CS 505 Final Report

Project Tittle: Grammar Ninja

Abstract:

Improving one’s English language writing is a significant challenge without access to proficient teachers that can provide valuable feedback. Given the recent rapid acceleration in generative models ability to understand language, we aim to develop a model/fine-tune a model to provide an interface that will generate feedback given text as an input. Our goal is provide quantitative benchmarks for language proficiency in six different areas: cohesion, syntax, vocabulary, phraseology, grammar, and conventions in a provided writing. Additionally, we also generate feedback at the inference layers to

provide concrete feedback as to how the input text can be improved. To conclude, our project aims to apply the concepts learned in class to a real-world challenge, by providing a interface to acquire feedback on English writing. By focusing on key areas of language skills and providing model generated actionable feedback, we hope to contribute a somewhat practical tool.

Problem Definition:

As we mentioned in our milestone report, this project aims to develop a Natural Language Processing (NLP) based system that can automatically evaluate and provide feedback on student argumentative essays. The system will focus on several key aspects of writing, including the effectiveness of arguments, grammar, use of evidence, syntax, and tone. The feedback can be either quantitative, in the form of scores in various categories, or qualitative, as generated English feedback that offers specific guidance and suggestions for improvement.

To this end, we have decided to split our problem into three separate parts - predicting feedback scores based on the writing, identifying the different parts of argumentative writing (think Parts-Of-Speech tagging but for sentences as opposed to words), and finally constructing a generative model to produce the corrected version of a given input sentence.

Approaches Taken:

model 1: Feedback Scores

As mentioned in the milestone report, this was dataset we chose to use to identify scoring the dataset on the following features - 'cohesion', 'syntax', 'vocabulary', 'phraseology', 'grammar', 'conventions’.

Feedback Prize - English Language Learning

Evaluating language knowledge of ELL students from grades 8-12

https://www.kaggle.com/competitions/feedback-prize-english-language-learning

First, I explored the data:

Given some input data (X), we predict the 6 scores (y):

	text_id	full_text	cohesion	syntax	vocabulary	phraseology	grammar	conventions
0	0016926B079C	I think that students would benefit from learn...	3.5	3.5	3.0	3.0	4.0	3.0
1	0022683E9EA5	When a problem is a change you have to let it ...	2.5	2.5	3.0	2.0	2.0	2.5

We want to predict regression scores around these bucket classifications:

```
# how many unique values does each column have?
df.nunique()
```

✓ 0.0s

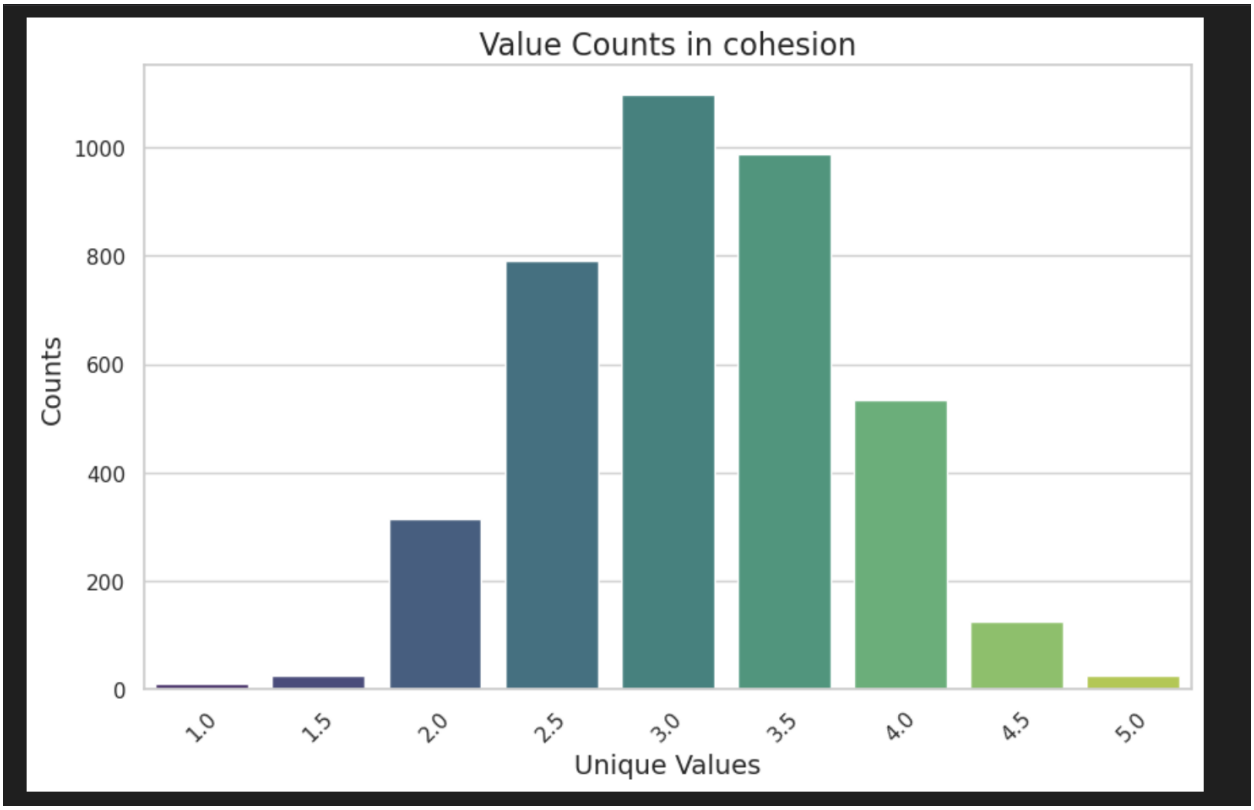
```
text_id        3911
full_text      3911
cohesion          9
syntax            9
vocabulary        9
phraseology       9
grammar           9
conventions       9
dtype: int64
```

Its 6 target features and input_txt.

We noticed a few things, most importantly that the target classes are imbalanced:

```
cohesion
3.0    1096
3.5     988
2.5     790
4.0     534
2.0     315
4.5     125
1.5      27
5.0      26
1.0      10
Name: count, dtype: int64
syntax
3.0    1250
3.5     867
2.5     839
2.0     410
4.0     388
4.5     100
1.5      29
5.0      17
1.0      11
Name: count, dtype: int64
vocabulary
3.0    1503
3.5    1007
...
5.0      25
1.5      20
1.0      15
Name: count, dtype: int64
```

For example, `cohesion`:

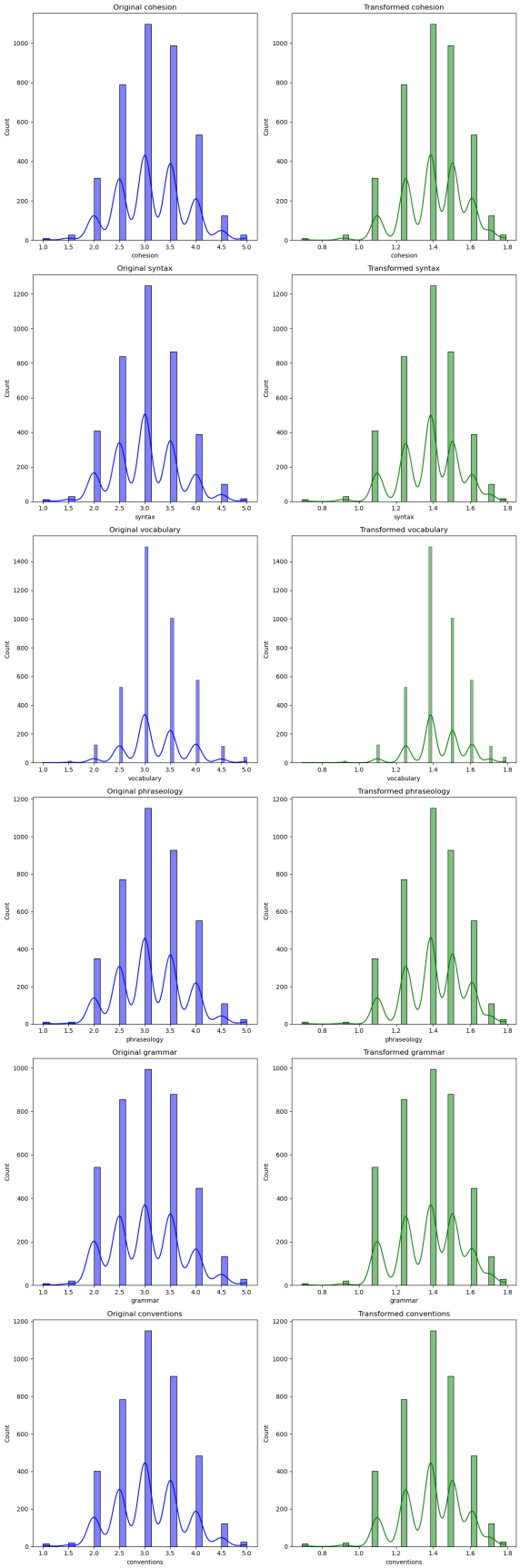


So, we now had to make a decision regarding how to re-balance the dataset, so the model can learn equally well on less represented data as well. Here, we considered some strategies:

```
---------------- FROM THE .ipynb, full source in Repo ----------

## Re-balancing using Target Transformation

- Transform the target variable to make the dataset more uniforr

- Reduces Skewness: Models typically assume that the target vari

- Improves Model Sensitivity to Minority Classes: In a skewed di

- Enhances Model Interpretability: Transformations can make the

Common target transformations include:
```

```
- Log Transformation: Useful for right-skewed distributions. How

- Square Root Transformation: Less aggressive than log transforr

- Box-Cox Transformation: A more general form of transformation

- Yeo-Johnson Transformation: Similar to Box-Cox but can be appl

Alternate Strategies:

- Under-sampling; Delete some data from rows of data from the ma
    - Limitation: This is hard to use when you don’t have a subs


- Copy rows of data resulting minority labels. In this case, cop
    - Limitation: copying current data and you don’t really pres


- SMOTE - Synthetic Minority Oversampling Technique
    - Synthetically generate new data based on implications of c

    - Limitation: If two different class labels have common neig
```

After some experimentation on small sample sizes with SMOTE, Box-Cox Transformation and Log Transformation we came to the conclusion that target rebalancing with logs would be the most appropriate to reconstruct an appropriate distribution.


