****

**دانشگاه آزاد اسلامی**

**واحد علوم و تحقیقات**

**دانشکده برق و کامپیوتر و مکانیک**

**گزارش کار پروژه نهایی درس گفتار پردازی رقمی**

**موضوع مدل Pre-trained فارسی در ASR**

**شرح مدل XLSR-Wav2Vec2**

**گردآورنده: صبا حصارکی**

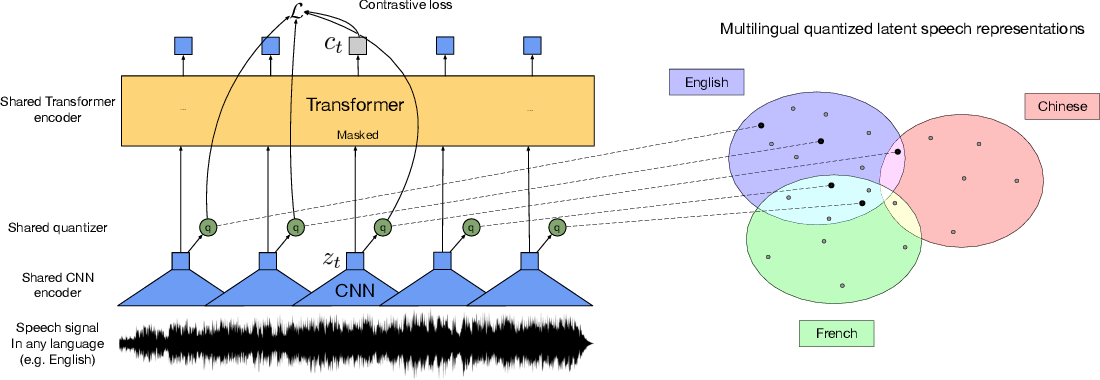
**استاد: دکتر کوچاری**

**بهمن 1401**

**معرفی مدل هایPre-trained شده فارسی برای ASR**

**Fine-tune XLSR-Wav2Vec2 برای ASR چند زبانه باtransformer**

Wav2Vec2 یک مدل از پیش آموزش دیده برای تشخیص خودکار گفتار (ASR) است و در سپتامبر 2020 منتشر شد. بلافاصله پس از اینکه عملکرد برتر Wav2Vec2 در مجموعه داده انگلیسی ASR Libri Speech نشان داده شد، هوش مصنوعی فیس بوک XLSR-Wav2Vec2 را ارائه کرد. XLSR مخفف بازنمایی گفتار بین زبانی است و به توانایی XLSR-Wav2Vec2 برای یادگیری بازنمایی گفتاری که در چندین زبان مفید است اشاره دارد.

****شبیه به Wav2Vec2، XLSR-Wav2Vec2 بازنمایی گفتار قدرتمندی را از صدها هزار ساعت گفتار در بیش از 50 زبان گفتار بدون برچسب می‌آموزد. مشابه مدل‌سازی زبان ماسک‌شده BERT، مدل بازنمایی‌های گفتاری متن‌شده را با mask تصادفی بردارهای ویژگی قبل از ارسال آن‌ها به شبکه transformer می‌آموزد.

**Wav2Vec2-Large-XLSR-53-Persian**

Fine-tune facebook/wav2vec2-large-xlsr-53 به زبان فارسی از دیتاست common voice استفاده می کند. هنگام استفاده از این مدل، باید که ورودی گفتار در فرکانس 16 کیلوهرتز نمونه برداری شده باشد.

**قطعه کد برای این مدل پیش آموزش**

ابتدا کتابخانه های مهم را ایمپورت می کنیم

|  |
| --- |
| # requirement packages !pip install git+https://github.com/huggingface/datasets.git !pip install git+https://github.com/huggingface/transformers.git !pip install torchaudio !pip install librosa !pip install jiwer !pip install hazm |

**پیش بینی مدل**

تابع normalizer: بعضی از کاراکتر های regex را نادیده می گیرد و نرمالیز کرده و کاراکتر های ویژه را حذف می کند

تابع character mapping: نگاشت کاراکتر های یونیکد به کاراکتر های فضای خالی و کاراکتر های حرف بزرگ به حروف کوچک در زبان فارسی

تابع speech file to array: صدا را لود کرده و در یک آرایه ریخته و فشرده می کند سپس دوباره با نرخ نمونه برداری 16khz نمونه برداری مجدد می کند

تابع predict: بعد ازresample که در مرحله ی قبل انجام شد . ویژگی ها را استخراج کرده و یک attention mask را به آن اعمال کرده و سپس decode را انجام داده و مدل Wav2Vec2ForCTC را به مدل pre-trained اعمال می کنیم

|  |
| --- |
| import librosa import torch import torchaudio from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor from datasets import load\_dataset  import numpy as np import hazm import re import string  import IPython.display as ipd  \_normalizer = hazm.Normalizer() chars\_to\_ignore = [     ",", "?", ".", "!", "-", ";", ":", '""', "%", "'", '"', "'",     "#", "!", "؟", "?", "«", "»", "ء", "،", "(", ")", "؛", "'ٔ", "٬",'ٔ', ",", "?",      ".", "!", "-", ";", ":",'"',""", "%", "'", """, "'", "-", "...", "\_", """, '"', '"' ]  # In case of farsi chars\_to\_ignore = chars\_to\_ignore + list(string.ascii\_lowercase + string.digits)  chars\_to\_mapping = {     'ك': 'ک', 'دِ': 'د', 'بِ': 'ب', 'زِ': 'ز', 'ذِ': 'ذ', 'شِ': 'ش', 'سِ': 'س', 'ى': 'ی',     'ي': 'ی', 'أ': 'ا', 'ؤ': 'و', "ے": "ی", "ۀ": "ه", "ﭘ": "پ", "ﮐ": "ک", "ﯽ": "ی",     "ﺎ": "ا", "ﺑ": "ب", "ﺘ": "ت", "ﺧ": "خ", "ﺩ": "د", "ﺱ": "س", "ﻀ": "ض", "ﻌ": "ع",     "ﻟ": "ل", "ﻡ": "م", "ﻢ": "م", "ﻪ": "ه", "ﻮ": "و", "ئ": "ی", 'ﺍ': "ا", 'ة': "ه",     'ﯾ': "ی", 'ﯿ': "ی", 'ﺒ': "ب", 'ﺖ': "ت", 'ﺪ': "د", 'ﺮ': "ر", 'ﺴ': "س", 'ﺷ': "ش",     'ﺸ': "ش", 'ﻋ': "ع", 'ﻤ': "م", 'ﻥ': "ن", 'ﻧ': "ن", 'ﻭ': "و", 'ﺭ': "ر", "ﮔ": "گ",     "\u200c": " ", "\u200d": " ", "\u200e": " ", "\u200f": " ", "\ufeff": " ", }  def multiple\_replace(text, chars\_to\_mapping):     pattern = "|".join(map(re.escape, chars\_to\_mapping.keys()))     return re.sub(pattern, lambda m: chars\_to\_mapping[m.group()], str(text))  def remove\_special\_characters(text, chars\_to\_ignore\_regex):     text = re.sub(chars\_to\_ignore\_regex, '', text).lower() + " "     return text  def normalizer(batch, chars\_to\_ignore, chars\_to\_mapping):     chars\_to\_ignore\_regex = f"""[{"".join(chars\_to\_ignore)}]"""     text = batch["sentence"].lower().strip()          text = \_normalizer.normalize(text)     text = multiple\_replace(text, chars\_to\_mapping)     text = remove\_special\_characters(text, chars\_to\_ignore\_regex)      batch["sentence"] = text     return batch   def speech\_file\_to\_array\_fn(batch):     speech\_array, sampling\_rate = torchaudio.load(batch["path"])     speech\_array = speech\_array.squeeze().numpy()     speech\_array = librosa.resample(np.asarray(speech\_array), sampling\_rate, 16\_000)      batch["speech"] = speech\_array     return batch   def predict(batch):     features = processor(batch["speech"], sampling\_rate=16\_000, return\_tensors="pt", padding=True)      input\_values = features.input\_values.to(device)     attention\_mask = features.attention\_mask.to(device)      with torch.no\_grad():         logits = model(input\_values, attention\_mask=attention\_mask).logits               pred\_ids = torch.argmax(logits, dim=-1)      batch["predicted"] = processor.batch\_decode(pred\_ids)[0]     return batch   device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") processor = Wav2Vec2Processor.from\_pretrained("m3hrdadfi/wav2vec2-large-xlsr-persian") model = Wav2Vec2ForCTC.from\_pretrained("m3hrdadfi/wav2vec2-large-xlsr-persian").to(device)  dataset = load\_dataset("common\_voice", "fa", split="test[:1%]") dataset = dataset.map(     normalizer,      fn\_kwargs={"chars\_to\_ignore": chars\_to\_ignore, "chars\_to\_mapping": chars\_to\_mapping},     remove\_columns=list(set(dataset.column\_names) - set(['sentence', 'path'])) )  dataset = dataset.map(speech\_file\_to\_array\_fn) result = dataset.map(predict)  max\_items = np.random.randint(0, len(result), 20).tolist() for i in max\_items:     reference, predicted =  result["sentence"][i], result["predicted"][i]     print("reference:", reference)     print("predicted:", predicted)     print('---') |

