The auto-regressive language models (e.g., GPT3 [BMR+20]) trained on human-written text can produce natural text as humans do. In this homework, you will implement and use various decoding algorithms, generate text using the pre-trained large language models (LLMs) on different generation tasks, evaluate the output text, and justify the limitations of current decoding methods. The lead TA for this assignment is Shuyu Gan (gan00067@umn.edu). Please communicate with the lead TA via Slack, email, or during office hours. This is an individual homework.

This assignment assumes that you have covered most of the search algorithms and evaluation metrics in text generation on Language Models: Search and Decoding (Feb 18) and Language Models: Evaluations and Applications (Mar 4). Please read the reading materials and lecture notes if you missed class.

In this homework,you are not required to implement language models from scratch. Instead, your goal is to build a complete **text generation research pipeline** using existing tools and models. The pipeline will cover: task selection, decoding, automatic evaluation, human evaluation, and analysis of outputs. Please follow the steps below, report the results from the **Tasks** in each step, and submit three deliverables: (1) a spreadsheet with results, (2) your codebase with Jupyter notebook, and (3) a PDF report

Step 1: Trying out different decoding algorithms using HuggingFace

```
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) and run an example script to generate text. For instance in the example above, once you load pre-trained autoregressive language models like GPT2 [RWC+19], the HuggingFace library allows you to select a variety of decoding algorithms.

Task 1 You should report the outputs from four different decoding algorithms covered in the class: *greedy search*, *beam search*, *top-k sampling*, and *top-p sampling*. You are not expected to implement these algorithms yourself. Instead, directly use the pre-implemented decoding functions in HuggingFace (e.g., the `generate()` method with proper arguments).

For a prompt like "Today, I believe we can finally," you should report **four output text from the four different decoding algorithms with the specific parameters** you used (e.g., beam size, n-best, k, p). For this step, make up five prompts yourself – you can use any random prompts you want.

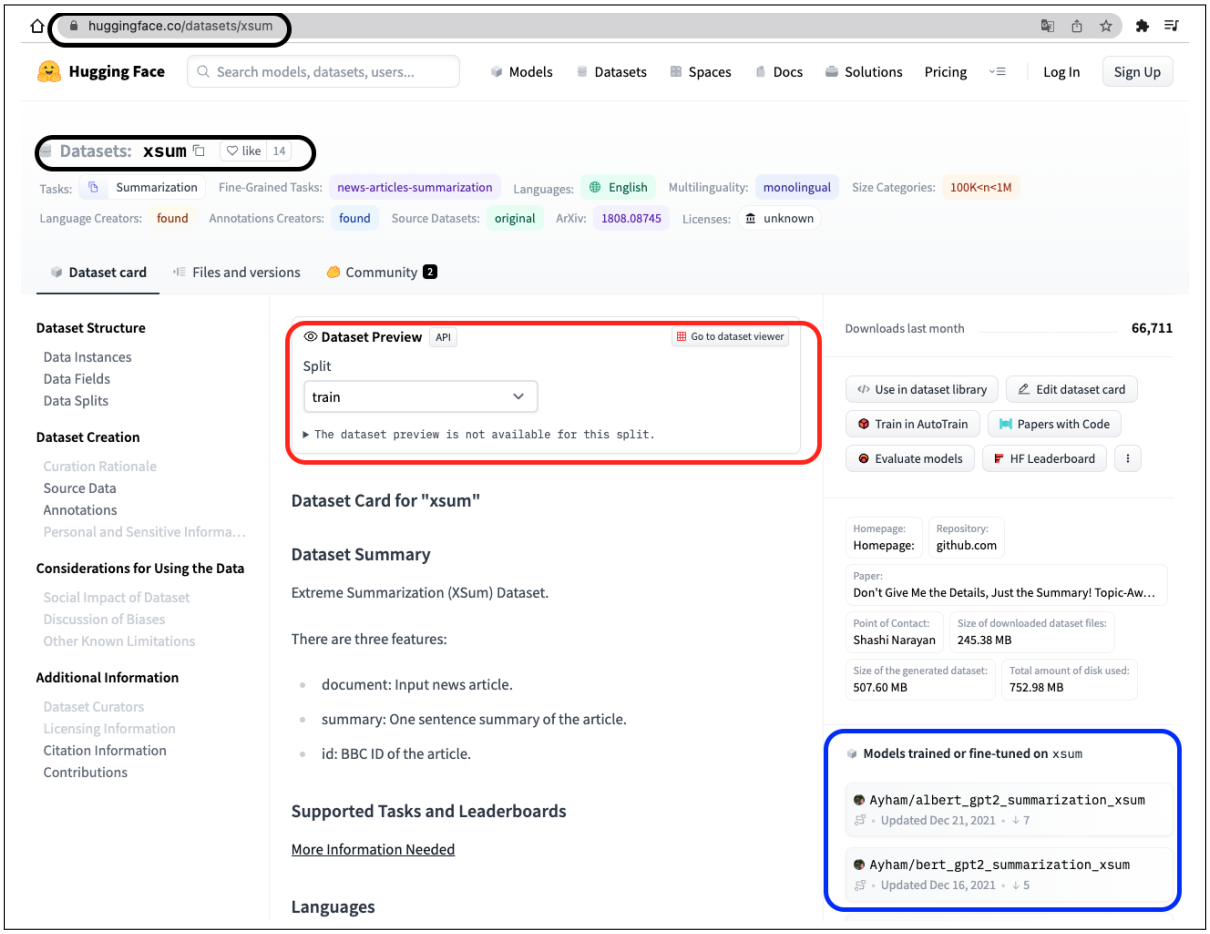

Figure 1: HuggingFace Dataset Interface: https://huggingface.co/datasets?task_categories=task_categories:summarization