Original cohesion
Transformed cohesion
Original syntax
Transformed syntax
Original vocabulary
Transformed vocabulary
Original phraseology
Transformed phraseology
Original grammar
Transformed grammar
Original conventions
Transformed conventions
Count
cohesion
syntax
vocabulary
phraseology
grammar
conventions

We have transformed the input data:

```
df_transformed['cohesion'] = np.log1p(df['cohesion'])
```

To compute the efficacy of the transformation, we compute the Shapiro-Wilk Test, Skewness, Kurtosis:

```
riginal cohesion - Shapiro-Wilk: p=0.0000, Skewness: 0.0353, Kurtosis: -0.1882
ransformed cohesion - Shapiro-Wilk: p=0.0000, Skewness: -0.4510, Kurtosis: 0.3881
riginal syntax - Shapiro-Wilk: p=0.0000, Skewness: 0.1256, Kurtosis: -0.0580
ransformed syntax - Shapiro-Wilk: p=0.0000, Skewness: -0.3752, Kurtosis: 0.3759
projectnb/cs505ws/projects/grammar_ninja_viint/envs/fb_scores/lib/python3.10/site-packages/sea
 with pd.option_context('mode.use_inf_as_na', True):
projectnb/cs505ws/projects/grammar_ninja_viint/envs/fb_scores/lib/python3.10/site-packages/sea
 with pd.option_context('mode.use_inf_as_na', True):
projectnb/cs505ws/projects/grammar_ninja_viint/envs/fb_scores/lib/python3.10/site-packages/sea
 with pd.option_context('mode.use_inf_as_na', True):
projectnb/cs505ws/projects/grammar_ninja_viint/envs/fb_scores/lib/python3.10/site-packages/sea
 with pd.option_context('mode.use_inf_as_na', True):
riginal vocabulary - Shapiro-Wilk: p=0.0000, Skewness: 0.2246, Kurtosis: 0.3588
ransformed vocabulary - Shapiro-Wilk: p=0.0000, Skewness: -0.2940, Kurtosis: 0.8074
riginal phraseology - Shapiro-Wilk: p=0.0000, Skewness: 0.0670, Kurtosis: -0.2391
ransformed phraseology - Shapiro-Wilk: p=0.0000, Skewness: -0.3963, Kurtosis: 0.2580
projectnb/cs505ws/projects/grammar_ninja_viint/envs/fb_scores/lib/python3.10/site-packages/sea
 with pd.option_context('mode.use_inf_as_na', True):
projectnb/cs505ws/projects/grammar_ninja_viint/envs/fb_scores/lib/python3.10/site-packages/sea
 with pd.option_context('mode.use_inf_as_na', True):
projectnb/cs505ws/projects/grammar_ninja_viint/envs/fb_scores/lib/python3.10/site-packages/sea
 with pd.option_context('mode.use_inf_as_na', True):
riginal grammar - Shapiro-Wilk: p=0.0000, Skewness: 0.2015, Kurtosis: -0.4158
ransformed grammar - Shapiro-Wilk: p=0.0000, Skewness: -0.2101, Kurtosis: -0.3099
riginal conventions - Shapiro-Wilk: p=0.0000, Skewness: 0.0769, Kurtosis: -0.1664
ransformed conventions - Shapiro-Wilk: p=0.0000, Skewness: -0.4272, Kurtosis: 0.4098
```

Our transformation has correctly changed the distributions' characteristics, often reducing right skewness but introducing left skewness and altering the tailedness (kurtosis). After these changes, the data now more similarly conforms to a normal distribution according to the Shapiro-Wilk test.

After this, we decided to choose a pre-trained generative model that understands semantics of language, and then further fine-tuning with our dataset. And additionally from above, we have, the following constraint - we want a model that is robust to non-normality.

After some research and considering a few generative models GPT-3.5/4, Llama, we decided to use BERT instead. This is because bert has both a cased and uncased version. So, we can experiment in more ways with tokenization.

```
from torch import nn
class BERT_Classifier(nn.Module):
    def __init__(self):
        super(BERT_Classifier, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-cased')
        self.drop = nn.Dropout(0.0)
        self.out = nn.Linear(768, 6)


    def forward (self, ids, mask, token_type_ids):
        _, pooled_output = self.bert(ids, attention_mask=mask, token_type_ids=token_type_ids, return_dict=False)
        output_2 = self.drop(pooled_output)
        output = self.out(output_2)
        return output

model = BERT_Classifier()
model.to(device)
```

- We are using the BERT model. We then added a `Dropout` and `Linear Layer` as well. We add these layers to ensure the model is able to regularize and classify the data better.
- In the forward loop, there are 2 outputs from the `BERT` model layer.
- The output of this, `pooled_output` is passed through the `Dropout` layer and then the `Linear` layer.
- We set the number of dimesnions in the `Linear` layer to be equal to the number of classes we have in the dataset.

This gives is the following output:

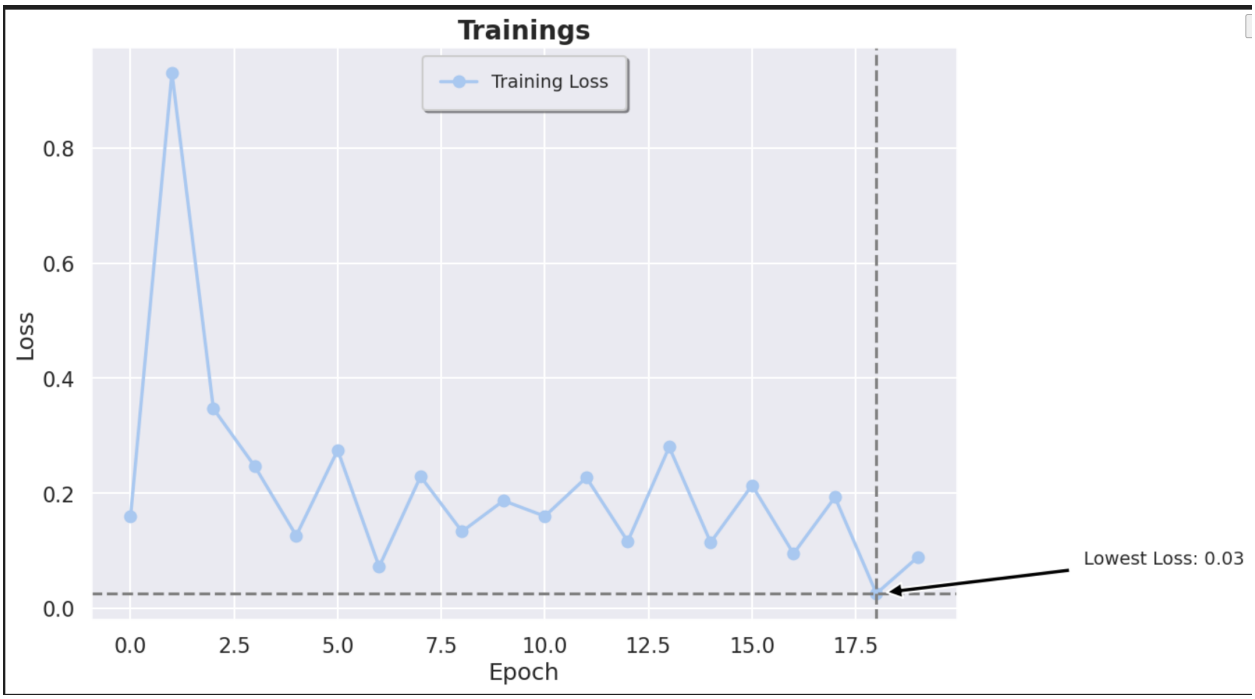
```
BERT_Classifier(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(28996, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_aff
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
```

```
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768,
              (key): Linear(in_features=768, out_features=768, I
              (value): Linear(in_features=768, out_features=768,
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768,
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementw
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072,
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768,
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwis
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=Tr
      (activation): Tanh()
    )
  )
  (drop): Dropout(p=0.0, inplace=False)
  (out): Linear(in_features=768, out_features=6, bias=True)
)
```

After reading latest consensus, experiments on hugging face and local experimentaton, we came to the conclusion that the following hyper-parameters are best for this model and data:

```
MAX_LEN = 200
TRAIN_BATCH_SIZE = 4
VALID_BATCH_SIZE = 4
EPOCHS = 45
LEARNING_RATE = 2e-05
tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
```

Now, we train on the full dataset:



model 2: Part of Writing Tagging

As mentioned in the milestone report this model automatically segments texts and classifies a range of argumentative and rhetorical elements. The primary elements identified by this model include:

- Position: Identifying the main stance or thesis of the essay.
- Lead: Recognizing introductory statements or premises.

- Rebuttal: Detecting responses to counterarguments.
- Claim: Identifying specific assertions or points made in support of the position.
- Evidence: Recognizing data, quotations, or other factual information supporting claims.
- Counterclaim: Detecting opposing viewpoints or arguments.
- Concluding Statement: Identifying concluding remarks or summaries of the argument.