**خروجی :**

|  |
| --- |
| reference: اطلاعات مسری است predicted: اطلاعات مسری است --- reference: نه منظورم اینه که وقتی که ساکته چه کاریه خودمونه بندازیم زحمت predicted: نه منظورم اینه که وقتی که ساکت چی کاریه خودمونو بندازیم زحمت --- reference: من آب پرتقال می خورم لطفا predicted: من آپ ارتغال می خورم لطفا --- reference: وقت آن رسیده آنها را که قدم پیش میگذارند بزرگ بداریم predicted: وقت آ رسیده آنها را که قدم پیش میگذارند بزرگ بداریم --- reference: سیم باتری دارید predicted: سیم باتری دارید --- reference: این بهتره تا اینکه به بهونه درس و مشق هر روز بره خونه شون predicted: این بهتره تا اینکه به بهمونه درسومش خرروز بره خونه اشون --- reference: ژاکت تنگ است predicted: ژاکت تنگ است --- reference: آت و اشغال های خیابان predicted: آت و اشغال های خیابان --- reference: من به این روند اعتراض دارم predicted: من به این لوند تراج دارم --- reference: کرایه این مکان چند است predicted: کرایه این مکان چند است --- reference: ولی این فرصت این سهم جوانی اعطا نشده است predicted: ولی این فرصت این سحم جوانی اتان نشده است --- reference: متوجه فاجعهای محیطی میشوم predicted: متوجه فاجایهای محیطی میشوم --- reference: ترافیک شدیدیم بود و دیدن نور ماشینا و چراغا و لامپهای مراکز تجاری حس خوبی بهم میدادن predicted: ترافیک شدید ی هم بودا دیدن نور ماشینا و چراغ لامپهای مراکز تجاری حس خولی بهم میدادن --- reference: این مورد عمل ها مربوط به تخصص شما می شود predicted: این مورد عملها مربوط به تخصص شما میشود --- reference: انرژی خیلی کمی دارم predicted: انرژی خیلی کمی دارم --- reference: زیادی خوبی کردنم تهش داستانه predicted: زیادی خوبی کردنم ترش داستانه --- reference: بردهای که پادشاه شود predicted: برده ای که پاده شاه شود --- reference: یونسکو predicted: یونسکو --- reference: شما اخراج هستید predicted: شما اخراج هستید --- reference: من سفر کردن را دوست دارم predicted: من سفر کردم را دوست دارم |

**ارزیابی مدل**

ابتدا با کتابخانه ی hazm نرمالایز را انجام می دهیم و بعد کاراکتر هایی که باید نگاشت شوند ویا نادیده گرفته شوند را در یک متغیر ذخیره می کنیم و سپس کاراکتر های ویژه را حذف کرده و normalization را که شامل حذف کاراکتر های ویژه و جابه جا کردن چندین کاراکتر است را انجام می دهیم و بعد prediction برای مدل را انجام می دهیم که شامل استخراج ویژگی ها و اعمال attention mask است بعد از آن مدل از پیش آموزش دیده شده را به دیتاست نگاشت می کنیم

|  |
| --- |
| import librosa import torch import torchaudio from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor from datasets import load\_dataset, load\_metric  import numpy as np import hazm import re import string  \_normalizer = hazm.Normalizer() chars\_to\_ignore = [     ",", "?", ".", "!", "-", ";", ":", '""', "%", "'", '"', "'",     "#", "!", "؟", "?", "«", "»", "ء", "،", "(", ")", "؛", "'ٔ", "٬",'ٔ', ",", "?",      ".", "!", "-", ";", ":",'"',""", "%", "'", """, "'", "-", "...", "\_", """, '"', '"' ]  # In case of farsi chars\_to\_ignore = chars\_to\_ignore + list(string.ascii\_lowercase + string.digits)  chars\_to\_mapping = {     'ك': 'ک', 'دِ': 'د', 'بِ': 'ب', 'زِ': 'ز', 'ذِ': 'ذ', 'شِ': 'ش', 'سِ': 'س', 'ى': 'ی',     'ي': 'ی', 'أ': 'ا', 'ؤ': 'و', "ے": "ی", "ۀ": "ه", "ﭘ": "پ", "ﮐ": "ک", "ﯽ": "ی",     "ﺎ": "ا", "ﺑ": "ب", "ﺘ": "ت", "ﺧ": "خ", "ﺩ": "د", "ﺱ": "س", "ﻀ": "ض", "ﻌ": "ع",     "ﻟ": "ل", "ﻡ": "م", "ﻢ": "م", "ﻪ": "ه", "ﻮ": "و", "ئ": "ی", 'ﺍ': "ا", 'ة': "ه",     'ﯾ': "ی", 'ﯿ': "ی", 'ﺒ': "ب", 'ﺖ': "ت", 'ﺪ': "د", 'ﺮ': "ر", 'ﺴ': "س", 'ﺷ': "ش",     'ﺸ': "ش", 'ﻋ': "ع", 'ﻤ': "م", 'ﻥ': "ن", 'ﻧ': "ن", 'ﻭ': "و", 'ﺭ': "ر", "ﮔ": "گ",     "\\u200c": " ", "\\u200d": " ", "\\u200e": " ", "\\u200f": " ", "\\ufeff": " ", }  def multiple\_replace(text, chars\_to\_mapping):     pattern = "|".join(map(re.escape, chars\_to\_mapping.keys()))     return re.sub(pattern, lambda m: chars\_to\_mapping[m.group()], str(text))  def remove\_special\_characters(text, chars\_to\_ignore\_regex):     text = re.sub(chars\_to\_ignore\_regex, '', text).lower() + " "     return text  def normalizer(batch, chars\_to\_ignore, chars\_to\_mapping):     chars\_to\_ignore\_regex = f"""[{"".join(chars\_to\_ignore)}]"""     text = batch["sentence"].lower().strip()          text = \_normalizer.normalize(text)     text = multiple\_replace(text, chars\_to\_mapping)     text = remove\_special\_characters(text, chars\_to\_ignore\_regex)      batch["sentence"] = text     return batch   def speech\_file\_to\_array\_fn(batch):     speech\_array, sampling\_rate = torchaudio.load(batch["path"])     speech\_array = speech\_array.squeeze().numpy()     speech\_array = librosa.resample(np.asarray(speech\_array), sampling\_rate, 16\_000)      batch["speech"] = speech\_array     return batch   def predict(batch):     features = processor(batch["speech"], sampling\_rate=16\_000, return\_tensors="pt", padding=True)      input\_values = features.input\_values.to(device)     attention\_mask = features.attention\_mask.to(device)      with torch.no\_grad():         logits = model(input\_values, attention\_mask=attention\_mask).logits               pred\_ids = torch.argmax(logits, dim=-1)      batch["predicted"] = processor.batch\_decode(pred\_ids)[0]     return batch   device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") processor = Wav2Vec2Processor.from\_pretrained("m3hrdadfi/wav2vec2-large-xlsr-persian") model = Wav2Vec2ForCTC.from\_pretrained("m3hrdadfi/wav2vec2-large-xlsr-persian").to(device)  dataset = load\_dataset("common\_voice", "fa", split="test") dataset = dataset.map(     normalizer,      fn\_kwargs={"chars\_to\_ignore": chars\_to\_ignore, "chars\_to\_mapping": chars\_to\_mapping},     remove\_columns=list(set(dataset.column\_names) - set(['sentence', 'path'])) ) dataset = dataset.map(speech\_file\_to\_array\_fn) result = dataset.map(predict)  wer = load\_metric("wer") print("WER: {:.2f}".format(100 \* wer.compute(predictions=result["predicted"], references=result["sentence"]))) |

