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \times \text{Mask M} = \begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix}$$

Mask M

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

This has been simplified for demonstration purposes. (The softmax would also need to be modified)

Masked Self-Attention

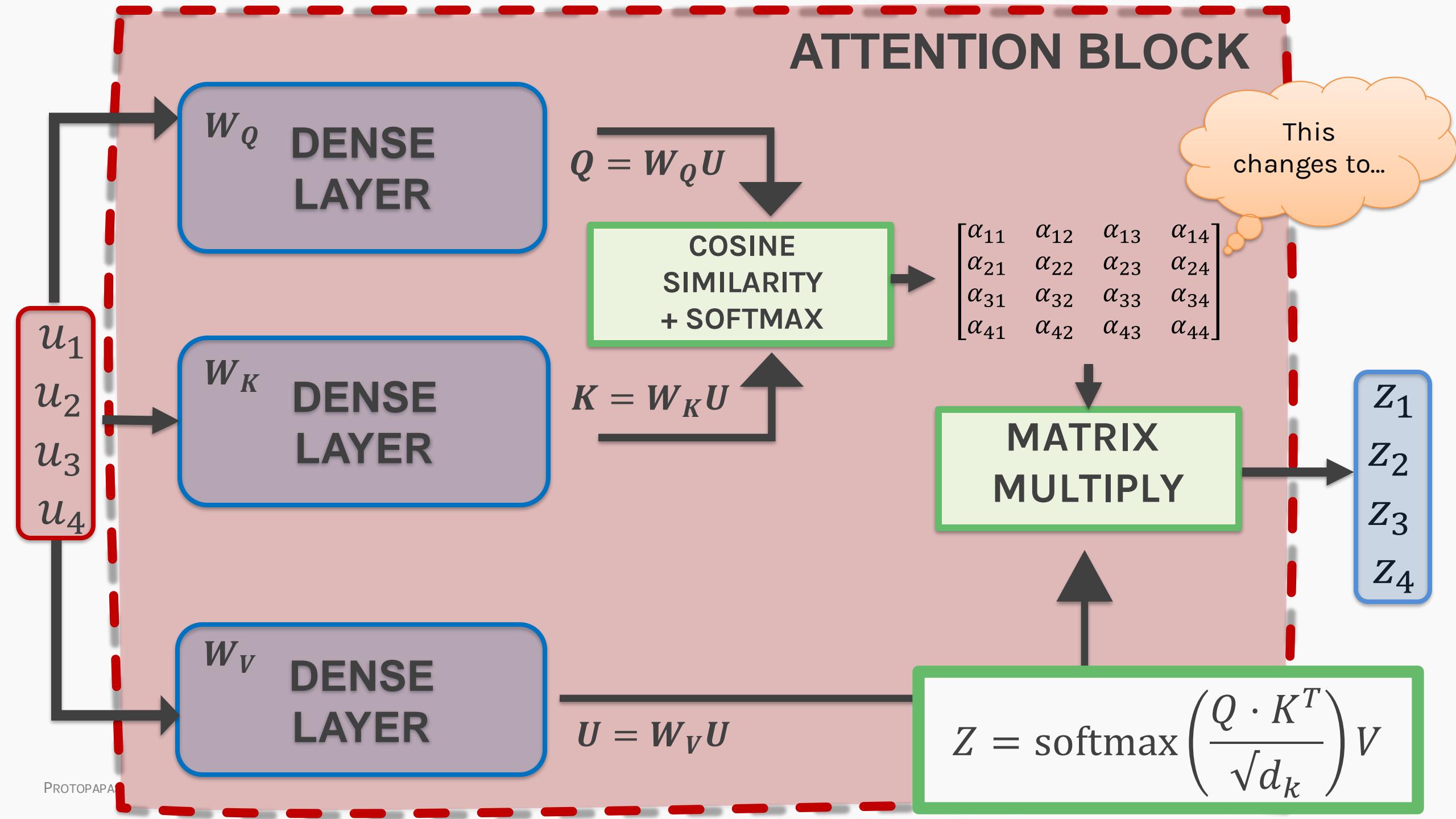
- Now when this is multiplied by the values, we get the desired outcome.

Remember, this is just another representation of the token embeddings, just like query and keys.

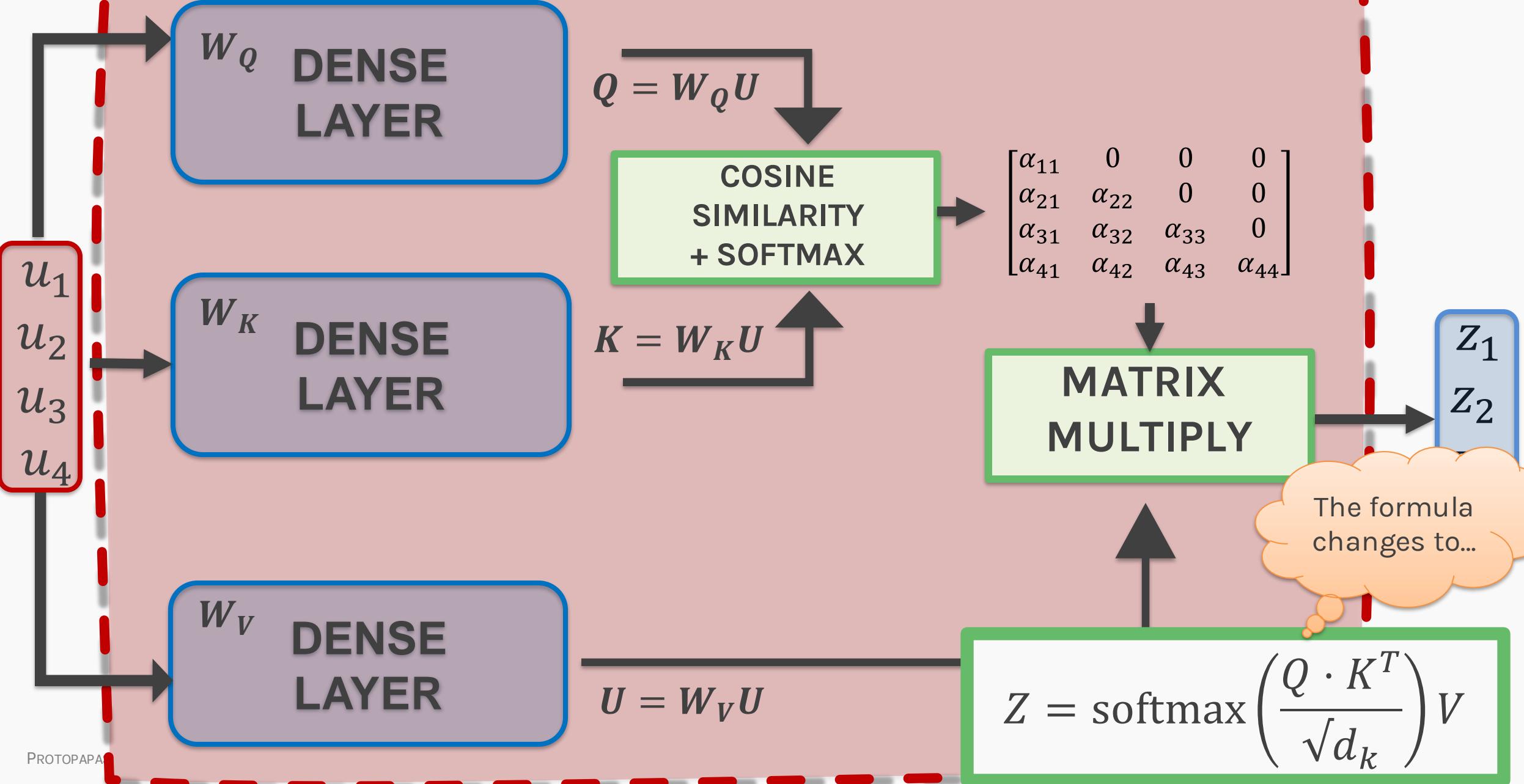
$$\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \times \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} \alpha_{11}V_1 \\ \alpha_{21}V_1 + \alpha_{22}V_2 \\ \alpha_{31}V_1 + \alpha_{32}V_2 + \alpha_{33}V_3 \\ \alpha_{41}V_1 + \alpha_{42}V_2 + \alpha_{43}V_3 + \alpha_{44}V_4 \end{bmatrix}$$

