Note: This assignment assumes that you have covered most of the search algorithms and evaluation metrics in text generation in the November 1st lecture on Search Algorithms. Please read the reading materials and lecture notes if you missed class.

When trained on human-written text, auto-regressive language models (e.g., GPT3 [BMR+20]) can produce natural text as humans do. In this homework, you will implement various decoding algorithms, generate text using pre-trained large language models (LLMs) on generation tasks, evaluate the output text, and justify the limitations of current decoding methods. Note that this homework is quite easy since you don't actually need to implement anything from scratch; instead, you will make a complete pipeline of text generation research including decoding, automatic and human evaluation, and analysis. Here are the steps you need to follow and you have to report outputs from the **Tasks** of each step in your submission of the spreadsheet, codebase, and final report.

Step 1: Implementation of decoding algorithms

```
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

prompt = "Today I believe we can finally"
input_ids = tokenizer(prompt, return_tensors="pt").input_ids

/* generate up to 30 tokens */
outputs = model.generate(input_ids, do_sample=False, max_length=30)
tokenizer.batch_decode(outputs, skip_special_tokens=True)

/* step 1 */
outputs1 = model.YourDecodingAlgorithmToImplement1(input_ids)
outputs2 = model.YourDecodingAlgorithmToImplement2(input_ids)
..

```

You can first go to HuggingFace API on text generation (link) and simply run existing scripts to generate text. If you load pre-trained autoregressive language models like gpt2, you can use a variety of decoding algorithms. You should report the outputs from four different decoding algorithms in this step: *greedy search*, *beam search*, *top-k sampling*, and *top-p sampling* (or nucleus sampling). You don't actually need to implement these algorithms by yourself. Instead, I encourage* you to use pre-implemented decoding functions in HuggingFace, like greedy_search().

Task 1.1. For a prompt like "Today I believe we can finally", you should show the **output sequence(s) from the four decoding algorithms with the specific parameters** you used (e.g., beam size, n-best, k, p). For this step, you can choose any random prompt. You must also calculate the **likelihood of each output sequence** by the log sum of every token logit.

*If you write each algorithm from scratch, you will receive a bonus point.