**خروجی**

مقدار خطای تست بدست آمده

|  |
| --- |
| Test Result: WER: 32.20% |

**Fine-tuning XLSR-Wav2Vec2 for Multi-Lingual ASR with hugging Face Transformers**

**جزئیات آموزش**

مدل‌های pre-train در fair-seq پیاده‌سازی می‌شوند. دو معماری را با encoder ویژگی یکسان اما تنظیمات transformerمتفاوت ارزیابی می‌شود: پایه با 12 بلوک، ابعاد حالت 768، بعد داخلی (FFN) 3072 و 8 attention head. و بزرگ با 24 بلوک، ابعاد مدل 1024، ابعاد داخلی 4096 و 16 attention head; هر دو از 0.1 dropout استفاده می کنند. برای Base، ما 250 هزار نمونه یا 15.6 ثانیه صدا را برش می دهیم و تا 1.4 میلیون نمونه را روی هر GPU بسته بندی می کنیم. برای Large، ما 320 هزار نمونه را برش می دهیم و تا 1.2 میلیون نمونه را روی یک GPU قرار می دهیم. دسته ها با استفاده از ضریب α ∈ {0.5، 1} نمونه برداری می شوند. ما از 16 GPU برای مجموعه داده های کوچک (معمولاً تک زبانه) و 64 پردازنده گرافیکی برای مجموعه داده های بزرگ (معمولاً چند زبانه) استفاده می کنیم و از Adam استفاده می شود که در آن نرخ یادگیری برای 10٪ اول به روز رسانی ها به اوج افزایش می یابد. 1e-5 (پایه) یا 1e-3 (بزرگ)، و سپس به صورت خطی در مجموع 250 هزار به روز رسانی کاهش مییابد .

**Fine tuning**

برای fine tuning مدل، classifier ‌ای که واژگان خروجی کار downstream مربوطه را نشان می‌دهد در بالای مدل اضافه می‌کنیم و روی داده‌های برlabel دار با Connectionist Temporal Classification (CTC آموزش می‌دهیم، a). وزن encoder ویژگی در زمان fine tune به روز نمی شود. ما بهترین تنظیم نرخ یادگیری را در e-5,6e-5] بر اساس میزان خطای مجموعه توسعه‌دهندگان تعیین می‌کنیم. برنامه نرخ یادگیری دارای سه مرحله است: warm up برای 10٪ اول به روز رسانی، ثابت نگه داشتن برای 40٪ و سپس کاهش خطی برای بقیه. برای common voice برای 20 هزار به‌روزرسانی Fine tune می‌شود.

**مدل های pre-trained**

از معماری Base استفاده می شود. برای common voice، یک مدل فارسی را در ساعت 1350 و ده مدل تک زبانه را در هر مجموعه پیش‌آموزشی آموزش می‌دهیم. برای مقایسه با مدل انگلیسی، مدل‌های چندزبانه Base و Large را با 1350 ساعت داده آموزش داده می شود : 793 ساعت صدای گفتار از 10 زبان ارزیابی به اضافه 557 ساعت صدای فارسی. ما زبان های کم منبع را با α = 0.5 نمونه برداری می شود و مدلی با α = 1 برای مقایسه (نامتوازن) آموزش داده می شود . برای fine-tune چند زبانه، واژگان واجی را در بین زبان‌ها جدا می‌کنیم یا به اشتراک می‌گذاریم.

برای BABEL، یک مدل تک زبانه در هر یک از 14 زبان، و همچنین یک مدل چند زبانه Base و Large را در مجموع 650 ساعت صدای گفتار به ده زبان آموزش می دهیم. از آنجایی که مقدار داده در هر زبان نسبت به Common Voice متعادل‌تر است، از α = 1 استفاده می‌شود. از همان صدای گفتاری برای پیش‌آموزش و تنظیم دقیق استفاده می‌شود و از مجموعه کاراکترهای جداگانه برای تنظیم دقیق چند زبانه استفاده می‌شود.

یک مدل بزرگ در 53 زبان MLS، Common Voice و BABEL آموزش می‌دهیم که از 56 هزار ساعت داده گفتاری تشکیل شده است. این مدل را XLSR-53 نام دارد.

**قطعه کد مربوطه**

|  |
| --- |
| %%capture !pip install datasets==1.13.3 !pip install transformers==4.11.3 !pip install torchaudio !pip install librosa !pip install jiwer |

**Prepare Data, Tokenizer, Feature Extractor**

مدل‌های ASR گفتار را به متن transcribe می‌کنند، و به هر دو استخراج‌کننده ویژگی نیاز داریم که سیگنال گفتار را به قالب ورودی مدل پردازش کند، به عنوان مثال. یک بردار ویژگی و یک نشانه ساز که فرمت خروجی مدل را به متن پردازش می کند. در hugging face transformerها، مدل XLSR-Wav2Vec2 با یک توکنایزر به نام Wav2Vec2CTCTokenizer و یک استخراج کننده ویژگی به نام Wav2Vec2FeatureExtractor همراه است.. لایه خطی برای classification هر نمایش زمینه به یک کلاس نشانه استفاده می‌شود پس از pre-train، یک لایه خطی در بالای embedding‌های BERT برای classification بیشتر اضافه می‌شود. اندازه خروجی این لایه با تعداد token‌ های موجود در واژگان مطابقت دارد، که به وظیفه پیش‌آموزشی XLSR-Wav2Vec2 بستگی ندارد، بلکه فقط به مجموعه داده label شده مورد استفاده برای fine tuning بستگی دارد. برای این نوت بوک از زبان فارسی استفاده می شود. کد زبان با پیشوند برای فارسی "fa" است. عالی است، اکنون می‌توانیم از API ساده Datasets hugging face برای دانلود داده‌ها استفاده کنیم.

|  |
| --- |
| ## Prepare Data, Tokenizer, Feature Extractor |

|  |
| --- |
| from datasets import load\_dataset, load\_metric, Audio  common\_voice\_train = load\_dataset("common\_voice", "fa", split="train+validation") common\_voice\_test = load\_dataset("common\_voice", "fa", split="test") |

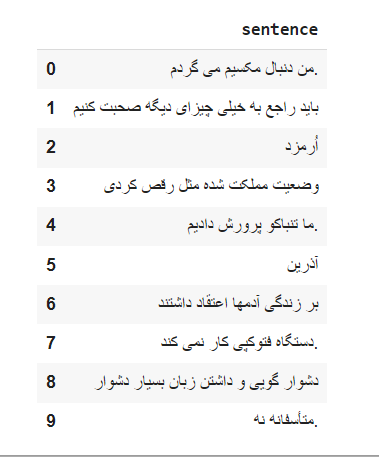
بسیاری از مجموعه‌های داده ASR فقط متن هدف، "جمله" را برای هر آرایه صوتی "صوتی" و فایل "مسیر" ارائه می‌کنند. Common Voice در واقع اطلاعات بسیار بیشتری در مورد هر فایل صوتی، مانند "لهجه" و غیره ارائه می دهد.

|  |
| --- |
| common\_voice\_train = common\_voice\_train.remove\_columns(["accent", "age", "client\_id", "down\_votes", "gender", "locale", "segment", "up\_votes"]) common\_voice\_test = common\_voice\_test.remove\_columns(["accent", "age", "client\_id", "down\_votes", "gender", "locale", "segment", "up\_votes"]) |

حالا یک تابع برای نمایش چند نمونه تصادفی از مجموعه داده می نویسیم و آن را چند بار اجرا می کنیم تا برای رونویسی متنی به دست بیاوریم.

|  |
| --- |
| from datasets import ClassLabel import random import pandas as pd from IPython.display import display, HTML  def show\_random\_elements(dataset, num\_examples=10):     assert num\_examples <= len(dataset), "Can't pick more elements than there are in the dataset."     picks = []     for \_ in range(num\_examples):         pick = random.randint(0, len(dataset)-1)         while pick in picks:             pick = random.randint(0, len(dataset)-1)         picks.append(pick)          df = pd.DataFrame(dataset[picks])     display(HTML(df.to\_html())) |

|  |
| --- |
| show\_random\_elements(common\_voice\_train.remove\_columns(["path", "audio"]), num\_examples=10) |

**خروجی**

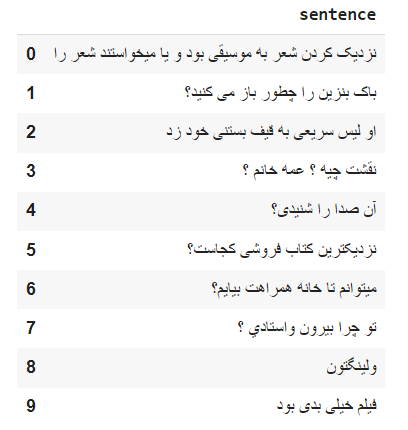
پس از ترجمه جملات transcribe شده، این زبان بیشتر با متن نوشته شده بیش تر از noisy dialogue مطابقت دارد .

