

# TrashGPT

**Paarth Tandon**

ptandon  
@umass.edu

**Ronan Salz**

rsalz  
@umass.edu

**Ed Almusalamy**

malmusalamy  
@umass.edu

**Cooper Gibbs**

cgibbs  
@umass.edu

## 1 Problem statement

For our semester project, our group wanted to explore the possible applications of NLP in the domain of entertainment. We took inspiration from recent popular projects like AI Seinfeld (Parrish, 2023) and the recent prominence of voice generation models within internet culture (Verma, 2023). Using these ideas as motivation, our group sought to experiment generating episodes of popular podcasts using the latest text generation models. We specifically chose to target the Trash Taste podcast, a podcast focusing on a group of YouTubers who moved to Japan. They cover a wide variety of topics including aspects of nerd culture such as video games and manga, travel, food, and life as immigrants. To this end, we trained a variety of models on transcripts of these episodes as well as experimented with prompting using state-of-the-art models, and evaluated their output on a variety of different content-generation-related tasks.

### 1.1 Codebase

Our codebase can be viewed [here](#). Note that we do not reference specific files in this report, as they are very clearly explained in the README.md that is present in each folder. These include instructions on how to set it up, and how to use it.

The “appendix” that is mentioned throughout the report is also in the codebase.

## 2 What you proposed vs. what you accomplished

Below is a short list of what we proposed to do in our project proposal presented in rough chronological order. Accompanying each item is a statement of whether we accomplished that task, any relevant notes about that task, as well as the corresponding section of the report where more detail can be found.

1. Collect, preprocess, and label podcast data
  - **Accomplished.** See Section 4.
2. Fine-tune GPT-2 Medium on collected dataset
  - **Accomplished.** In addition to GPT-2 Medium, we we also trained GPT-2 Small, Bloom 560M, and LLaMA 7B. See Section 5.
3. Experiment with modern state-of-the-art prompt-focused models
  - **Accomplished.** We performed a small set of trial experiments using prompting through the OpenAI’s GPT-4 API. See Section 5.6 and 6.1.
4. Create an evaluation framework and perform evaluations of all tested models’ generations
  - **Accomplished.** See Section 6.
5. Review evaluations and qualitatively analyze failure cases of our models
  - **Accomplished.** See Section 5 and 7.

## 3 Related work

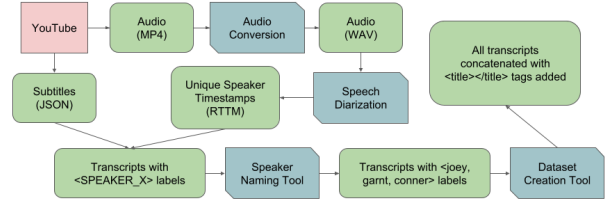
While modern research has not explored automatic content generation in the precise manner we approached for the scope of this project, there has been study into many closely-related topics. Of particular note is recent work focusing on conversation mining and speech similarity. Mitra et al. introduced the Intent and Conversational Mining framework that is intended to assist in the curation, annotation, and labelling of large scale conversational data (Mitra et al., 2022). This framework utilizes a denoising autoencoder trained on the SAMsum data set to provide users labelled summaries of conversation logs. Users can then

comment and correct summaries as needed, outputting labelled data sets to train further models on (Mitra et al., 2022).

Such a framework falls in line with recent papers in the realm of podcast summarization. Because of their unique structure and format, many pre-existing summarization techniques do not maintain their efficacy (Purver et al., 2006). Recent papers have begun to fill in the gaps by focusing on podcast transcriptions, the same kind of data we examine in our proposed work (Nicholls, 2021; Martikainen et al., 2022). A specific approach to podcast summarization has explored the discretization of podcast transcriptions into “transcript chunks” (Song et al., 2022). These chunks have shown to improve summaries by minimizing speech disfluencies and recognition errors which reduces hallucinations in the summary (Song et al., 2022). Models have been trained specifically to assist with this task and identify portions of speech that are markedly “out of character” with the rest of the podcast’s dialogue: Reddy et al. made use of an annotated Spotify data set to train a model capable of identifying portions of podcasts as off-topic or extraneous to an episode’s proposed focus (Reddy et al., 2021). Such off-topic identification has also seen application to other dialogue-based media, such as customer service calls (Zweig et al., 2006).

The exact task of podcast generation itself has actually seen preliminary study, too. Zahoor et al. in their 2021 work combined question answering with extractive text summarization to generate podcasts on-the-fly using text-to-speech engines (Zahoor and A.Sofi, 2021), which clearly shares a great deal of similarities with our proposed work. The primary difference between Zahoor et al.’s approach and ours is in the content of the generated podcast. Zahoor et al. rely on the summarization of web pages to create content, and one issue they found with this approach is the unreliability of the summarization procedure; often they found that the resulting summaries would not include important contextual information (Zahoor and A.Sofi, 2021). Since our work focuses instead on NLG techniques, we did not anticipate running into similar issues.

