Assignment 1: Large Language Models for Text Generation

CS 410/510 Large Language Models Fall 2024

Greg Witt

Q1. Describe three differences between Llama 3.2 models and Phi-3.5 model.

Llama 3.2 Models

Llama 3.2 Mobile Devices Llama 3Heard of Models

Overview

Trained on a mix of publically avaialbe online data, Multiple languages seem to be the forte of this model, support of **128K context length** as the use cases explicitly mention the models ability to perform well in this category, unlike the Phi Model which claims to be supportive but is adviced from doing so within it's terms of accountability. Support architecture is based on **GQA (Grouped Query Atttention)**

Llama models offer a **Lama Guard 3 Model** to support the regulation of safety for input and output of the model. The models we are using aren't Fine-tuned or Tool Use training, but trained with a multilingual and long-context training.

Three Differences

- Non-Instructional Models Used, Text Only for Experiment, which leads to performanced very unfavorably in reasoning
- Models use **GQA (Grouped Query Attention)**
- Fine Tuned for broader NLP tasks, used for generalization

Phi-3.5 Model

Discover the Multi-Lingual Phi-3.5 SLMs

Technical Report

Overview

released in April 2024, With various varies **Phi-3.5-mini**, **Phi-3.5-vision**, and **Phi-3.5-Moe** (Mixture of Experts Model). The Mini model specifically features of **128k Context Length**

Microsoft pre-trained using multi-lingual synthetic and filtered data. Post-trained with Supervised Fine-Tuning (SFT), Proximal Policy Optimization (PPO) and (Direct Preference Optimization)

with regard to **Long Context** it was able to perform roughly **1% point higher average** compare to **Llama-3.1-8B-Instruct** models. Beating the LLama models in **GovReport**,**Qasper** by only a slight margin. This Bench mark suffered compared to Llama-3.1-8B with **Ruler: a retriever-based benchmark for long context understanding** losing to the Llama model with an average of 4% points. but appears to beat the Llama models with an average of 6% for **ReportQA** for code understanding and context for Python, Java, an Typescript. Suggested for **Retrieval-Augmentd Generation** due to it's high performance for knowledge-intensive tasks.

Three Differences

- 3.8 Billion Parameters, with **Blocksparse attention module**
- **Decoder-Only** Transfomer model
- Extensive pre-training for instructional chat format

Q2. Generate a story of 200 words that starts with the words *“Once upon a time”* using each of these models.

You should have 3 outputs in total.

Below are three instances of the requested models. Each was executed **three times** the last run is featured below the model's generation cell. the **additional stories** are featured *below* the final **Llama** model and the **Phi** model. the link will take you to a git repo that has the images stored.

an **analysis** will below each model and an in depth explaination will be featured there.

Install Required Packages

In [ ]:
```
# pip install transformers
# pip install torch
```

Llama 3.2 - 1B:

Hugging Face Model Card

Download Llama-3.2 1B

In [2]:
```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

llama_32_1B = "meta-llama/Llama-3.2-1B"
```

```
# creates a Tokenizer specifically for the llama_model requested
llama_32_1b_tokenizer = AutoTokenizer.from_pretrained(llama_32_1B)

# Set the padding token ID to be the same as the EOS token ID
llama_32_1b_tokenizer.pad_token_id = llama_32_1b_tokenizer.eos_token_id

llama_32_1b_model = AutoModelForCausalLM.from_pretrained(llama_32_1B, torch_

llama_32_1b_model = llama_32_1b_model.to('cpu')
```