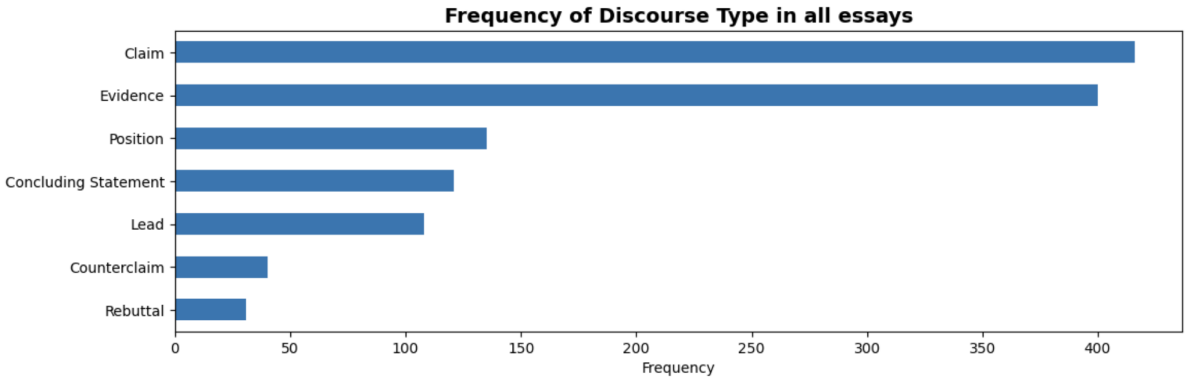
The training data used is taken from https://www.kaggle.com/competitions/feedback-prize-2021/data?select=train.csv

The first step was to explore the initial two datasets. We found the smaller dataset mostly had duplicates of the first one, so we continued with just the larger dataset.

	id	discourse_id	discourse_start	discourse_end	discourse_text	discourse_type	discourse_type_num	predictionstring
0	423A1CA112E2	1.622628e+12	8.0	229.0	Modern humans today are always on their phone....	Lead	Lead 1	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 1...
1	423A1CA112E2	1.622628e+12	230.0	312.0	They are some really bad consequences when stu...	Position	Position 1	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
2	423A1CA112E2	1.622628e+12	313.0	401.0	Some certain areas in the United States ban ph...	Evidence	Evidence 1	60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
3	423A1CA112E2	1.622628e+12	402.0	758.0	When people have phones, they know about certa...	Evidence	Evidence 2	76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 9...
4	423A1CA112E2	1.622628e+12	759.0	886.0	Driving is one of the way how to get around. P...	Claim	Claim 1	139 140 141 142 143 144 145 146 147 148 149 15...

The dataset was structured as shown above. There were overall about 15k documents/texts in the training dataset (which we also had access to). The CSV file shown above had all these texts split into the respective rhetorical parts.

```
labels: {'Concluding Statement', 'Evidence', 'Position', 'Claim', 'Rebuttal', 'Lead', 'Counterclaim'}
Number of texts : 15594
```



This problem could've been treated as sequence classification problem or a token classification problem. We decided to structure it as a token classification similar to the POS tagging we did in HW05.

- Using the id, discourse type and prediction string columns we concatenated the texts on their id, each word was tagged with a discourse type.

	id	essay_text	tags
0	0000D23A521A	[Some, people, belive, that, the, so, called, ...	[Position, Position, Position, Position, Posit...
1	00066EA9880D	[Driverless, cars, are, exaclty, what, you, wo...	[Lead, Lead, Lead, Lead, Lead, Lead, Lead, Lea...
2	000E6DE9E817	[I, am, arguing, against, the, policy, change,...	[Position, Position, Position, Position, Posit...
3	001552828BD0	[Would, you, be, able, to, give, your, car, up...	[Lead, Lead, Lead, Lead, Lead, Lead, Lead, Lea...
4	0016926B079C	[I, think, that, students, would, benefit, fro...	[Position, Position, Position, Position, Posit...

- Now this text had to be converted to tokens suitable for our model. Since the first layer in our model was going to be a base pre-trained Longformer transformer, we went with the following tokenizer.

```
tokenizer = AutoTokenizer.from_pretrained('allenai/longformer-base-4096')
```

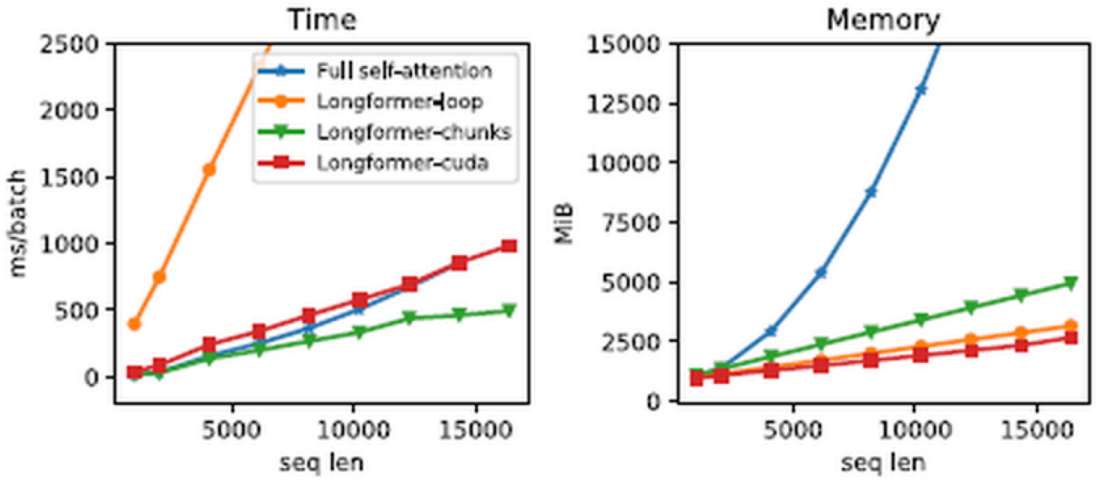
After converting the text to tokens, we needed to carefully align the tags with the tokenized text since some words would be split into multiple tokens.

	id	tokens	aligned_tags
0	0000D23A521A	[Some, Ġpeople, Ġbel, ive, Ġthat, Ġthe, Ġso, Ġ...	[Position, Position, Position, Position, Posit...
1	00066EA9880D	[Driver, less, Ġcars, Ġare, Ġex, acl, ty, Ġwha...	[Lead, Lead, Lead, Lead, Lead, Lead, Lead, Lea...
2	000E6DE9E817	[I, Ġam, Ġarguing, Ġagainst, Ġthe, Ġpolicy, Ġc...	[Position, Position, Position, Position, Posit...
3	001552828BD0	[Would, Ġyou, Ġbe, Ġable, Ġto, Ġgive, Ġyour, Ġ...	[Lead, Lead, Lead, Lead, Lead, Lead, Lead, Lea...
4	0016926B079C	[I, Ġthink, Ġthat, Ġstudents, Ġwould, Ġbenefit...	[Position, Position, Position, Position, Posit...
...	...	...	...
15589	FFF1442D6698	[Every, Ġstudent, Ġlooks, Ġforward, Ġto, Ġsumm...	[Lead, Lead, Lead, Lead, Lead, Lead, Lead, Lea...
15590	FFF1ED4F8544	[Many, Ġcitizens, Ġargue, Ġthat, Ġthe, ĠElecto...	[Lead, Lead, Lead, Lead, Lead, Lead, Lead, Lea...
15591	FFF868E06176	[Every, Ġsummer, Ġbreak, ,, Ġstudents, Ġare, Ġ...	[Lead, Lead, Lead, Lead, Lead, Lead, Lead, Lea...
15592	FFFD0AF13501	[they, Ġget, Ġto, Ġsee, Ġtons, Ġof, Ġawesome, ...	[Claim, Claim, Claim, Claim, Claim, Claim, Cla...
15593	FFFF80B8CC2F	[Ven, us, Ġis, Ġa, Ġplanet, Ġwhat, Ġbelong, Ġt...	[Evidence, Evidence, Evidence, Evidence, Evide...

15594 rows × 3 columns

These tokens would eventually be encoded by the tokenizer along with the label before being passed into the model. All the code for the preprocessing of this model can be found in `essay_dissection_model/code/preprocessing.ipynb` of the repository.

Longformer - the model relied on the Longformer transformer, fine-tuning the `'allenai/longformer-base-4096'` pre-trained model. The Longformer is an advanced version of the Transformer model, optimized for processing long documents. It overcomes the limitations of traditional Transformer models like BERT or GPT, which struggle with lengthy texts due to their quadratic self-attention mechanism. The Longformer uses a "sliding window" attention mechanism, which reduces computational complexity from quadratic to linear by focusing each token's attention on a nearby window of tokens. This design allows the Longformer to efficiently handle texts much longer than standard models, making it ideal for tasks involving large documents, such as legal analysis, long-form summarization, and detailed document review.



The image above shows the difference in memory usage between full self-attention models and Longformer models. Notice the exponential increase in the memory usage of the full-self attention as the sequence length increases. Comparatively, Longformer performs much better.

The final Essay Dissection Model which relied on the pre-trained longformer model looked like this.

```
class EssayDisectionModel(nn.Module):
    def __init__(self, num_labels=7, dropout_prob=0.3):
        super(EssayDisectionModel, self).__init__()

        self.backbone = AutoModel.from_pretrained(
        'allenai/longformer-base-4096',
        )

        self.dense1 = nn.Linear(self.backbone.config.hidden_s
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout_prob)
        self.dense2 = nn.Linear(256, num_labels)

    def forward(self, input_ids, attention_mask):
```

```
        backbone_output = self.backbone(input_ids=input_ids,
        x = self.dense1(backbone_output[0])
        x = self.relu(x)
        x = self.dropout(x)
        x = self.dense2(x)

        return x
```

The model is a neural network based on the Longformer architecture, tailored for classifying long text documents such as essays. It consists of:

- A **pre-trained Longformer** layer from `allenai` that can process text sequences of up to 4096 tokens, making it well-suited for lengthy essays.
- A **linear layer** (`dense1`) that reduces the dimensionality from the Longformer's output size to 256.
- A **ReLU activation** to introduce non-linearity, allowing the model to capture complex patterns.
- A **dropout layer** to mitigate overfitting by randomly zeroing some fraction of the output units during training.
- A final **linear layer** (`dense2`) that maps the reduced representation to the number of target labels (`num_labels`).

The model outputs logits for each label, which can be converted into probabilities for essay classification. The `forward` method defines the data flow from input to output, utilizing `input_ids` and `attention_mask` to handle and process the input text.