Finally, we did a brief survey into recent work that focuses on efficient fine-tuning and training of LLMs, as we encountered consistent challenges because of a lack of compute. We considered ap-



Data Pipeline

proaches described in Bakker et al. and Hu et al. such as few-shot fine-tuning and input token prepending for LLaMa, one of our tested models (Bakker et al., 2022; Hu et al., 2023; Zhang et al., 2023).

## 4 Dataset

Our dataset is a collection of transcribed episodes of the Trash Taste Podcast. Each episode is roughly two hours long. Here is a [playlist](#) of each episode that was used in the dataset. The goal of this project was to fine-tune a decoder-based language model on these transcripts. This fine-tuning process will be the same as the original pre-training task, next token prediction. To bring the data into a usable state, it must go through many steps.

### 4.1 Data preprocessing

To better understand how the data is formed, we will step through each stage of the data pipeline. First, we collected both the audio and subtitle data for 100 episodes of the Trash Taste podcast. This was done using a tool called yt-dlp. The next task was to determine at what times each member of the podcast is speaking, so that we can associate their lines with their names in the data. This was done using a technique called “speech diarization”, which takes audio and returns when each unique speaker in the audio is speaking. We took advantage of a premade [implementation](#). While the details of this process are outside the scope of this report, it essentially works by using transformer based models to infer the timestamps. Taking both the raw JSON subtitles from yt-dlp and the calculated timestamps from “speech diarization”, we created a script that generates data like this:

```

<SPEAKER_01> i come back home and i'm like all right </SPEAKER_01>
<SPEAKER_02> and then i just sit on my desk like oh i </SPEAKER_02>
<SPEAKER_02> i get more motivated to get back in the </SPEAKER_02>
<SPEAKER_02> grind i'm like i'm like doubly charged </SPEAKER_02>
<SPEAKER_01> oh no i get that sometimes as well but </SPEAKER_01>
<SPEAKER_01> like if it's especially after like a </SPEAKER_01>
<SPEAKER_00> long because you know we've been on the </SPEAKER_00>
<SPEAKER_03> grind for quite a while now because </SPEAKER_03>
<SPEAKER_00> we're you know preparing for the tour </SPEAKER_00>
  
```

<SPEAKER\_03> yeah you know which i guess by the time </SPEAKER\_03>

Next, we have to manually change each `SPEAKER_X` to the actual name of the host. This is important because we want our models to understand who is speaking. To speed up this process, we wrote a tool that allows a user to quickly listen to a snippet of audio spoken by a given `SPEAKER_X` and provide a manual label for that speaker. After this process, the transcripts look like this:

```
<joe> i come back home and i'm like all right </joe>
<connor> and then i just sit on my desk like oh i </connor>
<connor> i get more motivated to get back in the </connor>
<connor> grind i'm like i'm like doubly charged </connor>
<joe> oh no i get that sometimes as well but </joe>
<joe> like if it's especially after like a </joe>
<garnt> long because you know we've been on the </garnt>
<connor> grind for quite a while now because </connor>
<garnt> we're you know preparing for the tour </garnt>
<connor> yeah you know which i guess by the time </connor>
```

As you can see, each line is now associated with its speaker's name. In this podcast there are three speakers: Joey, Connor, and Garnt. To keep the task from becoming too complex, we excluded any episodes that featured a guest. This exclusion was done before we downloaded the 100 episodes.

Finally, we concatenated all the episodes into one big file (for ease of loading), and incorporated the title of the podcast into the data as well:

```
...
<title>animals we could beat in a fight </title>
<joe> - I'm stuck in the north. </joe>
<joe> Help me out, bro. </joe>
...
```

After which, the file is tokenized by the the chosen model's tokenizer, and is ready to be used for training.

The audio data and speech diarization are also used to condition a voice cloning model that will allow us to generate audio performances of the automatically generated text. We chose [tortoise-tts](#) as our voice cloning model. This is also a transformer based model, but as before with speech diarization, the mechanisms of this model are outside the scope of this report.

## 4.2 Data annotation

It was mentioned earlier that we created a tool that allows the user to easily label each unique speaker with the correct host. We found that creating this tool drastically sped up the manual labelling process. What originally took five to seven minutes per episode could be done in under one minute after some practice. It worked by using the timestamps from the speech diarization process to collect three of the longest audio clips from each unique speaker. The user can then listen to those clips, and type in the name of the speaker

they hear. The tool then automatically replaces that speaker's ID with the host's name. We experimented with using automatic methods to do this naming process, but none of them seemed promising. Using a service like Amazon Mechanical Turk was unreasonable as well due to costs and time.

