جامعة الامير مقرن بن عبد العزيز
University of Prince Mugrin

Department of Artificial Intelligence

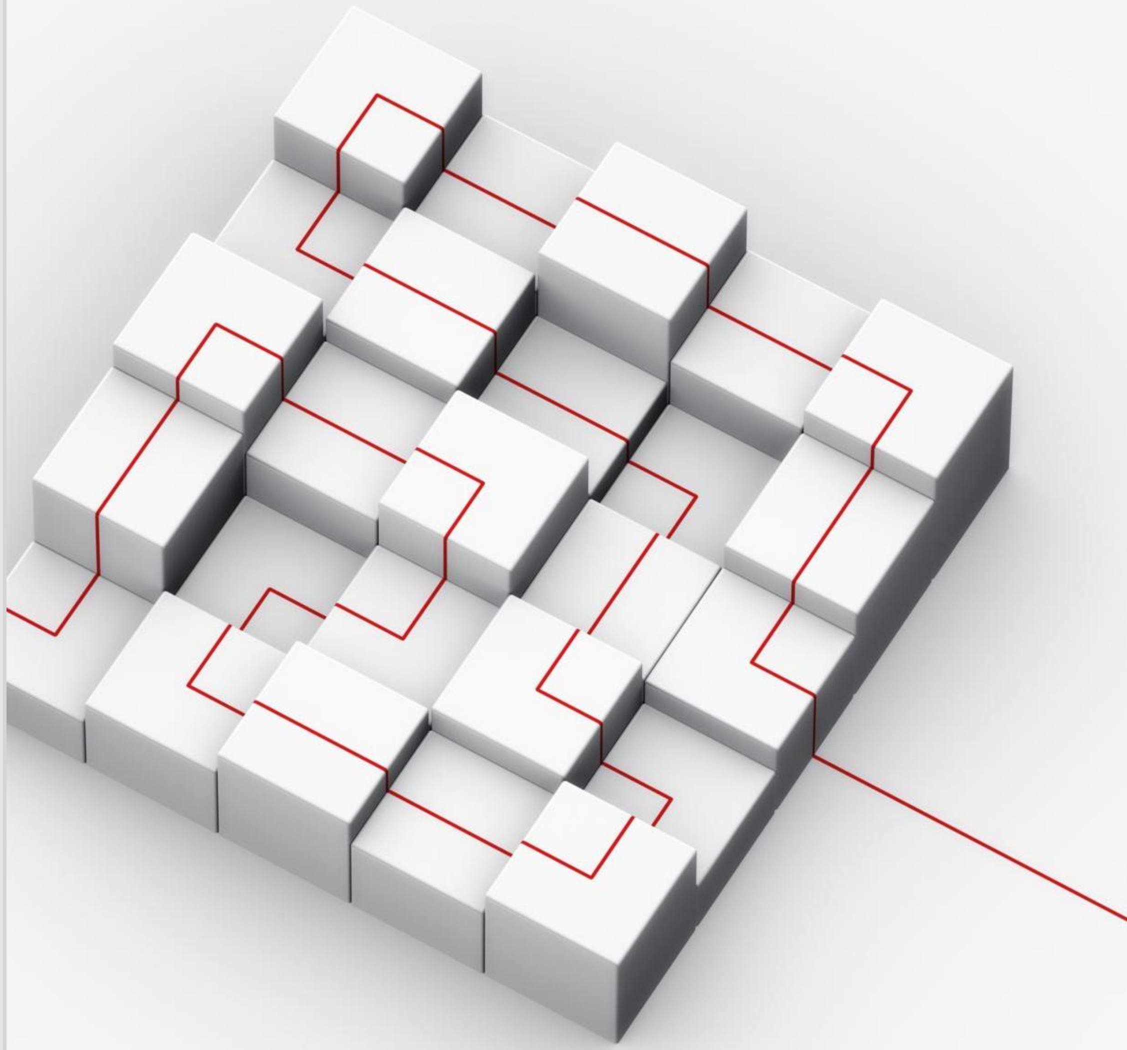
College of Computer and Cyber Sciences

Introduction to Deep Learning

Transformer-Based Language Model with PyTorch

1. Learning Objectives

- Understand the fundamental concepts of **Transformers**
- Understand the architecture and components of a **Transformer-based language** model.
- Load and preprocess a real-world text dataset (WikiText-2) using **Hugging Face**.
- Implement a Transformer **encoder** for next-word prediction using PyTorch.
- **Train** the model on sequential word data.
- Use the trained model to **generate top predictions** for next-word inference.



2. Explanation of Key Concepts

Hugging Face is a leading company and open-source platform in the field of **ML**. It provides tools, models, and libraries that make it easier for developers and researchers to build, deploy, and share machine learning applications.

Why is it Important?

- **Ease of Use**: Hugging Face simplifies complex ML workflows, especially for those working with transformers.
- **State-of-the-Art Models**: Hugging Face provides access to the latest breakthroughs in AI, helping users leverage cutting-edge models without needing extensive computational resources. It also give user access to lead multiple types of datasets.