values

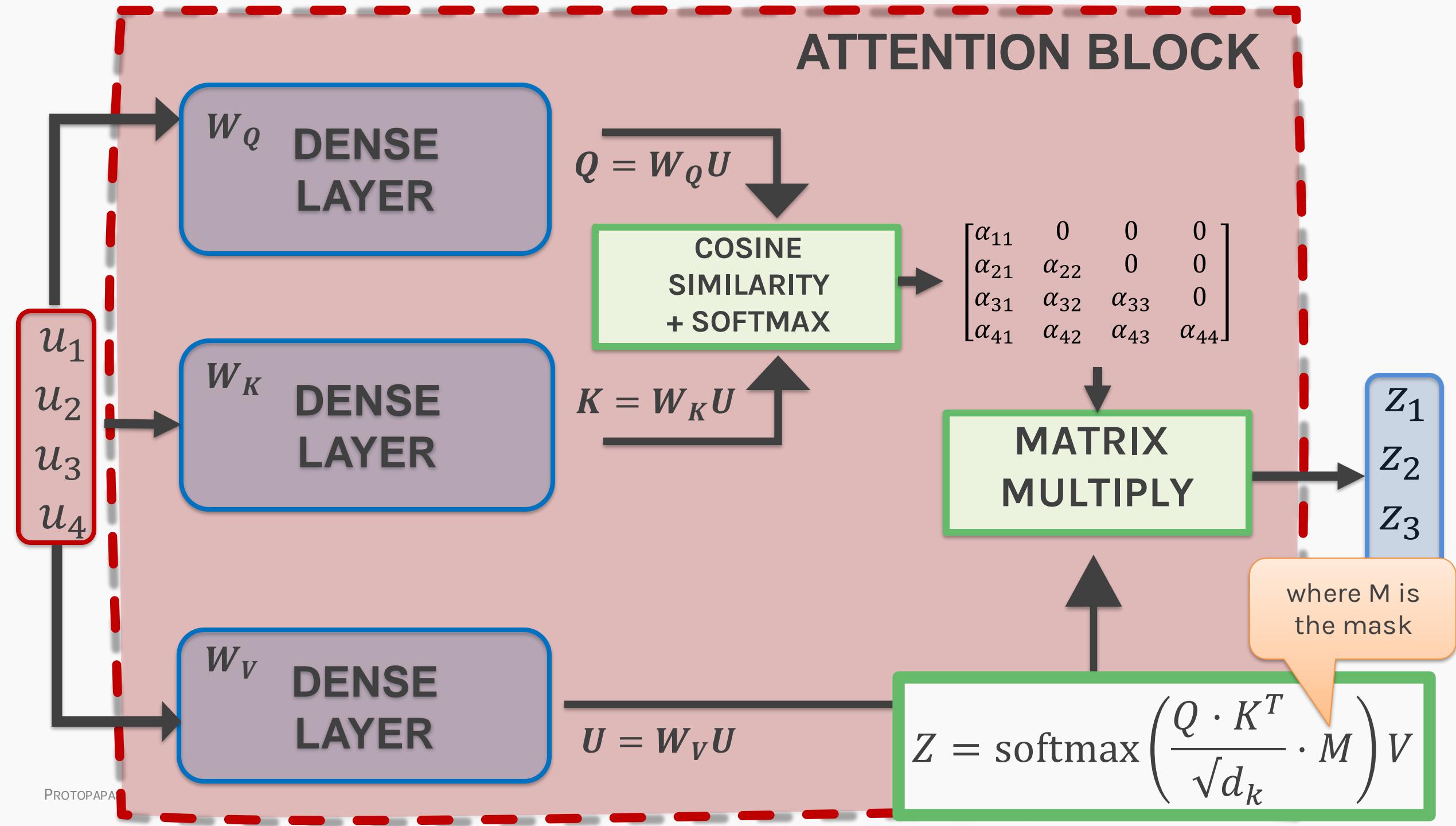
ATTENTION BLOCK



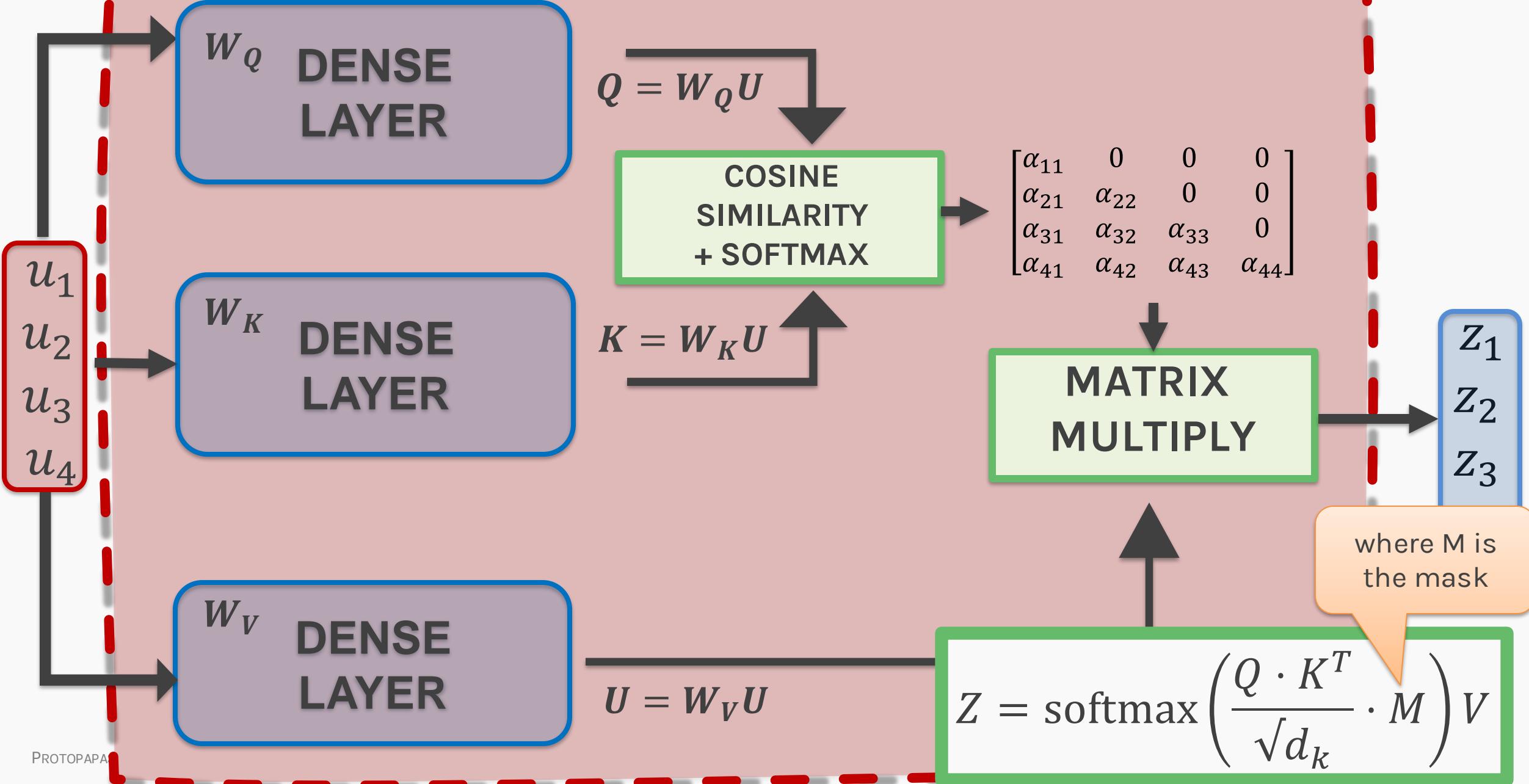
ATTENTION BLOCK



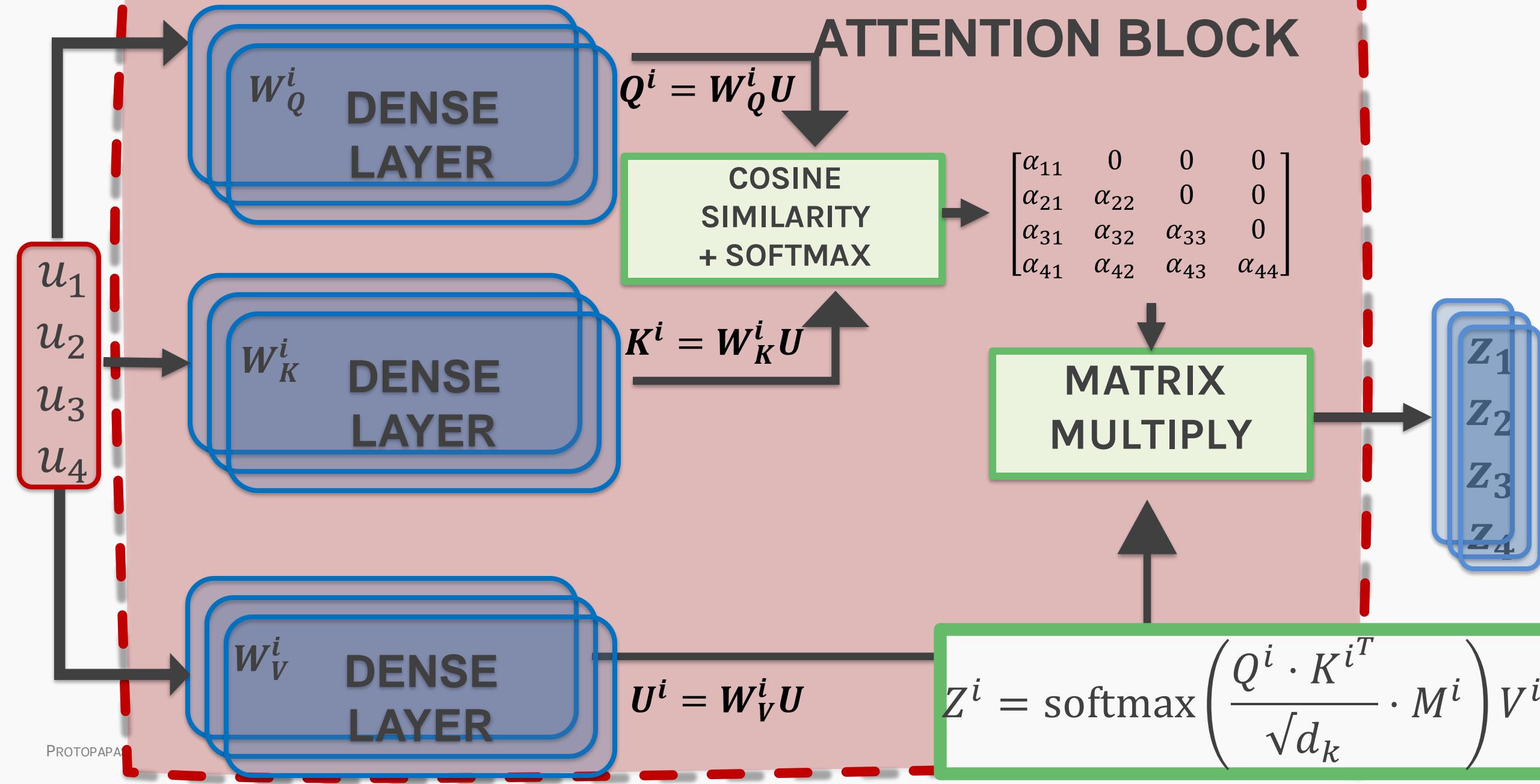
ATTENTION BLOCK