Generate A Story with Llama 3.2 1B

In [3]:
```
# Our Story Prompt
story_prompt = "Once upon a time"

# Encode the prompt into token IDs
prompt_ids = llama_32_1b_tokenizer.encode(story_prompt, return_tensors="pt")

# Create an attention mask
attention_mask = prompt_ids.ne(llama_32_1b_tokenizer.pad_token_id)

# Generate a response from llama_3.2-1B
outputs = llama_32_1b_model.generate(prompt_ids,
                                     attention_mask=attention_mask,
                                     max_length=200,
                                     do_sample=True,
                                     num_return_sequences=1,
                                     pad_token_id=llama_32_1b_tokenizer.eos_token_id,
                                     temperature=0.93,
                                     top_k=30,
                                     top_p=0.90,
                                     repetition_penalty=1.2
                                    )

# Decode the generated response
generated_tokens = outputs[0]

generated_story = llama_32_1b_tokenizer.decode(generated_tokens, skip_specia
print(generated_story)
```

```
Starting from v4.46, the `logits` model output will have the same type as th
e model (except at train time, where it will always be FP32)
```

```
Once upon a time, there was the King of all Men and his beautiful Wife. He g
ave her an apple from which he believed she would never eat anything again.
She began to cry when it first hit her mouth but then stopped as soon as her
lips touched that magical fruit – and so did everything else in sight! The K
ing had lost hope for another woman’s love forever until one day…
…He woke up with 42 women!
That morning, while everyone went about their usual business; breakfasts wer
e made,
lunches eaten
dinnners served &
all this took place without even missing out on brushing your teeth or takin
g off those damn makeup brushes you use every night before bed just because
they smell nice?
There was no need at all since none of them could be found anywhere near us
anymore either! All over town these days? No problemo!
All we needed here today though came right down front & personal too – only
thing anyone knew who lived alone now more often than
```

Measure of Perplexity

In [5]:
```python
import torch

# Extract the Logits from the Model based on the Inputs
with torch.no_grad():
    outputs = llama_32_1b_model(prompt_ids)
    logits = outputs.logits

# shift the input_ids to the right to determine the next token for the model
shift_logits = logits[:, :-1, :].contiguous()
shift_labels = prompt_ids[:, 1:].contiguous()

# calculate the log likelihood based on Cross EntropyLoss
loss_fct = torch.nn.CrossEntropyLoss(ignore_index=llama_32_1b_tokenizer.pad_
# determine the loss value to exponentiate
loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.v

# exponentiate the loss
perplexity = torch.exp(loss)

# return the results
print(f"Perplexity for Llama 3.2 1B Model: {round(perplexity.item(),2)}")
```

```
Perplexity for Llama 3.2 1B Model: 14.88
```

Measure Token Type Ratio

In [12]:
```python
prompt_text = "Once upon a time "

tokens = prompt_text.split()

types = set(tokens)
ttr = len(types) / len(tokens)

print("Type-Token Ratio (TTR) for Llama 3.2 1B Model:", round(ttr,2))
```

```
Type-Token Ratio (TTR) for Llama 3.2 1B Model: 1.0
```

Analysis

The **temperature** of the model was adjusted to `0.93` to add some creativity to the response because anything higher or lower was way too incoherent to result in anything that was closely similar to a story. The results for `0.85` were very close to an incoherent rambling, this was due to the highly correlated nature of phrase combinations being used by the model resulting in repeating tokens, for this I added a **repetition_penalty** to the model to ensure the model had a restriction on it's token ramblings.

After noticing the repeating tokens I added a `top_p` of `0.90` . to ensure a probabilty cut off for it's token choices but add a bit of spice instead of always choosing the best possible, this along with the `temperature` modifications helped ensure some diverse responses that weren't just rambings from the training data. I also modified the `top_k` to `30` this modification was to ensure a increase the amount of tokens to chose. I determined the higher number other than `15` of something small like `5` ensured a larger amount for the model to choose from but with an additiona `top_p` being set to a `0.9` it would help choose the most appreate from this subset.

this resulted in a less repetitive story, although the ending was very unlikely, often times when runnign the Llama models they would often repeat words and add descriptions to the prompt. I ensure this was often overlooked by adding a `pad_token` this ensures that the ending is clearly defined for the prompt to ending, it also helped with the **TTR** and **Perplexity** the scores for the perplexity were fairly high, which does make sense for these models as they're just filling in words that aren't necessarily close to a particularly developed prompt. Often times prescribing a larger prompt increased the **TTR** but didn't result in a better story.