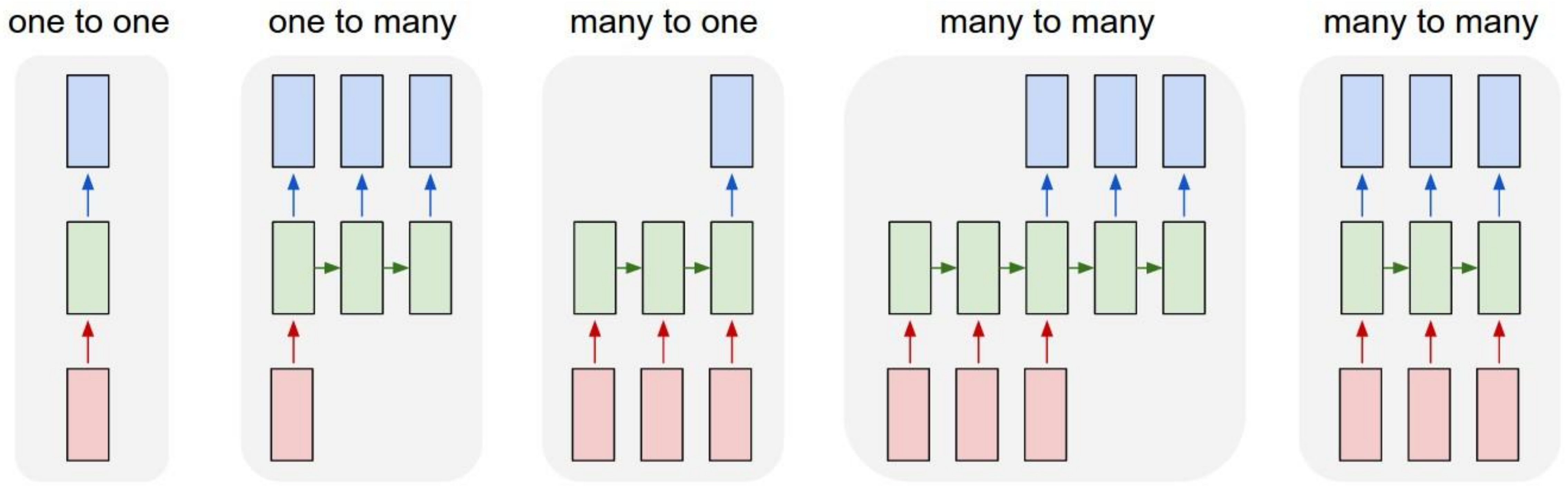
2. Explanation of Key Concepts

What is WikiText-2 dataset?

- A cleaned subset of Wikipedia, designed for training language models.
- It keeps punctuation and case.
- It’s often used in NLP for **language modeling** tasks.