## 4.3 Data quality

One can observe that our data pipeline has many steps that take a stochastic approach to preprocessing. In this section, we will discuss how this affected the quality of the data.

First of all, not all of the subtitles on YouTube were written by humans. Roughly half the dataset is comprised of YouTube's automatic subtitles. This in itself incorporates questionable interpretations of what the speaker was actually saying. In the future, it may be worth exploring tools such as OpenAI Whisper for automatic subtitles.

Next, the speech diarization process was far from perfect. There are many cases where it performs poorly. For example, if the hosts constantly speak over each other during an episode, the timestamps for those portions will not be accurate. This often leads to situations where the hosts complete each other's sentences like this example:

```
<connor> - But what's more terrifying
in my opinion than a blizzard </connor>
<joe> is the fact that in a dust storm </joe>
<joe> it can cause static electricity. </joe>
<joe> So basically you have thunder
strikes at ground level </joe>
```

In actuality, Joey spoke the line labelled as Connor, but since Connor was speaking right before Joey, so there was an error in the diarization process. While this phenomenon is not overwhelmingly present in the dataset, it does add some unwanted noise. As seen later, these artifacts appear in the resulting generations as well.

## 5 Methodology

Throughout this project, we took advantage of the extensive tools written for HuggingFace to train our models and generate text. This mainly included quantization tools ([peft](#), [bitsandbytes](#)) and their [training API](#). To prepare our dataset for training, we wrote custom PyTorch dataset definitions. This dataset definition provides a sample from the dataset of a length equal to the context size of the chosen model. This is to maximize training efficiency.

It is worth noting that we trained each model on the complete 100 episode dataset. To test the

models with data they had not seen yet, we used episodes that came out after the creation of the dataset. Since one episode is two hours long, it provides more than enough text to test with.

## 5.1 Decoding Strategy

We tested many different decoding strategies while working on this project. This included experimenting with the temperature, top-p, and top-k parameters, beam search, and a repetition penalty. The repetition penalty was interesting to work with, as our generations need to follow a strict format using the named tags. This means that if we set the penalty too high, the models will stop generating the tags, as they are marked as repetitious. In the future, it might be interesting to explore how the decoding strategies work when the tags are ignored in the penalty calculation. Here is the best decoding strategy we found, which we used on all the models:

```
sample_output = model.generate(
    input_ids=input_ids,
    do_sample=True,
    max_length=full_context_len,
    temperature=0.7,
    repetition_penalty=1.005
)
```

Note that the max length is determined by the context length of the model we are testing.

## 5.2 Baseline Approach (GPT-2 Small)

Our baseline model was GPT-2 Small. We chose this model as it was easy to run on our local computer (RTX 3060 12GB). This allowed for quick testing and debugging. The model was trained on the task of next token prediction on our dataset. Here are the hyperparameters chosen:

```
TrainingArguments(output_dir=model_dir,
    num_train_epochs=2,
    logging_steps=100,
    save_steps=5000,
    per_device_train_batch_size=2,
    warmup_steps=10,
    weight_decay=0.05,
    logging_dir=model_dir,
    report_to='none')
```

As you can see, we are using the HuggingFace Trainer to train our model. We found that it is a useful and easy to use training scheme. By default it uses the AdamW optimizer with an initial learning rate of  $5e-5$ . We chose to leave it as is (along with other good defaults suggested by HuggingFace), as hyperparameter tuning was not a large focus of our project. It is worth noting that we only trained it for two epochs. This was to keep it consistent with the larger models that we are training later, as we only had time to train them for two epochs.

After training was complete, we test all our models in a few different ways. Since there are not any applicable quantitative measures for our task, we use a qualitative evaluation framework. This is discussed later in the report.

First, to see how well the model understands a given context, we give it a sample of the test episode as a prefix, and allowed the model to continue the sample. Here is an example.

Prefix text (real text from the podcast):

```
<connor> it's just like it's it's very fun taking </connor>
<connor> them around but after like like multiple </connor>
<connor> days of this I've been getting this same </connor>
<connor> question I'm like I'm gonna lose </connor>
<connor> my mind being asked these questions over </connor>
<connor> and over again but it's been fun they </connor>
<connor> love the bullet trains everyone seems to </connor>
<connor> love bullet trains as well every foreign </connor>
<joe> person I think people from outside Japan </joe>
<joe> just like find that concept so insane </joe>
<joe> yeah because it's like well like I think </joe>
```

Real continuation:

```
<joe> like China is the only other country </joe>
<joe> that has trains that go that fast yeah I </joe>
<connor> think I think no Europe has high speed </connor>
<joe> rails does it yeah I don't think it's as </joe>
<garnt> well connected nearly as well connected </garnt>
<joe> yeah or run on time or because I think </joe>
<connor> like a million people a day use the </connor>
<garnt> shinkansen which is like insane amounts </garnt>
```