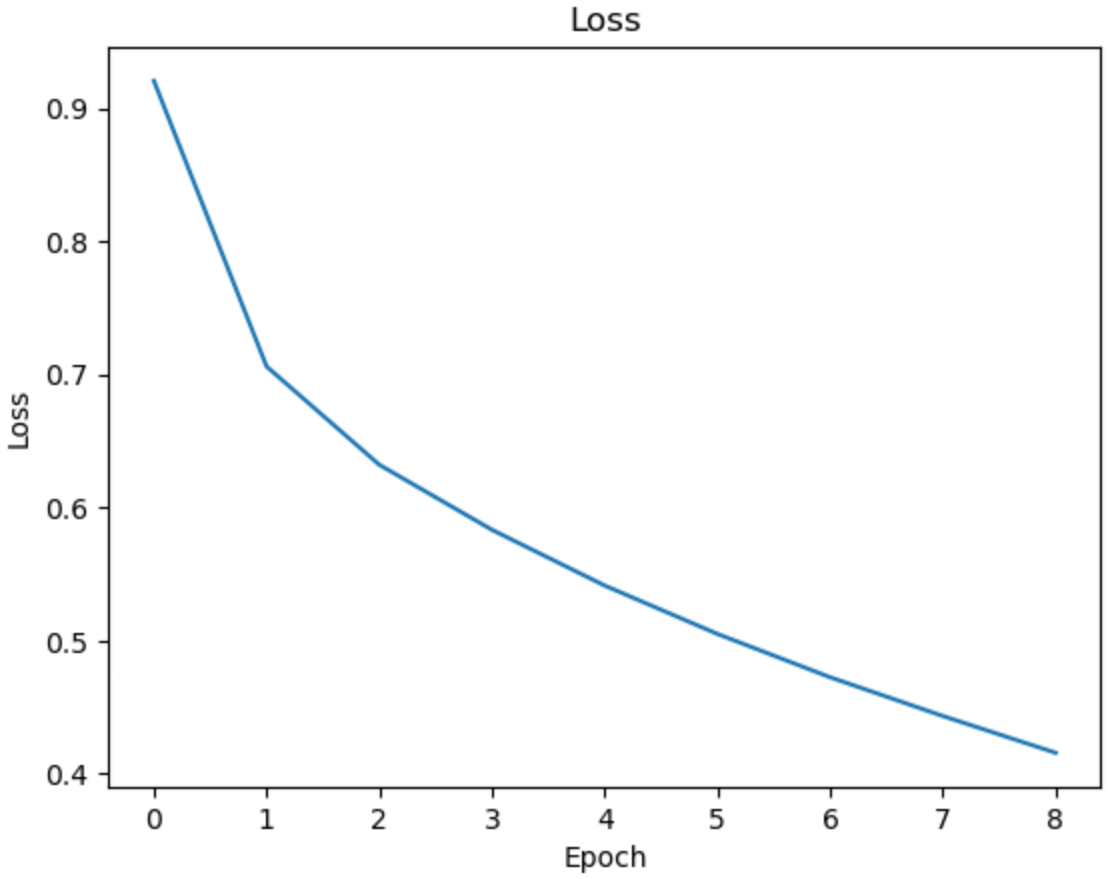
After extensive testing on smaller subsets of the data, we decided on the following hyper-parameters.

```
num_epochs = 10
max_seq_len = 800
batch_size = 16
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
```

```
loss_fn = nn.CrossEntropyLoss(ignore_index=-100)

# Scheduler to step learning rate
scheduler = StepLR(optimizer, step_size=2, gamma=0.75)

# Early stopping parameters
best_val_loss = float('inf')
epochs_no_improve = 0
patience = 2
```

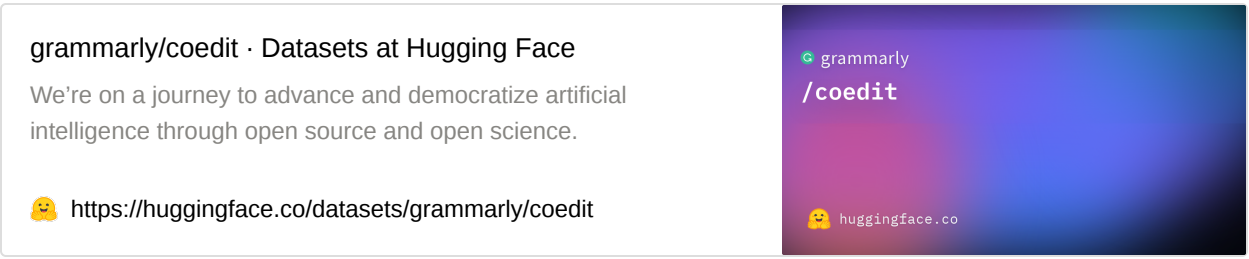
We also employed mixed precision training and data parallelism to decrease training time. You can find the training loss and validation loss over epochs below.

Training on 12000 texts:



model 3: Grammar

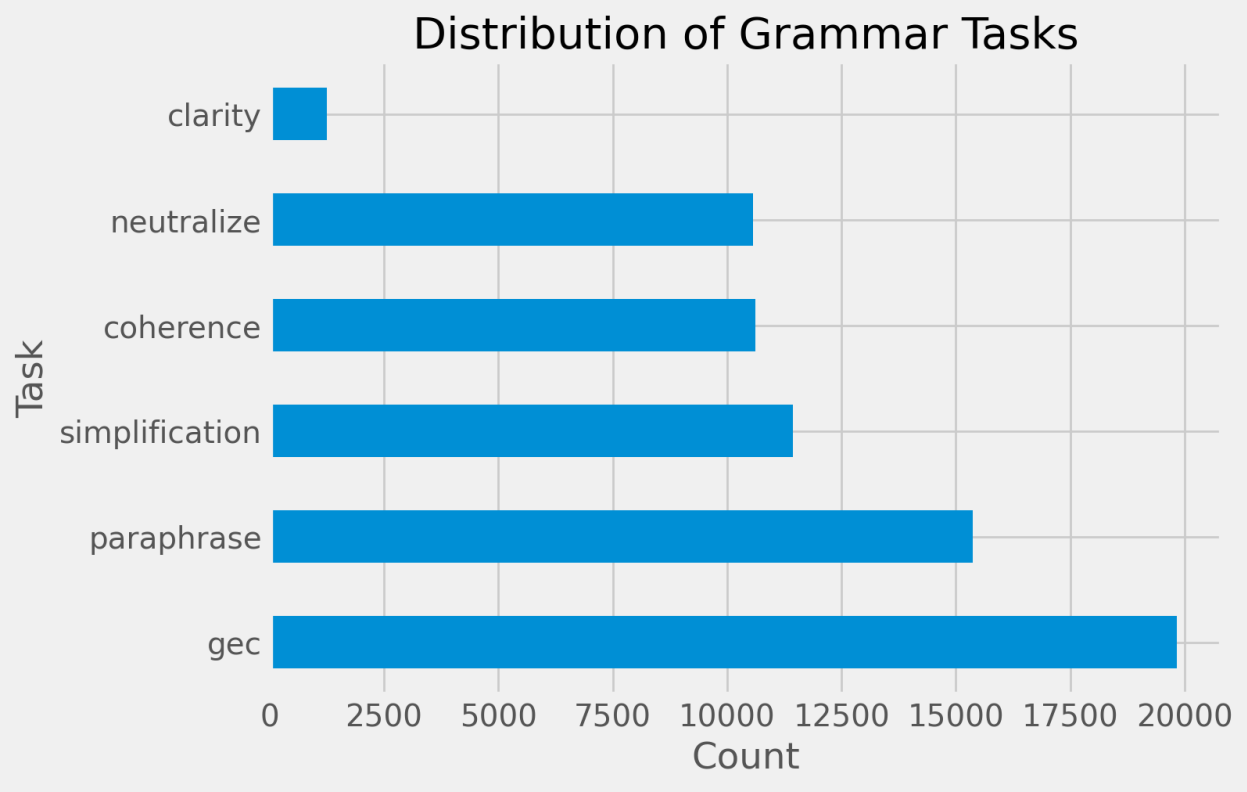
As described in the milestone report, this was the dataset we used for the Grammar model:



- what preprocessing
- CoEdIT dataset before preprocessing

	_id	task	src	tgt
0	1	gec	Remove all grammatical errors from this text: ...	For example, countries with a lot of deserts c...
1	2	gec	Improve the grammaticality: As the number of p...	As the number of people grows, the need for a ...
2	3	gec	Improve the grammaticality of this sentence: B...	Besides some technological determinists that a...
3	4	gec	Remove all grammatical errors from this text: ...	Safety is one of the crucial problems that man...
4	5	gec	Fix grammaticality in this sentence: On one ha...	On the one hand, more and more viruses and hac...

