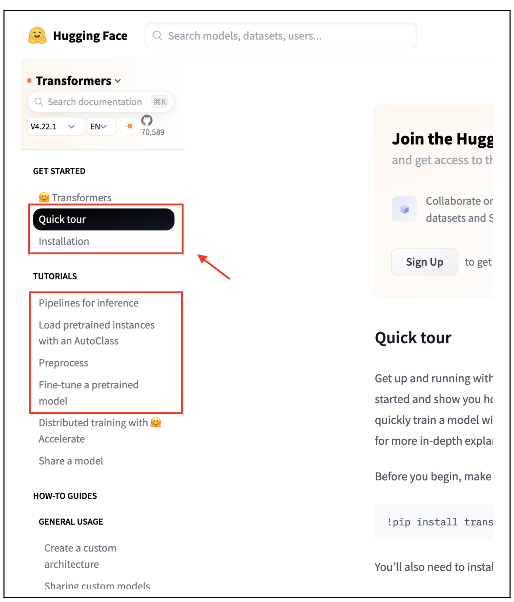
The lead TAs for this assignment are Shirley Hayati (hayat023@umn.edu). Please communicate with the lead TAs via Slack or during office hours. **All questions MUST be discussed in the homework channel (i.e., #HW2).** Questions through emails, Direct Messages, and other channels will not be answered.

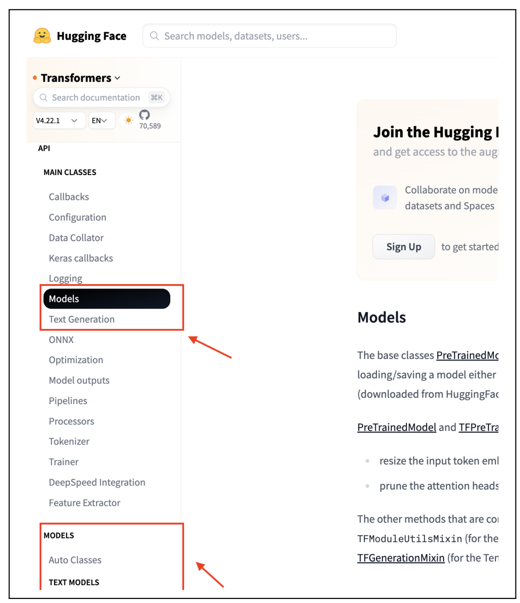
Prerequisite. This assignment assumes that you have programming experience with PyTorch, Jupyter Notebooks and Google Colab[1]. Going forward, we will be using Colab Pro. You can find information at these instructions regarding subscriptions and how to get reimbursement at the end of the semester. You have learned how to build a text classifier in the lectures. You may also learn the basic concept of pretraining and finetuning from the tutorials presented earlier by the TAs.

Overview. As part of this assignment, you will build your own text classifier using the HuggingFace library. By fine-tuning the pre-trained model implemented in HuggingFace model libraries on your dataset, you will replicate the high-performing text classifier and check how far you can reach out to the best performances by the state-of-the-art models on the Papers-with-Code leaderboard.

Academic Honesty Policy. Make sure to (a) cite any tools or papers you reference/use, and (b) credit anyone youve discussed the assignment with. It is considered academic dishonesty if you reference any tool/paper/person without proper attribution.

Step 1: Getting used to HuggingFace library





Follow the basic instructions on inference, model loading, preprocessing, and fine-tuning in the HuggingFace tutorial: https://huggingface.co/docs/transformers/quicktour. It is highly recommended that you install the library and run the commands in the tutorial in your Google Colab[2] or your local machine using Jupyter Notebook.[3] In the tutorial document, you can find some default classes implemented by HuggingFace by scrolling down the left menu. You must first understand these abstract classes in order to run their models. A tutorial on fine-tuning HuggingFace's pre-trained model can be found here: tutorial.