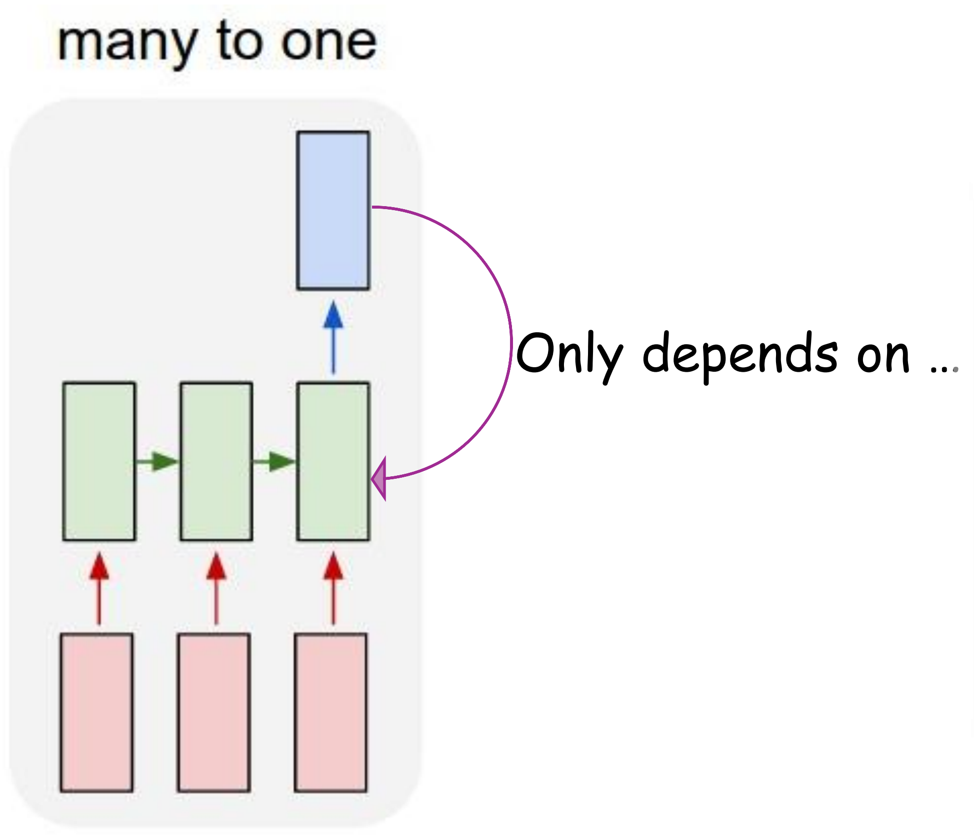
Transformers

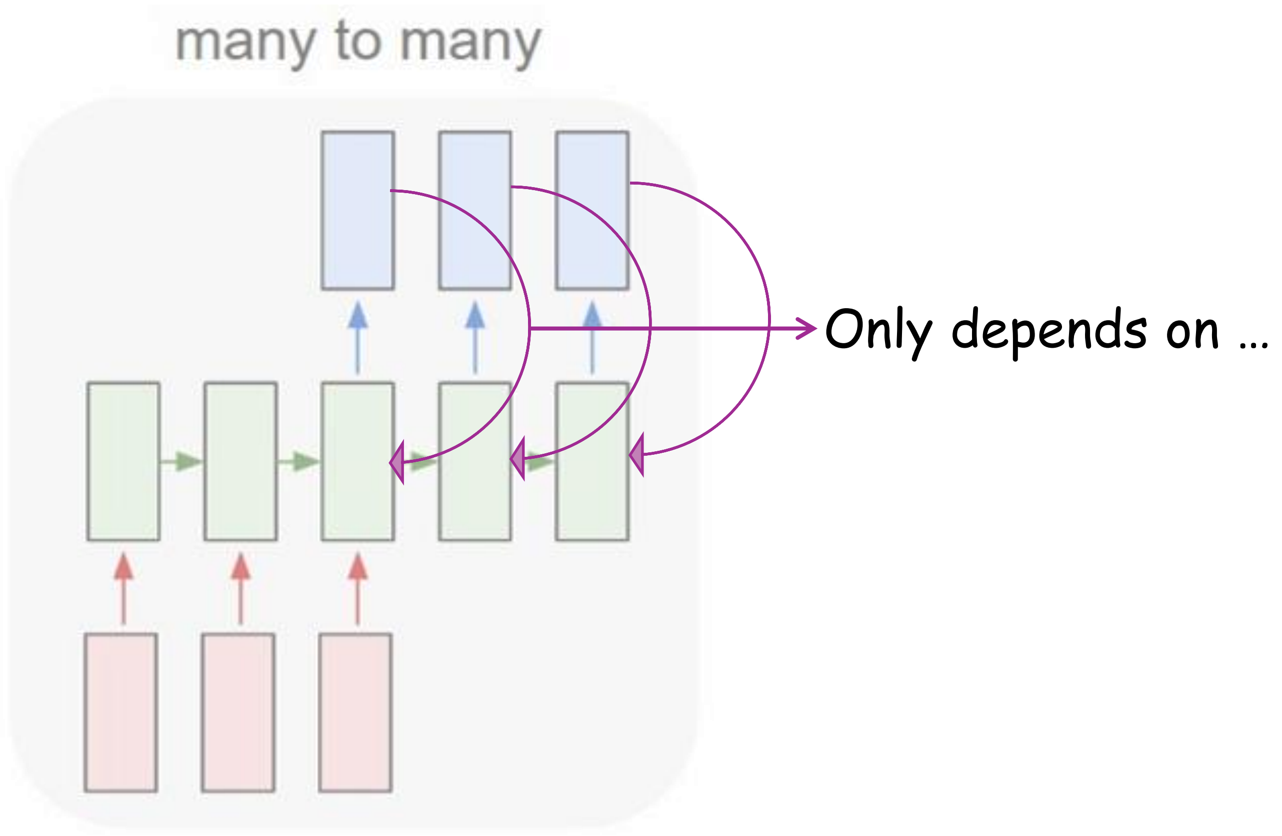
Types of Sequence Modeling Architectures



First process the input and generate a hidden representation for it
Then use it to generate an output

Modelling the problem

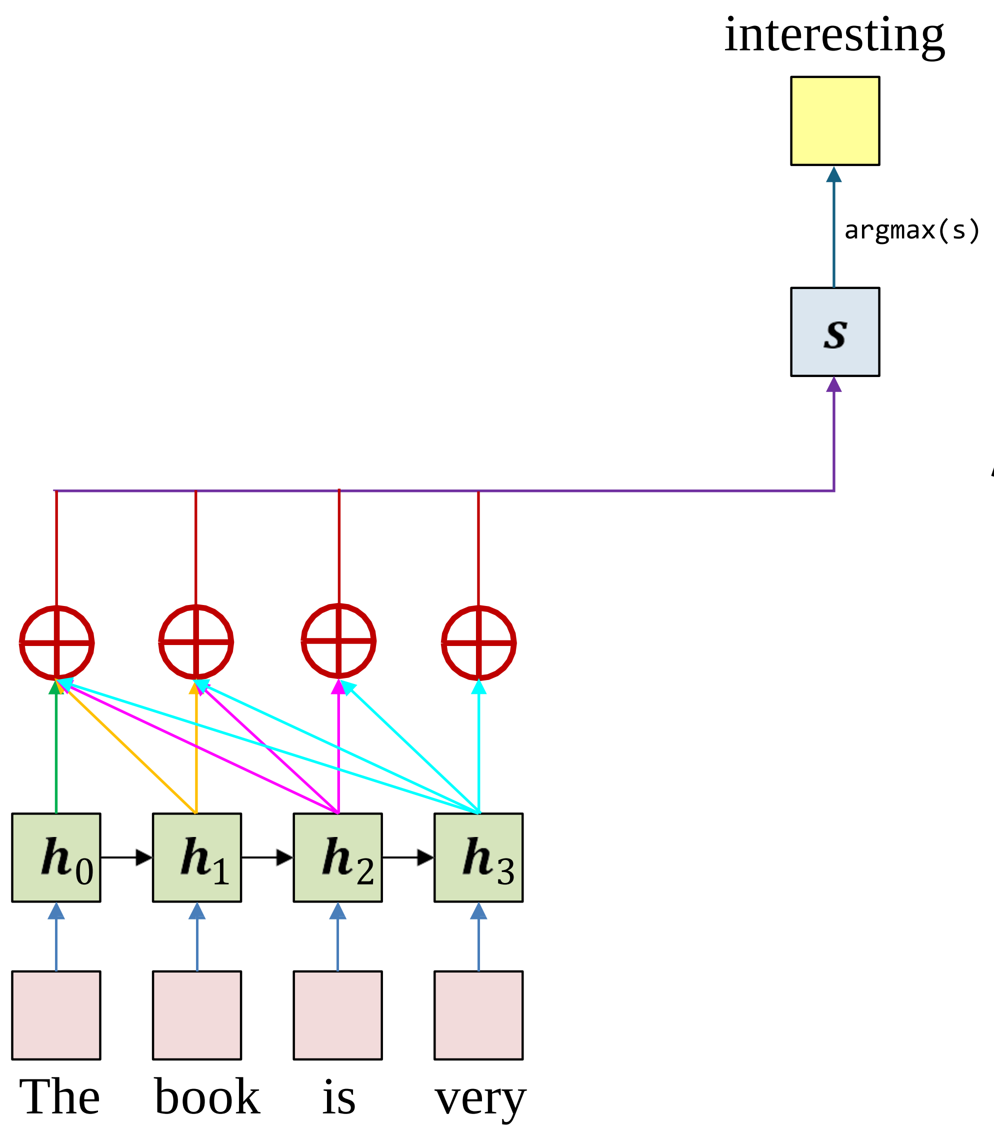




Problem: Each word that is output depends only on current hidden state, and not on previous outputs

