**Pre-trained models for text generation**

Pre-trained models can also be used for text generation.

**2. Why pre-trained models?**

00:04 - 00:30

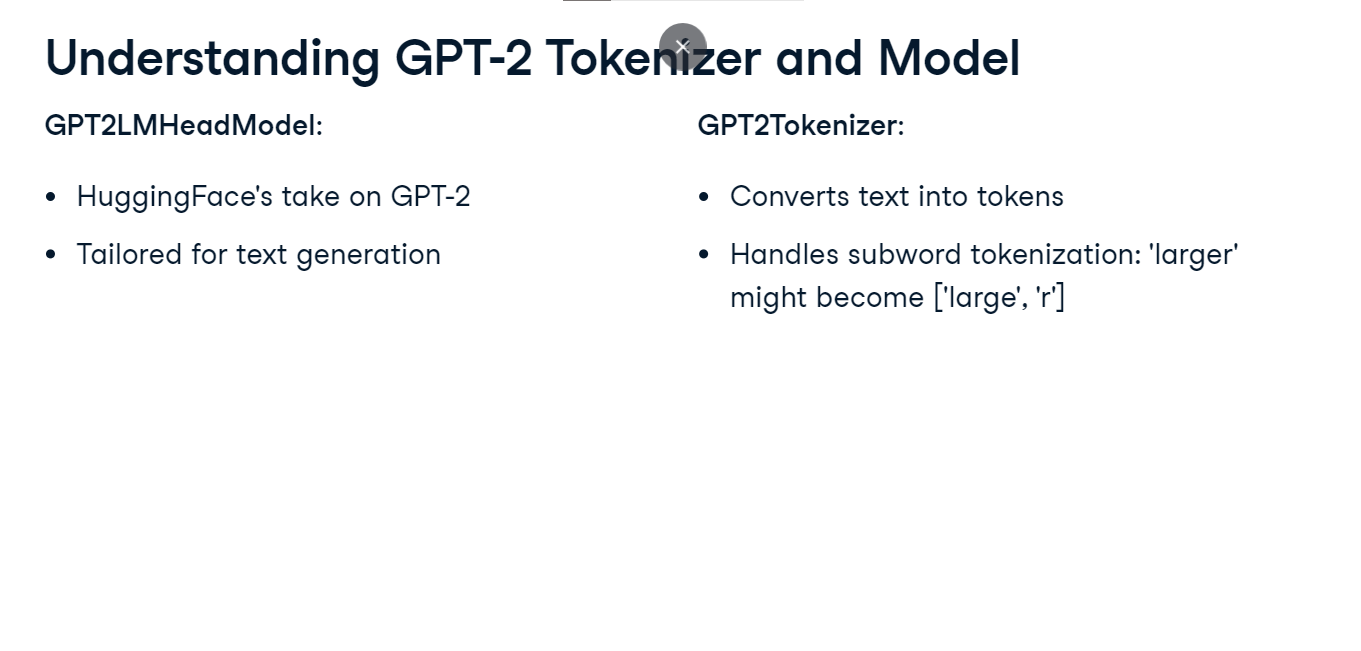
Building models from scratch can be effective for particular tasks, but pre-trained models offer advantages, having been trained on extensive datasets and delivering high performance in tasks like sentiment analysis, text completion, and language translation. However, they also pose challenges, such as high computational cost, significant storage requirements, and limited customization options.

**Pre-trained models in PyTorch**

Using PyTorch with Hugging Face Transformers gives us access to a library of pre-trained models. We'll try out GPT-2 and T5.

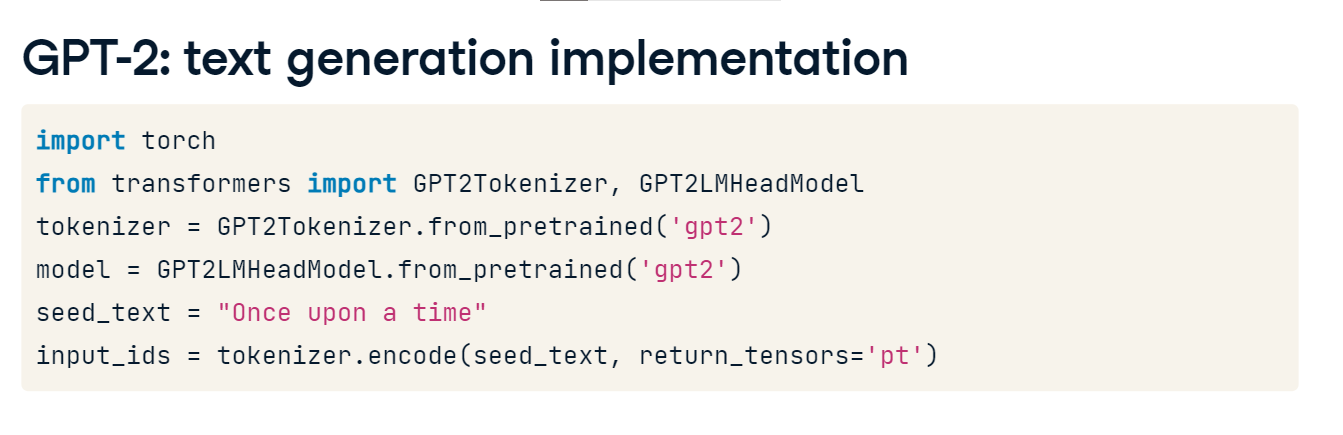
**Understanding GPT-2 Tokenizer and Model**