GPT-2 Small continuation:

```
<garnt> Japan is very, very small I think a lot of </garnt>
<garnt> the big cities in like Southeast Asia </garnt>
<garnt> are very small but you can still have </garnt>
<garnt> like a very traditionalist cuisine </garnt>
<garnt> you have a traditionalist culture as well </garnt>
<connor> and people are just like oh yeah </connor>
<connor> okay I'm gonna go into like a Southeast </connor>
<connor> Asian restaurant and I'm like oh my God </connor>
```

Next, we wanted to see how well the model can write about a certain topic. To inject a topic into the prefix, we used the title tag mentioned earlier. For example, when GPT-2 Small was given the prefix: <title> the differences between japanese and american people </title>, it generated:

```
<connor> and it's just like just the difference </connor>
<garnt> between japanese and american people </garnt>
<joe> right yeah right because like what </joe>
<joe> we're talking about is because </joe>
<joe> we're talking about here's a difference </joe>
<joe> that is so subtle that we don't really </joe>
<connor> know exactly what we're talking about because </connor>
<connor> we're talking about this difference between </connor>
<connor> japanese and american and like what does </connor>
<joe> japanese do to us and what does american do </joe>
<joe> to us and what does american do to us and </joe>
<joe> just like it's kind of like a very </joe>
<joe> different place to live in as well because </joe>
<joe> like this is like like a very different </joe>
<connor> place to live in Japan because </connor>
<connor> like the Japanese people live there </connor>
<connor> but it's kind of like a different </connor>
<connor> place to live in in Japan as well because </connor>
```

Our third and final test is open ended generation. To do this, we gave just the prefix < to the model, and saw what it generated. Here is a sample generation from GPT-2 Small:

```
<connor> - The problem is that you have to put it, </connor>
<joe> you have to put the thing in. </joe>
<connor> And I had to put the thing in. </connor>
<garnt> And I just found it really weird, </garnt>
<garnt> because I was like, I
was just like, "Oh, okay, </garnt>
<connor> what's going on?" </connor>
<garnt> - Like you guys were like,
"Oh yeah, this is the thing." </garnt>
<garnt> - Yeah. </garnt>
```

```

<garnt> - Yeah, so I was like, oh, okay. </garnt>
<connor> Yeah are we gonna have
another episode of Trash Taste? </connor>
<connor> Do we have another episode or not? </connor>
<connor> 'Cause we've got our new
episode, this one's out. </connor>
<connor> So let's wait for
the other one. </connor>
<connor> - Oh, we're gonna do that. </connor>
<connor> - We're gonna do that. </connor>
<connor> - Yeah, I hope so. </connor>
<connor> - But I don't know, </connor>
<connor> when would you like
make a new episode? </connor>
<connor> Maybe on Trash Taste, </connor>
<connor> maybe in like the future? </connor>
<connor> - Probably in the future. </connor>

```

Many more samples can be viewed in the appendix of our report. The samples in the appendix were not cherry-picked and vary greatly in quality. As seen in the example above, Connor completes Garnt's sentence: "because I was like, I was just like, "Oh, okay, what's going on?". This is the model learning the artifacts that already existed in the dataset.

The full qualitative analysis of the generations are discussed in a later section.

After training GPT-2 Small as a proof of concept, we wanted to train larger models. The models we trained are the following: GPT-2 Medium, Bloom 560M, and LLaMA 7B. The resulting generations of these models will not be discussed in this section, as this will focus on the training aspect. Results will be discussed in a later section, and generations can be viewed in the appendix.

### 5.3 GPT-2 Medium

Just like GPT-2 Small, we were able to train GPT-2 Medium on our own hardware (RTX 3060 12GB). We did not need to incorporate any quantization, check-pointing, nor gradient accumulation techniques to train the model. In this configuration it used almost the full VRAM available on the card. Here are the training hyperparameters used:

```

TrainingArguments(output_dir=model_dir,
                  num_train_epochs=2,
                  logging_steps=100,
                  save_steps=5000,
                  per_device_train_batch_size=2,
                  warmup_steps=10,
                  weight_decay=0.05,
                  logging_dir=model_dir,
                  report_to='none')

```

Once again, the learning rate was left as the default value, as it is adjusted during the AdamW optimization process. No other tricks were needed to train the model. In brief, the resulting generations were similar to GPT-2 Small.