بعضی از transcribeشامل برخی کاراکترهای خاص هستند، مانند .?!;:. بدون یک مدل زبان، طبقه‌بندی chunk ‌های گفتار به چنین کاراکترهای خاصی بسیار سخت‌تر است، زیرا آنها واقعاً با واحد صدای مشخصه مطابقت ندارند. به عنوان مثال، حرف "" sصدایی واضح دارد، در حالی که کاراکتر ویژه "." صدای واضحی ندارد. همچنین برای درک معنای سیگنال گفتار، معمولاً نیازی به گنجاندن کاراکترهای خاص در transcribe نیست. علاوه بر این، متن را normalize می کنیم تا فقط حروف کوچک داشته باشد و یک separator token کلمات را در پایان اضافه می کنیم.

|  |
| --- |
| import re chars\_to\_ignore\_regex = '[\,\?\.\!\-\;\:\"\"\%\'\"\']'  def remove\_special\_characters(batch):     batch["sentence"] = re.sub(chars\_to\_ignore\_regex, '', batch["sentence"]).lower() + " "     return batch |

|  |
| --- |
| common\_voice\_train = common\_voice\_train.map(remove\_special\_characters) common\_voice\_test = common\_voice\_test.map(remove\_special\_characters) |

|  |
| --- |
| show\_random\_elements(common\_voice\_train.remove\_columns(["path","audio"])) |

****

**خروجی**

بیش تر نویسه‌های ویژه را از transcribe حذف کرده‌ایم و آنها را فقط به حروف کوچک normalize کرده‌ایم

در CTC، طبقه‌بندی chunk گفتار به حروف رایج است. یک تابع نگاشت می نویسیم که تمام transcribe ها را به یک transcribe طولانی concatenate می کند و سپس رشته را به مجموعه ای از کاراکترها تبدیل می کند. مهم است که آرگومان batched=True را به تابع map(...) منتقل کنیم تا تابع نگاشت به همهtranscribe ها به طور همزمان دسترسی داشته باشد.

|  |
| --- |
| def extract\_all\_chars(batch):   all\_text = " ".join(batch["sentence"])   vocab = list(set(all\_text))   return {"vocab": [vocab], "all\_text": [all\_text]} |

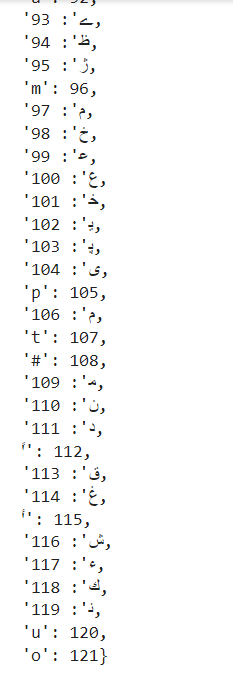
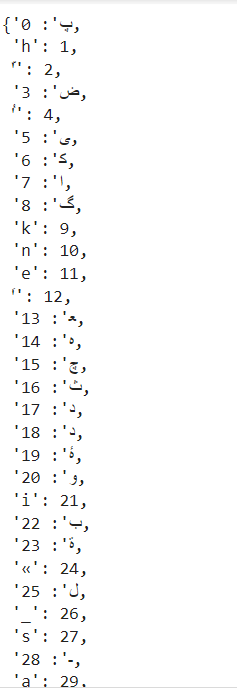
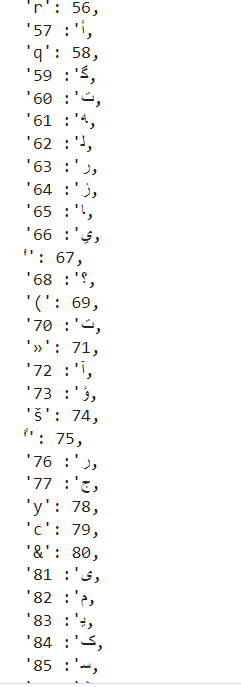
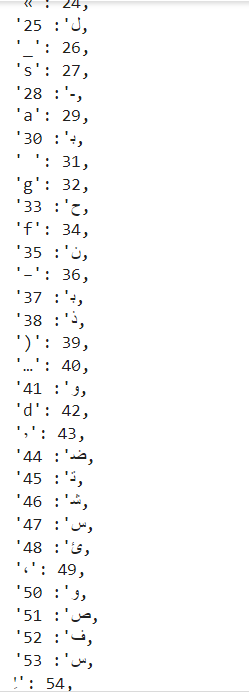
|  |
| --- |
| vocab\_train = common\_voice\_train.map(extract\_all\_chars, batched=True, batch\_size=-1, keep\_in\_memory=True, remove\_columns=common\_voice\_train.column\_names) vocab\_test = common\_voice\_test.map(extract\_all\_chars, batched=True, batch\_size=-1, keep\_in\_memory=True, remove\_columns=common\_voice\_test.column\_names) |

حالا یک union از تمام حروف متمایز در مجموعه داده آموزشی و مجموعه داده آزمایشی ایجاد می کنیم و لیست حاصل را به یک فرهنگ لغت شمارش شده تبدیل می کنیم

.

|  |
| --- |
| vocab\_list = list(set(vocab\_train["vocab"][0]) | set(vocab\_test["vocab"][0])) |

|  |
| --- |
| vocab\_dict = {v: k for k, v in enumerate(vocab\_list)} vocab\_dict |

****

**خروجی**

کاراکترهای خاص را حذف نکردیم زیرا:

مدل باید یاد بگیرد که چه زمانی یک کلمه تمام می‌شود پیش‌بینی کند وگرنه پیش‌بینی مدل همیشه دنباله‌ای از کاراکترها خواهد بود که جدا کردن کلمات از یکدیگر را غیرممکن می‌کند.

برای اینکه واضح‌تر شود که " " کلاس توکن خود را دارد، به آن یک کاراکتر قابل مشاهده تر می دهیم |. علاوه بر این، همچنین یک نشانه "UNK" اضافه می کنیم تا مدل بعداً بتواند با کاراکترهایی که در مجموعه آموزشی Common Voice با آنها مواجه نشده اند مواجه شود.

در نهایت، یک توکن padding را نیز اضافه می‌کنیم که با علامت خالی CTC مطابقت دارد. "توکن خالی" جزء اصلی الگوریتم CTC است.

|  |
| --- |
| vocab\_dict["|"] = vocab\_dict[" "] del vocab\_dict[" "] |

|  |
| --- |
| vocab\_dict["[UNK]"] = len(vocab\_dict) vocab\_dict["[PAD]"] = len(vocab\_dict) len(vocab\_dict) |

حالا دیکشنری کامل است و از 39 توکن تشکیل شده است، به این معنی که لایه خطی که در بالای نقطه Checkpoint از پیش آموزش دیده XLSR-Wav2Vec2 اضافه می کنیم، بعد خروجی 39 بعد خواهد داشت.

|  |
| --- |
| import json with open('vocab.json', 'w') as vocab\_file:     json.dump(vocab\_dict, vocab\_file) |

|  |
| --- |
| from transformers import Wav2Vec2CTCTokenizer  tokenizer = Wav2Vec2CTCTokenizer("./vocab.json", unk\_token="[UNK]", pad\_token="[PAD]", word\_delimiter\_token="|") |

**XLSR-Wav2Vec2 Feature Extractor**

یک checkpoint pre-trainedقرار است که داده های ورودی آن کم و بیش از همان توزیعی که بر روی آن آموزش دیده است نمونه برداری شده باشد. سیگنال‌های گفتاری یکسانی که با دو سرعت مختلف نمونه‌برداری می‌شوند، توزیع بسیار متفاوتی دارند، قبل از fine tune یک pre-trained checkpoint یک مدل ASR، بررسی اینکه نرخ نمونه‌گیری داده‌هایی که برای پیش‌آموزش مدل استفاده شده است با نرخ نمونه‌گیری مجموعه داده‌های مورد استفاده برای fine tune مدل مطابقت دارد، بسیار مهم است.