GPT2LMHeadModel is HuggingFace's take on GPT-2, tailored for tasks like text generation. GPT2Tokenizer converts text into numerical tokens. It includes subword tokenization, where words can be split into smaller units or 'subwords' to capture more nuanced meanings. For example, 'larger' might be tokenized into 'large' and 'r'.



**GPT-2: text generation implementation**

We start by importing the GPT2 modules from the transformers library. We initialize the GPT-2 model and tokenizer using the from\_pretrained method with the argument 'gpt2'. The tokenizer converts our input text into a format the model understands. Next, we set a seed text, "Once upon a time," to serve as our story's opening line. This seed text is encoded into input tensors using the tokenizer. The flag return\_tensors equals 'pt' specifies that we want these tensors in PyTorch format.



**GPT-2: text generation implementation II**

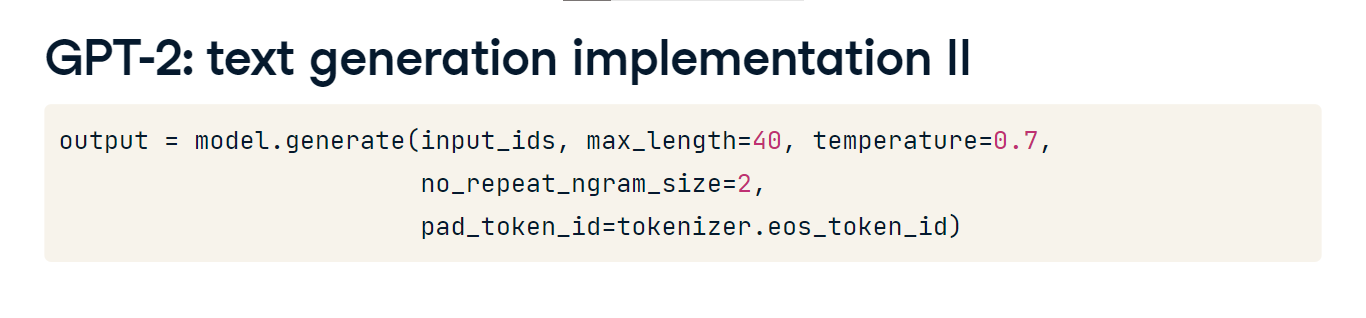
Now, we generate text using our model.

We set a maximum length for our generated text to 40 tokens using the max\_length argument.

The temperature parameter, set to 0-point-7, controls the randomness of the output, with lower values reducing randomness.

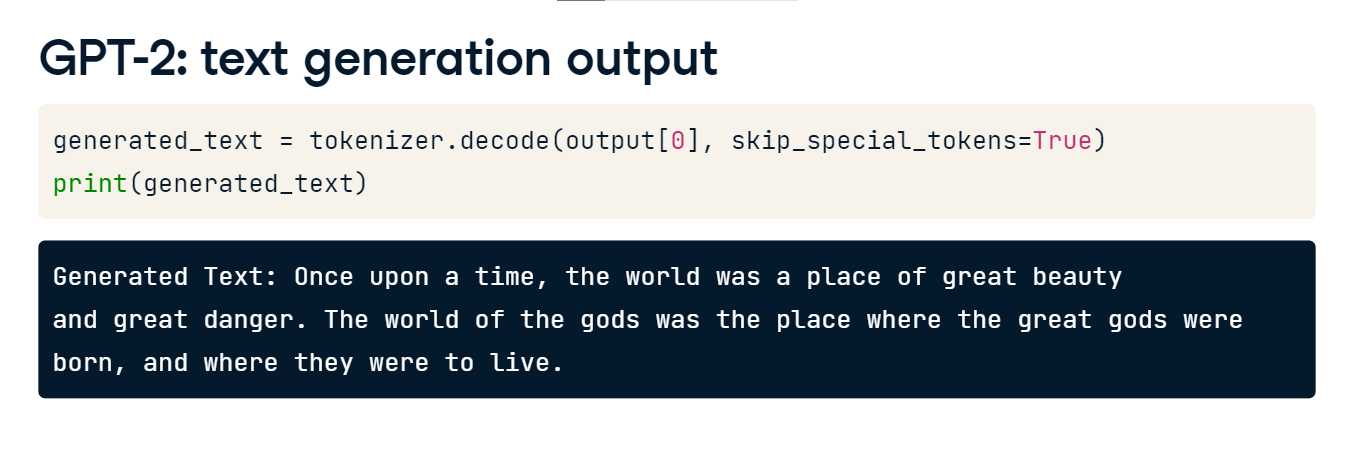
The no\_repeat\_ngram\_size parameter, set to two, prevents consecutive word repetition in the generated text.

