1.Who are you (mini-bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

My real name is 刘欣迪 (Xindi Liu), but I usually use my online English name Dylan Liu. I'm a freelance programmer (AI related) with 7 years of experience. One of my main incomes now is prizes from data science competition platforms (Topcoder, Kaggle and DrivenData, although I have no income yet on Kaggle, only a solo gold medal).

2.What motivated you to compete in this challenge?

I would love to participate in various competitions involving deep learning, especially tasks involving natural language processing or LLM . At the same time, competition prize money is also one of my main incomes.

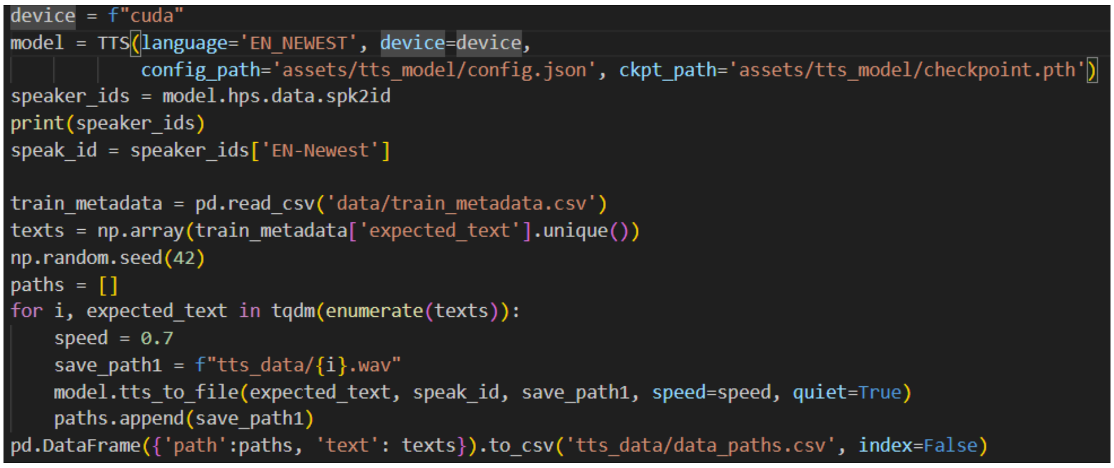
3.High level summary of your approach: what did you do and why?

At first, I treated this task as a multimodal (speech and text) classification task. I tried to use a speech feature extraction model and a text feature extraction model to extract the features of speech and text respectively, and then used a classifier for binary classification. After trying different models, I chose whisper-medium (only the encoder part) as the speech feature extraction model and bge-large as the text feature extraction model. The cv (cross validation) score of this solution was ~0.32.
Then I tried to use the cosine similarity of the two features as the binary classification result of the model. The cv score of this solution was ~0.3.
I spent some time thinking about the room for improvement: text information has no pronunciation details, how to get the accurate pronunciation of text? I thought of using a text-to-speech (TTS) model to convert text into standard speech, and then compare it with speech data.
I used the EN_NEWEST model of MeloTTS to generate standard speech, and then trained a model with it according to the previous method. The cv score of the classifier solution was ~0.28, and the cv score of the cosine similarity solution was ~0.26.
In my experience, putting all the information in one model input works best. So I tried combining the speech data and the standard speech into a new unified language input, and treated this task as a speech binary classification task. This solution achieved a cv score of ~0.24.
I then tried different data augmentations, which only slightly affected the cv score, reducing it by ~0.01. At this time, the lb (leaderboard) score was ~0.228. I achieved this score about a month before the end of the competition.
For the rest of the competition, I did not succeed in improving the model's performance. I tried various speech feature extraction models + various hyperparameters, and tried to generate new data with the TTS model, but unfortunately these methods did not significantly improve the model results.
In the end, I combined the 10 models from the previously trained models and used a cv-based weight searching script to automatically search for weights. Due to the limitation of inference time, each model only used half of it's folds, thus forming the final submission.

4.Do you have any useful charts, graphs, or visualizations from the process?

t the beginning, I made some simple visualizations: label distribution, grade distribution, data type distribution, and data length distribution.