### 5.4 Bloom 560M

We chose Bloom, as it was a larger model than GPT-2, and had a newer checkpoint as well. Bloom was also trained on multilingual data, which could give it more information about the

culturally charged discussions that happen on the podcast about Japan. This is the first model that we trained where we had to take advantage of cloud computing. To train Bloom 560M, we subscribed to [Lambda Labs](#), an online platform which provides spot instances with GPUs. The GPU that was used to train Bloom was the A10 with 24GB of VRAM. Here are the training hyperparameters used:

```

TrainingArguments(output_dir=model_dir,
                  num_train_epochs=2,
                  logging_steps=10,
                  # save_steps=100,
                  save_steps=5000,
                  per_device_train_batch_size=1,
                  warmup_steps=10,
                  weight_decay=0.05,
                  logging_dir=model_dir,
                  report_to='none',
                  fp16=True,
                  gradient_accumulation_steps=4,
                  gradient_checkpointing=True)

```

As you can see, we needed to take advantage of certain training tricks to fit the model and its gradients in 24GB of VRAM. This included loading the model in 16-bit precision, performing gradient accumulation, and enabling gradient check-pointing. All of these methods reduce the amount of VRAM that is needed to train the model to roughly 13GB. In brief, the generations were more creative and entertaining, while somewhat less coherent.

Since the context size is doubled from GPT-2 to Bloom (1024 to 2048), we noticed that the model seems to diverge near the end of its generation. For example, we noticed if the model generates the word "ugh", it will continue to do so until the end of the generation. Here is an example of the phenomenon:

```

<garnt> And then I go to Google translate, </garnt>
<garnt> and ugh, I was getting it done. </garnt>
<garnt> - [Joey] Okay. </garnt>
<garnt> - I was like, (coughing). </garnt>
<garnt> And I was like, (coughing). </garnt>
<garnt> - [Joey] ugh. </garnt>
<garnt> - I was like, </garnt>
<garnt> I was like, (coughing), and I was like, </garnt>
<garnt> "Please, please help me." </garnt>
<garnt> ugh, I was like, "What the fuck" </garnt>
<garnt> and like, I was trying to
look ugh in my ugh ugh in ugh. </garnt>
<garnt> ugh, ugh, ugh, ugh, ugh, ugh, ugh, ugh, </garnt>
... ughs continue ...

```

This also happens with some other words like "no", and the suffix "iverse". Examples of these can be found in the appendix. Adjusting the repetition penalty to be higher did not improve the results, so it seems as though these tokens have a very high probability to be generated when they are already present in the context.

### 5.5 LLaMA 7B

The last model we trained was LLaMA 7B. We chose this model, as it is an example of the largest LLM that we could feasibly train on Lambda Labs. Also, since it was trained on more data than the



other models, we expect that its baseline language understanding will be leveraged well in this specific generation task. Once again, we were limited by our compute budget, and used an A10 with 24GB of VRAM. To be able to train LLaMa, we had to take advantage of many training tricks. First, the model had to be quantized using the [bit-sandbytes](#) library. We used 8-bit quantization, as 16-bit was too large. Next, we had to incorporate a LoRA adapter to reduce the VRAM used while training. This was done using the [peft](#) library. Finally, we still had to enable gradient checkpointing and accumulation to be able to train the model in 24GB. We used the following hyperparameters for the adapter and the model:

```
LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=["q_proj", "v_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

TrainingArguments(
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    num_train_epochs=1,
    warmup_steps=100,
    fp16=True, # must be enabled when loading in 8-bit mode
    logging_steps=1,
    output_dir=model_dir
)
```

We did not experiment with different adapters, as training LLaMA for two epochs already took up 30 hours of training time. Training for longer would bode to be too expensive for our group to complete.

LLaMA has a context size of 2048, and similarly to Bloom, LLaMA also generated artifacts at the end of its generations. However, instead of focusing on a specific word like "ugh", it got stuck generating partial tags. Here is an example of the phenomenon:

```
<connor> - I wasn't like, </connor>
<connor> "If I don't get this, I'm gonna be a." </connor>
</ </ </ </ </ </connor </connor </connor>
... partial tags continue ...
```

## 5.6 GPT-4 Exploration

In addition to above approaches, we also tried a small sample of prompting techniques in conjunction with the recently released GPT-4 model via OpenAI's JavaScript API. Given GPT-4's newness and high demand, we encountered several problems that were specific to using this model alone. We were subject to frequent timeouts, a small context window, as well as the associated cost of using the API. With this in mind, we decided to use GPT-4 in only a zero- and one-shot approach as an opportunity to survey and demo the latest NLP

technologies instead of collecting rigorously evaluated data.

In our exploration, we performed three sets of five trials using various prompting techniques. We were first curious about GPT-4's innate knowledge of our dataset, the Trash Taste podcast. Our first set of trials ("BASELINE PROMPTS") was focused around figuring out whether or not Trash Taste had appeared in GPT-4's data at any point. To this end, we formulated a set of zero-shot generative prompts. In these prompts, we would prompt the model with a basic one sentence question that contained as minimal information about Trash Taste as possible. One such prompt only named the podcast without naming the hosts, while another prompt named the three hosts by their online usernames, but did not use their real names.