XLSR-Wav2Vec2 روی داده های صوتی Babel، Multilingual LibriSpeech (MLS) و Common Voice از قبل آموزش داده شد. قبلا مجموعه داده‌ها با فرکانس 16 کیلوهرتز نمونه‌برداری شده‌اند، به طوری که Common Voice که در 48 کیلوهرتز نمونه‌برداری می‌شود، برای آموزش باید به 16 کیلوهرتز کاهش پیدا کند. بنابراین، در ادامه باید داده های fine tune را به 16 کیلوهرتز کاهش دهیم.

یک object feature extractor برای XLSR-Wav2Vec2 برای sampling به پارامترهای زیر نیاز دارد:

: feature\_size دنباله ای از بردارهای ویژگی را به عنوان ورودی می گیرند. در حالی که طول این دنباله به متفاوت است، در Wav2Vec2، اندازه ویژگی 1 است زیرا مدل بر روی سیگنال گفتار خام با اندازه ی 2 آموزش داده شده است.

:sampling\_rate نرخ نمونه برداری که مدل بر روی آن آموزش می بیند.

: padding\_value برای inference دسته‌ای، ورودی‌های کوتاه‌تر باید با یک مقدار خاص padd شوند.

:do\_normalize اینکه آیا ورودی باید نرمال شود یا خیر. معمولاً مدل‌های گفتاری هنگام normalization ورودی بهتر عمل می‌کنند

:return\_attention\_mask آیا مدل باید از یک Attention mask برای inference دسته‌ای استفاده کند یا خیر. مدل های XLSR-Wav2Vec2 همیشه باید از attention mask استفاده کنند.

|  |
| --- |
| from transformers import Wav2Vec2FeatureExtractor  feature\_extractor = Wav2Vec2FeatureExtractor(feature\_size=1, sampling\_rate=16000, padding\_value=0.0, do\_normalize=True, return\_attention\_mask=True) |

|  |
| --- |
| from transformers import Wav2Vec2Processor  processor = Wav2Vec2Processor(feature\_extractor=feature\_extractor, tokenizer=tokenizer) |

**پیش پردازش داده ها**

تا اینجا به مقادیر واقعی سیگنال گفتار توجه نشده ، فقط transcribe را بررسی کرده‌ایم. علاوه بر جمله، مجموعه داده های ما شامل دو مسیر نام ستون و صدا می باشد. path مسیر مطلق فایل صوتی را بیان می کند.

|  |
| --- |
| common\_voice\_train[0]["path"] |

|  |
| --- |
| common\_voice\_train[0]["audio"] |

در اینجا نرخ نمونه برداری 48000 khz است

|  |
| --- |
| {'path': '/root/.cache/huggingface/datasets/downloads/extracted/3b0e0ce56990e1c035c009dd3032760da75614498aab0a4090ea91c8d4e8a843/cv-corpus-6.1-2020-12-11/fa/clips/common\_voice\_fa\_19258088.mp3',  'array': array([ 0.0000000e+00,  5.9539030e-14,  7.0466365e-17, ...,         -2.8262571e-11, -1.5154249e-11, -5.9851811e-12], dtype=float32),  'sampling\_rate': 48000} |

|  |
| --- |
| common\_voice\_train = common\_voice\_train.cast\_column("audio", Audio(sampling\_rate=16\_000)) common\_voice\_test = common\_voice\_test.cast\_column("audio", Audio(sampling\_rate=16\_000)) |

نمونه برداری مجدد با نرخ نمونه برداری 16000 khz

|  |
| --- |
| common\_voice\_train[0]["audio"] |

|  |
| --- |
| {'path': '/root/.cache/huggingface/datasets/downloads/extracted/3b0e0ce56990e1c035c009dd3032760da75614498aab0a4090ea91c8d4e8a843/cv-corpus-6.1-2020-12-11/fa/clips/common\_voice\_fa\_19258088.mp3',  'array': array([ 1.4601147e-14,  5.1826401e-15, -3.5990877e-14, ...,         -9.6792444e-12, -3.6583240e-11, -2.1919251e-11], dtype=float32),  'sampling\_rate': 16000} |

**پخش صدا**

|  |
| --- |
| import IPython.display as ipd import numpy as np import random  rand\_int = random.randint(0, len(common\_voice\_train)-1)  ipd.Audio(data=common\_voice\_train[rand\_int]["audio"]["array"], autoplay=True, rate=16000) |

|  |
| --- |
| rand\_int = random.randint(0, len(common\_voice\_train)-1)  print("Target text:", common\_voice\_train[rand\_int]["sentence"]) print("Input array shape:", common\_voice\_train[rand\_int]["audio"]["array"].shape) print("Sampling rate:", common\_voice\_train[rand\_int]["audio"]["sampling\_rate"]) |

**خروجی**

|  |
| --- |
| Target text: در دانشگاه لیورپول هستم  Input array shape: (73344,) Sampling rate: 16000 |

داده ها یک آرایه 1 بعدی هستند، نرخ نمونه برداری همیشه با 16 کیلوهرتز مطابقت دارد و متن هدف نرمال می شود. می‌توانیم از Wav2Vec2Processor برای پردازش داده‌ها به فرمت مورد انتظار Wav2Vec2ForCTC برای آموزش استفاده کنیم. دوباره از تابع map(...) استفاده خواهیم کرد.

ابتدا، داده‌های صوتی را به سادگی با call batch ["audio"] بارگیری و نمونه‌گیری مجدد می‌کنیم. دوم اینکه مقادیر input\_values را از فایل صوتی بارگذاری شده استخراج می کنیم. در مورد ما، این فقط شامل عادی سازی می شود، اما برای سایر مدل های گفتاری، این مرحله می تواند مطابق با استخراج باشد، به عنوان مثال. ویژگی های Log-Mel سوم، transcribe را برای label زدن شناسه ها encode می کنیم.

توجه: این تابع نگاشت نمونه خوبی از نحوه استفاده از کلاس Wav2Vec2Processor است. در زمینه "normal"، فراخوانی پردازشگر (...) به روش فراخوانی Wav2Vec2FeatureExtractor هدایت می شود.

|  |
| --- |
| def prepare\_dataset(batch):     audio = batch["audio"]      # batched output is "un-batched"     batch["input\_values"] = processor(audio["array"], sampling\_rate=audio["sampling\_rate"]).input\_values[0]          with processor.as\_target\_processor():         batch["labels"] = processor(batch["sentence"]).input\_ids     return batch |

|  |
| --- |
| common\_voice\_train = common\_voice\_train.map(prepare\_dataset, remove\_columns=common\_voice\_train.column\_names, num\_proc=4) common\_voice\_test = common\_voice\_test.map(prepare\_dataset, remove\_columns=common\_voice\_test.column\_names, num\_proc=4) |

**آموزش**

داده ها به گونه ای پردازش می شوند که ما آماده شروع راه اندازی pipe line آموزشی هستیم. از آموزش hugging face استفاده می کنیم که برای آن باید موارد زیر را انجام دهیم.

ابتدا داده ها را جمع آوری می کنیم . برخلاف اکثر مدل‌های NLP، XLSR-Wav2Vec2 طول ورودی بسیار بیشتری نسبت به طول خروجی دارد. به عنوان مثال، یک نمونه با طول ورودی 50000 دارای طول خروجی بیش از 100 نیست. با توجه به اندازه های ورودی بزرگ، بسیار کارآمدتر است که دسته های آموزشی را به صورت داینامیک قرار دهیم، به این معنی که تمام نمونه های آموزشی باید فقط به طولانی ترین نمونه در خود اضافه شوند.. بنابراین، XLSR-Wav2Vec2 به جمع‌آوری داده‌های padding ویژه نیاز دارد که در زیر آن را تعریف می‌کنیم.