Lastly, the pad\_token\_id is set to the ID of the end-of-sentence (EOS) token, which means the model pads the output with this token if it's shorter than the maximum length of 40 tokens.



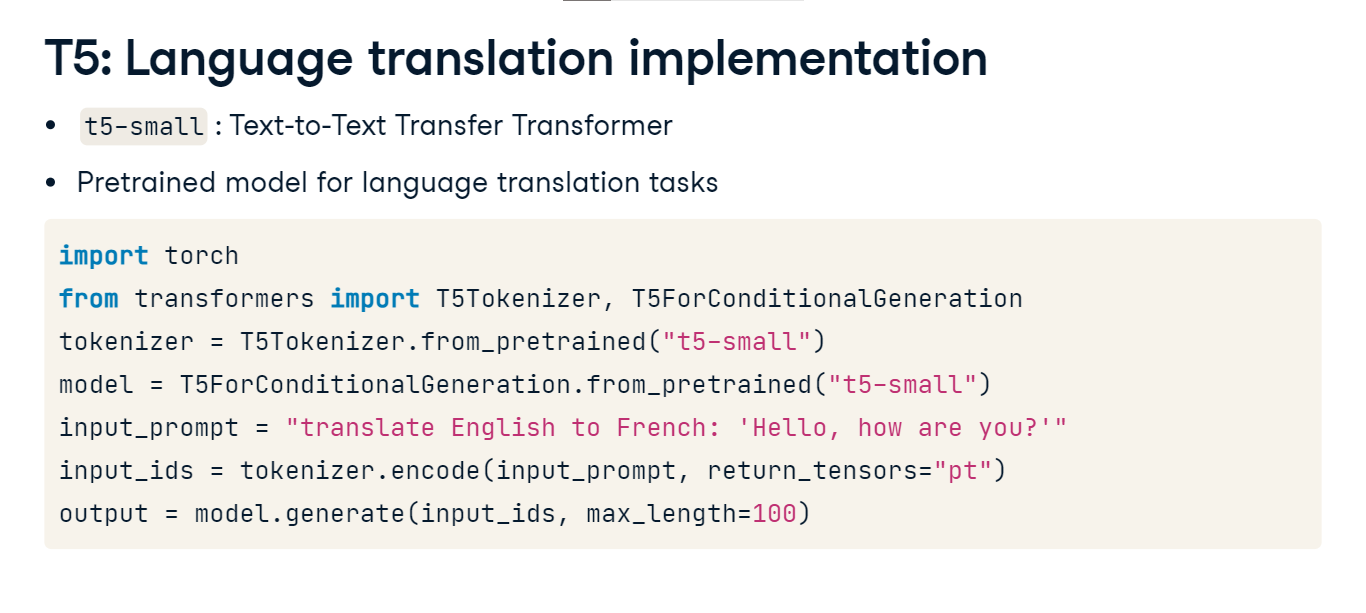
**GPT-2: text generation output**

Finally, we use the decode method to convert the token IDs in our output tensor back into text. skip\_special\_tokens equals True ensures that any special tokens used by the model for internal purposes, such as beginning- or end-of-sentence markers, are not included in the final output. The printed text demonstrates our GPT-2 model's successful story generation from the provided seed text.