[1] **Note:** When using Google Colab, please avoid selecting A100 GPUs as they rapidly consume your allotted compute units. Instead, opt for L4 or T4 GPUs for a more efficient use of resources.

[2] https://colab.research.google.com/

[3] https://jupyter.org/

Step 2: Choose a Task and Dataset

You can now select a task and dataset from the list in Table 1. Please contact the lead TA a week before the deadline if you wish to choose another task and/or dataset. It is recommended that you read the original paper that describes the dataset first. After that, you can download the raw dataset or load it from the pre-formatted HuggingFace dataset. Below are links to the Papers-with-Code leaderboard, original paper, raw dataset, and HuggingFace dataset. Check what model has currently the best score on your dataset in the leaderboard.

Table 1: List of classification tasks and dataset. The following list contains links to the PapersWithCode leaderboard, HuggingFace formatted dataset, and the original paper. Question answering (QA) tasks could be viewed as a classification task that predicts the appropriate start and end position of your answer span given a question. Natural Language Inference (NLI) and Human-vs-GPT language detection tasks could be viewed as classification tasks as well, as they predict the final labels (e.g., entail/contradict/neutral, human/GPT) given a pair of two texts. Some datasets may be too large to train efficiently. In this case, it is fine to sample a sensible amount for each class label and include the details in the report.

Tasks	**Labels**	**Size**	**Datasets**
Sentiment classification	2 (Positive, Negative)	70K	SST2 (leaderboard, HF dataset, paper)
Sentiment classification	3 (Positive, Negative, neutral)	122K	DynaSent (leaderboard, HF dataset, paper)
Multi-style Classification	2-10	2K-260K	xSLUE (dataset, paper)
Politeness classification	Range (Very Impolite to Very Polite)	11K	StanfordPoliteness (dataset, paper)
Paper acceptance	2 (Accept, Reject)	14K	PeerRead (dataset, HF dataset, paper)
Social classification	Offensive, intent, lewd, target/in-group	147K	Social Bias Inference (SBIC) (leaderboard, HF dataset, paper)
Hate Speech	3 (offensive, hatespeech, neither)	24K	Hate Speech Detection (HSD) (leaderboard, HF dataset, paper)
Bias Classification	Biased, Non-Biased	50k	BEADs (dataset, paper)
Bias Aspect Categorization	Gender, Race, Age, Religion, Mental Health, Disability, Nationality, Socioeconomic Status	50k	BEADs (dataset, paper)
Natural Language Inference	3 (neutral, contradict, entail)	570K	SNLI (leaderboard, HF dataset, paper)
Natural Language Inference	3 (neutral, contradict, entail)	432K	MNLI (leaderboard, HF dataset, paper)
Textual Similarity	2 (similar, not similar)	5.8K	MRPC (leaderboard, HF dataset, paper)
Commonsense Reasoning	multi-choice QA	3K-43K	Winograd Challenge (leaderboard, HF dataset, paper)
Commonsense Reasoning	multi-choice QA	12K	CommonsenseQA (leaderboard, HF dataset, paper)
Question Answering	Span prediction	142K	SQuAD 2.0 (leaderboard, HF dataset, paper
Visual Question Answering	Span prediction	22M	GQA (leaderboard, HF dataset, paper)
Semantic Evaluation (SemEval)*	-	-	SemEval (2022), SemEval (2023), SemEval (2024), SemEval (2025), Other SemEval tasks in HF dataset
Human-vs-AI text detection	2 (human, AI)	436K	DeepfakeTextDetect (HF dataset, paper)