Problem effect: **limits the model's understanding of context**

Transformers to the Rescue



$$\boldsymbol{s} = \begin{bmatrix} \text{interesting} & 0.52 \\ \text{great} & 0.23 \\ \text{boring} & 0.10 \\ \text{difficult} & 0.09 \\ . & . \\ . & . \\ . & . \end{bmatrix}$$

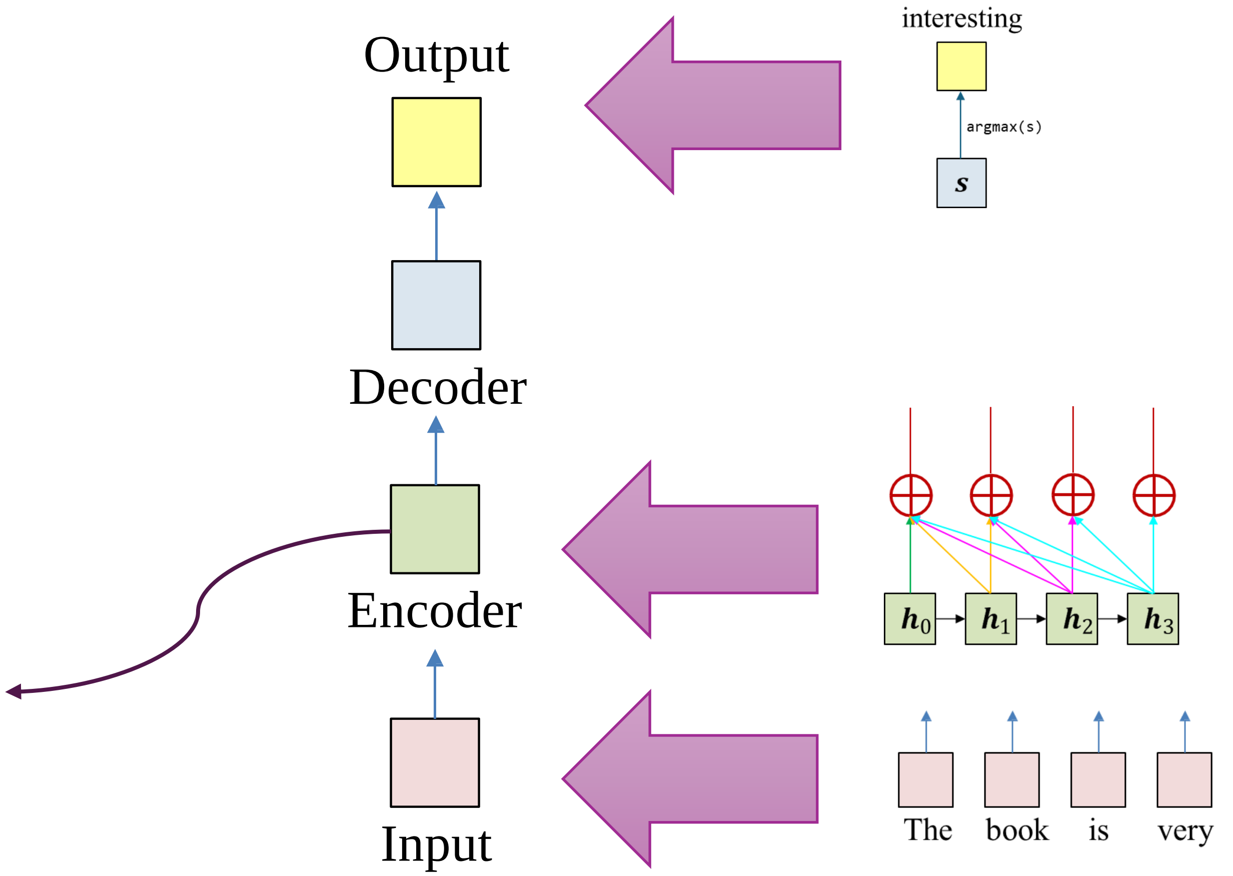
Self-attention

Self-attention = Full context access

- **Every token** attends to **all previous tokens**
- The **output representation** is built from a **weighted combination of all previous hidden states**, not just the current one.

“Attention Is All You Need”

Where Does This Fit in the Bigger Picture?



What is the Encoder?

The **encoder** is the part of the Transformer that processes the input sequence.