****

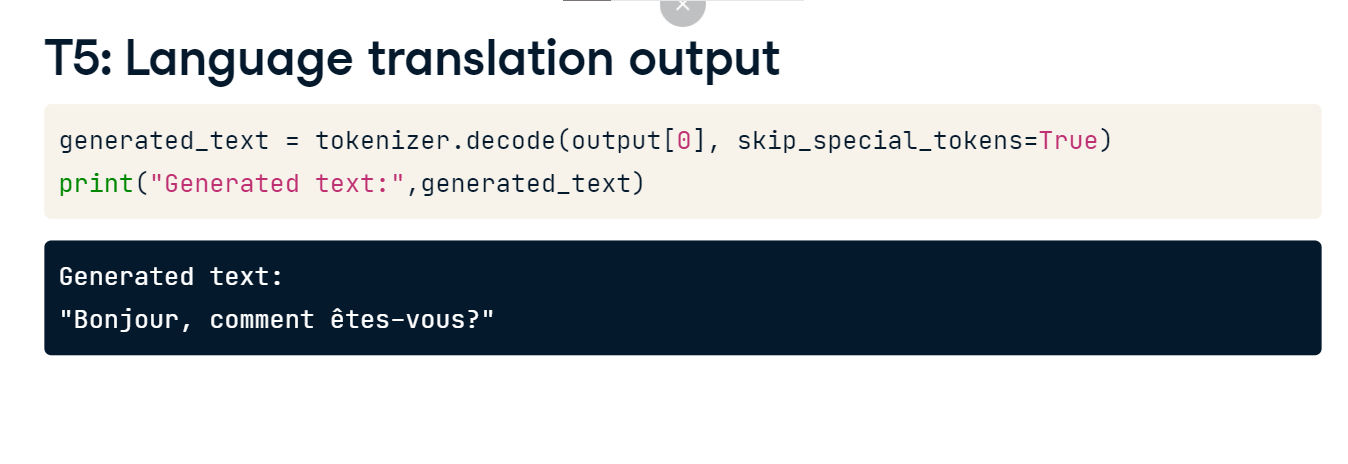
**T5: Language translation implementation**

Another text generation task involves language translation, where T5-small (Text-to-Text Transfer Transformer) is a specialized model. We import the necessary modules similar to GPT-2, only this time it's T5Tokenizer and T5ForConditionalGeneration. We initialize the T5 model and tokenizer using the 't5-small' model name. For language translation, we prepare an input prompt that spells out the task: "translate English to French," followed by the sentence "Hello, how are you?" we wish to translate. This prompt is encoded using the tokenizer, resulting in input IDs. These IDs are fed into the model for translation, with a max\_length of 100 to accommodate longer translations if needed.



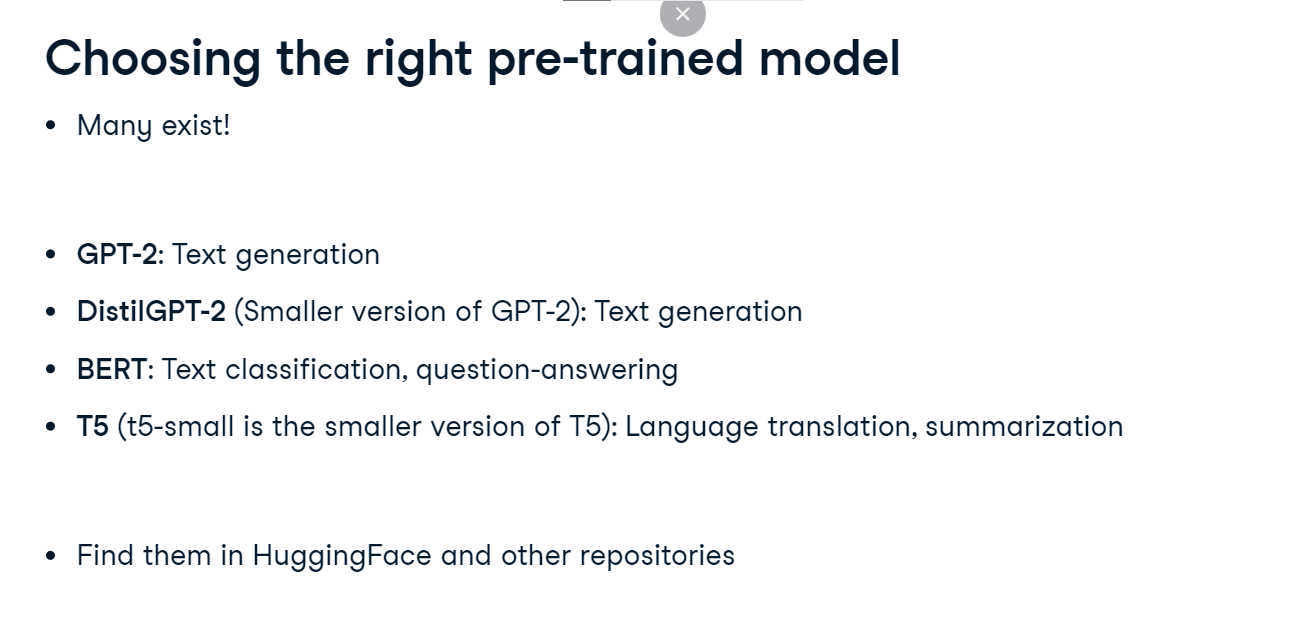
**T5: Language translation output**

We convert the generated output back to text using the decode function. Printing the generated text reveals a successful translation of our input to French. Occasionally, the generated text might not be accurate, given this is the smaller model and would need further tuning for better translation.