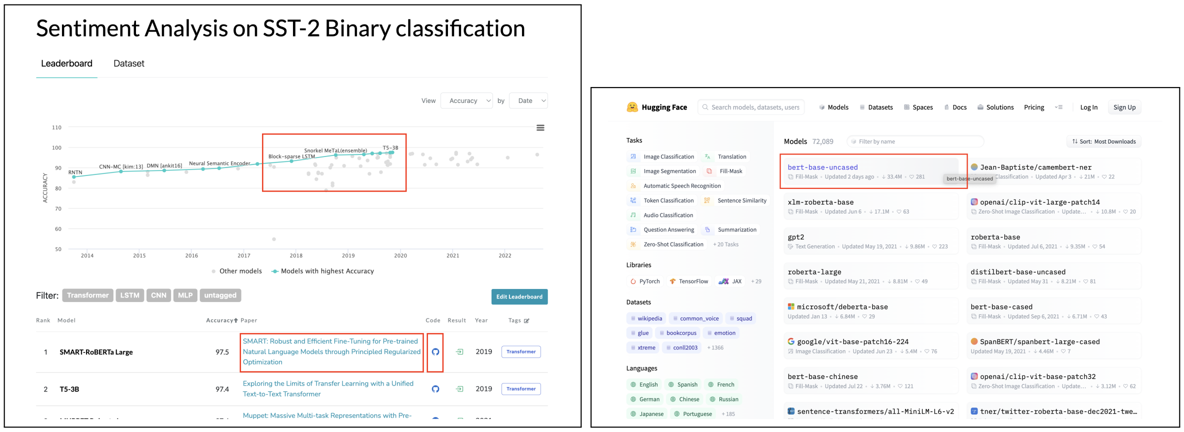


Figure 1: The Papers-with-Code leaderboard of the dataset SST-2 for binary classification task (left) and HuggingFace's model cards on pre-trained language models, like `bert-base-uncased` (right)

Step 3: Choose a Model and Replicate it

The next step is to choose a model to replicate. You can (1) choose one of HuggingFace's pre-trained models, such as BERT, GPT2, or RoBERTa[4] (See Figure 1 (right)), and (2) fine-tune it. You have to "train" the model, by writing your own training script for fine-tuning the pre-trained model on your target dataset. Note: you are **not** allowed to use the default `Trainer` function in HuggingFace like below.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_imdb["train"],
    eval_dataset=tokenized_imdb["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,

)

trainer.train()
```

Instead, you need to implement your own `Trainer` like `CustomTrainer` and then inherit the default `Trainer` except for `_inner_training_loop` function. You can check how the default `_inner_training_loop` is implemented. In your customized `_inner_training_loop` function, you can just copy the code in the default `_inner_training_loop` function, but please understand how your training process is implemented as discussed in class, such as multiple epochs of training, forward and backward propagation, gradient update methods, gradient clipping, and parameter updating.

From the TA's HuggingFace tutorial, here is an example CustomTrainer.

```
class CustomTrainer(Trainer):
    def _inner_training_loop(
        self, batch_size=None, args=None, resume_from_checkpoint=None, trial=None,
                                              ignore_keys_for_eval=None
    ):
        number_of_epochs = args.num_train_epochs
        start = time.time()
        train_loss=[]
        train_acc=[]
```

[4] I understand you have no idea what BERT/GPT is. We will cover them soon in the class so stay tuned.

```
        eval_acc=[]

        criterion = torch.nn.CrossEntropyLoss().to(device)
        self.optimizer = torch.optim.Adam(model.parameters(), lr=args.learning_rate)
        self.scheduler = torch.optim.lr_scheduler.StepLR(self.optimizer, 1, gamma=0.9)

        train_dataloader = self.get_train_dataloader()
        eval_dataloader = self.get_eval_dataloader()

        max_steps = math.ceil(args.num_train_epochs * len(train_dataloader))

        for epoch in range(number_of_epochs):
            train_loss_per_epoch = 0
            train_acc_per_epoch = 0
            with tqdm(train_dataloader, unit="batch") as training_epoch:
                training_epoch.set_description(f"Training Epoch {epoch}")
                for step, inputs in enumerate(training_epoch):
                    inputs = inputs.to(device)
                    labels = inputs['labels']

                    # forward pass
                    self.optimizer.zero_grad()
                    # output = ... # TODO Implement by yourself

                    # get the loss
                    # loss = criterion((output[?], labels) # TODO Implement by
                                                                yourself
                    train_loss_per_epoch += loss.item()