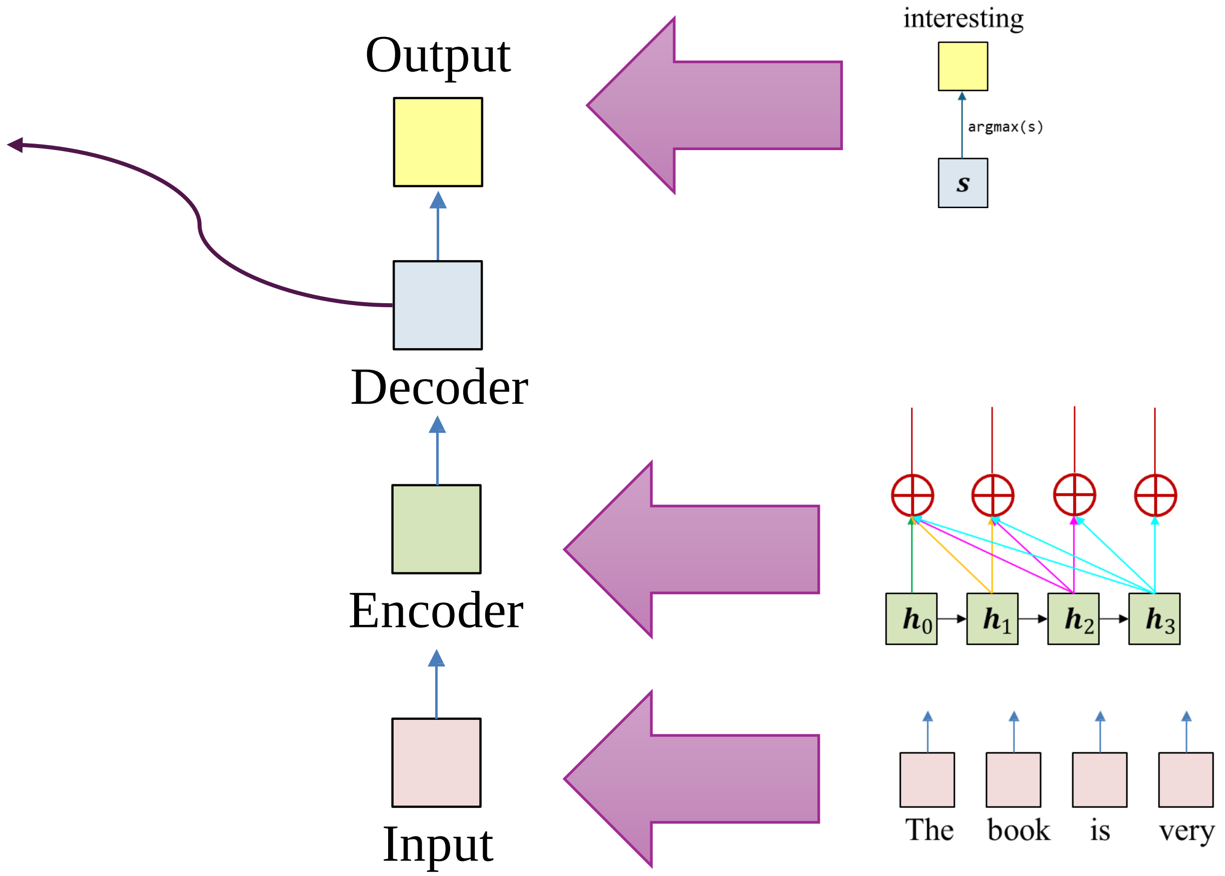
It reads all the input tokens (like "The book is very") and transforms them into hidden representations that capture meaning and context.

Where Does This Fit in the Bigger Picture?

What is the Decoder?

The **decoder** is used when we want to generate a full output sequence, like in translation:
"The book is difficult“ → “الكتاب صعب”

It takes the encoder’s output and step by step generates the target sequence.

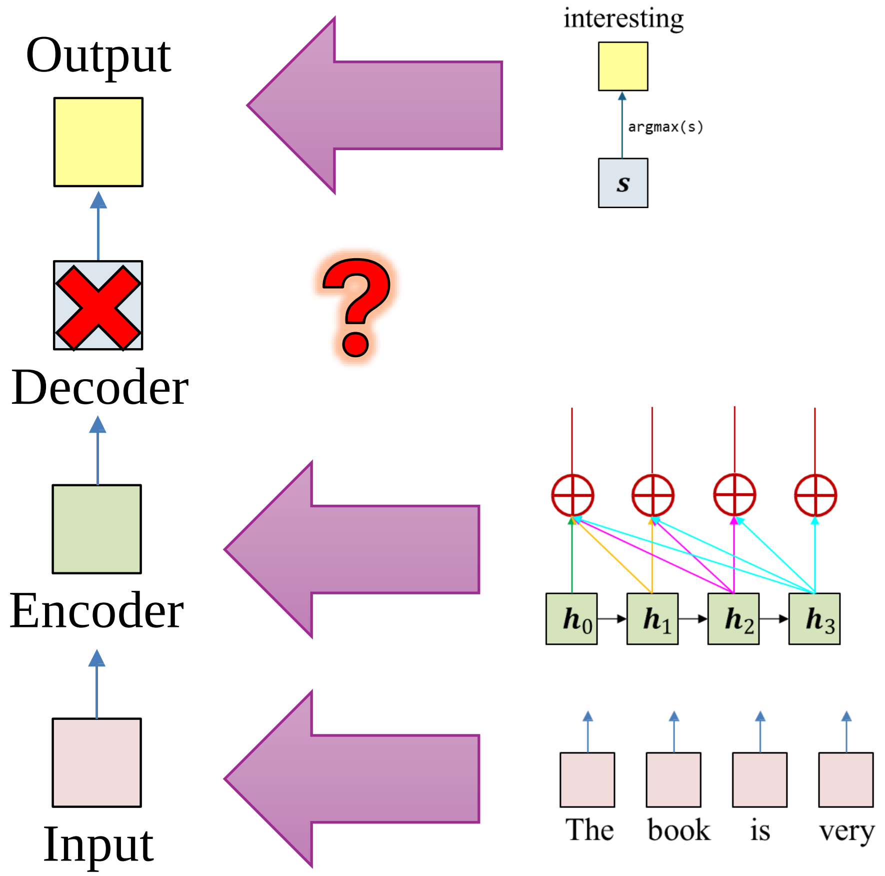


Why Don't We Use the Decoder Here?

In this example, we are only **predicting the next word** ("interesting") based on a given prompt.