In addition, you must calculate the **per-token log-likelihood** or the **perplexity** of each generated output sequence. For details, see the official tutorial: Perplexity calculation in HuggingFace.

Reporting. Prepare a **spreadsheet**. In the first tab, each row corresponds to one prompt. The columns should include: the prompt, four outputs (one per decoding algorithm), the decoding parameters, and the perplexity/log-likelihood. Thus, the first tab should be an N-by-7 table, where N=5. You will add further tabs in later steps. As an example, your spreadsheet should look like the following Table 1(one row shown here).

Prompt	Greedy	Beam	Top-k	Top-p	Parameters	PPL/LL
Today, I believe...	...	...	...	...	beam=5; k=50; p=0.95	loglik=-123.4; ppl=45.6

Table 1: An example row of the spreadsheet format for Step 1 results.

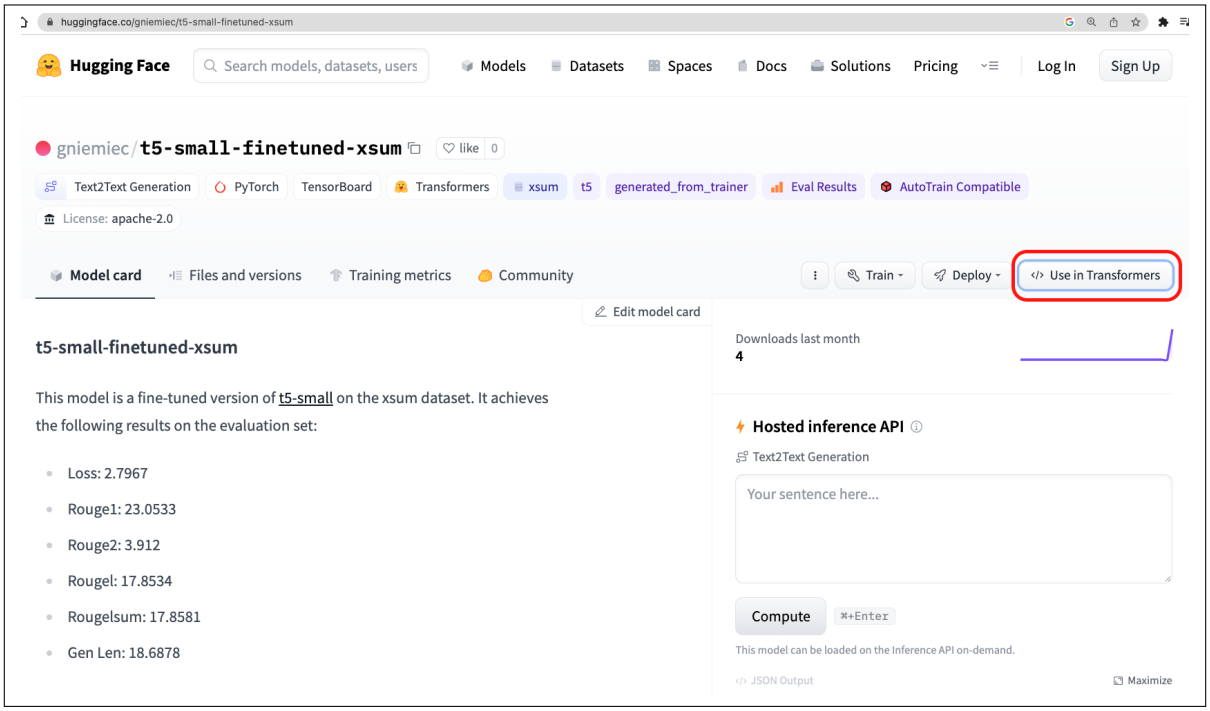

Figure 2: Loading the pre-trained T5 model fine-tuned on XSUM dataset

Step 2: Decoding for downstream generation tasks

Perplexity is an intrinsic method to evaluate your language model. Now, we evaluate the language model with an extrinsic evaluation. First, select a **specific task** for evaluating decoding algorithms where the task provides reference text. For instance, you can choose a dataset called XSUM [NCL18] 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:

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

Various tasks are available, such as machine translation, abstraction summarization, dialogue generation, paraphrasing, and style transfer, where the input and output text (reference text) are provided. Please check available tasks and fine-tuned models in HuggingFace Tasks.

Once you choose a specific task and dataset for your text generation, you then need to get actual outputs from the generation model. This homework does not require you to train your own generation models from scratch on the target dataset. Instead, on the dataset page, you can find the existing fine-tuned models (blue box in Figure 1).[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.

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

Task 2 On the test set (or development set if there is no test set given) of the dataset you choose, you can simply generate the output summary or responses using the default decoding functions in HuggingFace. 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. Below is an example script to load the model and decode a batch of test input:

```
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)
..
```

Specifically, you will use the four decoding algorithms implemented in Step #1 and generate output texts on the test set of your target task. If your test set is more than 50 samples, please only use the *first 50 samples* in the following evaluation.

Reporting. In the **second tab of your spreadsheet**, make one row for each input sample. Each row should include: the input text, four generated outputs, and the reference text. Thus, the second tab should be an N-by-6 table, where N is the number of test samples (maximum 50), as illustrated in Table 2(One row example).

Input	Greedy	Beam	Top-k	Top-p	Reference
(Sample input)	...	...	...	...	(Gold reference)

Table 2: An example row of the spreadsheet format for Step 2 results.

Step 3: Automatic and Human Evaluation

Lastly, you evaluate quality of your generated text by choosing the right evaluation metric for your task. Since this is not an open-ended generation task, you can use reference-based evaluation metrics to determine how similar your generated outputs are to the reference text.

```

/* generate your own summary using different decoding algorithms */
outputs1 = model.YourDecodingAlgorithmToImplement1(input_ids)
outputs2 = model.YourDecodingAlgorithmToImplement2(input_ids)
..
token_outputs1 = tokenizer.batch_decode(outputs1, skip_special_tokens=True)
..
```

Task 3.1 There are two categories of automatic evaluation: content overlap-based metrics (e.g., BLEU [PRWZ02], ROUGE [LH03]) and model-based metrics (e.g., Word Mover's distance [KSKW15], BERT score [ZKW+20]). You should choose **at least one** content-overlap metric **and** at least one model-based metric, and **measure these metric scores of your decoded outputs in Step #2 with respect to the reference text**. Please consider which automatic metrics would be appropriate for your task and provide a justification in the report. By reading the original papers of the dataset or task you are using, you can find which evaluation metrics are used on which tasks. Again, it is not necessary to implement these metrics from scratch; instead, you can find pre-implemented evaluation metrics at Huggingface's evaluate-metric. For instance, below is an example usage of Huggingface's evaluate function for BLEU metric:

```
>>> predictions = ["hello there general kenobi", "foo bar foobar"]
>>> references = [
...     ["hello there general kenobi", "hello there !"],
...     ["foo bar foobar"]
... ]
>>> bleu = evaluate.load("bleu")
>>> results = bleu.compute(predictions=predictions, references=references)
>>> print(results)
{"bleu": 1.0, "precisions": [1.0, 1.0, 1.0, 1.0], "brevity_penalty": 1.0,
"length_ratio": 1.1666666666666667, "translation_length": 7,
"reference_length": 6}
```

As you decode your outputs, you need to report **the metric score of each sample** in a new column in the spreadsheet made in Step #2. The report should include **the averaged metric scores across all samples** (e.g., 50 test samples) and a comparison of the decoding algorithms that work.

Task 3.2 You may notice that automatic evaluation does not always accurately measure your task's performance. In this task, you will **devise two or three aspects of human evaluation related to your target task**; please check out what types of aspects (e.g., fluency, coherence, formality, typicality) could be used in the human evaluation of generated text in the lecture on Language Model Evaluation. Include a justification for these choices in your report. After this, each person in your team will **manually annotate scores of each aspect with a Likert scale (1-5)** and rate the level of agreement between the human evaluation scores of all of the team members.

This step aims to identify the gap between automatic evaluation metrics and human evaluations and show their difference in your report. Because human evaluation is time-consuming and costly, you can **only select the first 20 samples** from your test set.

To avoid any biases from the predicted outputs, you should make a third tab in your spreadsheet with a row for each of the first 20 test samples. Since this is an individual homework and you may not be an expert judge, please still provide **your own annotations** on the chosen evaluation dimensions (e.g., factuality, fluency, coherence) using a Likert scale (15). If possible, you are encouraged to collect additional annotations from peers or others to increase reliability, but this is optional.

Your spreadsheet should therefore include the first 20 test samples and the corresponding human evaluation scores. Report **the averaged automatic and human evaluation scores for the first 20 samples** and **analyze how they differ in practice** in your report.

Since humans are inconsistent and subjective, we next need to evaluate how consistent annotators

were with one another. You can use the nltk's nltk.metrics library to calculate the inter-annotator agreement (IAA) scores such as Krippendorff's alpha.[2] Below is an example code for the calculation of Krippendorff's alpha to measure the inter-annotator agreement.