                    #calculate gradients
                    loss.backward()
                    #update weights
                    self.optimizer.step()
                    train_acc_per_epoch += (output['logits'].argmax(1) == labels).sum
                                                                ().item()
            # adjust the learning rate
            self.scheduler.step()
            train_loss_per_epoch /= len(train_dataloader)
            train_acc_per_epoch /= (len(train_dataloader)*batch_size)


            eval_loss_per_epoch = 0
            eval_acc_per_epoch = 0
            with tqdm(eval_dataloader, unit="batch") as eval_epoch:
                eval_epoch.set_description(f"Evaluation Epoch {epoch}")
                # ... TODO Implement by yourself
            eval_loss_per_epoch /= (len(eval_dataloader))
            eval_acc_per_epoch /= (len(eval_dataloader)*batch_size)

            print(f'\tTrain Loss: {train_loss_per_epoch:.3f} | Train Acc: {
                                                    train_acc_per_epoch*100:.2f}%')
            print(f'\tEval Loss: {eval_loss_per_epoch:.3f} | Eval Acc: {
                                                    eval_acc_per_epoch*100:.2f}%')

        print(f'Time: {(time.time()-start)/60:.3f} minutes')
```

As part of your assignment or class project, you may have to change some parts of this training function or modify outputs from forward propagation. Your submitted code should include this customized `CustomTrainer` with the copied (or modified) version of `_inner_training_loop` function.

Step 4: Analyze your classifier's training and evaluate it on test set

Read carefully below what experiments and additional analyses should be included in your report. Missing items will result in point deductions.

- Description of the task and models with references to the original papers and model cards/repository.
- The kind of hardware you run your model on.
- How do you ensure your model has been trained correctly? Do you have a learning curve graph of your training losses from forward propagation? What does it look like? We highly recommend you use a tracking tool such as "Weights & Biases" (W&B). With a few lines of code, this lets you automatically track the progress of training and plot learning curves. You can then export the plots and add them to your report.
- Evaluation metrics used in your experiment.
- Test set performance and comparison with score reported in original paper AND leaderboard. A justification is needed if it differs from the reported scores.
- Training and inference time.
- Hyperparameters used in your experiment (e.g., number of epochs, learning parameter, dropout rate, hidden size of your model) and other details.
- Hypothesize what kinds of samples you might think your model would struggle with.
- Potential modeling or representation ideas to improve the errors.
- (optional) What was the most challenging part of this homework?

Step 5: Comparison with LLM predictions

In this step, you will compare your model's predictions with predictions from a large language model, like Llama3 or Qwen2.5. In your spreadsheet in the previous step, you can add one more column at the end, and include LLM predictions. In your report, you can report accuracies of LLM prediction with the ground-truth answers as well difference between model's predictions and LLM predictions with justification.

To complete this step, refer to the tutorial on vLLM. This step may also be completed using Huggingface alone, if you prefer this instead of vLLM. You will need to choose which model you would like to use. Given the memory constraints within Colab Pro, we recommend using a model which is 8B params or smaller, such as Llama 3.2-3B, or Qwen2.5-1.5B (NOTE: If you use llama models you will need to apply for a permission key from meta from within huggingface. You can read more here to find out how to get and use access tokens).

In order to get accurate predictions, you have to design your prompt appropriately. For instance, you should provide a description of your classification task, expected labels to predicted, and then input instances:

```
Task: Given text, predict the sentiment of the text.
The predicted label should be either Positive, Neutral, or Negative.