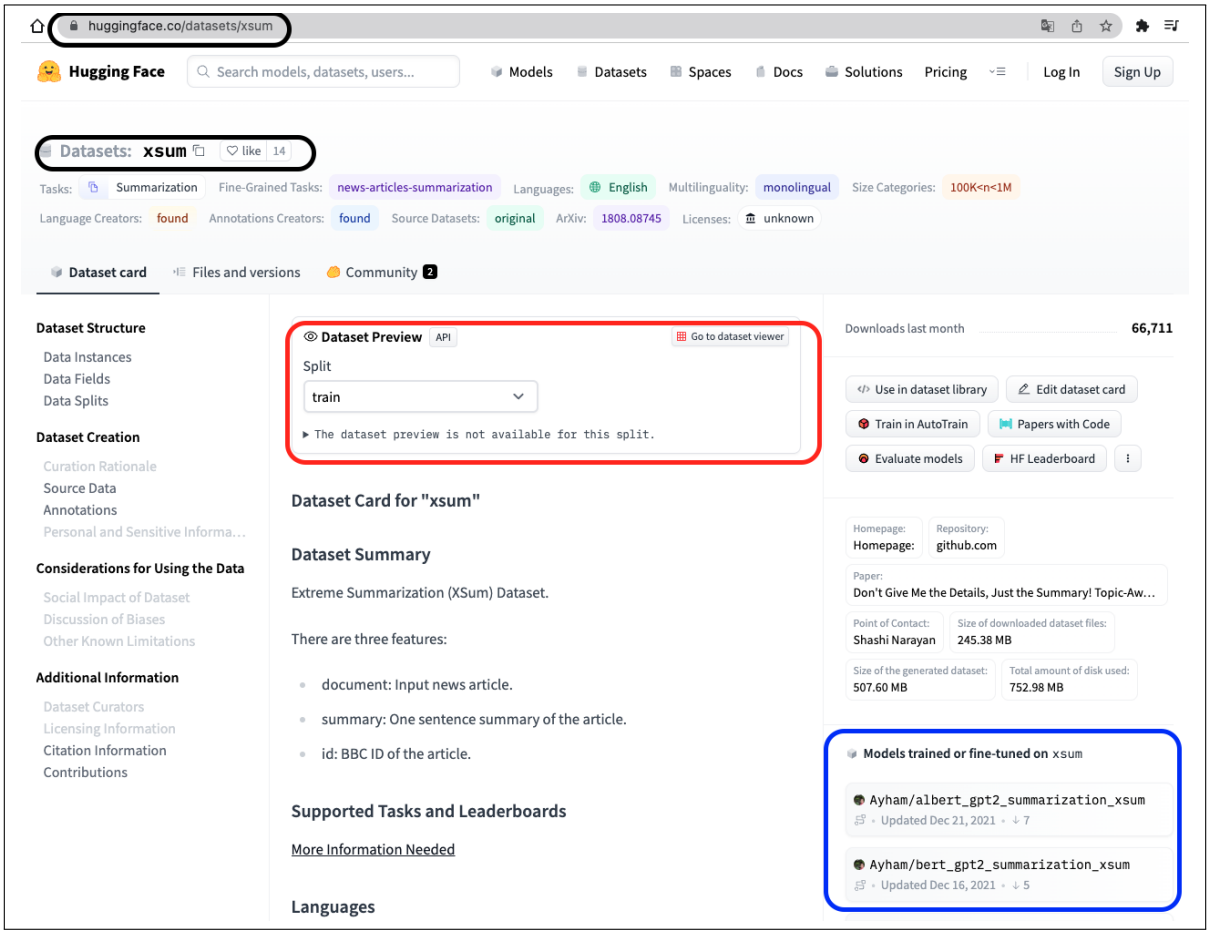

Figure 1: HuggingFace Dataset Interface: https://huggingface.co/datasets?task_categories=task_categories:summarization

Step 2: Decoding for downstream generation tasks

```
from datasets import load_dataset
dataset = load_dataset("xsum")
```

Your next step is to select a specific task for evaluating decoding algorithms where reference text is provided. A variety of tasks are available, such as machine translation, abstraction summarization, dialogue generation, paraphrasing, and style transfer, where the input text and output text (reference text) are provided. For instance, you can choose a dataset called XSUM from the summarization task in the HuggingFace dataset repository, as depicted in Figure 1. You can see some example instances from Dataset Preview and download the dataset easily from a few lines of code:

Task 2.1. Once you choose a specific task and dataset for your text generation, now you need to get actual outputs from the generation model. This homework does not require you to train your encoder-decoder model from scratch on the target dataset. On the dataset page, you can find the fine-tuned