```
from nltk import agreement
rater1 = [1,1,1]
rater2 = [1,1,0]
rater3 = [0,1,1]

taskdata=[[0,str(i),str(rater1[i])] for i in range(0,len(rater1))]+ \
         [[1,str(i),str(rater2[I])] for i in range(0,len(rater2))]+ \
         [[2,str(i),str(rater3[i])] for i in range(0,len(rater3))]
ratingtask = agreement.AnnotationTask(data=taskdata)
print("alpha " +str(ratingtask.alpha()))
```

Please report the agreement score in your report and explain how you think of it.

Deliverables

Please upload your code, spreadsheet, and report to Canvas by Oct 19 , 11:59pm.

Code: You should submit a **zipped file** containing the decoding algorithms with evaluation scripts with Jupyter Notebook.

Report and Spreadsheet: Submit a PDF report (maximum six pages, excluding references and appendix) and a spreadsheet (CSV, Excel, or Google Sheet). The spreadsheet must include three tabs corresponding to:

- **Step 1:** Decoding outputs for five prompts (N=5), four decoding methods, hyper-parameters, and perplexity/log-likelihood (N×7 table).
- **Step 2:** Decoded outputs for the first 50 test samples (N $\leq$ 50), including input text, four decoding outputs, reference text(N×6 table).
- **Step 3:** Evaluation of Step 2 results. Extend the Step 2 tab by adding new columns for automatic evaluation scores (e.g., ROUGE-L, BERTScore) for each decoding method across all samples. In addition, create a separate third tab for human evaluation of the first 20 samples, including your own Likert-scale ratings on the selected evaluation dimensions.(If you are able to collect extra annotations from peers or others, you may also include them, but this is optional.)

For the report, you must use the provided LaTeX template (link).

Formatting convention: All your files submitted should follow this naming convention: **CSCI5541-F25-HW4-{Name}.{zip,pdf,csv}**.

Rubric (25 points + 4 bonus points)

- **Task 1 : Implementation of Decoding Algorithms (12 points)**
  - All 4 decoding algorithms implemented, outputs in spreadsheet (+2 x 4 algorithms = +8)
  - Parameters of the algorithms included in spreadsheet (+1)

[2]The detailed agreement measurement on human annotations will be covered on an upcoming lecture on Dataset, Annotation, and Evaluation.

  - Prompt is consistent across all algorithms (+1)
  - Perplexity/Likelihood of each output is calculated and in spreadsheet (+2)
- **Task 2: Decoding for extrinsic evaluation (2 points)**
  - Nx6 spreadsheet correctly generated with inputs, four outputs, and reference text (+1)
  - No mistakes in results or code (+1)
- **Task 3.1 Automatic Evaluation (4 points)**
  - At least one overlap-based metric is implemented and justification included (+1)
  - At least one model-based metric is implemented and justification included (+1)
  - Metrics are calculated between reference text and decoded outputs (+1)
  - Average metric score across all samples is reported (+1)
- **Task 3.2 Human Evaluation (5 points)**
  - At least 2-3 aspects of human evaluation for the target task devised (+1)
  - Reasoning given behind choice of aspects (+1)
  - Majority/Average voting is implemented (+1)
  - Difference between human and automatic evaluation is highlighted in report (+1)
  - Inter-annotator agreement is calculated (+1)
- **Report (2 points)**
  - Report contains full information about models and dataset chosen, the requested spreadsheets and associated metric results, a comparison of decoding algorithms, and justification of design decisions. (+2)
  - Partial report submitted (i.e., some pieces of the report are missing) (+1)
- **Bonus Point (Max: +4 points)**
  - Trying out other advanced decoding algorithms implemented in HuggingFace (+1)
  - Multiple evaluation metrics (2 of each category(content overlap and model-metric) are implemented) (+1)
  - Any other interesting analyses performed (+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.

[KSKW15] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In Francis Bach and David Blei, editors, Proceedings of the 32nd International Conference on Machine Learning, volume 37 of Proceedings of Machine Learning Research, pages 957–966, Lille, France, 07–09 Jul 2015. PMLR.

[LH03] Chin-Yew Lin and Eduard Hovy. Automatic evaluation of summaries using n-gram co-occurrence statistics. In Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, pages 150–157, 2003.

[NCL18] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. ArXiv, abs/1808.08745, 2018.

[PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[RWC+19] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[ZKW+20] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020.