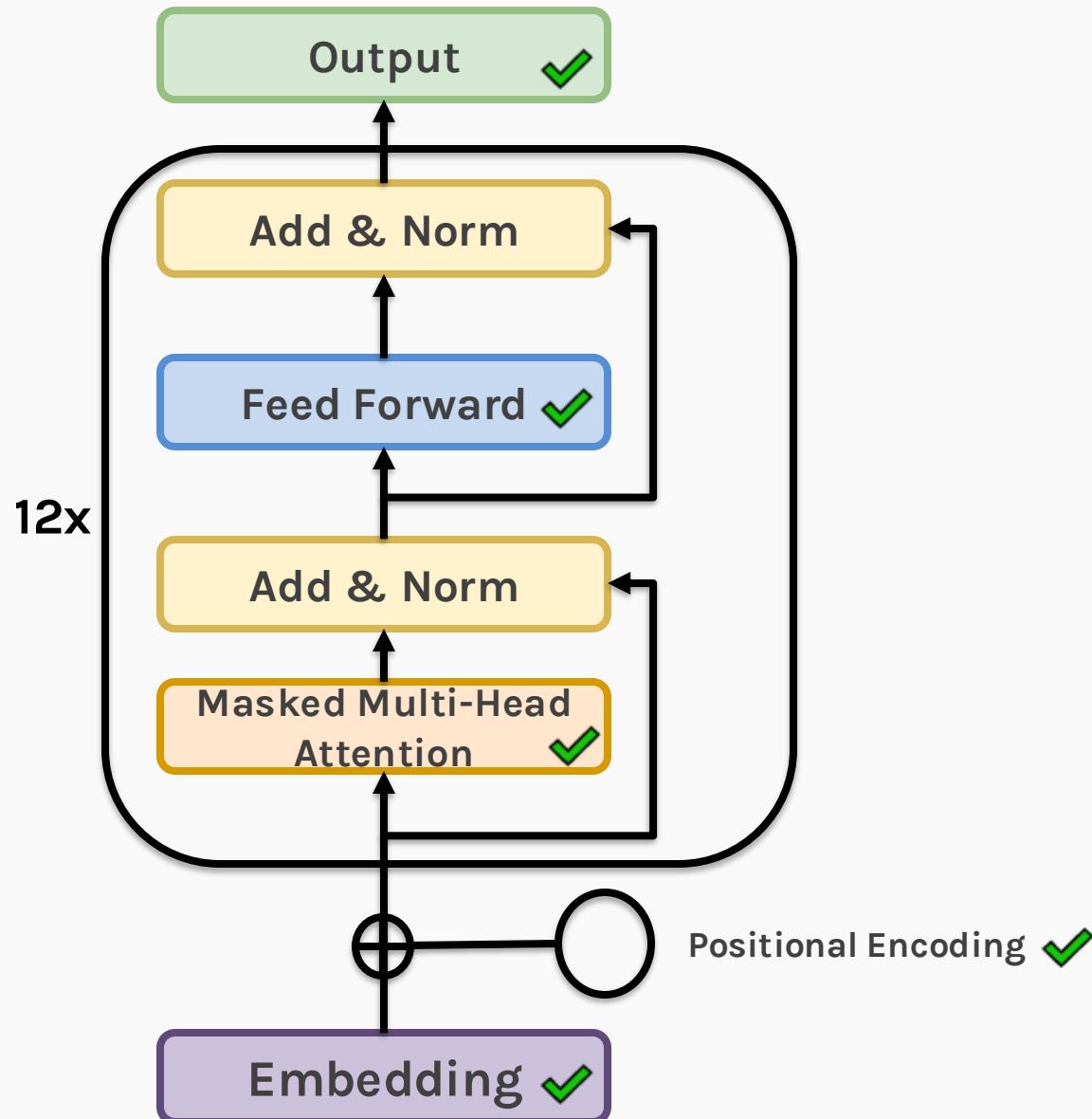
MASKED ATTENTION BLOCK



MASKED MULTI-HEAD ATTENTION BLOCK

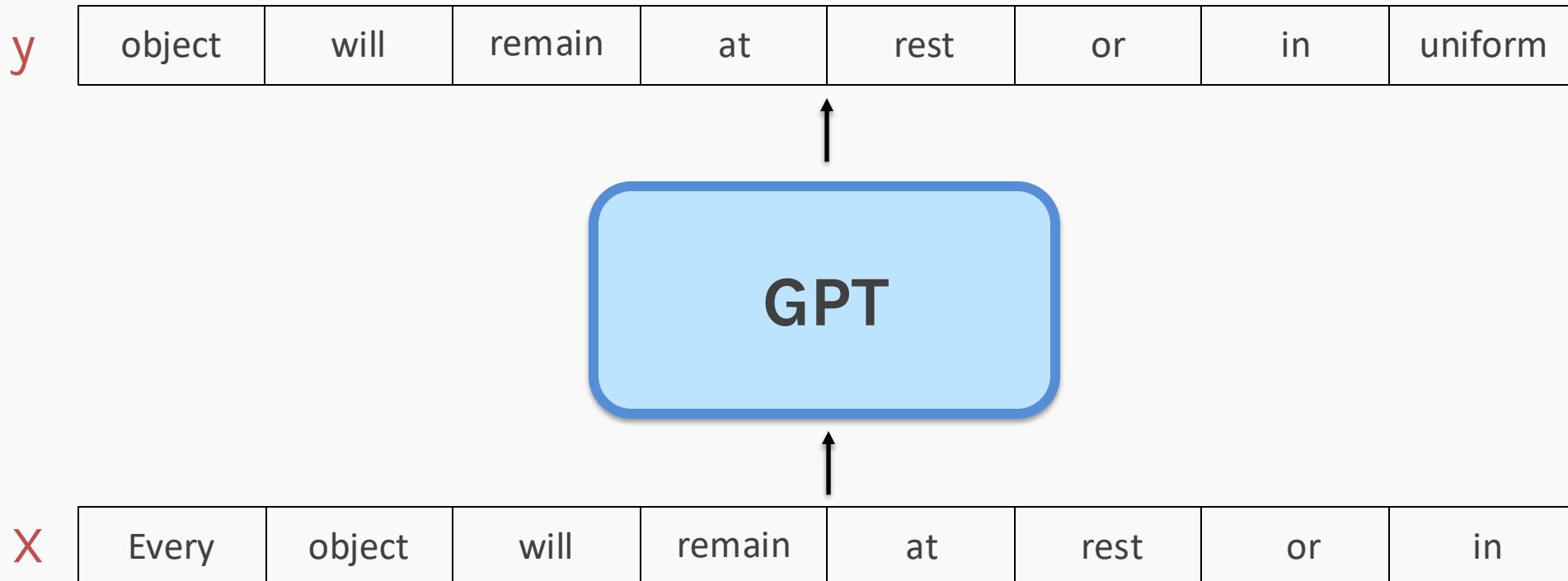


GPT – The Complete Picture



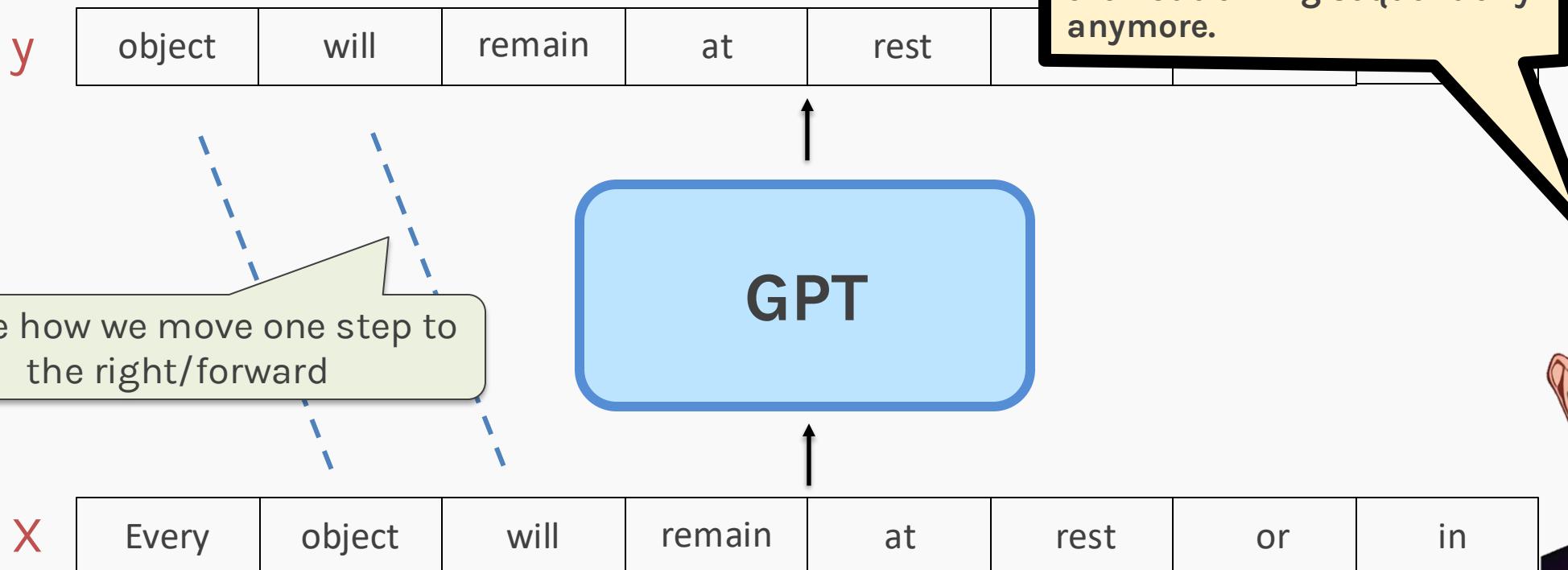
GPT - Training

- Now that we have figured out how to keep the model from extracting information from the future tokens, the model can be trained just like any other language model trained to do next word prediction.