The next set of trials ("ONE-SHOT CONVERSATION GENERATIONS") was used to observe GPT-4's creativity. We prompted the model with a one-shot generation task to generate conversations about topics that would reasonably appear on the podcast, but did not appear as topics within our training set. Specifically, we prompted the model with a natural language instruction to create a conversation about some Trash Taste-relevant topic, as well as a snippet of conversation from our training data, and directed the model to use the snippet as a reference for tone and style.

The final set of trials ("ZERO-SHOT CONVERSATION EXTENSIONS") was centered around GPT-4's short term consistency in terms of content and style. To examine this, we prompted the model with a zero-shot completion task. Specifically, the model was prompted with the first 10-15 lines of a conversation in our test set, and asked to continue the conversation in the same tone and style as it was being carried out.

View the appendix to read all generated samples.

## 5.7 Voice Cloning

To generate spoken performances of our lines, we wanted to take advantage of a voice cloning model. We used [tortoise-tts](#) for our voice cloning model. It is a transformer based model which conditions on a given set of example audio files, and generates a new file that attempts to imitate the voice provided. Examples of its usage are available in our code base, and here are some examples

of its performance: [Video 1](#), [Video 2](#), [Video 3](#).

While the audio was generated automatically, the images were added after the fact using video editing software.

## 6 Evaluation and Analysis Overview

Based on our discussions with the course staff, using popular quantitative measures would not be very applicable for our task. Because of this, we decided to create a qualitative evaluation framework, which three of our group members completed for a set of generations. This set included generations for three tasks: continuing a given set of lines from the podcast, open-ended generation (no prefix), and prompted generation. Prompting was done by taking advantage of the `<title>` tag. This allowed us to prompt the models like this:

```
<title> the differences between japanese and american people </title>  
... model's generation would follow
```

The annotators rated on three dimensions:

- **Coherence:** Is the next line consistent with the context and semantics of the excerpt? Is the formality of the next line consistent with the excerpt? Is the next line reasonable beyond just being a valid answer? Does it seem like a possible response for a conversation? Do you think that the speaker would actually say something like this?
- **Creativity:** Is the response exciting to the point where it appears to have been created by a human with a diverse imagination?
- **Entertainment (Podcast-specific properties):** Does the next line seem audience-friendly/like something a creator might say in light of the fact an audience will listen? (Ex: appropriate, perhaps clarifying for a layperson, maybe funny, meta-type philosophical, interest-raising, etc.)

Overall, entertainment should be judged due to the podcast-specific criteria, creativity should be a mix of excitement and imagination as described above and coherence is a mix of context, tone and reasonableness. Each generation was graded as an integer between one and five.

### 6.1 Analysis of the Qualitative Evaluations

As aforementioned, we generated responses for three different tasks: complete the passage, creative generations, and prompting with a title. Each

task had five generations to evaluate. There were many interesting results we came across and some surprisingly funny and entertaining generations that were quite captivating.

First, let's analyze how each model performed in each of the three tasks. Note that each cell of the below table is the average score over the five generations amongst the three annotators. Again, the scores were between one and five.

Complete the Passage			
Models	Entertain	Creativity	Coherence
GPT2 S	2.8	2.47	<b>2.73</b>
GPT2 M	2.6	2.47	2.47
Bloom	2.93	<b>3.0</b>	1.8
LLaMA	<b>3.27</b>	<b>3.0</b>	2.27

As we can see, the scores were fairly low with most results coming in less than three. We think the subpar results have to do with the difficulty in generation text that abides by and follows up on the passage given. Attempting to not contradict the former statements and to extend the thoughts creates quite a challenging feat. On the other hand, there were some promising models such as LLaMA which led the way for both entertainment and creativity with a decent coherence score.

Here is a particularly interesting and realistic generation by LLaMA:

"Garnt: 'yeah you know I mean they don't have a lot of crime in Japan yeah' Joey: 'you know it's a lot of like the people that do crime they're usually foreigners...'" Another interesting takeaway is that GPT-2 Small performed better than GPT-2 Medium. In fact, there was a lot less hallucination and GPT-2 Small seemed to pick up some common utterances in the word "like" which is commonly said by these hosts as they talk.

Next, let's look at how our different models perform on creative generations.

Creative Generations			
Models	Entertain	Creativity	Coherence
GPT2 S	2.87	3.0	2.53
GPT2 M	2.13	3.33	2.53
Bloom	3.47	3.6	2.93
LLama	<b>3.87</b>	<b>3.93</b>	<b>3.07</b>

As we can see, the results are generally much stronger here than in the passage completion, which is likely due to there being less restrictions in order to create a logical, coherent generation.