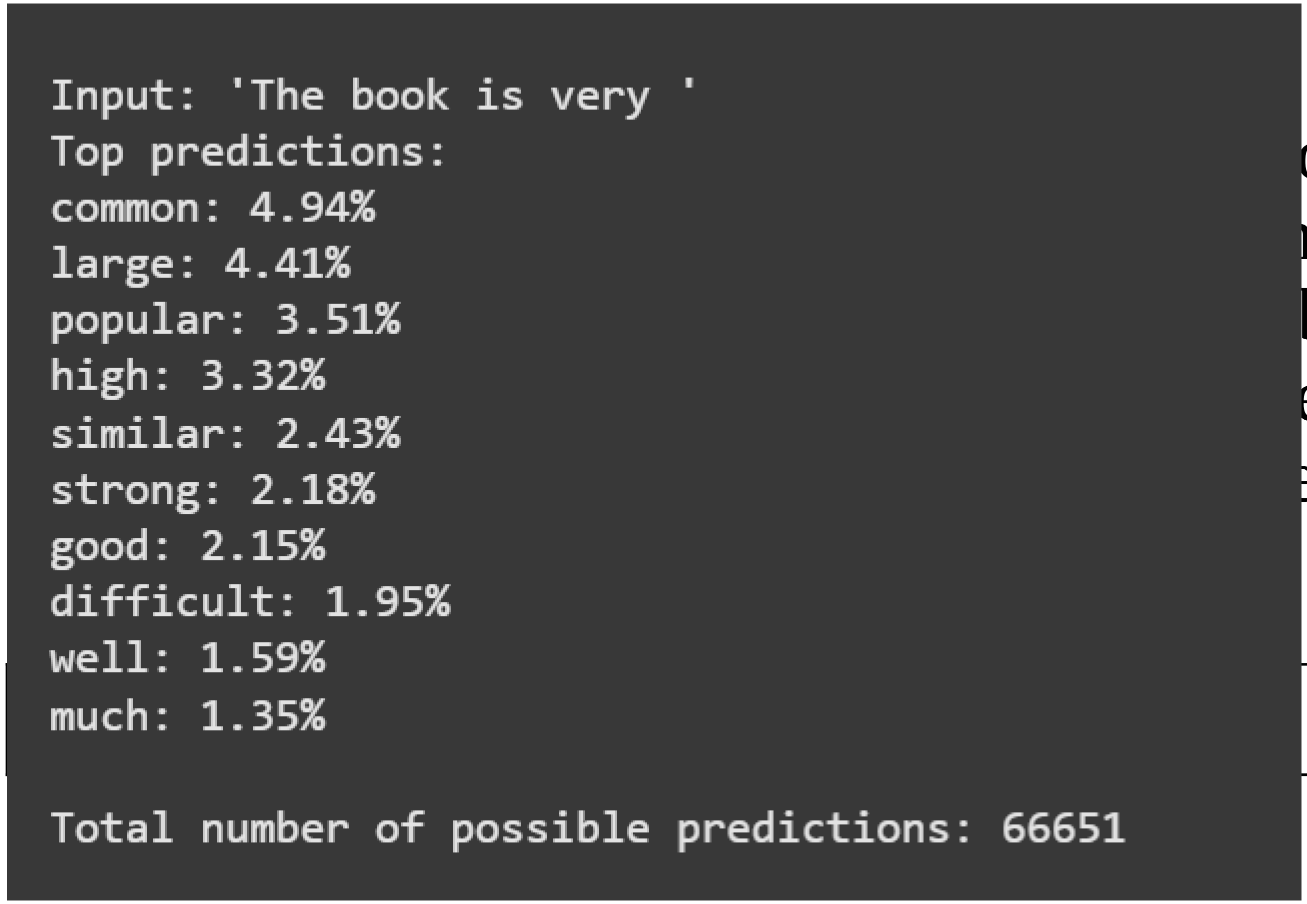
This is a **many-to-one task**, and we only need the encoder part.

So, the decoder is **not needed** in this type of task.



3. Activities

➢ **Exercise 1: Transformer-Based Language Model with PyTorch**



```
Input: 'The book is very '
Top predictions:
common: 4.94%
large: 4.41%
popular: 3.51%
high: 3.32%
similar: 2.43%
strong: 2.18%
good: 2.15%
difficult: 1.95%
well: 1.59%
much: 1.35%

Total number of possible predictions: 66651
```

ormer-based language model
a sequence. We will use the
basic preprocessing steps. We
encoder from scratch to learn
ext word given a sequence of

4. Tasks

- **Task 1**

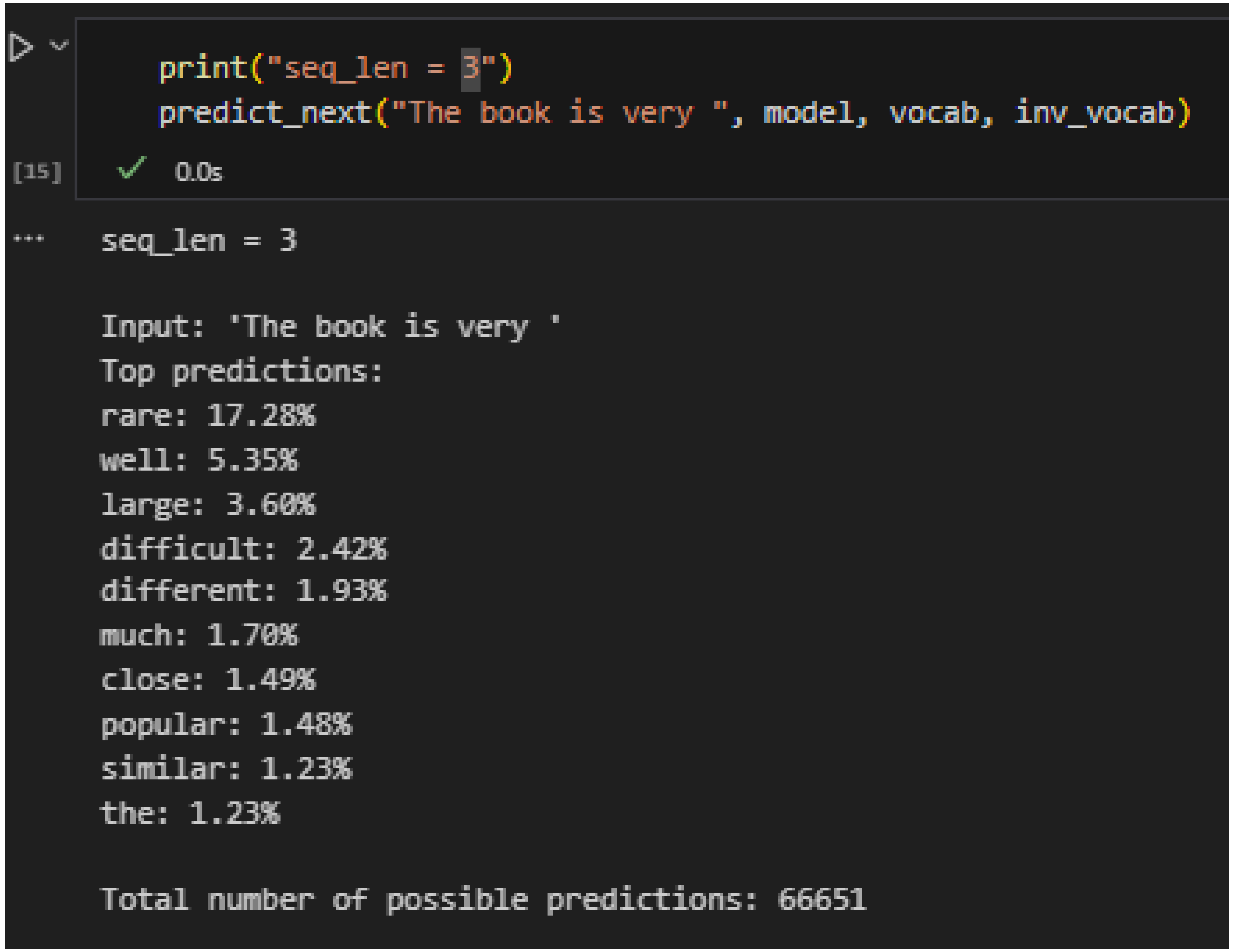
In this task, you will change the sequence length (seq_len) from 5 to another value (e.g., 3 or 7), re-run the training, and then use the predict_next function to compare how the model's predictions change.