GPT - Training

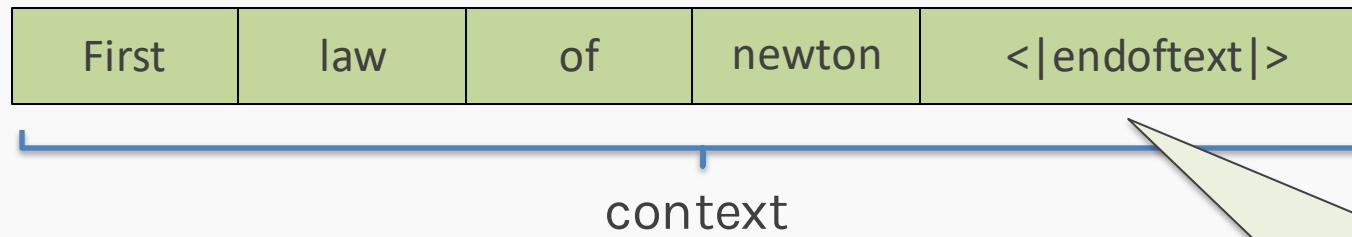
- Now that we have figured out how to keep the model from extracting information from the future tokens, the model can be trained just like any other language model trained to do next word prediction.



GPT - Inference

There are two ways GPT can generate text:

1. **Conditional**: We provide a starting prompt(context) and the model continues to generate text on that topic.



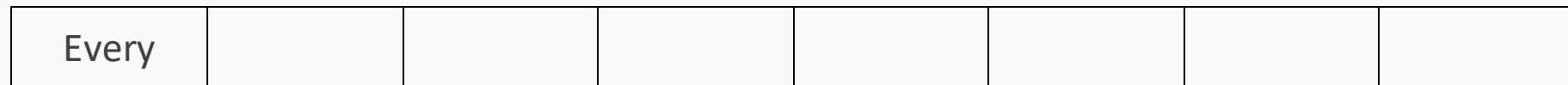
GPT uses a special implementation of the Conv1D layer in place of Dense layer that allows the model to input variable length sequences without the use of padding.

This is a special token used by the model to mark the end of the input or the starting point for predictions.

GPT - Inference

There are two ways GPT can generate text:

1. **Conditional**: We provide a starting prompt(context) and the model continues to generate text on that topic.



GPT - Inference

There are two ways GPT can generate text:

1. **Conditional**: We provide a starting prompt(context) and the model continues to generate text on that topic.

Every	object						
-------	--------	--	--	--	--	--	--



First	law	of	newton	< endoftext >	Every				
-------	-----	----	--------	---------------	-------	--	--	--	--

context

GPT - Inference

There are two ways GPT can generate text:

1. **Conditional**: We provide a starting prompt(context) and the model continues to generate text on that topic.

Every	object	will					
-------	--------	------	--	--	--	--	--



First	law	of	newton	< endoftext >	Every	object			
-------	-----	----	--------	---------------	-------	--------	--	--	--

context

GPT - Inference

There are two ways GPT can generate text:

1. **Conditional**: We provide a starting prompt(context) and the model continues to generate text on that topic.

Every	object	will	remain				
-------	--------	------	--------	--	--	--	--



First	law	of	newton	< endoftext >	Every	object	will		
-------	-----	----	--------	---------------	-------	--------	------	--	--

context

GPT - Inference

There are two ways GPT can generate text:

1. **Conditional:** We provide a starting prompt(context) and the model continues to generate text on that topic.

Every	object	will	remain	at			
-------	--------	------	--------	----	--	--	--



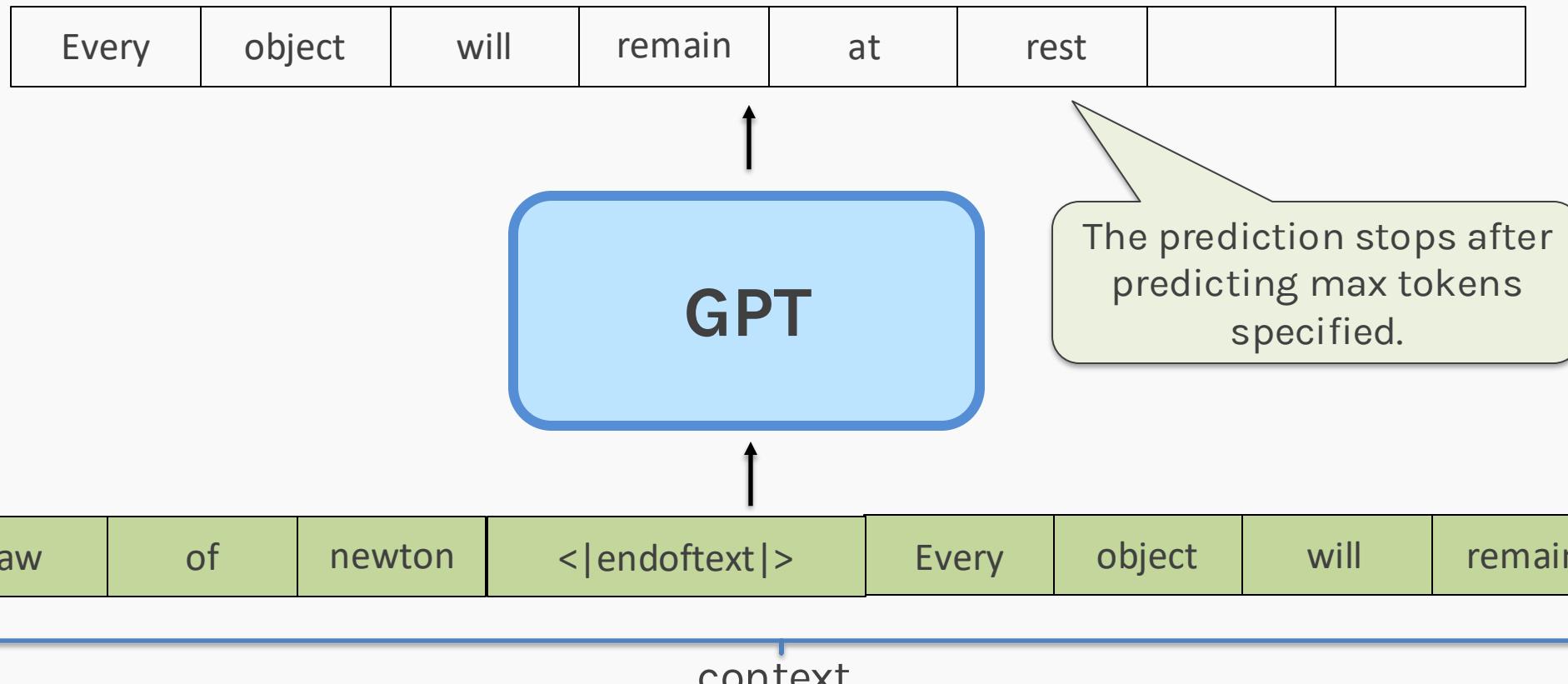
First	law	of	newton	< endoftext >	Every	object	will	remain	
-------	-----	----	--------	---------------	-------	--------	------	--------	--