Llama 3.2 - 3B:

Hugging Face Model Card

Download Llama 3.2 - 3B

In [7]:
```python
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

llama_32_3B = "meta-llama/Llama-3.2-3B"

# creates a Tokenizer specifically for the llama_model requested
llama_32_3b_tokenizer = AutoTokenizer.from_pretrained(llama_32_3B)

# Set the padding token ID to be the same as the EOS token ID
llama_32_3b_tokenizer.pad_token_id = llama_32_3b_tokenizer.eos_token_id

llama_32_3b_model = AutoModelForCausalLM.from_pretrained(llama_32_3B, torch_
llama_32_3b_model = llama_32_3b_model.to('cpu')
```

```
huggingface/tokenizers: The current process just got forked, after paralleli
sm has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(tr
ue | false)
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

Generate A Story with Llama 3.2 3B

In [14]:

```
# Your special prompt
story_prompt = "Once upon a time "

# Encode the prompt into token IDs
prompt_ids = llama_32_3b_tokenizer.encode(story_prompt, return_tensors="pt")

# Create an attention mask
attention_mask = prompt_ids.ne(llama_32_3b_tokenizer.pad_token_id)

# Generate a response from llama_3.2-3B
outputs = llama_32_3b_model.generate(prompt_ids,
                        attention_mask=attention_mask,
                        max_length=200,
                        do_sample=True,
                        num_return_sequences=1,
                        pad_token_id=llama_32_3b_tokenizer.eos_token_id,
                        temperature=0.83,
                        top_k=30,
                        top_p=0.90,
                        repetition_penalty=1.2
                    )

# Decode the generated response
generated_tokens = outputs[0]

generated_story = llama_32_3b_tokenizer.decode(generated_tokens, skip_specia
print(generated_story)
```

```
Once upon a time 2.0: The second coming of the world's most famous animated
film
By Chris HallamPosted on July 1, 2015 August 3, 2020 Posted in Film reviewsT
agged Disney, Donald Duck, Fantasia, Mickey Mouse, Pinocchio, Snow White and
the Seven Dwarfs No Comments on Once upon a time 2.0: The second coming of t
he world’s most famous animated film
Snow White and the Seven Dwarfs was released to critical acclaim in America
by Walt Disney Pictures (or as they were then known – Laugh-O-Gram Studio) w
ay back in December 1937. It remains arguably one of cinema’s greatest achie
vements. Its success paved the road for all future Disney animation producti
ons.
The original soundtrack had been so popular that it spawned an album which f
eatured three tracks from the movie. These included “Someday My Prince Will
Come”, “Whistle While You Work” and “Heigh
```

Measure Perplexity

In [10]:
```
import torch

# Extract the Logits from the Model based on the Inputs
with torch.no_grad():
    outputs = llama_32_3b_model(prompt_ids)
    logits = outputs.logits

# shift the input_ids to the right to determine the next token for the model
shift_logits = logits[:, :-1, :].contiguous()
shift_labels = prompt_ids[:, 1:].contiguous()

# calculate the log likelihood based on Cross EntropyLoss
loss_fct = torch.nn.CrossEntropyLoss(ignore_index=llama_32_3b_tokenizer.pad_
# determine the loss value to exponentiate
loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.v

# exponentiate the loss
perplexity = torch.exp(loss)

# return the results
print(f"Perplexity for Llama 3.2 3B Model: {round(perplexity.item(),2)}")
```

```
Perplexity for Llama 3.2 3B Model: 35.32
```

Measure Type-Token Ratio

In [11]:
```
prompt_text = "Once upon a time"

tokens = prompt_text.split()

types = set(tokens)
ttr = len(types) / len(tokens)