Also, we observe the LLaMA is the star performer by far with the best average results in all three evaluation criteria. This is not too surprising considering the previous results where LLaMA

was at the front of the pack especially in terms of entertainment and creativity. Here is a demonstrative excerpt from Llama’s creative generation:

“Garnt: ‘I mean, I think that’s why I really enjoyed Tetris. I really enjoyed playing it as a kid. I think it’s just because it was just fun. Yeah, it’s fun.’ Connor: ‘It’s a game where you don’t have to think. You just have to just do the thing. Yeah, and it was a lot of fun.’ Joey: ‘I think I learned to play it first in the arcades and I remember the only thing I could do. I would just keep going until I ran out of lives, and then I’d just go back to the beginning.’”

This generation seems like it could be a real conversation! On top of this, everything that was said is very agreeable and entertaining. This goes to show the power of LLaMA and backs its immense success in this category and in general.

Next, let’s look at how the models performed when prompted with a title. In particular, “What are the differences between Japanese and Americans?”

Note that LLaMA only had two generations below whereas the others had the usual five. This was due to running out of time and money to host the cloud instance.

Title Prompt			
Models	Entertain	Creativity	Coherence
GPT2 S	2.13	2.07	1.67
GPT2 M	2.53	2.53	<b>2.33</b>
Bloom	2.87	2.93	2.07
LLaMA	<b>3.17</b>	<b>3.0</b>	<b>2.33</b>

As with complete the passage we are seeing fairly low scores which is once again likely due to the constraints a prompt puts on the output text. Coherence is usually the lowest when this constraint exists which could be because the model gets confused trying to match the prompts request and in turn it lacks in its coherence and becomes fairly nonsensical.

Unsurprisingly, LLaMA leads the way once again with pretty solid performance with its lowest score being in the coherence category. In face, all four models have the lowest score in coherence likely due to the explanation above.

Here is an except from LLaMA’s generation:

“Joey: ‘um so i thought it would be interesting to compare the two cultures and i know that there’s a lot of people who are like well you know japanese and american culture is very similar like i mean i know like it’s’ Garnt: ‘a lot of people who are like oh yeah like we’re the same people yeah

we’re the same people yeah we’re the same people...’ ”

This excerpt shows how LLaMA is able to use this title prompt and actually attempt to juxtapose the cultures which is quite impressive. The other models would not even begin to do a comparison and most generations were completely off track and full of hallucinations and the like.

This goes to show the relative strength of LLaMA once again.

Along with these highlights there were many, many excerpt which did not make any sense. Many of these errors including repeating the same thing over and over and veering very far off topic into a pit of nothingness. As mentioned a few times before, performing under conditions degraded the quality of the responses such as when a title was supplied and when asked to continue a passage.

View the appendix to read all generated samples.

## 6.2 GPT-4 Qualitative Analysis

Our initial set of trials (“BASELINE PROMPTS”) seemed to indicate that GPT-4 has seen information relating to Trash Taste and its hosts. When prompted to “create a conversation that could occur on Trash Taste” with no further context provided, GPT-4 correctly named all three hosts of the podcast by both their real and screen names. It also correctly inferred the subject matter of the podcast, as all generations are consistently about anime-related topics. One notable omission is that GPT-4 did not ever name the podcast if it was not provided; when prompted to “generate a conversation between Connor, Garnt, and Joey on their podcast,” the words “Trash Taste” never appeared. While we lacked the time and resources to probe the model further in this regard, further prompt engineering could be focused on testing the model’s recall of the podcast title. Nevertheless, given the provided responses, we feel confident that GPT-4 has encountered both the podcast as well as each individual host of the podcast.

We were fairly impressed with GPT-4’s creative generations in our second set of trials (“ONE-SHOT CONVERSATION”). It maintained the topic and tone of the podcast effectively, and the generated conversations had clear grounding in the podcast’s theme of anime and nerd culture. It also perfectly mimicked the specific format of the ex-



ample conversation snippet. The one aspect we found lacking was the conveyance of style and the “human” aspect of the resulting generations. We felt that GPT-4’s responses had an element of cleanliness and sterility that was not found in the source text. Language was softened and opinions were expressed in safer manners, which caused clear discrepancies with the source text.

We observed similar behavior in the third set of trials (“ZERO-SHOT CONVERSATION EXTENSIONS”). GPT-4 had coherent and logical extensions of the provided conversation snippet; it exhibited clear understanding of what the topics of each conversation was about, and had no difficulty creating more relevant dialogue. In these extensions, though, the same stylistic sterility was observed. There was marginally more character and personality in the extensions compared to the one-shot conversation generations, but this component was still noticeably lacking overall.

View the appendix to read all generated samples.

## 7 Contributions of group members

Below is a rough list of contributions by teammate. While not listed below, all teammates contributed equally to report writing for both the proposal and final draft. All group members were satisfied with the work done by all other group members, and are confident that every individual did their fair share of the work.