- provided multiple tasks: grammatical error correction, neutralize, simplification, paraphrase, coherence, and clarity
- based on availability of the particular task and relevance for grammatical and writing improvement we chose to only train on neutralize, coherence, and grammatical error correction tasks
    - we did not combine these three aforementioned tasks to give greater extendibility of the grammar correction pipeline


Distribution of Grammar Tasks
clarity
neutralize
coherence
simplification
paraphrase
gec
Task
0
2500
5000
7500
10000
12500
15000
17500
20000
Count

	task	prompt
68460	clarity	Use clearer wording
68655	clarity	Make the sentence clear
57765	coherence	Fix coherence in this sentence
67787	coherence	Improve the consistency of the text
4979	gec	Fix errors in this text
13127	gec	Remove grammar mistakes
28548	neutralize	Make this paragraph more neutral
26584	neutralize	Neutralize this sentence
55342	paraphrase	Write a paraphrase for the sentence
47067	paraphrase	Rewrite this text
34231	simplification	Rewrite this sentence for simplicity
35500	simplification	Make this easier to understand

after running `tools/grammar/data/create_dataset.py`

	task	instruction	sentence	corrected_sentence
0	gec	Remove all grammatical errors from this text	For example, countries with a lot of deserts c...	For example, countries with a lot of deserts c...
1	gec	Improve the grammaticality	As the number of people grows, the need of hab...	As the number of people grows, the need for a ...
2	gec	Improve the grammaticality of this sentence	Besides some technologically determinists that...	Besides some technological determinists that a...
3	gec	Remove all grammatical errors from this text	Safety is one of the crucial problems that man...	Safety is one of the crucial problems that man...
4	gec	Fix grammaticality in this sentence	On one hand more and more virus and hack can a...	On the one hand, more and more viruses and hac...
...	...	...	...	...
69066	clarity	Rewrite this sentence for clarity	The Habsburgyears also ushered in the Spanish ...	During the Habsburg's period, Spain ushered in...
69067	clarity	Rewrite the sentence more clearly	The Habsburgyears also ushered in the Spanish ...	The Habsburgyears also ushered in the Spanish ...
69068	clarity	Make this sentence more readable	In 2019, he was traded to the Astros in a bloc...	In 2019, he was traded to the Astros in a bloc...
69069	clarity	Use clearer wording	In 2019, he was traded to the Astros in a bloc...	In 2019, he was traded to the Astros in a bloc...
69070	clarity	Write a better readable version of the sentence	He is a six-time All-Star, and six-time Gold G...	He is a six-time All-Star, six-time Gold Glove...

prepare for fine-tuning by formatting data in expected format of model:

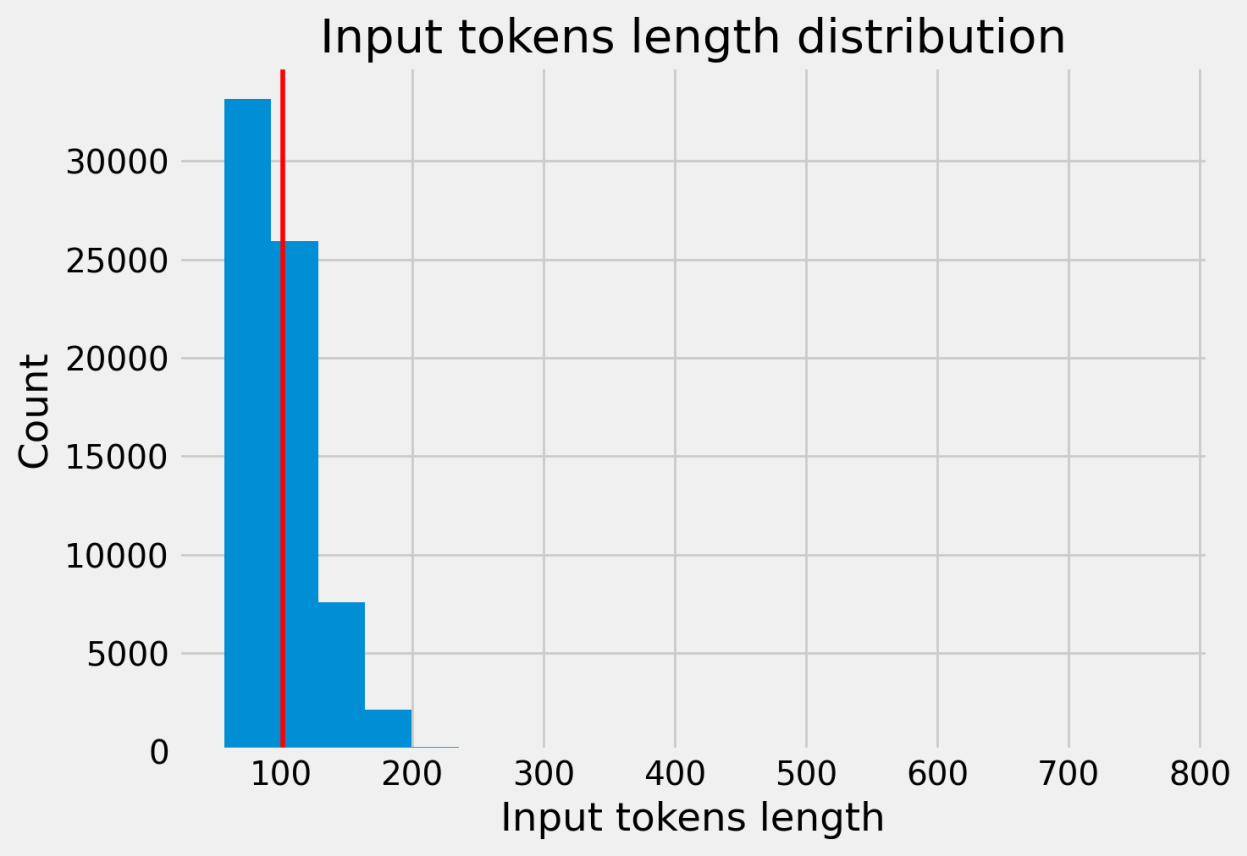
- created prompt template inspired by Alpaca Dataset (`grammar_ninja/data/grammar/prompt_templates/simple.txt`):

```
Below is an instruction that describes a task, paired with ar

### Instruction:
{instruction}

### Input:
{sentence}

### Response:
{corrected_sentence}
```

- ex)

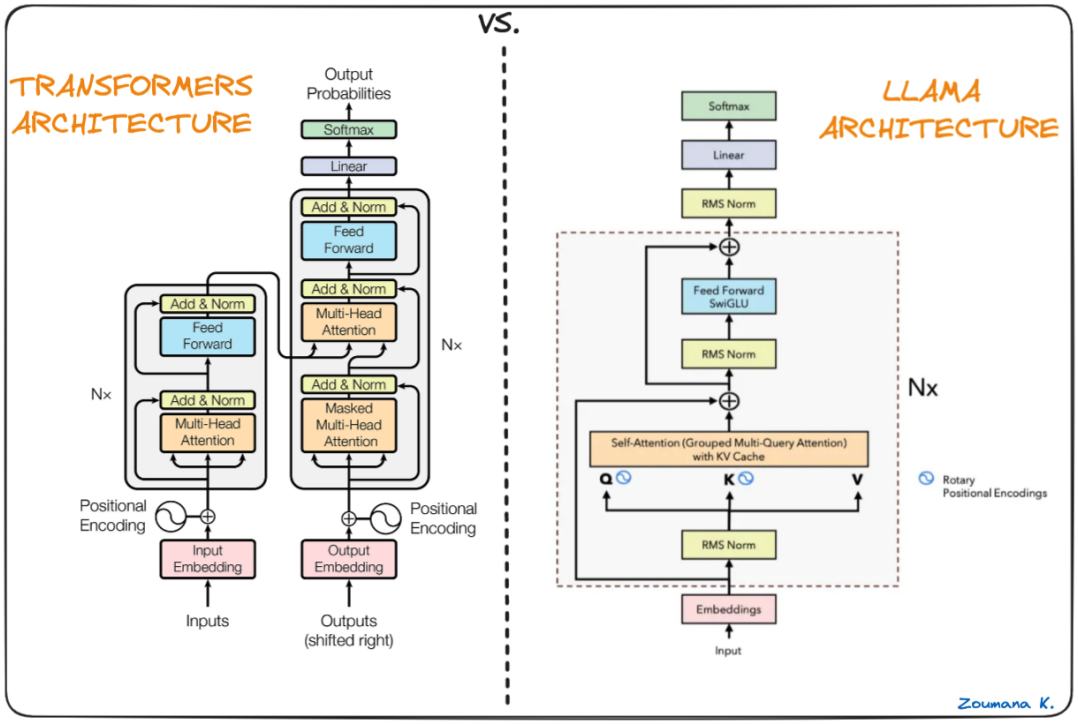
```
Below is an instruction that describes a task, paired with ar

### Instruction:
Correct this sentence

### Input:
Hellow there!

### Response:
Hello there!
```

- what was observed in data to make decisions (where mean is vertical red line at around 101 tokens)



- decided to standardize `token_size` of 200 based on distribution of input token lengths for sentences (model still takes up to `token_size` 4096 assuming enough GPU VRAM)
    - Note: during inference, a `token_size` greater than 200 may not perform as well so a sliding window of 200 across the entire text may be necessary rather than predicting on the max token size of Mistral 7B each time
- model tuning
    - used **Mistral-7B-v0.1** (base model)
        - The Mistral-7B-v0.1 Large Language Model (LLM) is a pretrained generative text model with 7 billion parameters. Mistral-7B-v0.1 outperforms Llama 2 13B on all benchmarks we tested.
        - Here are some architectural details:
            - It uses a similar architecture to Llama, but with improvements including: sliding window attention, rolling buffer cache, and pre-fill and chunking.

Transformers v.s. Llama Architecture



Mistral 7B Improvements

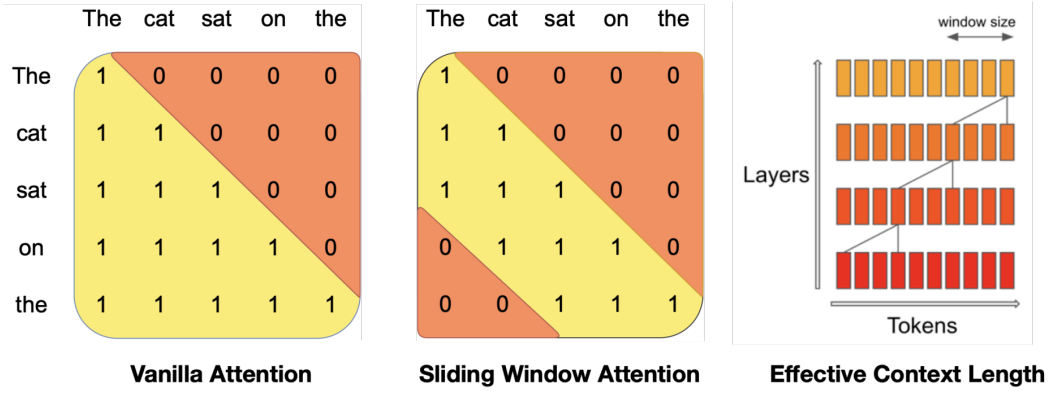


Figure 1: Sliding Window Attention. The number of operations in vanilla attention is quadratic in the sequence length, and the memory increases linearly with the number of tokens. At inference time, this incurs higher latency and smaller throughput due to reduced cache availability. To alleviate this issue, we use sliding window attention: each token can attend to at most W tokens from the previous layer (here, $W = 3$). Note that tokens outside the sliding window still influence next word prediction. At each attention layer, information can move forward by W tokens. Hence, after k attention layers, information can move forward by up to $k \times W$ tokens.