متریک ارزیابی در طول آموزش، مدل باید بر اساس میزان خطای کلمه ارزیابی شود. بر این اساس باید یک تابع compute\_metrics تعریف کنیم. ما باید یک ایست بازرسی از قبل آموزش دیده را بارگذاری کنیم و آن را به درستی برای آموزش پیکربندی کنیم.

**Training configuration**

پس از fine tune مدل، آن را به درستی بر روی داده های آزمایش ارزیابی می کنیم و تأیید می کنیم که آیا واقعاً یاد گرفته است که گفتار را به درستیtranscribe کند.

**Trainer setup**

ابتدا داده را جمع آوری می کنیم. بر خلاف جمع‌آوری‌های معمول داده، این جمع‌آوری داده با مقادیر ورودی و label متفاوت رفتار می‌کند و بنابراین برای توابع padding جداگانه روی آنها اعمال می‌شود. زیرا در گفتار ورودی و خروجی حالت‌های متفاوتی دارند، به این معنی که نباید با یک تابع padding برخورد کرد. مشابه با جمع‌آوری‌های معمول داده‌ها، توکن‌های padding در برچسب‌هایی با ۱۰۰- به طوری که آن نشانه‌ها هنگام محاسبه loss در نظر گرفته نمی‌شوند.

|  |
| --- |
| import torch  from dataclasses import dataclass, field from typing import Any, Dict, List, Optional, Union  @dataclass class DataCollatorCTCWithPadding:     """     Data collator that will dynamically pad the inputs received.     Args:         processor (:class:`~transformers.Wav2Vec2Processor`)             The processor used for proccessing the data.         padding (:obj:`bool`, :obj:`str` or :class:`~transformers.tokenization\_utils\_base.PaddingStrategy`, `optional`, defaults to :obj:`True`):             Select a strategy to pad the returned sequences (according to the model's padding side and padding index)             among:             \* :obj:`True` or :obj:`'longest'`: Pad to the longest sequence in the batch (or no padding if only a single               sequence if provided).             \* :obj:`'max\_length'`: Pad to a maximum length specified with the argument :obj:`max\_length` or to the               maximum acceptable input length for the model if that argument is not provided.             \* :obj:`False` or :obj:`'do\_not\_pad'` (default): No padding (i.e., can output a batch with sequences of               different lengths).         max\_length (:obj:`int`, `optional`):             Maximum length of the ``input\_values`` of the returned list and optionally padding length (see above).         max\_length\_labels (:obj:`int`, `optional`):             Maximum length of the ``labels`` returned list and optionally padding length (see above).         pad\_to\_multiple\_of (:obj:`int`, `optional`):             If set will pad the sequence to a multiple of the provided value.             This is especially useful to enable the use of Tensor Cores on NVIDIA hardware with compute capability >=             7.5 (Volta).     """      processor: Wav2Vec2Processor     padding: Union[bool, str] = True     max\_length: Optional[int] = None     max\_length\_labels: Optional[int] = None     pad\_to\_multiple\_of: Optional[int] = None     pad\_to\_multiple\_of\_labels: Optional[int] = None      def \_\_call\_\_(self, features: List[Dict[str, Union[List[int], torch.Tensor]]]) -> Dict[str, torch.Tensor]:         # split inputs and labels since they have to be of different lenghts and need         # different padding methods         input\_features = [{"input\_values": feature["input\_values"]} for feature in features]         label\_features = [{"input\_ids": feature["labels"]} for feature in features]          batch = self.processor.pad(             input\_features,             padding=self.padding,             max\_length=self.max\_length,             pad\_to\_multiple\_of=self.pad\_to\_multiple\_of,             return\_tensors="pt",         )         with self.processor.as\_target\_processor():             labels\_batch = self.processor.pad(                 label\_features,                 padding=self.padding,                 max\_length=self.max\_length\_labels,                 pad\_to\_multiple\_of=self.pad\_to\_multiple\_of\_labels,                 return\_tensors="pt",             )          # replace padding with -100 to ignore loss correctly         labels = labels\_batch["input\_ids"].masked\_fill(labels\_batch.attention\_mask.ne(1), -100)          batch["labels"] = labels          return batch |

|  |
| --- |
| data\_collator = DataCollatorCTCWithPadding(processor=processor, padding=True) |

سپس معیار ارزیابی تعریف می شود. معیار غالب در ASR، نرخ خطای کلمه (WER) است.

|  |
| --- |
| wer\_metric = load\_metric("wer") |

مدل دنباله ای از بردارهای logit را برمی گرداند: ymبردار logit y1 حاوی log-odds برای هر کلمه در واژگانی است که قبلاً تعریف شده ،بنابراینlen(yi)= config.vocab\_size. و به محتمل ترین پیش بینی مدل هستیم و بنابراین argmax(...) logits را می گیریم. همچنین، با جایگزین کردن -100 با pad\_token\_id و decode شناسه‌ها، لیبل های کدگذاری شده را به رشته اصلی برمی‌گردانیم و در عین حال مطمئن می‌شویم که توکن‌های متوالی در یک token در سبک CTC 1 گروه‌بندی نمی‌شوند.

|  |
| --- |
| def compute\_metrics(pred):     pred\_logits = pred.predictions     pred\_ids = np.argmax(pred\_logits, axis=-1)      pred.label\_ids[pred.label\_ids == -100] = processor.tokenizer.pad\_token\_id      pred\_str = processor.batch\_decode(pred\_ids)     # we do not want to group tokens when computing the metrics     label\_str = processor.batch\_decode(pred.label\_ids, group\_tokens=False)      wer = wer\_metric.compute(predictions=pred\_str, references=label\_str)      return {"wer": wer} |

حالا XLSR-Wav2Vec2 checkpoint از پیش آموزش دیده را دانلود می کنیم. pad\_token\_id توکنایزر باید برای تعریف pad\_token\_id مدل باشد یا در مورد Wav2Vec2ForCTC نیز توکن خالی CTC 2 باشد. همچنین loss را روی "mean" تنظیم می کنیم. از آنجایی که مجموعه داده بسیار کوچک است (حدود 6 ساعت داده آموزشی) و به دلیل اینکه Common Voice نویزی است، بنابراین fine tune پست بازرسی wav2vec2-large-xlsr-53 نیاز به hyper parameter tuning دارد. بنابراین، مقداری با مقادیر مختلف dropout، میزان learning rate ، تنظیم کنیم تا زمانی که آموزش به اندازه کافی پایدار به نظر برسد.

|  |
| --- |
| from transformers import Wav2Vec2ForCTC  model = Wav2Vec2ForCTC.from\_pretrained(     "facebook/wav2vec2-large-xlsr-53",      attention\_dropout=0.1,     hidden\_dropout=0.1,     feat\_proj\_dropout=0.0,     mask\_time\_prob=0.05,     layerdrop=0.1,     ctc\_loss\_reduction="mean",      pad\_token\_id=processor.tokenizer.pad\_token\_id,     vocab\_size=len(processor.tokenizer) ) |

|  |
| --- |
| model.freeze\_feature\_extractor() |

|  |
| --- |
| model.gradient\_checkpointing\_enable() |

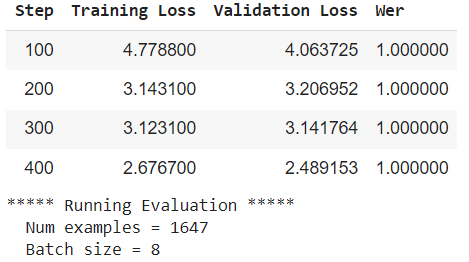
در مرحله آخر، تمام پارامترهای مربوط به آموزش را تعریف می کنیم. برای توضیح بیشتر در مورد برخی از پارامترها group\_by\_length با گروه بندی نمونه های آموزشی با طول ورودی مشابه در یک دسته، آموزش را کارآمدتر می کند. این می‌تواند با کاهش شدید تعداد کلی توکن‌های padding بی‌فایده که از طریق مدل ارسال می‌شوند، زمان تمرین را به میزان قابل توجهی افزایش دهد.