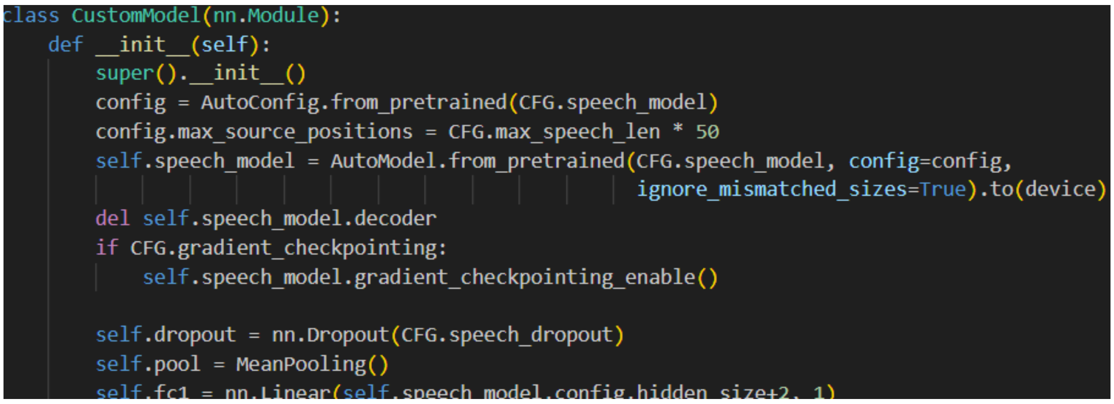
5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.



```python
device = f"cuda"
model = TTS(language='EN_NEWEST', device=device,
            config_path='assets/tts_model/config.json', ckpt_path='assets/tts_model/checkpoint.pth')
speaker_ids = model.hps.data.spk2id
print(speaker_ids)
speak_id = speaker_ids['EN-Newest']

train_metadata = pd.read_csv('data/train_metadata.csv')
texts = np.array(train_metadata['expected_text'].unique())
np.random.seed(42)
paths = []
for i, expected_text in tqdm(enumerate(texts)):
    speed = 0.7
    save_path1 = f"tts_data/{i}.wav"
    model.tts_to_file(expected_text, speak_id, save_path1, speed=speed, quiet=True)
    paths.append(save_path1)
pd.DataFrame({'path':paths, 'text': texts}).to_csv('tts_data/data_paths.csv', index=False)
```

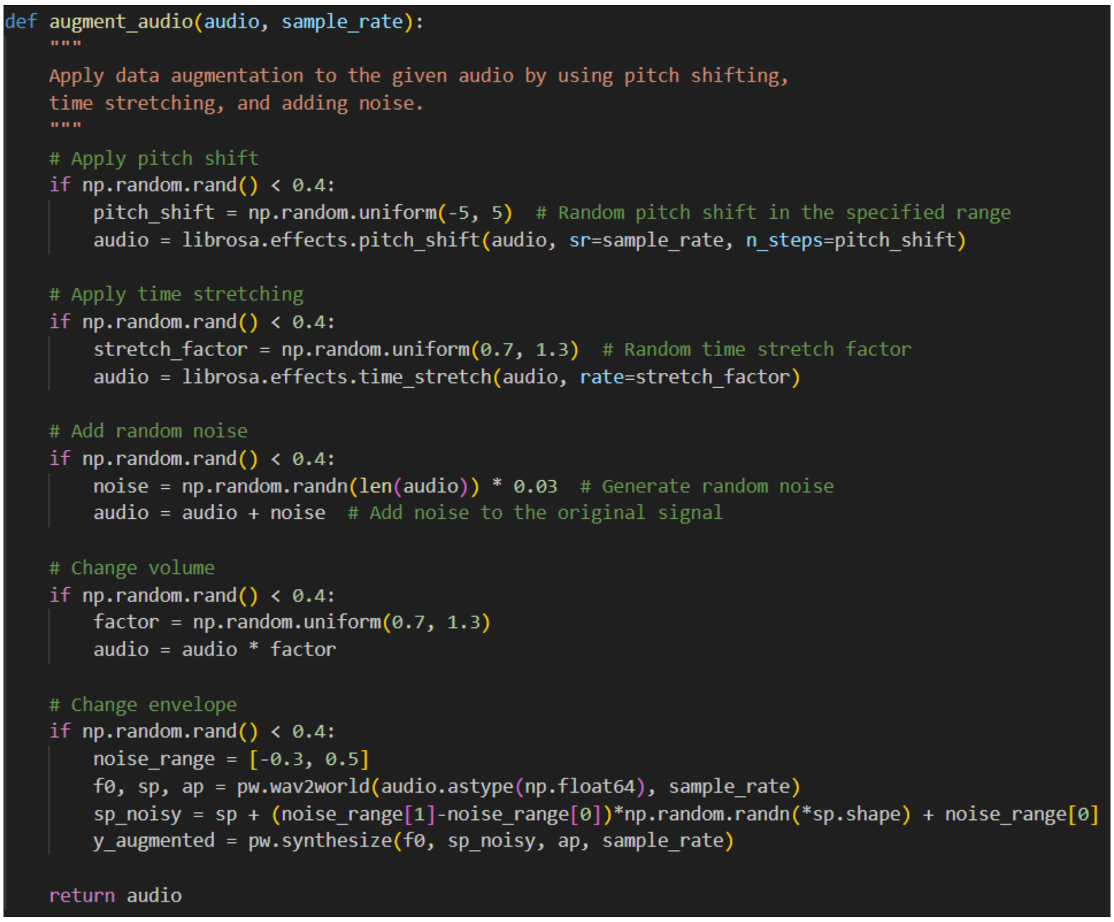
This section is where TTS generates label data. Speech labels enable the model to distinguish speech details, thus greatly improving model performance.



```python
class CustomModel(nn.Module):
    def __init__(self):
        super().__init__()
        config = AutoConfig.from_pretrained(CFG.speech_model)
        config.max_source_positions = CFG.max_speech_len * 50
        self.speech_model = AutoModel.from_pretrained(CFG.speech_model, config=config,
                                                      ignore_mismatched_sizes=True).to(device)
        del self.speech_model.decoder
        if CFG.gradient_checkpointing:
            self.speech_model.gradient_checkpointing_enable()

        self.dropout = nn.Dropout(CFG.speech_dropout)
        self.pool = MeanPooling()
        self.fc1 = nn.Linear(self.speech_model.config.hidden_size+2, 1)
```