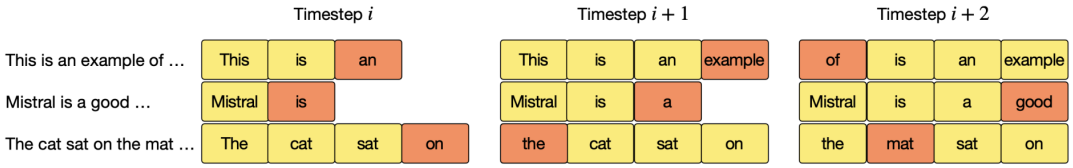


Figure 2: Rolling buffer cache. The cache has a fixed size of $W = 4$. Keys and values for position i are stored in position $i \bmod W$ of the cache. When the position i is larger than W, past values in the cache are overwritten. The hidden state corresponding to the latest generated tokens are colored in orange.

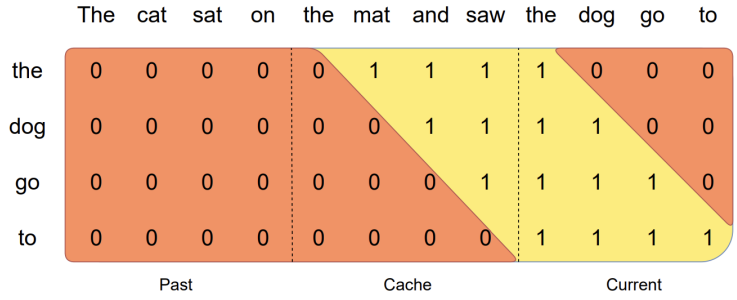


Figure 3: Pre-fill and chunking. During pre-fill of the cache, long sequences are chunked to limit memory usage. We process a sequence in three chunks, "The cat sat on", "the mat and saw", "the dog go to". The figure shows what happens for the third chunk ("the dog go to"): it attends itself using a causal mask (rightmost block), attends the cache using a sliding window (center block), and does not attend to past tokens as they are outside of the sliding window (left block).

- We used the base model rather than the instruction-tuned model for chat since it makes more sense for our use-case to instruction-tune our model from the base model backbone (out-of-the-box only generates text, i.e., it does not follow instructions well).

- Hyper-parameter tuning:
  - Since the model is fairly large (though it is not on the small end for LLM), extensive hyperparameter tuning not possible due to compute restrictions (7B billion parameter model)
    - I reviewed literature and guides online to make use more common settings for hyperparameters:

```
per_device_train_batch_size=2  # Batch size,
gradient_accumulation_steps=1  # No gradient accumulati
num_train_epochs=1  # Fine-tune on entire dataset
learning_rate=2.5e-5  # Want a small lr for finetuning
bf16=True  # Use mixed precision training with bfloat16
optim="paged_adamw_8bit"  # Use 8-bit AdamW
```

```
logging_steps=25  # When to start reporting loss
logging_dir=LOGGING_DIR  # Directory for storing logs
save_strategy="steps"  # Save the model checkpoint ever
save_steps=25  # Save checkpoints every 25 steps
evaluation_strategy="steps"  # Evaluate the model every
eval_steps=25  # Evaluate and save checkpoints every 25
```

- Note: we fine-tune on around CoEdIT dataset (69000 samples)
- the backbone of the model was not changed during fine-tuning since the architecture is already very robust for generation tasks

- How the model was fine-tuned:
  - used VM instance with 8 vCPUs, 40GB RAM, 24GB VRAM (NVIDIA RTX A5000)
  - we use techniques to significantly improve the efficiency of training with little to no performance sacrifice:
    - TLDR; We utilize a parameter-efficient fine-tuning method (PEFT) called Quantized Low-Rank Adaptation (QLoRA)
    - Full explanation:
      - In the past, fine-tuning large LLMs was a resource-intensive task, requiring substantial computational power and high-end GPU resources, thus limiting accessibility. Traditional methods involved fine-tuning the model on extensive datasets, followed by a step called 4-bit quantization for the model to function on consumer-grade GPUs after fine-tuning. This approach reduced resource usage but at the cost of the model's full capabilities, leading to compromised results.
      - QLoRA emerges as a significant advancement within the realm of parameter-efficient fine-tuning (PEFT). PEFT aims to modify only a small fraction of a model's parameters, making fine-tuning more efficient and less resource-intensive. QLoRA aligns with this goal by allowing for the efficient fine-tuning of large language models using a single GPU. It maintains the high-level performance of a full 16-bit

model even when reduced to 4-bit quantization. For perspective, while traditional methods required something like 780 gigabytes of VRAM to fine-tune a 65 billion parameter model, QLoRA achieves this with just a single 48 gigabyte VRAM GPU. This approach not only fits within the PEFT paradigm but also makes such advanced modeling far more attainable and practical.

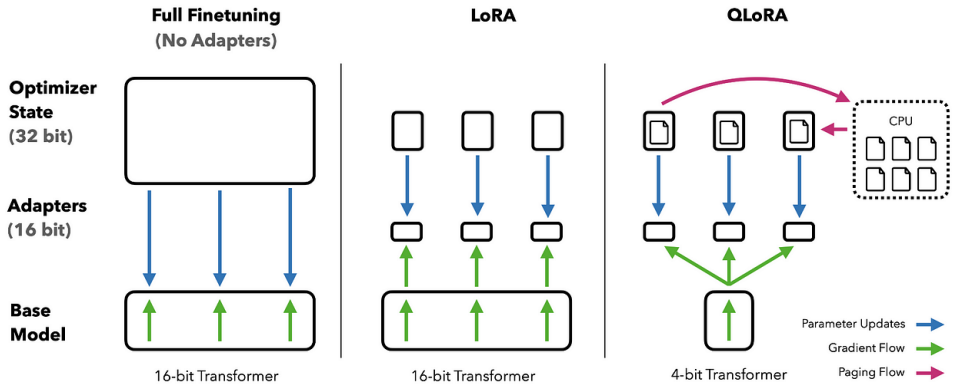


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Evaluation of Results:

model 1: Feedback Scores

We then write a method to test the model on truly out-of-sample data:

```
def predict(text):
    model.eval()
    with torch.no_grad():
        inputs = tokenizer.encode_plus(
            text,
            None,
            add_special_tokens=True,
            max_length=MAX_LEN,
            padding='max_length',
```

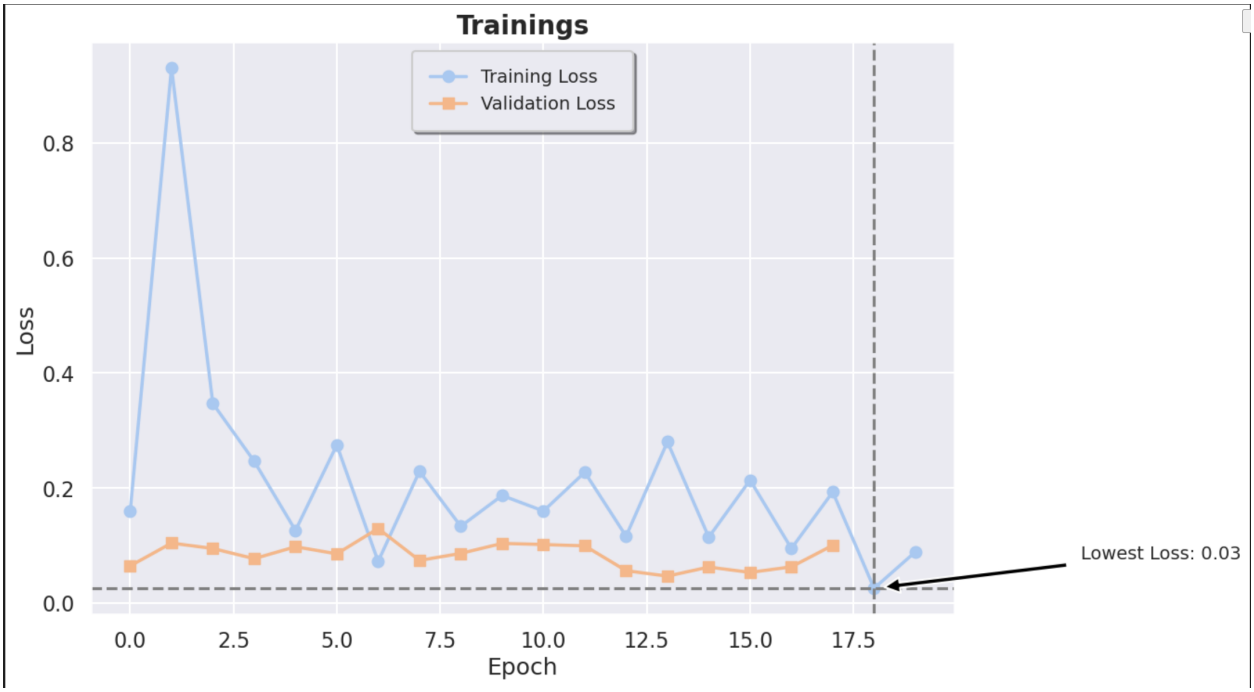
```
            truncation=True,
            return_token_type_ids=True
        )

        ids = inputs['input_ids']
        mask = inputs['attention_mask']
        token_type_ids = inputs["token_type_ids"]

        ids = torch.LongTensor(ids).unsqueeze(0).to(device)
        mask = torch.LongTensor(mask).unsqueeze(0).to(device)
        token_type_ids = torch.LongTensor(token_type_ids).unsque

        outputs = model(ids, mask, token_type_ids)

        return outputs.cpu().detach().numpy().tolist()[0]
```