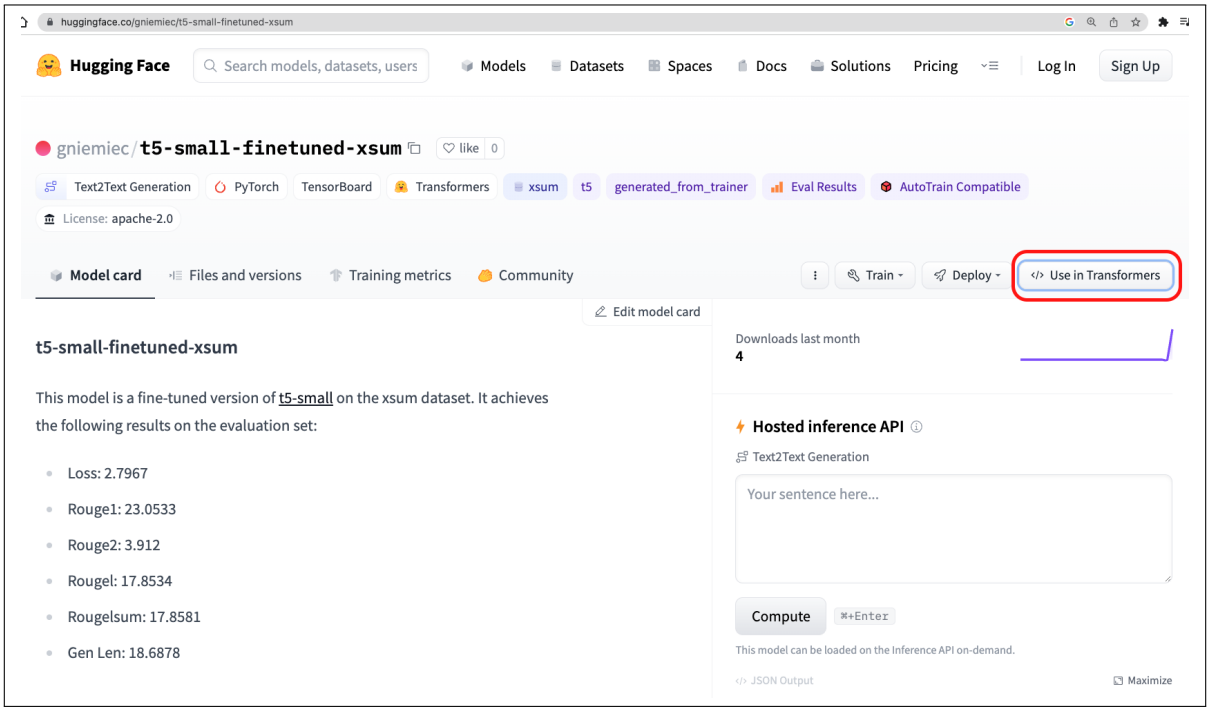

Figure 2: Loading the pre-trained T5 model fine-tuned on XSUM dataset

summarization models (blue box in Figure 1).[†] For instance, you can load the small T5 model already fine-tuned on XSUM dataset https://huggingface.co/gniemiec/t5-small-finetuned-xsum, as shown in Figure 2. On the top right, click the USE IN TRANSFORMERS button to access lines of code for loading the model into HuggingFace.

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained("gniemiec/t5-small-finetuned-xsum")

model = AutoModelForSeq2SeqLM.from_pretrained("gniemiec/t5-small-finetuned-xsum")

/* generate your own summary using different decoding algorithms */
outputs1 = model.YourDecodingAlgorithmToImplement1(input_ids)
outputs2 = model.YourDecodingAlgorithmToImplement2(input_ids)
..
tokenizer.batch_decode(outputs1, skip_special_tokens=True)
..
```

Task. From the test set of the dataset, you can simply generate the output summary. In this step, you have to write your own script to load the fine-tuned model and decode output text given input text from the test samples. More specifically, you will use the four decoding algorithms implemented in Step #1 and generate output text on the test set of the target task you choose. If your test set is more than 100 samples, please only use the first 100 samples in the following evaluation.

[†]If the dataset page does not include the model list, you can search the dataset name in the model cards.

You first need to make a spreadsheet where each row is a text sample in the test set. Then, you need to make four additional columns and copy&paste four output texts from the four decoding algorithms. For instance, your spreadsheet now should be N-by-5 where N is the number of the test set (maximum 100).

Step 3: Automatic and Human Evaluation

Lastly, you will need to evaluate your generated text. You select the right evaluation metric for your task. You can find information about the evaluation on text generation in Slides 52-60 of the Search Algorithms lecture. Since this is not an open-ended generation task, you can use your evaluation metrics to determine how similar your decoded outputs are to the reference text.[‡]

```
1
2  /* generate your own summary using different decoding algorithms */
3  outputs1 = model.YourDecodingAlgorithmToImplement1(input_ids)
4  outputs2 = model.YourDecodingAlgorithmToImplement2(input_ids)
5  ..
6  token_outputs1 = tokenizer.batch_decode(outputs1, skip_special_tokens=True)
7  ..
```

Task 3.1. You could choose **at least one** of the content overlap metrics (e.g., BLEU, ROUGE, CIDER, METEOR, PYRAMID) and model-metric metrics (e.g., Word Mover's distance, BERT score), and **measure these metric scores of your decoded outputs with respect to the reference text**. Please consider which automatic metrics would be appropriate for your task and provide a justification in the report.

Again, it is not necessary to implement these metrics from scratch; instead, you can find pre-implemented evaluation metrics at Huggingface's evaluate-metric. As you decode your outputs, you need to report **the metric score of each sample** in a separate column in the spreadsheet. The report should include **the averaged metric scores across all samples** (e.g., 100 test samples) and a comparison of the decoding algorithms that work.

Task 3.2. You may notice that automatic evaluation does not accurately measure your task's performance. Here, you will **devise two or three aspects of human evaluation related to your target task**, then **manually evaluate them with a Likert scale (1-5)** by yourself. Please check out what types of aspects (e.g., fluency, coherence, formality, typicality) could be used in the human evaluation of generated text in Slide 62 of the Search Algorithms lecture.

Since you are not expert judges, or your judgment could be subjective, I do not evaluate the quality of your scores. This step aims to identify the gap between automatic evaluation metrics and human evaluations. Because human evaluation is time-consuming and costly, you can [**only select the first 50 samples** from your test set. To avoid any biases from the predicted outputs, you are supposed to make a new tab in your spreadsheet only with test samples and manually label the two or three aspects of text quality as extra columns. Report the averaged automatic and human evaluation scores for the first 50 sentences and describe how they differ in practice.

[‡]I believe you also learned the limitation of single reference-based evaluation as well in the class

Deliverable

Please upload your (1) spreadsheet (e.g., CVS, Excel files, Google Spreadsheet) of your decoded outputs with evaluation scores from both automatic and human measurements, (2) zipped code of the decoding algorithms with evaluation script, and (3) technical report (maximum 4 pages) to Canvas by Dec 8, 11:59pm. The spreadsheet, code, and report should reflect the outcomes from Tasks 1.1, 2.1, 3.1, and 3.2.

References

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.