context

GPT - Inference

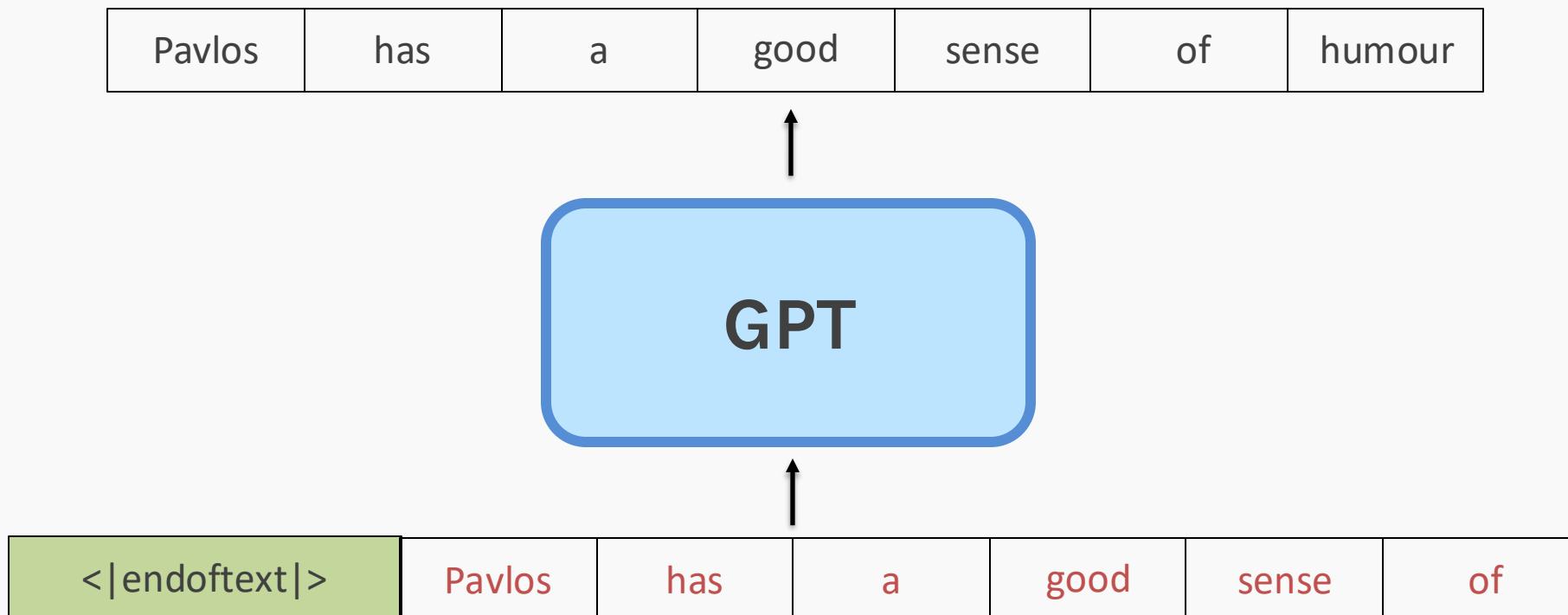
There are two ways GPT can generate text:

1. **Conditional**: We provide a starting prompt(context) and the model continues to generate text on that topic.



GPT - Inference

2. Unconditional: We start with `</endoftext|>` which acts as a special start token to have the model generate words on topic of its own choosing.

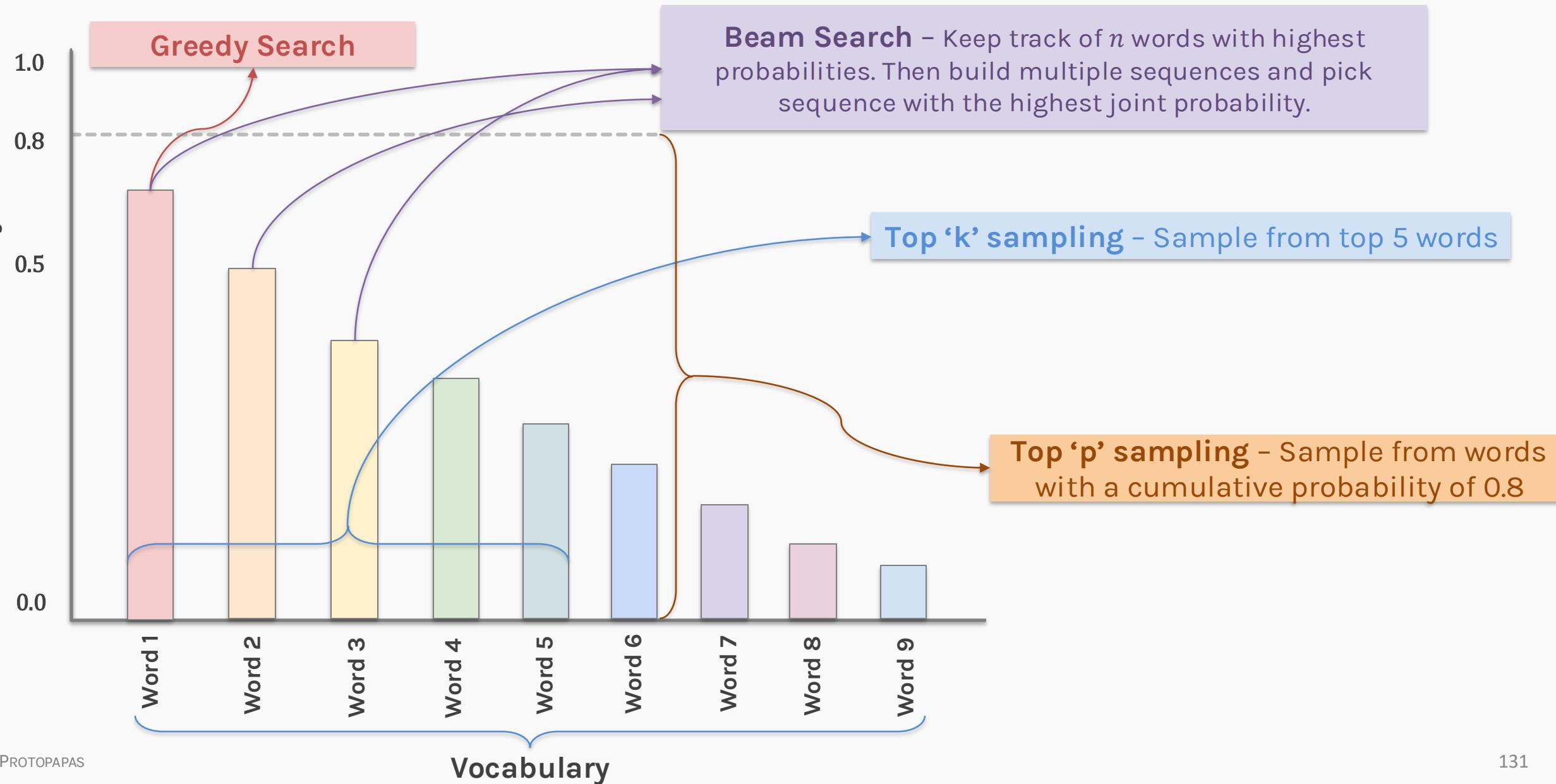


GPT - Inference

There are also different methods on how words are picked when generating text:

- Greedy Search
- Beam Search
- Top ‘k’ sampling
- Top ‘p’ sampling

GPT - Inference



Evolution of GPT

- Since its inception in 2018, GPT has received some major upgrades in terms of performance, thanks to the increasing number of parameters with each iteration.