**Choosing the right pre-trained model**

We've explored two pre-trained models, but many more exist, and knowing which one to choose is key. While GPT-2 excels in text generation, its smaller sibling, DistilGPT-2, also specializes in similar tasks. BERT is optimal for text classification and question-answering tasks. T5 and t5-small are well-suited for language translation and summarization. The HuggingFace library is the official source for these models. They can also be found in other repositories and frameworks, so it's up to you to explore!



**Evaluation metrics for text generation**

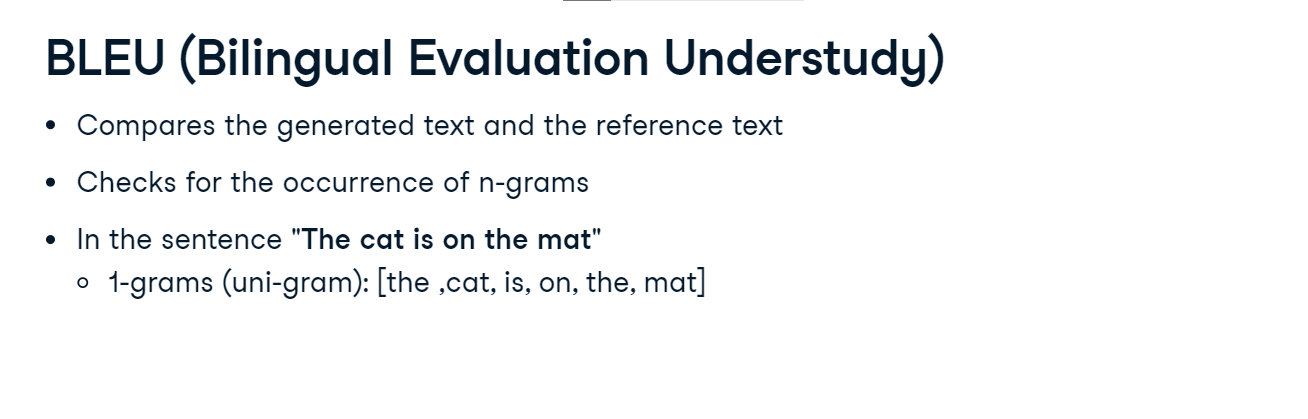
Great work! Let's evaluate our text generation models.

**Evaluating text generation**

Text generation aims to create human-like text, posing a unique evaluation challenge where traditional metrics like accuracy or F1 score can fall short. Instead, we evaluate the quality and relevance of the generated text using metrics like BLEU and ROUGE, which compare it to reference texts, evaluating its quality more closely with how humans perceive language.

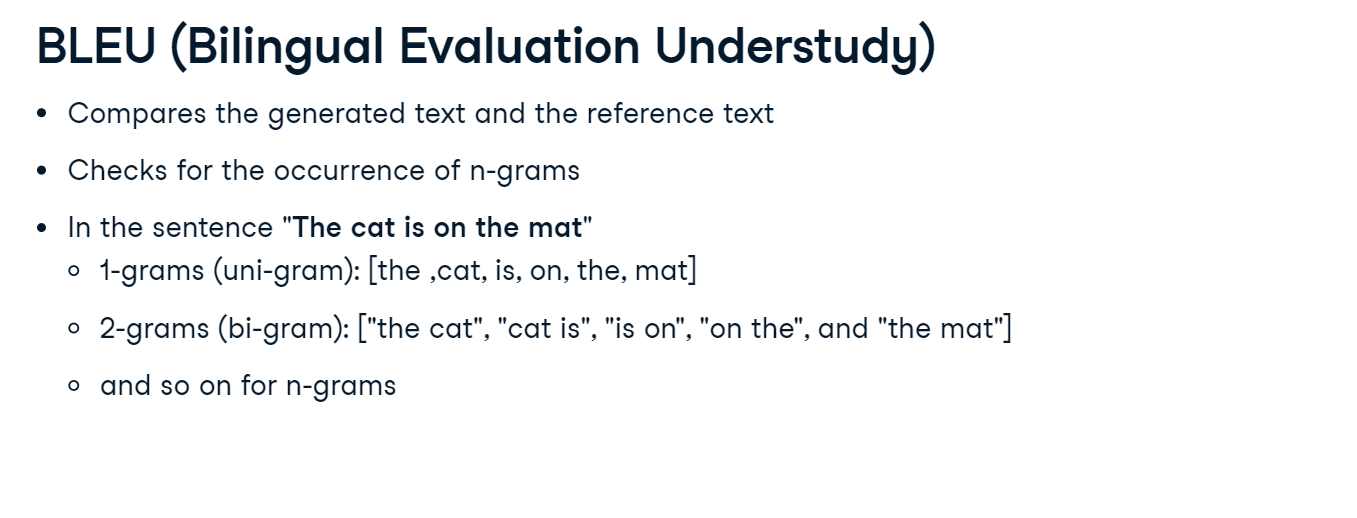
**BLEU (Bilingual Evaluation Understudy)**

To do this, we employ BLEU (Bilingual Evaluation Understudy), which compares the generated text with a reference text by examining the occurrence of n-grams. But what's an n-gram? In a sentence like 'the cat is on the mat', the 1-grams or uni-grams are each individual word, the 2-grams or bi-grams are 'the cat', 'cat is', and so on. The more the generated n-grams match the reference n-grams, the higher the BLEU score. A perfect match results in a score of 1-point-0, while zero would mean no match.