میزان یادگیری و کاهش وزن به صورت تجربی تنظیم شدند تا زمانی که fine tune پایدار شود. توجه داشته باشید که این پارامترها به شدت به مجموعه داده های Common Voice بستگی دارند و ممکن است برای سایر مجموعه داده های گفتاری کمتر از حد مطلوب باشند.

|  |
| --- |
| from transformers import TrainingArguments  training\_args = TrainingArguments(   # output\_dir="/content/gdrive/MyDrive/wav2vec2-large-xlsr-turkish-demo",   output\_dir="./wav2vec2-large-xlsr-persian-demo",   group\_by\_length=True,   per\_device\_train\_batch\_size=16,   gradient\_accumulation\_steps=2,   evaluation\_strategy="steps",   num\_train\_epochs=30,   fp16=True,   save\_steps=100,   eval\_steps=100,   logging\_steps=10,   learning\_rate=3e-4,   warmup\_steps=500, )  save\_total\_limit=2, |

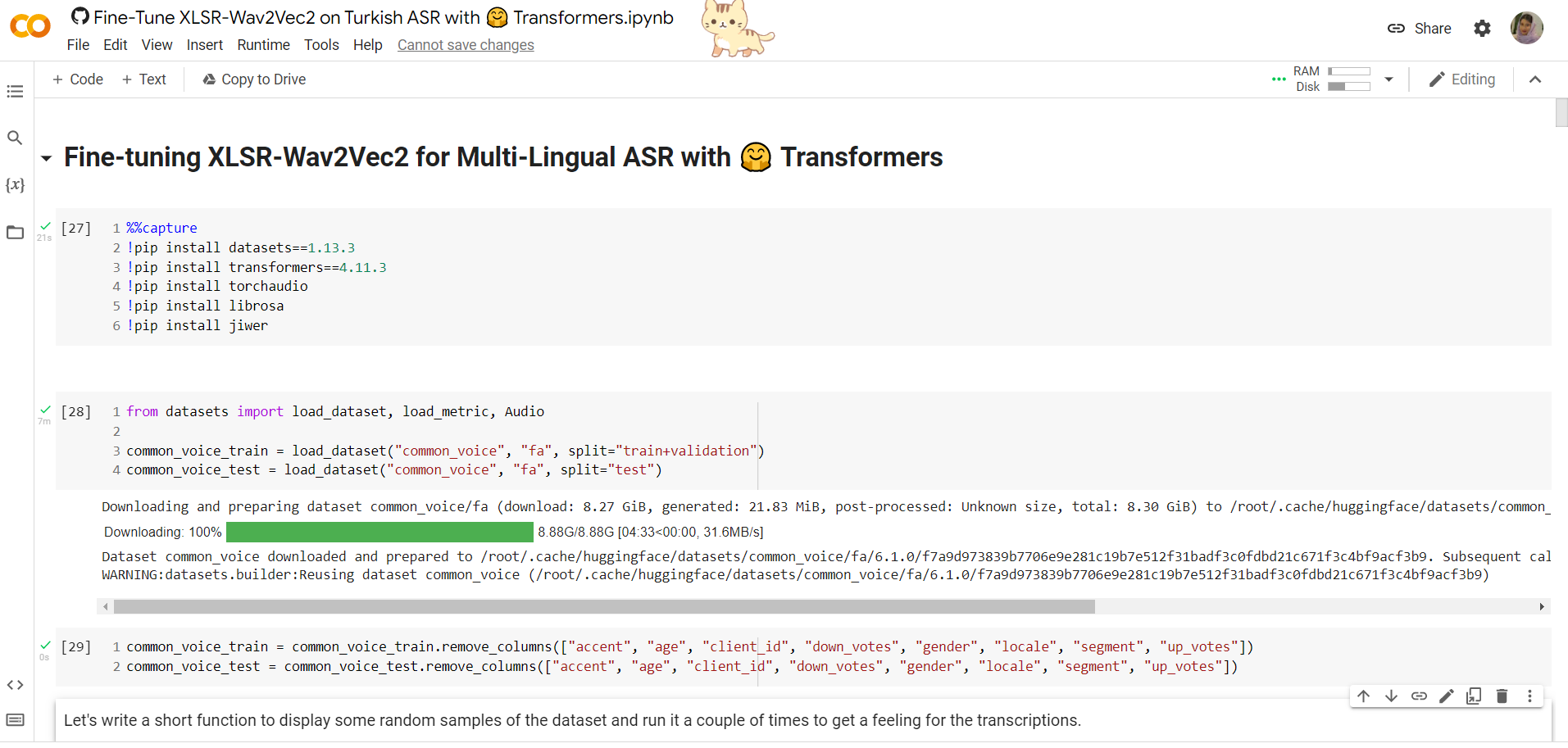
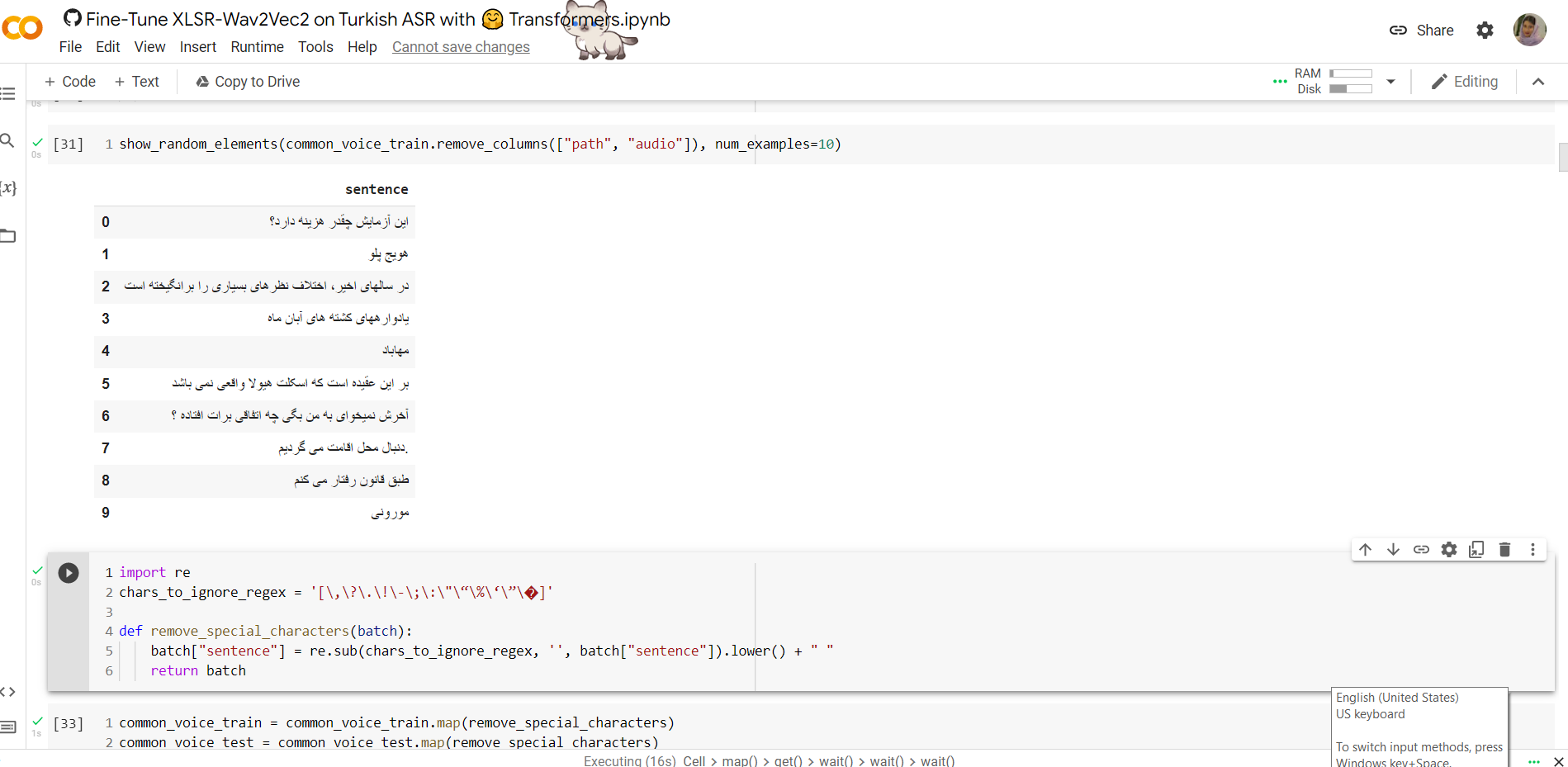
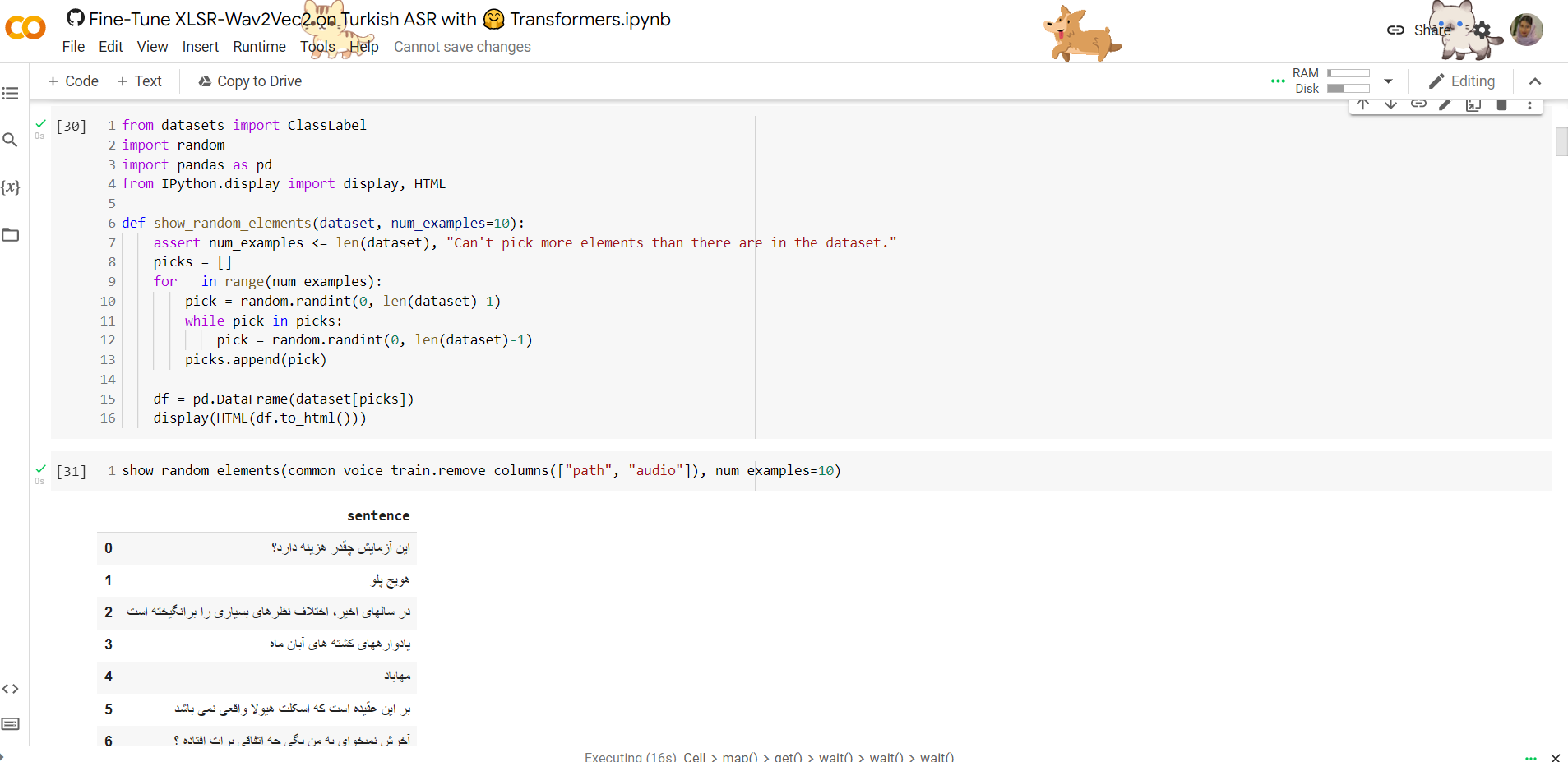
|  |
| --- |
| from transformers import Trainer  trainer = Trainer(     model=model,     data\_collator=data\_collator,     args=training\_args,     compute\_metrics=compute\_metrics,     train\_dataset=common\_voice\_train,     eval\_dataset=common\_voice\_test,     tokenizer=processor.feature\_extractor, ) |