Input: The weather is great
Output:
```

Sentence label (gold/pred) (Correct/Wrong)	Result (P/R/F)	Example (gold and predicted token labels)	Error types	Cause (Term)	Cause (Definition)	Surface/parsing patterns	Potential solutions	Difficulty
none / definition	{'p': 0.2, 'r': 0.134, 'f': 0.161}	We thus utilized Reuters news articles referred to as ' Reuters-21578 , ' which has been widely used in text classification v . We used a prepared SAn exception is the method proposed in (McCallure and Nigam , 1999) , which , instead of labeled texts , uses unlabeled texts , pre - determined categories , and keywords defined by humans for each category . We thus utilized Reuters news articles referred to as ' Reuters-21578 , ' which has been widely used in text classification v . We used a prepared SAn exception is the method proposed in (McCallure and Nigam , 1999) , which , instead of labeled texts , uses unlabeled texts , pre - determined categories , and keywords defined by humans for each category .	False Positive	Over-generalization: technical term bias	Over-generalization (which has | is the method proposed | uses): surface pattern bias		Heuristics: filter out multi-term/definition cases	

Error types for term prediction	#	%
Over-generalization: technical term bias	26	28.9
Missing definition	13	14.4
Incomplete phrase	11	12.2
Wrong data preprocessing	3	3.3
Over-generalization (is a): surface pattern bias	3	3.3
Over-generalization (is): surface pattern bias	2	2.2
Over-generalization (has a): surface pattern bias	1	1.1
Over-generalization (which has | is the method proposed | uses): surface pattern bias	1	1.1

Potential solutions to fix errors	#	%
Heuristics: filter out term/definition only cases	22	20.8
Parse features	19	17.9
Rule: surface patterns	12	11.3
Better encoder	11	10.4
UNK represetnation	6	5.7
Pattern generalization	6	5.7
Annotation: definition vs description	4	3.8
Entity detection	4	3.8
Heuristics: filter out multi-term/definition cases	3	2.8
? (extremely difficult)	3	2.8
POS features	3	2.8
Heuristics: filter out non-adjacent term:definition pairs	2	1.9

Figure 2: Example error annotations (top), example error causes (bottom left), and fixes (bottom right) from the test set for the term-definition detection task [KHS+20]. The ground-truth test set has no term and definition annotated, while the model predicts Reuters-21579 and SAn exception as terms, and been widely used in text classification v. and unlabeled texts pre-determined categories as definitions.

Step 6: Annotate error types and ideas to fix them

Run your model on the test set and collect at least 20[5] incorrectly predicted samples from the test set. If your task has a specific test set from the benchmark, you can use them. You now create a Google spreadsheet and store each error sample in each row with the following information in separate columns:

- Input text
- Ground-truth label (from the original data)
- Predicted label with a confidence score (i.e., softmax output from your classifier with respect to the ground-truth label)

Go through each row and manually label them in the following categories:

- Types of errors, e.g., false positive or false negative
- Types or causes, e.g., over-generalization, surface pattern bias
- Potential solutions to fix the cause, e.g., more training samples[6], linguistic features, some rules
- Rank your annotations by frequencies and show two tables of distributions of error types and solutions