print("Type-Token Ratio (TTR) for Llama 3.2 3B Model:", round(ttr,2))
```

```
Type-Token Ratio (TTR) for Llama 3.2 3B Model: 1.0
```

Additional Stories

Analysis

The Llama 3.2 3B clearly differs in size with 3 billion parameters allows for more fine tuning of the models parameters and likely a better outcome for instructional models. however I could notice a change in the resulting stories, this increase of parameters made a difference in this models performance after I tuned some of the parameters in a similar fashion to Llama 3.2 1B model.

There wasn't a need to modify a lot of parameters compared to the the other model, it seemed as if I had optimized the `top_p` and `top_k` to optimal numeric ranges from the Llama 3.2 1B model, so I used them again for Llama 3.2 3B.

The stories produced were very intriguing , I found the one produced about *Buddha* very mystical it reminded me of the *Lotus Sutra*, which it could have possibly been trained on.

Overall the stories were very coherent at times and interesting, often times less jiberish but there were a few times when the prompt resulted in a regurgiation of extra tokens that seemed like they were pulled from training data. Often times I would experience repetition so I kept the `repetition_penalty` to 1.2 just like the Llama 3.2 1B.

the **perplexity** resulted in a higher number compared to the smaller model which might be from the larger model size but the **TTR** remained close to `1.0` . which means it had a decent TTR meaning the resulting tokens were diverse and translates to higher cretivity in stories which is great for this experiment, as stories should be inviting and engauging. This is great the high perplexity means that the model produced mostly what it thought of as a moderate level of uncertainty in predictions for the next token. I think this might have helped the model produce more human friendly and more coherent stories compared to the 1B model, as the *Buddha* story read like a profound scripture.

Phi 3.5-Mini-Instruct:

Hugging Face Model Card

Download Phi 3.5 - Mini - Instruct Model

In [1]:
```
from transformers import AutoTokenizer, AutoModelForCausalLM

phi_35_mini_inst = "microsoft/Phi-3.5-mini-instruct"

phi_tokenizer = AutoTokenizer.from_pretrained(phi_35_mini_inst)
phi_model = AutoModelForCausalLM.from_pretrained(phi_35_mini_inst)
```

```
/opt/homebrew/Cellar/jupyterlab/4.1.6_1/libexec/lib/python3.12/site-package
s/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter a
nd ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_instal
l.html
  from .autonotebook import tqdm as notebook_tqdm
Loading checkpoint shards: 100%|██████████████████████████████| 2/2 [00:15<0
0:00,  7.66s/it]
```

Generate a Story with Phi 3.5 - Mini Instruct

In [4]:
```
prompt_text = "Generate a 200 word story that begins with, 'Once upon a time

# Tokenize the input text
inputs = phi_tokenizer(prompt_text, return_tensors="pt")

# Generate text
outputs = phi_model.generate(inputs.input_ids,
                             max_length=200,
                             temperature=0.85,
                             do_sample=True
                             )

# Decode the generated text
```

```
generated_story = phi_tokenizer.decode(outputs[0], skip_special_tokens=True)

print(generated_story)
```

```
Generate a 200 word story that begins with, 'Once upon a time' incorporate a
nature theme and make it scary, and mysterious in mood, without using words
related to 'fear', 'dark', 'mysterious', 'scary', 'night', or 'monster'.

In a dense, shadow-draped forest, where sunlight seldom danced through the a
ncient canopy, there existed a silence so profound that even the whispers of
the wind seemed subdued. Once upon a time, under this ethereal stillness, a
tale unfurled, woven from threads of enigma and the haunting beauty of natur
e itself.

An old oak, gnarled with secrets and wisdom from ages past, stood solitary a
midst its brethren. Its bark bore the intricate carvings of forgotten lore,
and its hollows whisper
```

Measure of Perplexity

In [6]:

```
import torch

# Extract the Logits from the Model based on the Inputs
with torch.no_grad():
    outputs = phi_model(inputs.input_ids)
    logits = outputs.logits

# shift the input_ids to the right to determine the next token for the model
shift_logits = logits[:, :-1, :].contiguous()
shift_labels = inputs.input_ids[:, 1:].contiguous()

# calculate the log likelihood based on Cross EntropyLoss
loss_fct = torch.nn.CrossEntropyLoss(ignore_index=phi_tokenizer.pad_token_id
# determine the loss value to exponentiate
loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.v

# exponentiate the loss
perplexity = torch.exp(loss)

# return the results
print(f"Perplexity for Phi-3 Model: {round(perplexity.item(),2)}")
```

```
Perplexity for Phi-3 Model: 18.66
```

Measure Type Token Ratio

In [7]:

```
prompt_text = "Generate a 200 word story that begins with, 'Once upon a time

tokens = prompt_text.split()

types = set(tokens)
ttr = len(types) / len(tokens)

print("Type-Token Ratio (TTR) for Phi-3 Model:", round(ttr,2))
```

```
Type-Token Ratio (TTR) for Phi-3 Model: 0.86
```

Additional Stories

Analysis

The **Phi 3.5** model was by far the best performing model compared to all three. but in a way Phi 3.5 almost acts as the control for this experiment, to ensure that we fully understood how to tweak and modify the hyper parameters of the previous models, which were quite bad at taking direction.

Phi 3.5 was terrific at producing stories, so much so that I asked for specific flavored stories after **instructing** it to add `mystery, and nature themes` this resulted in three scary sounding results, and this made me smile, just in time for Halloween.

As with the other models I modified the `temperature` and `max_length` to ensure there was some creativity and a boundary for the responses being 200 words, as previously mentioned I was very impresses with the results. the Phi 3.5 model was able to produce not supprisingly a **much lower perplexity** indicating the model had a greater certainty on which tokens would satisfy my prompts. the prompt for this model was much larger and this might be a factor in the reduced **TTR** of `0.86` . However this didn't really adversely affect the resulting model's response. it was coherent and often when read slowly in the dark quite spooky.

Q3 Investigate the Relationship between your quality metrics (type-token ration and perplexity) and the size of the models. What Trends do we notice? Discuss the implications of these findings in terms of model scalability and performance

Overaly I think the size of the models is important to consider for scalability. The larger llama 3.2 models with parameters of 1 and 3 billion weren't instructional models but still didn't perform the best for story telling. the Phi 3.5 mini model is also a decoder specific model likely less performance heavy as it will only decode and not encode-decode the resulting tokens. After the experiment for all three models was conducted the **TTY** and **Perplexity** scores for the Phi3.5 model were incredible compared to the LLama 3.2 models. However, this was slightly intentional, as the LLama 3.2 models weren't the best at taking the general prompt of 'Once Upon a time' without some modifications to the `temperature` and the modifications to the `token probabilities` through `top_k` and `top_p` to ensure creative responses. Phi 3.5 is advertised as a model that is "able to be deployed on low powered systems" it is described as a simple solution for mobile devices and low energy computers, and with it's compatability with **Retrieval Augment Generation** it could be a great model for scalable mobile applications or chatbots that will need reasoning skills with low resource utilization, and the ability to access and reason with external documenation.

Q4 The Phi-3.5 Model includes "Instruct" its name. Did you notice any differences in the generated text compared to the other models? Did you have to modify your approach for this model?

The Phi-3.5 is called an "Instruct" model because it has gone through additional training to take instructions from users in the form of prompts. It was easy to notice the differences from the output of the llama models after changing the prompts to reflect this difference. For the Llama models I didn't provide direction for the style of the story as it would derail the whole experiment. for Phi 3.5 instead of taking the phrase *"Once upon a time"* I instructed it to create the 200 word story, and this resulted in a much more clear response that was even in the style I chose, for the three responses I asked it to generate something very spooky to be relatively on point with the halloween season. This was the prompt I decided was the most fun, *"Generate a 200 word story that begins with, 'Once upon a time' incorporate a nature theme and make it scary, and mysterious"*. I also modfied the *temperature* to `temperature=0.85`

Q5 For your best performing model, how would you improve its performance further? Experiment with a few parameters that control the generation strategy (e.g., sampling or beam search) and/or parameters that manipulate the model output logits (temperature, top-k decoding, top-p decoding, repetition penalty, etc.). Present your results obtained using different settings. Describe what you tried and whether it improved your results. Explain why your results improved or did not improve.

Download Phi3.5 Mini Instruct Model

In [1]:
```
from transformers import AutoTokenizer, AutoModelForCausalLM

phi_35_mini_inst = "microsoft/Phi-3.5-mini-instruct"

phi_tokenizer = AutoTokenizer.from_pretrained(phi_35_mini_inst)
phi_model = AutoModelForCausalLM.from_pretrained(phi_35_mini_inst)
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