Write your thoughts…

5. References

- [1706.03762] Attention Is All You Need
- LSTM RNN in Keras: Examples of One-to-Many, Many-to-One & Many-to-Many | dl-question-bank – Weights & Biases
- The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.
- The Annotated Transformer
- Tokenizers
- NLP Preprocessing Steps in Easy Way - Analytics Vidhya
- TransformerEncoder — PyTorch 2.6 documentation
- Salesforce/wikitext · Datasets at Hugging Face
- Welcome to PyTorch Tutorials — PyTorch Tutorials 2.6.0+cu124 documentation

When the sq=3



```
print("seq_len = 3")
predict_next("The book is very ", model, vocab, inv_vocab)
```

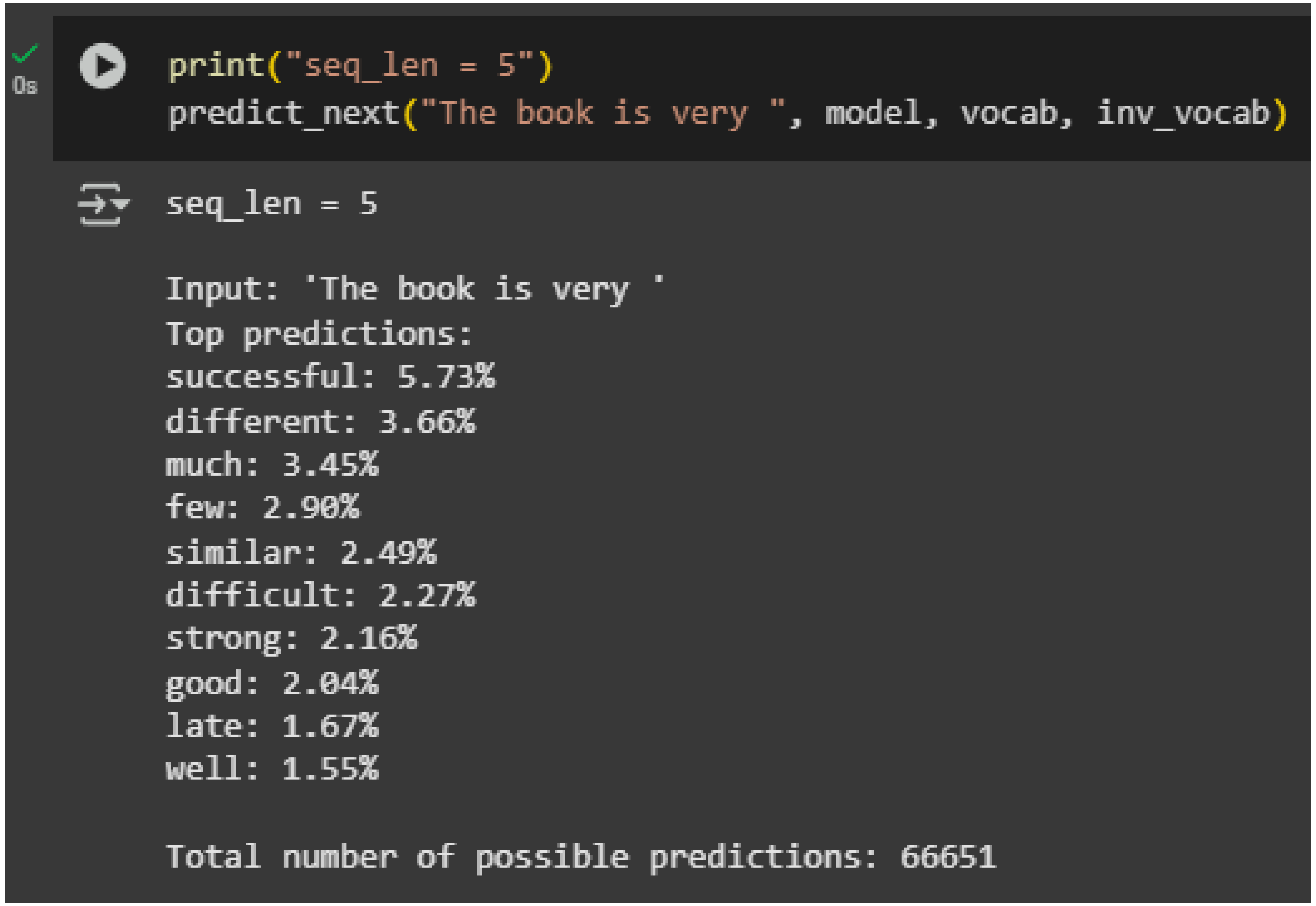
[15] ✓ 0.0s

```
seq_len = 3

Input: 'The book is very '
Top predictions:
rare: 17.28%
well: 5.35%
large: 3.60%
difficult: 2.42%
different: 1.93%
much: 1.70%
close: 1.49%
popular: 1.48%
similar: 1.23%
the: 1.23%

Total number of possible predictions: 66651
```