If you report the confidence score of the predicted labels (the last Linear layer's softmax score) on the samples in another column, you will receive a bonus point.

[5] If the errors in set samples are less than 20, it is fine to report errors less than 20.
[6] Simply labeling most examples with "more training data" without any justification will lose points

Figure 2 shows example error annotations with their causes and fixes for the term-definition detection task.[7] Please note that these types of causes and fixes are specific to the term-definition detection task, so they are not applicable to your task. You should figure out your own error types, causes, and fixes.

Visualize errors and perform qualitative analysis (Optional for Bonus points)

In this step, you will visualize the errors with other correctly predicted samples (randomly chosen up to 500) in a 2-dimensional semantic space and explore an overall view of how they are projected. Figure 3 shows an example visualization on QNLI dataset.

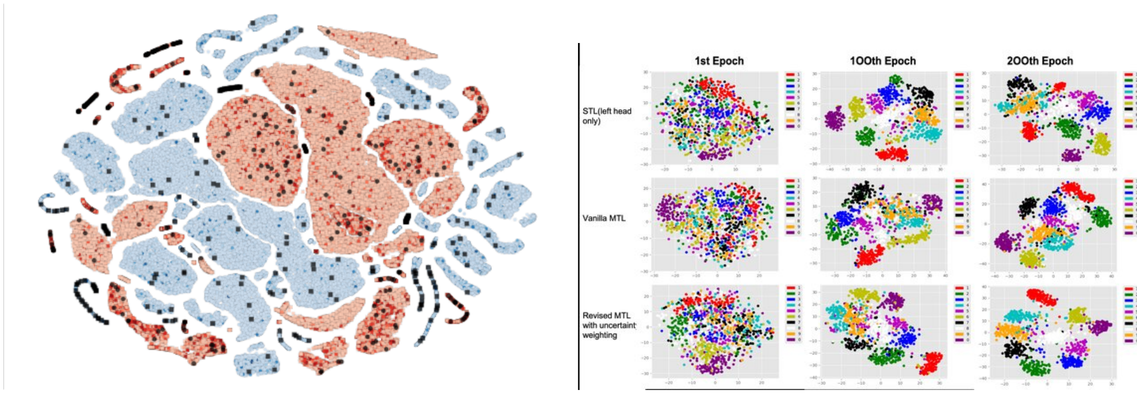


Figure 3: Example t-SNE projection of QNLI dataset (left) or multi-task learning (right) [GLS+19]. For QNLI dataset, red square and blue circles indicate the QNLI labels whether or not the question is answerable. Incorrectly predicted samples (black) are almost randomly located in the classifiers embedding space. For multi-task learning, as the training epochs increase (1st, 100th, and 200th), the clear separation of different label space has been observed.

Semantic space: First, you can take vector representations of correct and incorrect samples from the classifiers output hidden state (HuggingFace's model output class). Then, you project them onto reduced dimensions (i.e., 768 dimension → 2 dimensions) using dimension reduction methods like PCA (code) or t-SNE [vdMH08] (code), following tutorials like this. Finally you can visualize the 2-dimensional scatter plot in your report with the observation you found.

When you visualize the scatter plot, please consider the following tips:

- Use Matplotlib (link) or other visualization library for visualization.
- Choose different colors and/or shapes for correct and incorrect samples to distinguish them. (red for Positive labels, blue for Negative labels, black for incorrectly predicted labels)
- Use a legend to indicate the type of items.
- Display the model's confidence in each sample as transparency using *alpha* variable (example).

Parameter-Efficient Fine-Tuning (PEFT) with LoRA on LLaMA-3 (Optional for Bonus points)

In this bonus section, you will use the **Parameter-Efficient Fine-Tuning (PEFT)** library for applying the **LoRA** technique on the **LLaMA-3-8B-Instruct** model. You will measure performance improvements and analyze memory and training efficiency. Make sure to install the following core libraries and load the model:

[7] The task that detects spans of scientific terms and definitions defined in text

- Transformers
- PEFT GitHub
- LLaMA-3-8B-Instruct

For dataset, you must convert it to an **instruction-style dataset** with `instruction`, `input`, and `output` fields, such as from the following site. You must use the same dataset for your non-Lora classification model.

You can read more information about Lora and its tutorial.

Here is a skeleton code for fine-tuning LLM with Lora. Note that this code is incomplete and only serves as an example only.