- Paarth: Defined the goals of the project. Researched methods to attempt. Organized road-map of tasks to complete. Created data pipeline. Wrote code to train and test models. Worked with Lambda Labs.
- Ronan: Surveyed current state of the art to see if any pre-existing models had been fine-tuned for the task of podcast generation; headed interaction with GPT-4, including prompt engineering and analysis; evaluated generations from four primary models.
- Ed: Explored text-to-speech frameworks to use the generated text and make a podcast from it, created a tool to help automate labeling data, helped label the data, evaluated the quality of the generated text
- Cooper: Created average error table for models/ generation categories and analyzed re-

sults; related work research on fine-tuning LLM’s; scored generations

## 8 Conclusion

One of the main takeaways is that these relatively small LLMs performed fairly well and in some case much better than we thought. In many case the results were fairly reasonable and matched the correct output format. LLaMA proved to be the strongest model in general and excelled in creative and entertaining generations. The fact that we used a quantized version of LLaMA leads us to wonder how much stronger (or [possibly] worse) the results could have been if we used the non-quantized or larger versions of LLaMA. GPT-2 Small, GPT-2 Medium and Bloom all performed decently, but a bit below what we expected. They were all roughly the same in terms of quality with no clear superior.

If given the opportunity to continue our project, we would have explored the utilization of non-quantized versions of LLaMA, in addition to experimenting with larger language models. This approach aims to address the limitations observed in the models we have trained. Also, we believe that we could have much stronger results if the dataset was professionally transcribed, instead of using automated methods. The transcript schema we developed could be improved as well. For example, there is no way to correctly transcribe if two or more speakers are talking over each other. Despite this, we believe that our work has the potential to extend into the domain of automatic podcast generation.

In conclusion, we have developed a preliminary pipeline capable of generating new transcripts and audio for a given podcast, based on its existing audio and transcripts. With more improvements to each aspect of the pipeline, we believe that this could potentially be a practical tool in the future for creative generations.

## 9 AI Disclosure

- Did you use any AI assistance to complete this proposal? If so, please also specify what AI you used.
  - No. We did not use AI to write any code or prose for this report.

## References

- Bakker, M. A., Chadwick, M. J., Sheahan, H. R., Tessler, M. H., Campbell-Gillingham, L., Balaguer, J., McAleese, N., Glaese, A., Aslanides, J., Botvinick, M. M., and Sumnerfield, C. (2022). Fine-tuning language models to find agreement among humans with diverse preferences.
- Hu, Z., Lan, Y., Wang, L., Xu, W., Lim, E.-P., Lee, R. K.-W., Bing, L., Xu, X., and Poria, S. (2023). Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models.
- Martikainen, K., Karlgren, J., and Truong, K. (2022). Exploring audio-based stylistic variation in podcasts. In *Proc. Interspeech 2022*, pages 2343–2347.
- Mitra, S., Ramnani, R., Ranjan, S., and Sengupta, S. (2022). ICM : Intent and conversational mining from conversation logs. In *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 403–406, Edinburgh, UK. Association for Computational Linguistics.
- Nicholls, A. S. (2021). A neural model for text segmentation.
- Parrish, A. (2023). What’s the deal with this AI seinfeld stream? — theverge.com. <https://www.theverge.com/23581186/ai-seinfeld-twitch-stream-chatgpt>. [Accessed 7-April-2023].
- Purver, M., Kording, K. P., Griffiths, T. L., and Tenenbaum, J. B. (2006). Unsupervised topic modelling for multi-party spoken discourse. In *Annual Meeting of the Association for Computational Linguistics*.
- Reddy, S., Yu, Y., Pappu, A., Sivaraman, A., Rezapour, R., and Jones, R. (2021). Detecting extraneous content in podcasts. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1166–1173, Online. Association for Computational Linguistics.
- Song, K., Li, C., Wang, X., Yu, D., and Liu, F. (2022). Towards abstractive grounded summarization of podcast transcripts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4407–4418, Dublin, Ireland. Association for Computational Linguistics.
- Verma, P. (2023). Their voices are their livelihoods. now ai is taking that away.
- Zahoor, S. and A.Sofi, S. (2021). Automatic podcast generation. *Journal of University of Shanghai for Science and Technology*.
- Zhang, R., Han, J., Zhou, A., Hu, X., Yan, S., Lu, P., Li, H., Gao, P., and Qiao, Y. (2023). Llama-adapter: Efficient fine-tuning of language models with zero-init attention.
- Zeig, G., Siohan, O., Saon, G., Ramabhadran, B., Povey, D., Mangu, L., and Kingsbury, B. (2006). Automated quality monitoring for call centers using speech and NLP technologies. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Demonstrations*, pages 292–295, New York City, USA. Association for Computational Linguistics.