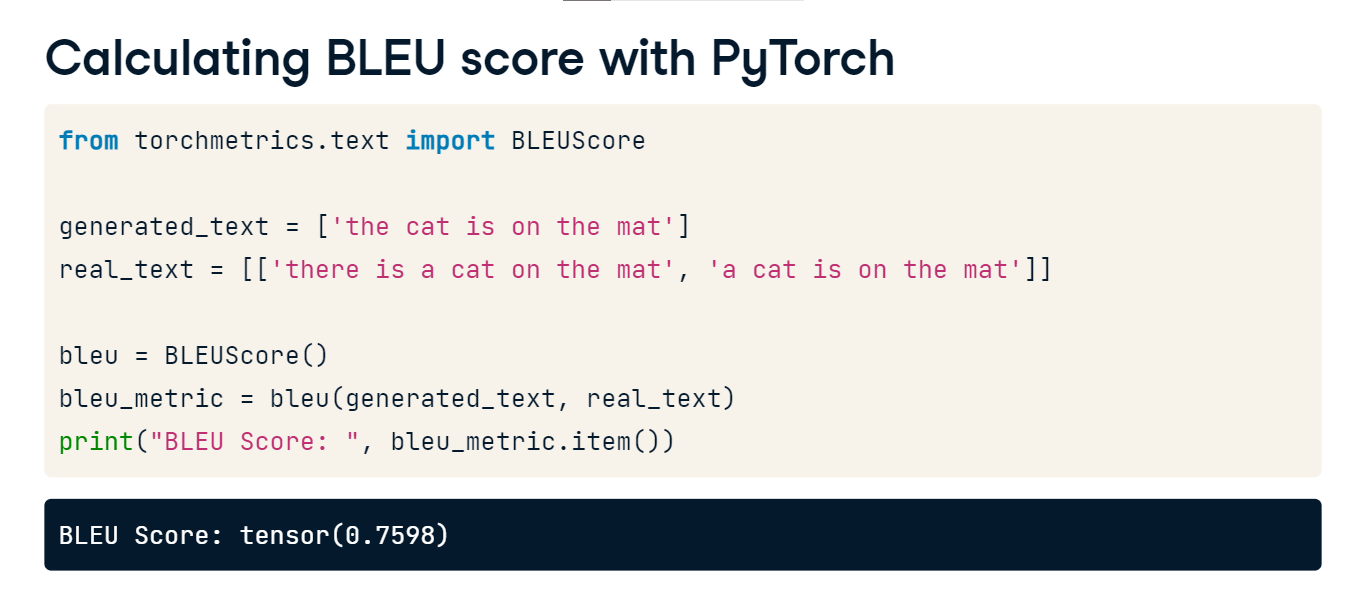
**Calculating BLEU score with PyTorch**

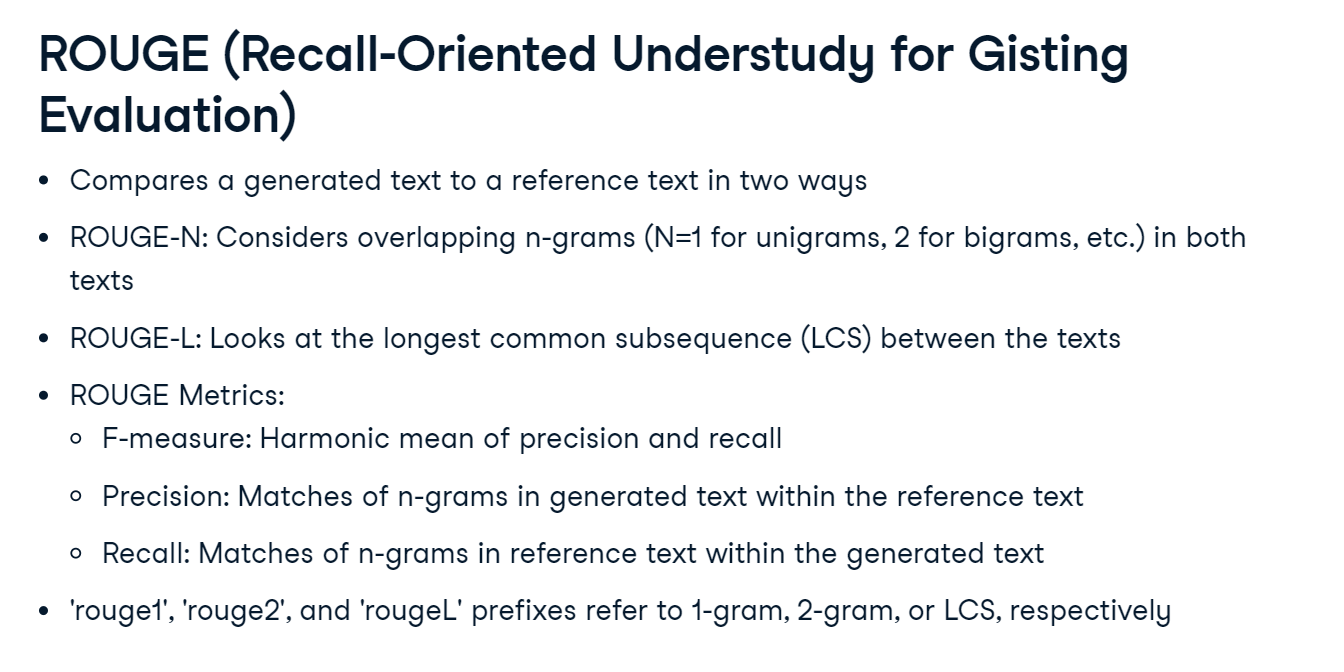
To calculate the BLEU score with PyTorch, we import BLEUScore from torchmetrics-dot-text. We initialize our predicted and target texts. In this case, we compare the generated text 'the cat is on the mat' with two reference texts. We initialize and calculate the BLEU score and call the instance of bleu by passing generated text and real text. The resulting score is approximately 0-point-76, representing the average precision of the n-grams in the generated text that also appear in the reference text.

****

**ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**

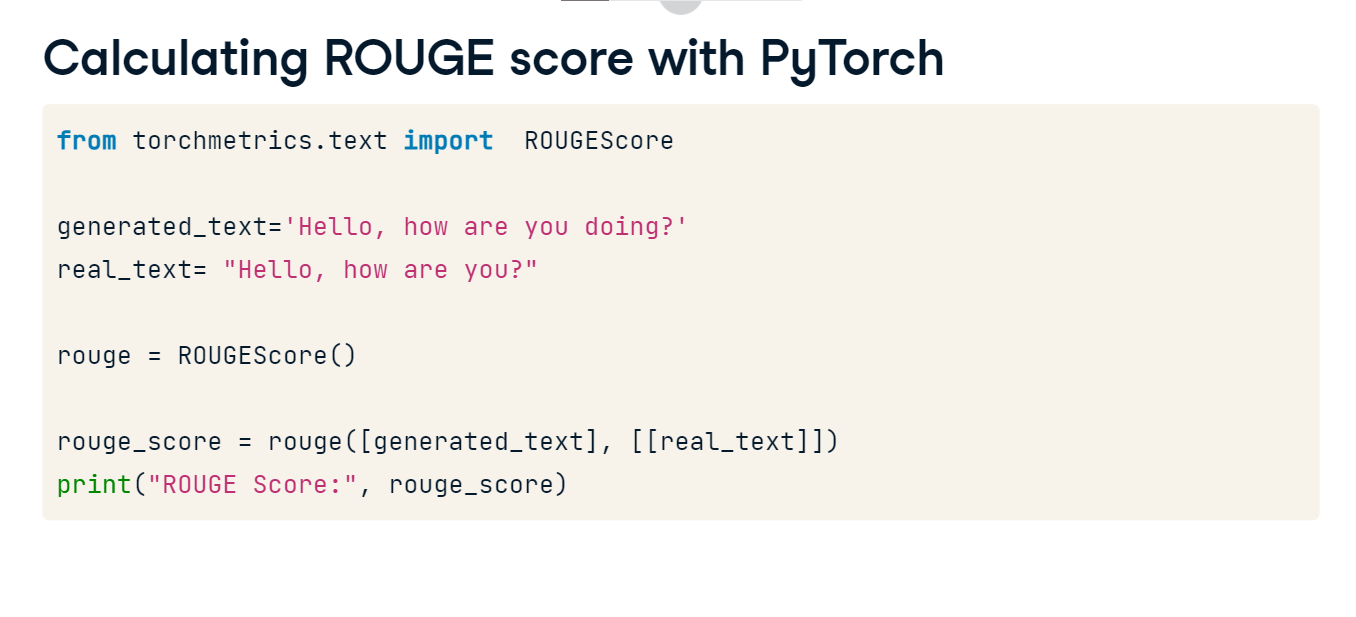
ROUGE (Recall-Oriented Understudy for Gisting Evaluation) assesses generated text against reference text in two ways. ROUGE-N examines overlapping n-grams, with N representing the n-gram order. ROUGE-L checks for the longest common subsequence (LCS), the longest shared word sequence between the generated and reference text. ROUGE has three metrics: F-measure, precision, and recall. F-measure is the harmonic mean of precision and recall. Precision checks matches of n-grams in the generated text that are in the reference text. Recall checks for matches of n-grams in the reference text that appear in the generated text. The prefixes 'rouge1', 'rouge2', and 'rougeL' specify the n-gram order or LCS.