```
# Install required packages if not already
# !pip install transformers datasets peft accelerate

from transformers import AutoTokenizer, AutoModelForCausalLM, Trainer,
                                            TrainingArguments, DataCollatorForSeq2Seq
from datasets import load_dataset
from peft import LoraConfig, get_peft_model

# ------------------------------
# 1. Load instruction dataset
# ------------------------------
# Example: dataset with 'instruction' and 'output' columns
dataset = load_dataset("json", data_files={"train": "train.json", "validation": "val.
                                            json"})

# ------------------------------
# 2. Load tokenizer and model
# ------------------------------
model_name =  # Replace with your base model
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token  # ensure padding token

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map="auto",  # uses all GPUs if available
    load_in_8bit=True,  # optional for memory efficiency
)

# ------------------------------
# 3. Tokenize dataset
# ------------------------------
def tokenize_fn(example):
    inputs = "Instruction: " + example["instruction"] + "\nResponse: "
    targets = example["output"]
    full_text = inputs + targets
    tokenized = tokenizer(full_text, truncation=True, max_length=512)
    return tokenized

tokenized_dataset = dataset.map(tokenize_fn, batched=False)

# ------------------------------
# 4. Setup LoRA
# ------------------------------
lora_config = LoraConfig()#insert config

model = get_peft_model(model, lora_config)
```

Compare the F1 scores and accuracy for predictions by the LoRA-tuned model, the original Llama-3-8B-Instruct model, and your fine-tuned model from Step 1-5.

In your report, provide a small table or chart comparing baseline vs. LoRA-tuned performance and write discussion of the results.

Deliverables

Please upload your code, google sheet link, and report to Canvas by Sep 21, 11:59pm.

Code: You should submit a **zipped file** containing - your training/inference scripts or a link to your GitHub repository - Your script for comparing with GPT predictions.

Report and Spreadsheet: Maximum **six pages PDF** and other supplementary documents such as spreadsheets for error analysis. The page limit of homework doesn't include references and an appendix with additional information. For report, you must use this LaTex template (link). In case you haven't used LaTex for scientific writing, this is a great opportunity to learn how scientists and researchers write their manuscripts using this typesetting tool called LaTex. Here is a tutorial for LaTex with Overleaf.

Formatting convention: All your files submitted should follow this naming convention: **CSCI5541-S25-HW2-{First Name}-{Last Name}.{zip,pdf,csv}**.

Rubric (20 points + 5 bonus points)

- Code looks good, i.e., each cell in the Jupyter Notebook runs without error and outputs intended results. (+3)
- Description of the task, dataset, models, and hardware used (+1)
- Includes appropriate references (+1)
- Explains how they checked their model was trained correctly using learning curve graphs or other appropriate information (+2)
- Specifies evaluation metrics used in the experiment (+1)
- Discusses test set performance and comparison with score reported in original paper or leaderboard. Includes justification if it differs from the reported scores. (+2)
- Includes training and inference time (+1)
- Includes hyperparameters used in the experiment (+1)
- Hypothesis of model performance and/or some kind of discussion about what they found in their incorrectly labeled samples (+1)
- Minimum of ten incorrectly predicted test samples with their ground-truth labels (+1)
- Discusses potential modeling or representation ideas to improve the errors (+1)
- Annotation of error types and potential fixes (+2)
- Comparison of incorrectly predicted samples with LLM (+1)
- Follow the suggested formats (files, report format, etc) (+2)

Bonus:

- Model's confidence score for the incorrectly predicted samples in the test set (Bonus +1)
- Error visualizations and qualitative analysis (Bonus +2)
- Instruction-tuning LLM with Lora (Bonus +2)

References

[GLS+19] Ting Gong, Tyler Lee, Cory Stephenson, Venkata Renduchintala, Suchismita Padhy, Anthony Ndirango, Gokce Keskin, and Oguz Elibol. A comparison of loss weighting strategies for multi task learning in deep neural networks. IEEE Access, PP:1–1, 09 2019.

[KHS+20] Dongyeop Kang, Andrew Head, Risham Sidhu, Kyle Lo, Daniel Weld, and Marti A. Hearst. Document-level definition detection in scholarly documents: Existing models, error analyses, and future directions. In Muthu Kumar Chandrasekaran, Anita de Waard, Guy Feigenblat, Dayne Freitag, Tirthankar Ghosal, Eduard Hovy, Petr Knoth, David Konopnicki, Philipp Mayr, Robert M. Patton, and Michal Shmueli-Scheuer, editors, Proceedings of the First Workshop on Scholarly Document Processing, pages 196–206, Online, November 2020. Association for Computational Linguistics.

[vdMH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. Journal of Machine Learning Research, 9:2579–2605, 2008.