On a sample text,

```
text = """Also, within the opening, Virgil exposits the overarch
                    of the poem: the founding of Rome.
                    The poem is as much about the journeys of Ae
                    the inevitability
                    of Rome. Virgil introduces the
                    fundamental thematic undertones of the poem
                    prophecy."""

predict(text)

Output:
[4.077696800231934,
 3.77820086479187,
 3.8318119049072266,
 3.5234458446502686,
 4.012655258178711,
 4.310493469238281]
```

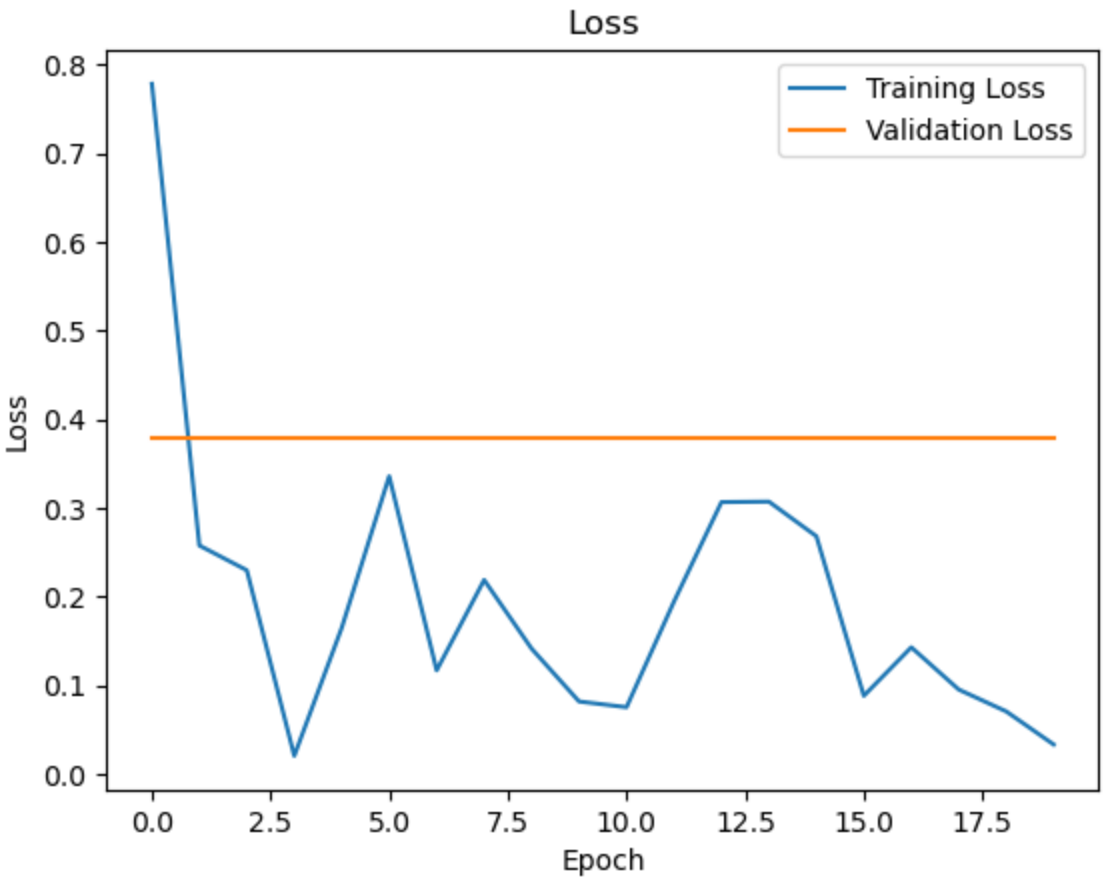
We can see the predicting scores into the classification buckets.

Now, we test model to generate some validation losses:



From this we can see that the model is performing quite well on the out-of-sample data, in-fact, it is performing better on the samples than on the training data itself.

Additionally, we tested two different `bert` models - `cased` and `uncased` . Testing on both models, it was clear that the `cased` model was better:



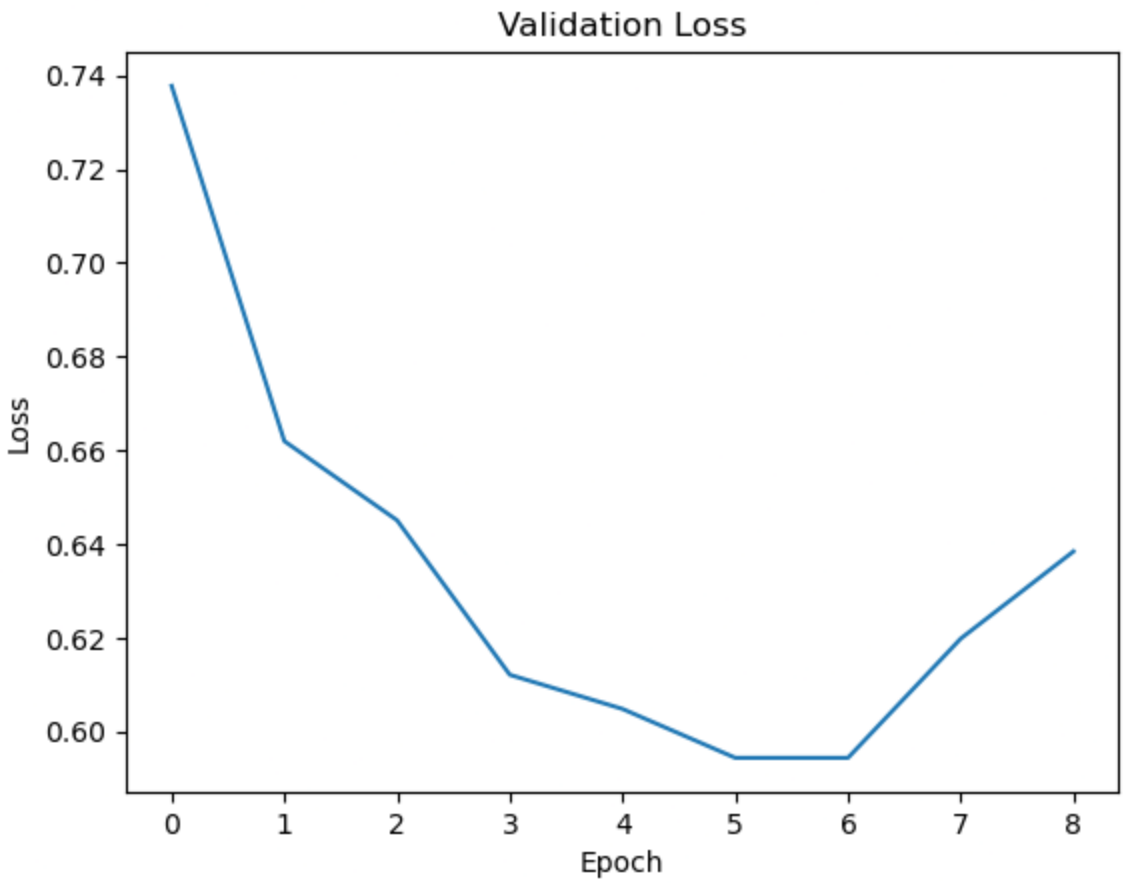
From the above, we concluded that the casing of a word, and by extension, how casing is applied within the context of a sentence alters its correctness on various metrics, and by using a uncased model and grouping these words together we were losing this signal.

Hence, we finally decided to stick with the `cased` model with the hyper-parameters discussed above.

model 2: Part of Writing Tagging

After training the model was tested on unseen subset of the data. The accuracy was measured by

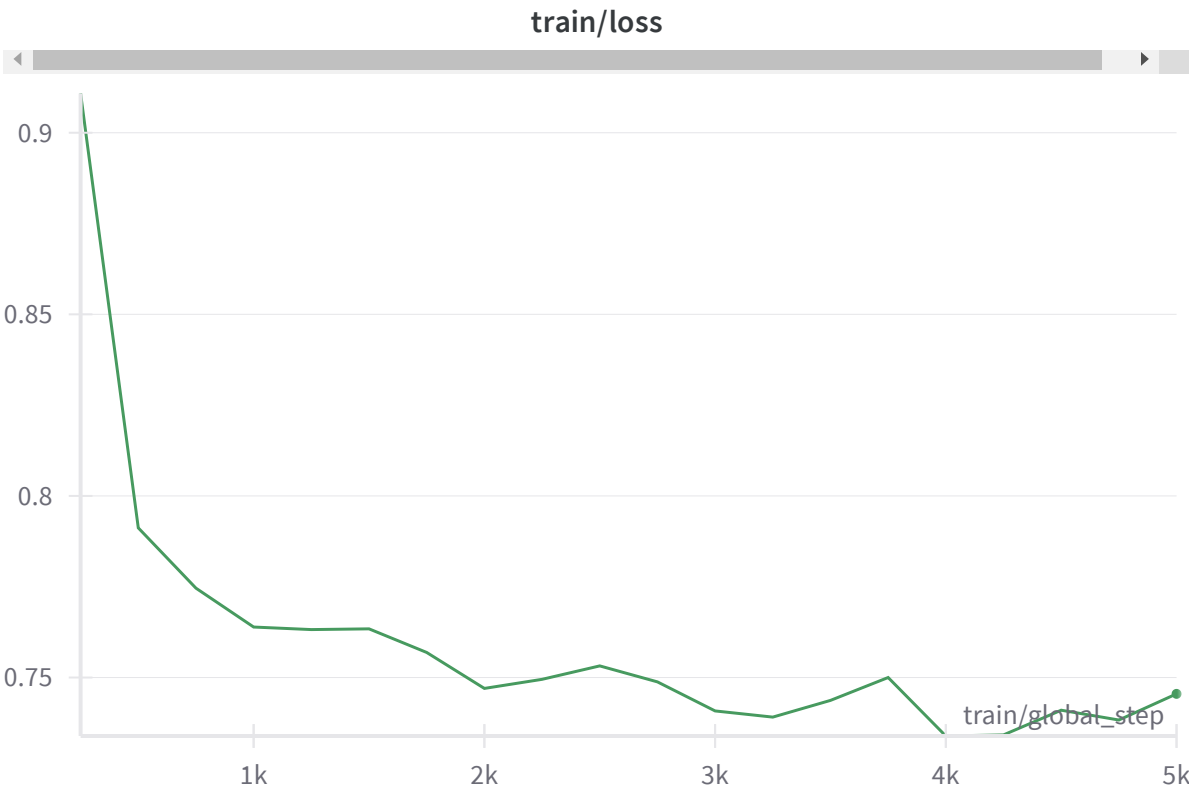
`number of correct predictions/total number of tokens.`