When the sq=5



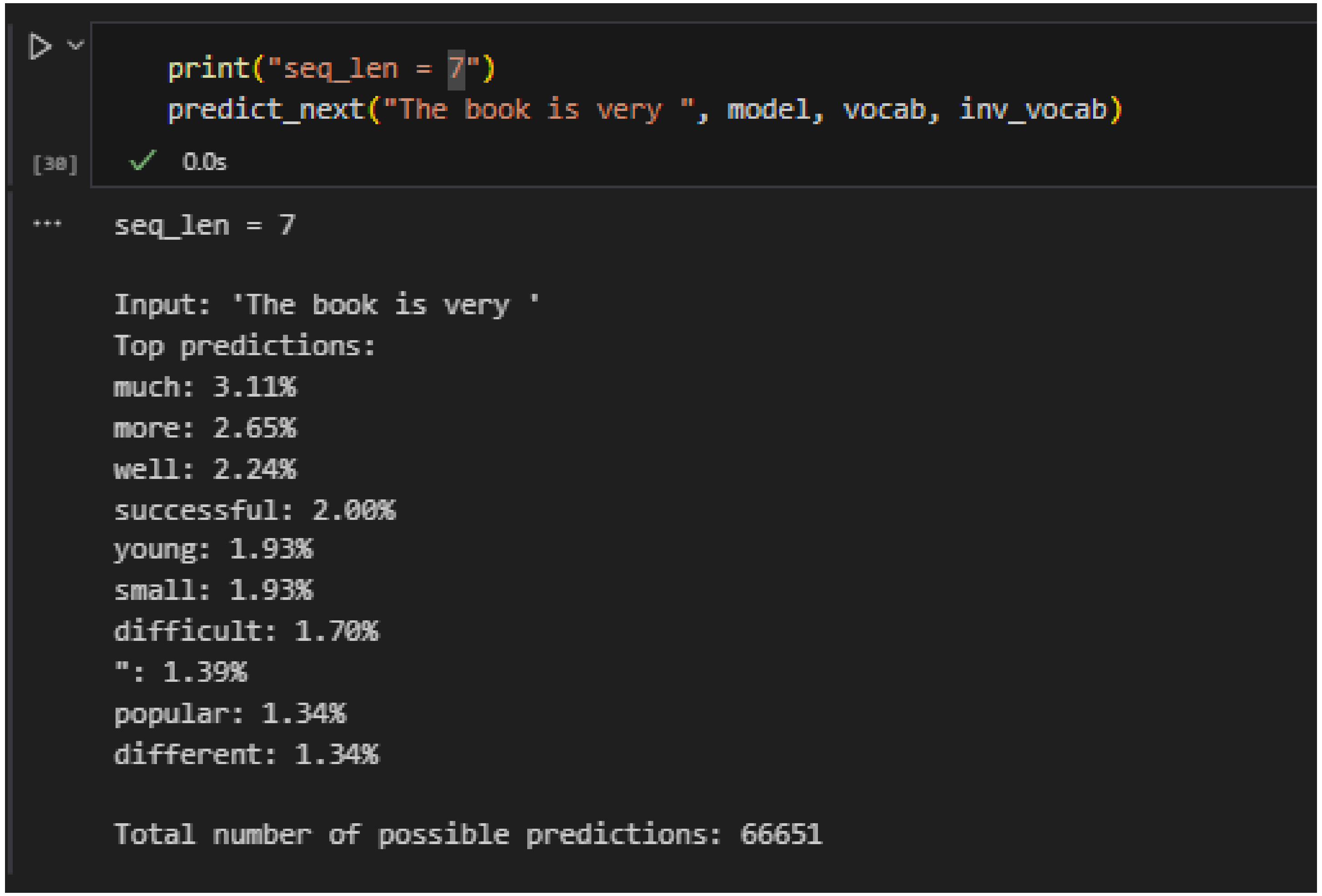
```
print("seq_len = 5")
predict_next("The book is very ", model, vocab, inv_vocab)
```

```
seq_len = 5

Input: 'The book is very '
Top predictions:
successful: 5.73%
different: 3.66%
much: 3.45%
few: 2.90%
similar: 2.49%
difficult: 2.27%
strong: 2.16%
good: 2.04%
late: 1.67%
well: 1.55%

Total number of possible predictions: 66651
```

When the sq=7



```
print("seq_len = 7")
predict_next("The book is very ", model, vocab, inv_vocab)
```

[30] ✓ 0.0s

```
seq_len = 7

Input: 'The book is very '
Top predictions:
much: 3.11%
more: 2.65%
well: 2.24%
successful: 2.00%
young: 1.93%
small: 1.93%
difficult: 1.70%
": 1.39%
popular: 1.34%
different: 1.34%

Total number of possible predictions: 66651
```

Thoughts

- I think at least half of the predictions in all sequence lengths were suitable.
- The loss is decreasing during training
- Although some of the predictions aren't suitable, it may be due to the low number of epochs (10), increasing it to 50 or 100 might give a much better performance model.
- Comparing the word 'difficult':

Seq_length = 3:	2.42% likelihood
Seq_length = 5:	2.27% likelihood
Seq_length = 7:	1.70% likelihood