Generate a Story with Phi 3.5 - Mini Instruct

In [6]:
```
prompt_text = "Generate a 200 word story that begins with, 'Once upon a time

# Tokenize the input text
```

```python
inputs = phi_tokenizer(prompt_text, return_tensors="pt")

# Generate text
outputs = phi_model.generate(inputs.input_ids,
                             max_length=500,
                             temperature=0.72,
                             top_k=25,
                             top_p=0.85,
                             num_beams=5,
                             do_sample=True,
                             repetition_penalty=1.5
                             )

# Decode the generated text
generated_story = phi_tokenizer.decode(outputs[0], skip_special_tokens=True)

print(generated_story)
```

```
Generate a 200 word story that begins with, 'Once upon a time' incorporate a
nature theme and make it scary, and mysterious.

Once upon a time, in a dense, ancient forest shrouded in perpetual twilight,
the trees whispered secrets as old as time itself. The air was thick with an
eerie silence, broken only by the occasional rustle of unseen creatures skul
king in the underbrush. A chilling breeze carried the scent of damp earth an
d decay, hinting at the presence of something sinister lurking just beyond t
he veil of shadows.

In the heart of this foreboding wilderness stood a dilapidated, gnarled tree
, its twisted branches reaching out like skeletal fingers towards the moonli
t sky. Legends spoke of a malevolent spirit residing within its hollow trunk
, feasting on the souls of those who dared to venture too close.

One fateful night, a group of intrepid explorers, driven by curiosity and a
thirst for adventure, stumbled upon this cursed grove. Ignoring the warnings
etched into their memories, they approached the ominous arboreal sentinel.

As they drew nearer, the atmosphere grew increasingly oppressive, suffocatin
g their senses with a palpable dread. Suddenly, a low, guttural growl echoed
through the forest, sending shivers down their spines. The ground trembled b
eneath their feet, and the very air seemed to vibrate with an otherworldly e
nergy.

The explorers turned to flee, but before they could take a single step, the
ground gave way beneath them, plunging them into darkness. When they regaine
d consciousness, they found themselves trapped within the tree's hollow, sur
rounded by an eerie luminescence emanating from within.

Fear gripped their hearts as they realized they were not alone. Shadows danc
ed along the walls, forming grotesque shapes that seemed to mock their feebl
e attempts at escape. The air grew colder, and a sense of impending doom hun
g heavy in the confined space.

With each passing moment, the expl
```

Measure Perplexity

In [10]:
```
import torch

# Extract the Logits from the Model based on the Inputs
with torch.no_grad():
    outputs = phi_model(inputs.input_ids)
    logits = outputs.logits

# shift the input_ids to the right to determine the next token for the model
shift_logits = logits[:, :-1, :].contiguous()
shift_labels = inputs.input_ids[:, 1:].contiguous()

# calculate the log likelihood based on Cross EntropyLoss
loss_fct = torch.nn.CrossEntropyLoss(ignore_index=phi_tokenizer.pad_token_id
# determine the loss value to exponentiate
loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.v

# exponentiate the loss
perplexity = torch.exp(loss)

# return the results
print(f"Perplexity for Phi-3 Model: {round(perplexity.item(),2)}")
```

```
Perplexity for Phi-3 Model: 18.66
```

Measure Token Type Ratio

In [9]:
```
prompt_text = "Generate a 200 word story that begins with, 'Once upon a time

tokens = prompt_text.split()

types = set(tokens)
ttr = len(types) / len(tokens)

print("Type-Token Ratio (TTR) for Phi-3 Model:", round(ttr,2))
```

```
Type-Token Ratio (TTR) for Phi-3 Model: 0.86
```

By adding these parameters such as `num_beams=5` this increased the results significantly in a qualitiative way. The diction style and format of horror is very obvious in this short section of the story. The story contains common pacing and thematic authenticity highlighting ominous noises, mist, and human emotions such as *heavy hearts*.

With the addition of the increased `beams` the heuristics approach is used to choose the tokens ensuring the mystery, nature and horror elements were retained and it overall performed very well.

After testing the results of this approach a few times I found it necessary to increased the word count to determine how frightening these stories could be. Although they don't have any ending, often they're left with a cliff hanger of suspense, moderately appropriate for this genre.

I increased the creativity by reducing the `temperature to 0.72 and top_k to 25` and `top_p` to `0.85` to limit the possible token counts and decreased the top probabilites and the `perplexity` and `TTR` weren't adversely affected.