	GPT - 1	GPT-2	GPT-3
Parameters	117 Million	1.5 Billion	175 Billion
Decoder Layers	12	48	96
Context Token Size	512	1024	2048
Hidden Layer	768	1600	12288
Batch Size	64	512	3.2M

Evolution of GPT

- Since its inception in 2018, GPT has received some major upgrades in terms of performance, thanks to the increasing number of parameters with each iteration.

	GPT - 1	GPT-2	GPT-3
Parameters	117 Million	1.5 B	175B
Decoder Layers	12	48	2048
Context Token Size	512	1024	
Hidden Layer	768	1600	12288
Batch Size	64	512	3.2M

Fun Fact: The current version of GPT (GPT-4) is speculated to have **1.7 Trillion** parameters!



Outline

- Transformers Recap
- BERT
- Practical problems in Large Language Models
- GPT
- **Hugging Face**

Hugging Face

Since the community is aware of difficulties in training language models from scratch, it has become standard to upload the trained weights and make them available to the community.

For the case of BERT, it is [340 million](#), or 1.4 GB assuming float32 format, which is much smaller than the dataset and can be stored in user-grade machines.

And can fit in most consumer-grade GPUs.

Hugging Face: What is it?

It is a [service](#) that enables you to download pre-trained models and datasets, [free of charge](#).

They also have a subscription model that enable you to train and serve NLP models. Aimed at labs or startups.

Hugging Face: Who use it?

More than 50,000 organizations are using Hugging Face



Allen Institute for AI
Non-Profit • 251 models



AI at Meta
Company • 1907 models



Amazon Web Services
Company • 15 models



Google
Company • 689 models



Intel
Company • 158 models



Microsoft
Company • 286 models



Grammarly
Company • 7 models



Writer *Enterprise*
Company • 8 models

Hugging Face: What can you find?

This platform has available almost all the state-of-the-art models in NLP.



Model Hub

Featured models

Browse the model hub to discover, experiment and contribute to new state of the art models.

Explore models

The screenshot shows the Hugging Face Model Hub interface. At the top, there's a navigation bar with tabs for Tasks, Libraries, Datasets, Languages, Licenses, and Other. Below this is a search bar labeled "Filter Tasks by name". The main content area is titled "Models 597,128" with a "Filter by name" button. The page is sorted by "Trending". The list of models includes:

- CohereForAI/c4ai-command-r-plus
- NexaAIDev/Octopus-v2
- mistral-community/Mixtral-8x22B-v0.1
- jetmoe/jetmoe-8b
- google/gemma-1.1-7b-bit
- ai21labs/Jamba-v0.1
- v2ray/Mixtral-8x22B-v0.1
- CohereForAI/c4ai-command-r-plus-4bit
- CohereForAI/c4ai-command-r-v0.1
- databricks/dbrx-instruct
- stabilityai/cosxl
- google/codegemma-7b-bit
- stabilityai/stablelm-2-12b

Hugging Face: What can you find?

You can explore the predictions using the hosted API.

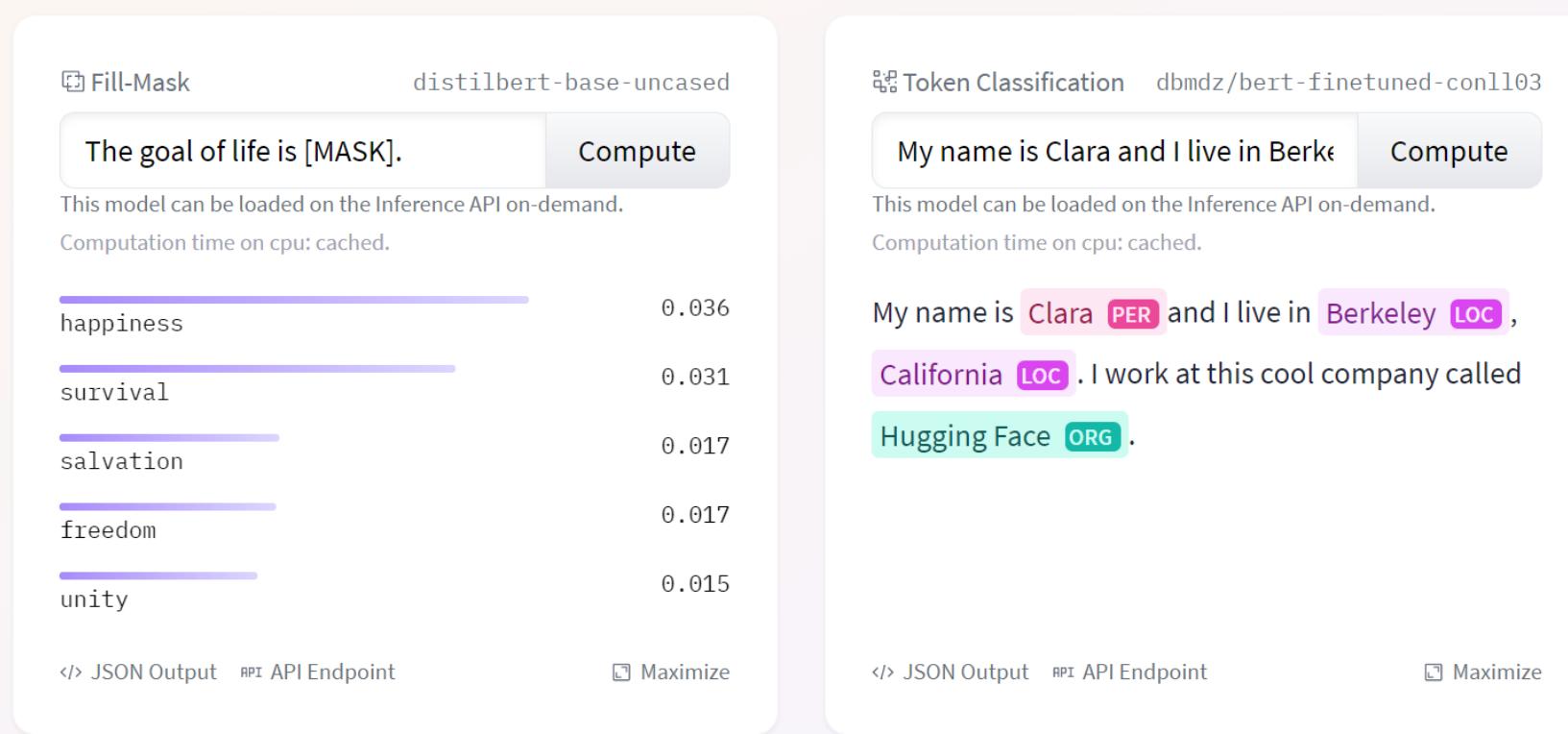
API

On demand

Inference API

Serve your models directly from Hugging Face infrastructure and run large scale NLP models in milliseconds with just a few lines of code.

[See pricing](#)



Hugging Face: What can you find?