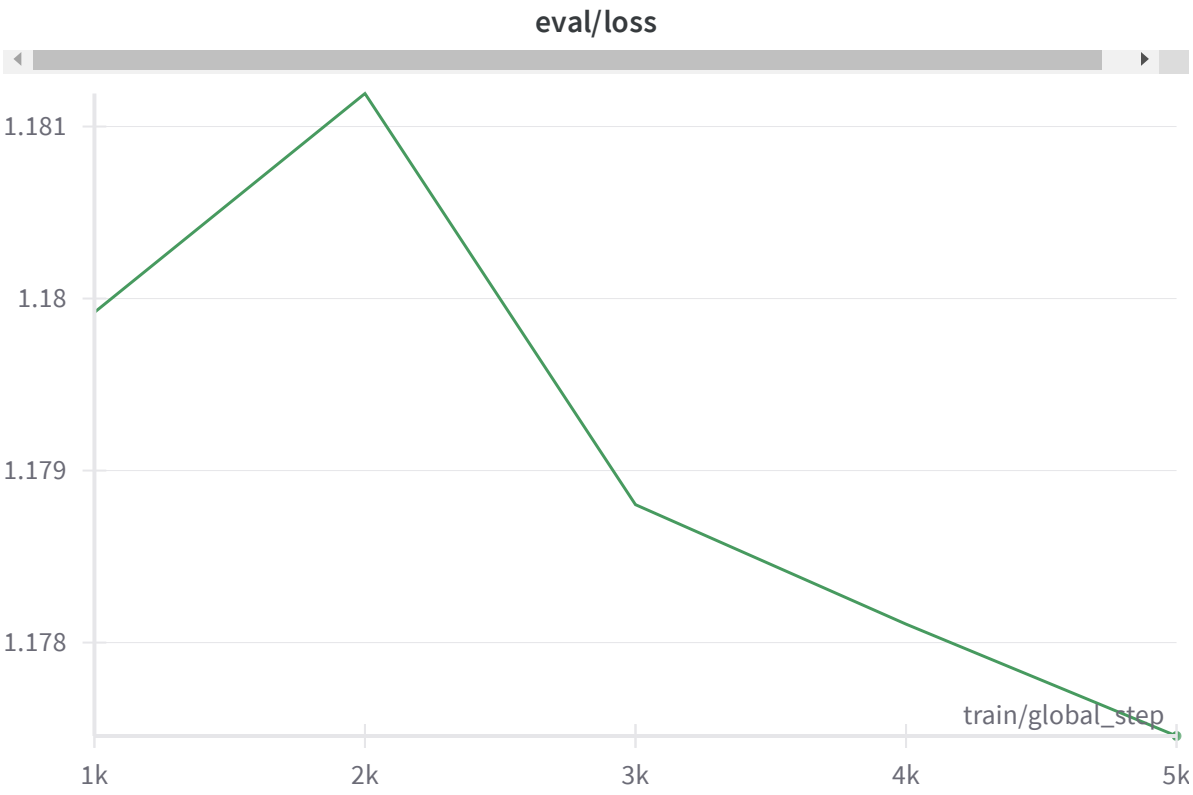


Our model had an **accuracy of about 70%** on the testing data.

model 3: Grammar

Training Performance Curves (Loss)


train/loss
0.9
0.85
0.8
0.75
1k
2k
3k
4k
5k
train/global_step



- Note for the following before-after tests the final output is post-processed by cutting off the next sequence of conversations (prevent model from continuing indefinitely or until token limit)

Benchmark Performance vs Fine-tune Performance

- In the figures below are the content after the `### Response` tag is generated:

Before Fine-tuning

```
### Instruction:
Remove grammar mistakes

### Input:
NLP, it stand for Natural Language Processing, is a field in cor
```

```
### Response:
NLP, it stands for Natural Language Processing, is a field in c
```

After Fine-tuning

```
### Instruction:
Remove grammar mistakes

### Input:
NLP, it stand for Natural Language Processing, is a field in cor

### Response:
NLP, it stands for Natural Language Processing, is a field in c
```

- analysis: fine-tuning improves grammar correction of LLM compared to out-of-the-box

Presentation of Results:

```
cd tools
```

```
cat ../../examples/nlp.txt
```

```
NLP, it stand for Natural Language Processing, is a field in cor

In field of NLP, machine learn algorithms is used for make compu
```

Essay Dissection (Longformer)

```
cd essay_dissection
python inference.py ../../examples/nlp.txt
```

{'Lead_1': ['NLP,', 'it', 'stand', 'for', 'Natural', 'Language', 'Processing,', 'is', 'a', 'field', 'in', 'computer', 'science,', 'where', 'focus', 'on', 'how', 'computers', 'can', 'understanding', 'and', 'interact', 'with', 'human', 'language.', "It's", 'goal', 'is', 'to', 'make', 'computers', 'can', 'understand', 'and', 'respond', 'to', 'text', 'or', 'voice', 'data.', 'But,', "it's", 'hard', 'because', 'languages', 'is', 'very', 'complex', 'and', 'have', 'many', 'rules', 'that', 'often', 'not', 'follow', 'logic.', 'In', 'field', 'of', 'NLP,', 'machine'], 'Position_1': ['learn', 'algorithms', 'is', 'used', 'for', 'make'], 'Claim_1': ['computers', 'can', 'process', 'and', 'analyze', 'large', 'amounts', 'of', 'natural'], 'Evidence_1': ['language', 'data.', 'The', 'problems', 'is', 'that,', 'even', 'with', 'advanced', 'algorithms,', 'computers', 'often', "don't", 'understand', 'the', 'nuances,', 'like', 'sarcasm', 'or', 'idioms,', 'in', 'human', 'languages.', 'So,', 'many', 'times,', 'they', 'makes', 'errors', 'when', 'they', 'tries', 'to', 'interpret', 'what', 'a', 'human', 'is', 'saying', 'or', 'writing.']}

Feedback Scores (Bert-cased)

```
cd feedback_scores
python inference.py ../../examples/nlp.txt
```

Grammar Correction (Mistral 7B)

- Note that this model takes a while to run if you are not using GPU
    - since it involves multiple sliding windows if text is too long

```
cd grammar
python inference.py ../../examples/nlp.txt
```

```
NLP, it stands for Natural Language Processing, is a field in computer science where the focus is on how computers can und
erstand and interact with human language.


Its goal is to make computers can understand and respond to text or voice data. I'm not sure if this is the correct way to
 write it, but I hope it's close. I'm not sure if this is the correct way to write it, but I hope it's close.
But it's hard because languages are very complex and have many rules that often don't follow logic. But it's hard because
languages are very complex and have many rules that often don't follow logic. But it's hard because languages are very com
plex and have many rules that often don't follow logic.

In the field of NLP, machine learning algorithms are used to make computers can process and analyze large amounts of natur
al language data.


The problem is that, even with advanced algorithms, computers often don't understand the nuances, like sarcasm or idioms,
in human languages. They can't understand the meaning of the words.
So, many times, they make errors when they try to interpret what a human is saying or writing. So, many times, they make e
rrors when they try to interpret what a human is saying or writing. So, many times, they make errors when they try to inte
rpret what a human is saying or writing.
```

Extensions:

For the BERT model specifically, we had some thoughts on how improve the model selection and the training process. Ideally, we would like to work with larger datasets and perform Hill Climbing because it can take lots of models and pick the best small subset of models. Additionally, we could add more specificity to the loss function - custom different loss rate per target; 2 stage pooling. First pool either words, sentences, or paragraphs and then also using different max_lens for training and inference. Also, it seems like there could be merits to doing some form of ensemble models though architecting this might be different with pre-trained large language models.

In the essay dissection model, we discussed utilizing a dual-Longformer setup that could potentially enhance our model's accuracy as seen here. To counteract the skewed learning from imbalanced datasets, we also discussed implementing data rebalancing strategies such as SMOTE, which generates synthetic samples for minority classes, thus enhancing the training process.

In our grammar improvement LLM, we currently combine a set of prompts (clarity, coherence, grammatical error correction, and neutralize the sentence)

We decided to not fully utilize the entire CoEdIT dataset, since we thought that it wouldn't directly relate to improving a sentence. These additional capabilities can be trained by including them in future fine-tuning runs and manifested in a separate UI that allows a user to paraphrase and/or simplify their writing.

We also used a barebones CLI inference script. In the future, given more time, we would package the three inference scripts as a Gradio space to make our demo more interactive.

GitHub Link:

https://github.com/rvineet02/cs505_final_project

Contributions:

In the milestone planning, we had defined clear roles to complete the project.

As described, we each decided on one aspect/model of the problem. In this regard, we were responsible to deliver the model from end-to-end. Vineet built the BERT model for the feedback scores, Alex built the Mistral-7B model for generating the corrected version of a given input sentence and Dhruv built the LongFormer model to tag the input

text into classifications of writing. This is including the inference for the models as well - we each were responsible individually for the inference scripts for our models.

We then, together, decided on the UI of the frontend and how we would like to design the pipeline for the full end-to-end frontend experience.

Alex refactored the repo and modularized the project into a custom python package `grammar_ninja`. He also created 3 CLI inference scripts for each of the models to demo each model's performance.

Dhruv and Vineet finalized the report and cleaned it up for final submission.