****

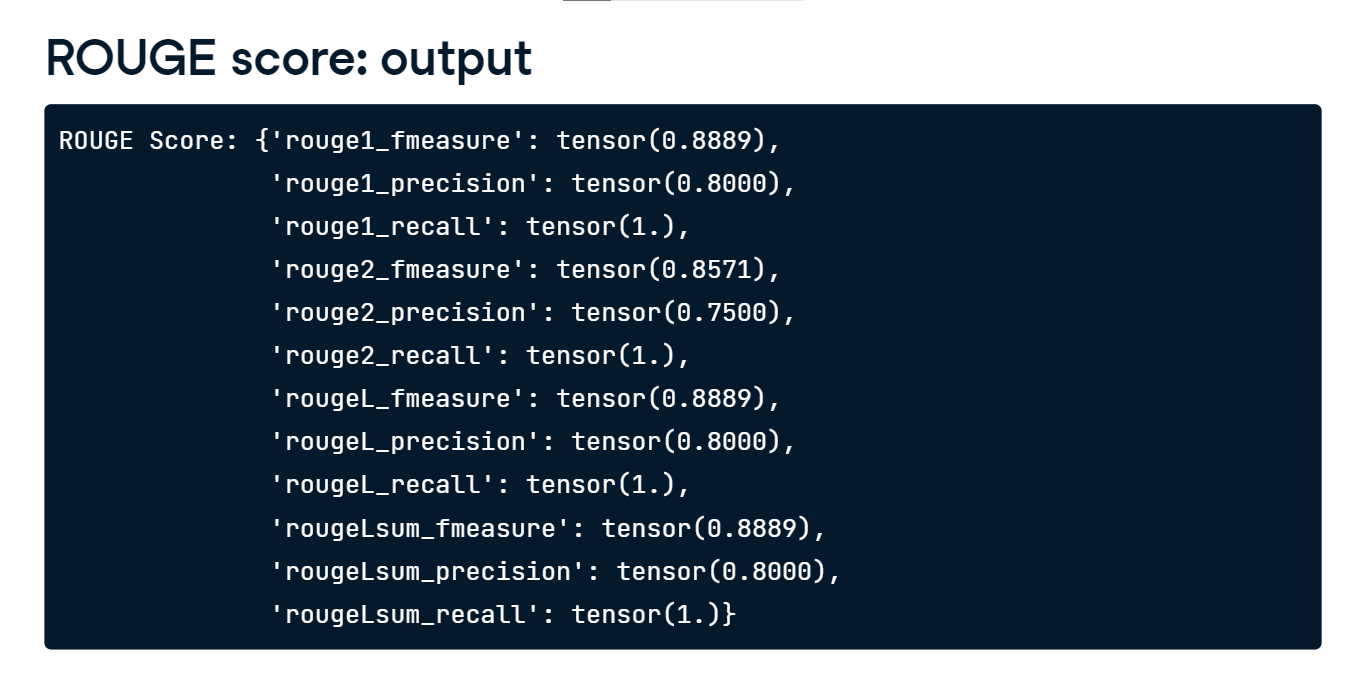
**Calculating ROUGE score with PyTorch**

We import the ROUGEScore module from torchmetrics-dot-text to calculate the ROUGE Score. We define both generated and real text, where the real text represents the model's ideal output, and the generated text is the actual output. We then initialize the ROUGEScore module and apply it to our texts to obtain the ROUGE Score, displayed in the next slide.



**ROUGE score: output**

In the ROUGE Score output, we first see the rouge1\_fmeasure, precision, and recall. These are the F1 score, precision, and recall, respectively, based on single words or uni-grams in the text. Next, we have the rouge2\_fmeasure, precision, and recall. These metrics consider two consecutive words or bi-grams in the text. Then we see rougeL\_fmeasure, precision, and recall. The "L" stands for "longest", representing the longest common subsequence between the generated and real text. Lastly, rougeLsum\_fmeasure, precision, and recall consider the longest matching sequences, accounting for all such sequences in the text and summing them up. Each of these provides a different perspective on the quality and similarity of the generated text. A score of 0-point-88 means that 88% of the generated text matches the real text.



**Considerations and limitations**

As with most metrics, there are considerations to keep in mind. ROUGE and BLEU center around word presence without delving into semantic understanding. They are sensitive to the length of the generated text, and the quality and choice of reference text play a crucial role in the score outcomes