You can also find datasets on HuggingFace that can be used to multiple downstream tasks.

The datasets can also be filtered on the basis of the task category and even subtask category.

The image shows two side-by-side search interfaces from the Hugging Face Datasets website. Both interfaces have a top navigation bar with tabs: Tasks, Sizes, Sub-tasks, Languages, Licenses, and Other. A search bar is located at the top of each interface.

Left Interface (Tasks View):

- Multimodal:** Visual Question Answering
- Computer Vision:** Depth Estimation, Image Classification, Object Detection, Image Segmentation, Text-to-Image, Image-to-Text, Image-to-Image, Image-to-Video, Unconditional Image Generation, Video Classification, Text-to-Video, Zero-Shot Image Classification, Mask Generation, Zero-Shot Object Detection, Text-to-3D, Image-to-3D, Image Feature Extraction
- Natural Language Processing:** Text Classification, Token Classification, Table Question Answering, Question Answering, Zero-Shot Classification, Translation, Summarization, Feature Extraction, Text Generation, Text2Text Generation, Fill-Mask, Sentence Similarity, Table to Text, Multiple Choice, Text Retrieval
- Audio:** Text-to-Speech, Text-to-Audio, Automatic Speech Recognition, Audio-to-Audio, Audio Classification, Voice Activity Detection
- Tabular:** Tabular Classification, Tabular Regression, Tabular to Text, Time Series Forecasting
- Reinforcement Learning:** Reinforcement Learning, Robotics
- Other:** Graph Machine Learning

Right Interface (Sub-tasks View):

- language-modeling** **multi-class-classification** **extractive-qa**
- named-entity-recognition** **natural-language-inference**
- open-domain-qa** **sentiment-classification**
- masked-language-modeling** **multiple-choice-qa**
- multi-label-classification** **document-retrieval**
- topic-classification** **closed-domain-qa** **text-scoring**
- closed-book-qa** **open-book-qa**
- tabular-multi-class-classification** **tabular-multi-label-classification**
- semantic-similarity-scoring** **entity-linking-retrieval**
- fact-checking-retrieval** **parsing** **multi-class-image-classification**
- semantic-similarity-classification** **fact-checking** **part-of-speech**
- image-captioning** **news-articles-summarization**
- intent-classification** **hate-speech-detection** **dialogue-modeling**
- multi-input-text-classification** **text-simplification**
- sentiment-analysis** **dialogue-generation** **sentiment-scoring**
- abstractive-qa** **acceptability-classification**
- speaker-identification** **instance-segmentation**
- semantic-segmentation** **multi-label-image-classification**
- visual-question-answering** **slot-filling** **explanation-generation**
- coreference-resolution** **lemmatization**
- entity-linking-classification** **word-sense-disambiguation**
- keyword-spotting** **rdf-to-text**
- multivariate-time-series-forecasting** **open-domain-abstractive-qa**
- news-articles-headline-generation** **audio-emotion-recognition**
- univariate-time-series-forecasting** **face-detection**
- utterance-retrieval** **multiple-choice-coreference-resolution**
- document-question-answering** **conversational**
- tabular-single-column-regression** **audio-intent-classification**
- audio-language-identification** **panoptic-segmentation**
- language-identification** **task-planning**

Hugging Face: What can you find?

It also provides licenses info, size and the metadata of the dataset.

There are roughly 133,136 such datasets available across 110+ languages!

The screenshot shows the Hugging Face dataset search interface with three main filter sections: Languages, Licenses, and Sizes.

Languages: A list of 110+ languages, each represented by a small icon and text. Some examples include English, Chinese, French, Spanish, German, Russian, Japanese, Portuguese, Korean, Arabic, Italian, Vietnamese, Hindi, Turkish, Polish, Indonesian, Dutch, Thai, Bengali, Swedish, Czech, Catalan, Persian, Finnish, Tamil, Danish, Romanian, Ukrainian, Hungarian, Greek, Telugu, Slovenian, Bulgarian, Malay, Urdu, Malayalam, Hebrew, Marathi, Slovak, Estonian, Basque, Norwegian, Croatian, Kannada, Gujarati, Serbian, Swahili, code, Lithuanian, Latvian, Indonesian, Panjabi, Icelandic, Maltese, Sinhala, Afrikaans, Tagalog, Nepali, Oriya, Galician, Albanian, Yoruba, Kazakh, Irish, Azerbaijani, Amharic, Armenian, Assamese, Welsh, Esperanto, Burmese, Georgian, Norwegian Bokmål, Belarusian, Macedonian, Khmer, Hausa, Uzbek, Igbo, Kyrgyz, Mongolian, Latin, Cebuano, Kinyarwanda, Somali, Javanese, Norwegian Nynorsk, Pashto, Breton, Ganda, Xhosa, English, Tatar, Zulu, Lao, Yue Chinese, Uyghur, Bosnian, Sanskrit, Tajik, Kurdish, Minangkabau, Sindhi, Asturian, Central Kurdish, Luxembourgish, Scottish Gaelic, Western Frisian, Pedi, Wolof, Yiddish, Occitan, Guarani, Twi, Sundanese, Haitian, Turkmen, Shona, Bashkir, Divehi, Tibetan, Maithili, Faroese, Achinese, Bambara, Tswana, Kabyle, Buginese, Chichewa, Yakut, Chuva, Lingala, Māori, Malagasy, Interlingua, Tigrinya, Iloko, Egyptian Arabic, Banjar, Southern Sotho, Waray (Philippines), Upper Sorbian, Javanese, Eastern Mari, Afar, Quechua, Santali, Balinese, Tsonga.

Licenses: A list of various licenses, each with a small icon and text. Some examples include mit, apache-2.0, openrail, cc-by-4.0, other, cc-by-sa-4.0, cc-by-nc-4.0, cc-by-nc-sa-4.0, cc0-1.0, cc, afl-3.0, cc-by-nc-nd-4.0, cc-by-sa-3.0, gpl-3.0, creativeml-openrail-m, llama2, odc-by, wtpl, cc-by-3.0, bigscience-openrail-m, unlicense, artistic-2.0, cc-by-2.0, bsd, agpl-3.0, gpl, odbl, gfdl, bsd-3-clause, cc-by-nc-sa-3.0, c-uda, cc-by-nd-4.0, pddl, openrail++, cc-by-nc-2.0, cc-by-nc-3.0, gpl-2.0, bigscience-bloom-rail-1.0, bigcode-openrail-m, cdla-sharing-1.0, bsd-2-clause, leol-3.0.

Sizes: A list of size ranges, each with a small icon and text. Some examples include n<1K, 1K<n<10K, 10K<n<100K, 100K<n<1M, 1M<n<10M, 10M<n<100M, 100M<n<1B, 1B<n<10B, 10B<n<100B, 100B<n<1T, n>1T.

Hugging Face: How to Use?

HuggingFace has an available Python API that gives access to datasets, tokenizers, and models. The models are available for PyTorch and Tensorflow.



Open Source

Transformers

Transformers is our natural language processing library and our hub is now open to all ML models, with support from libraries like [Flair](#), [Asteroid](#), [ESPnet](#), [Pyannote](#), and more to come.

[Check documentation](#)



huggingface@transformers:~

```
from transformers import AutoTokenizer, AutoModelForMaskedLM  
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")  
model = AutoModelForMaskedLM.from_pretrained("bert-base-uncased")
```

Thank you!