This is the definition of the model. The original model input length was 30s and had a decoder part, but this competition did not require such a long input. Too long input would obviously slow down the model operation speed, and a decoder was not needed. Based on the distribution of data length, I set the input length to 16s and deleted the decoder part of the model to speed up the model running, and this allowed me to integrate more models.



```python
def augment_audio(audio, sample_rate):
    """
    Apply data augmentation to the given audio by using pitch shifting,
    time stretching, and adding noise.
    """
    # Apply pitch shift
    if np.random.rand() < 0.4:
        pitch_shift = np.random.uniform(-5, 5)  # Random pitch shift in the specified range
        audio = librosa.effects.pitch_shift(audio, sr=sample_rate, n_steps=pitch_shift)

    # Apply time stretching
    if np.random.rand() < 0.4:
        stretch_factor = np.random.uniform(0.7, 1.3)  # Random time stretch factor
        audio = librosa.effects.time_stretch(audio, rate=stretch_factor)

    # Add random noise
    if np.random.rand() < 0.4:
        noise = np.random.randn(len(audio)) * 0.03  # Generate random noise
        audio = audio + noise  # Add noise to the original signal

    # Change volume
    if np.random.rand() < 0.4:
        factor = np.random.uniform(0.7, 1.3)
        audio = audio * factor

    # Change envelope
    if np.random.rand() < 0.4:
        noise_range = [-0.3, 0.5]
        f0, sp, ap = pw.wav2world(audio.astype(np.float64), sample_rate)
        sp_noisy = sp + (noise_range[1]-noise_range[0])*np.random.randn(*sp.shape) + noise_range[0]
        y_augmented = pw.synthesize(f0, sp_noisy, ap, sample_rate)

    return audio
```

This part is data augmentation. Although data augmentation did not significantly improve model performance, different data augmentations can improve the diversity of the model to some extent and can also improve the performance of model ensemble.

6.Please provide the machine specs and time you used to run your model.

●CPU (model): intel i7
●GPU (model or N/A): NVIDIA 4090
●Memory (GB): 48GB
●OS: Linux
●Train duration: ~60h
●Inference duration: ~3h55m (on the competition's virtual machine)

7.Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

When I trained different models, there was a small chance (about 2%) that the gradients would vanish each time I trained a new model. When the gradients vanish, you can clearly see that the gradients become very small (compared to the same step of other folds), the model almost stops converging, and eventually the cv of this fold becomes very bad. This phenomenon occurs very randomly and has nothing to do with the random seed, and it can be solved by retraining the faulty fold without any modification.

My current solution is to retrain the faulty fold. Although the models I chose did not experience gradient vanishing, when you check the cv results of the model, if the loss details of a fold are higher than the loss of other folds, such as the loss of a fold is greater than 0.28, and the loss of other folds is 0.24-0.22, you need to retrain this fold using the above method. Hopefully this will not happen.

8.Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

When looking for different solutions, I also used other TTS models to analyze the impact of different TTS models on the results. They are all MeloTTS models, but different versions and different accents.

9.How did you evaluate performance of the model other than the provided metric, if at all?

Inference Speed. Under the premise of achieving the same score, the shorter the inference time, the better. Shorter inference time means more models in the ensemble.

10.What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

I tried to generate different labels with different TTS for data augmentation. This hurt the results slightly.
I tried to generate more data with Faker library combined with the TTS model, but it did not improve the model.
Filtering the training data based on training loss (i.e. removing data with very large loss because these data are mislabeled) did not help the results at all, even if these data were obviously mislabeled.
I tried wav2vec2 and other speech/audio models. In theory, wav2vec2 for speech recognition is also a good speech feature extraction model, but its performance was much worse than whisper. At first I thought I didn't find the right hyperparameters, but after wasting a lot of time I gave up wav2vec2 and other models.
I tried to use the audio encoder part from Qwen2-audio-7B as a speech feature extraction model, but its performance was not good, even worse than wav2vec2.

11.If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

I will try audio LLM or speech generation models such as Qwen2-audio-7B. The current means seem to have reached a bottleneck, and to break through the bottleneck requires a larger model or more data.

12.What simplifications could be made to run your solution faster without sacrificing significant accuracy?

My model is already very fast that 1 model/1 fold takes only 10-15min to finish submission test. If you need a faster solution, you just need to train fewer models and do more hyperparameter tuning to get an efficiency solution.