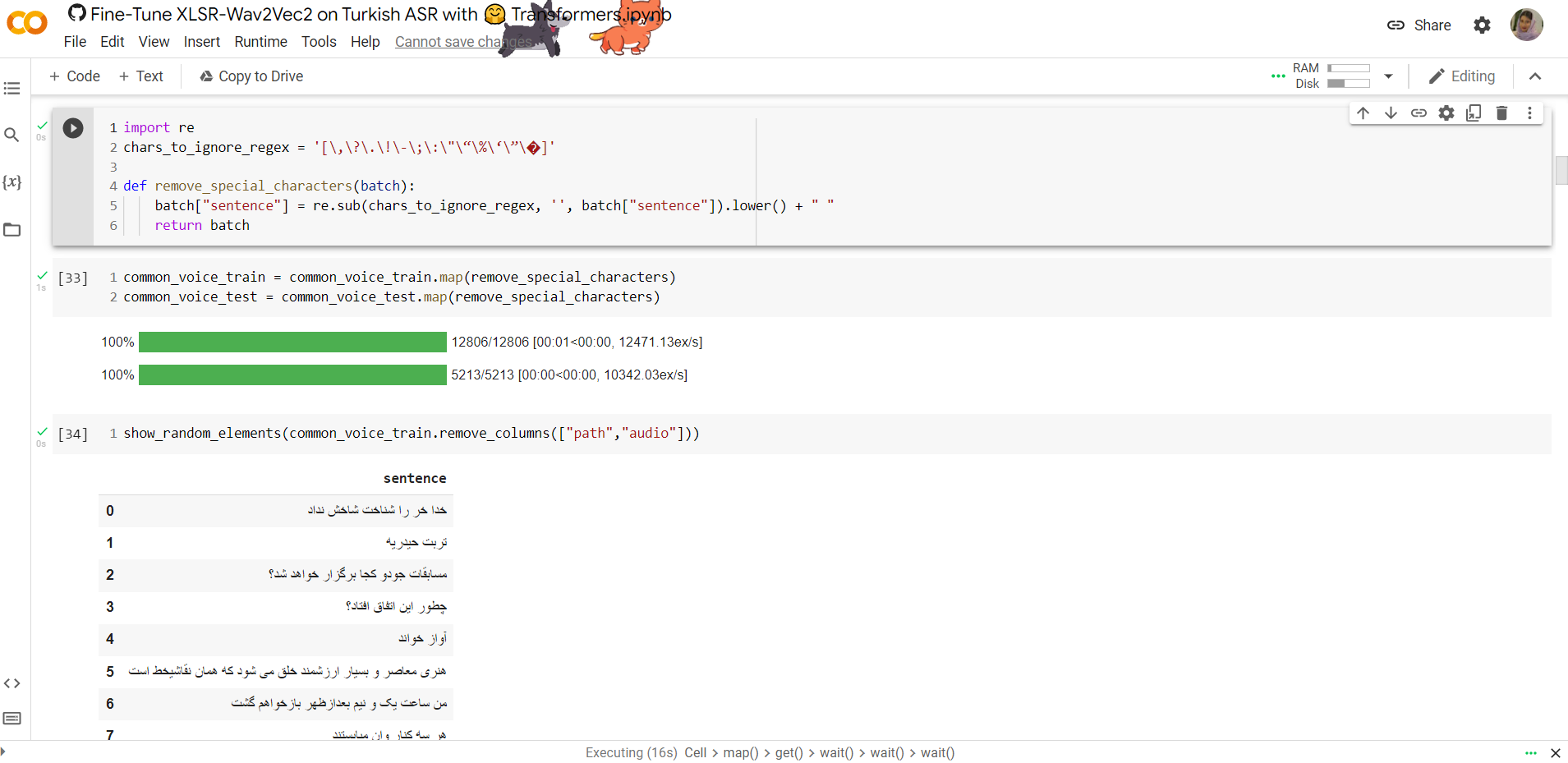
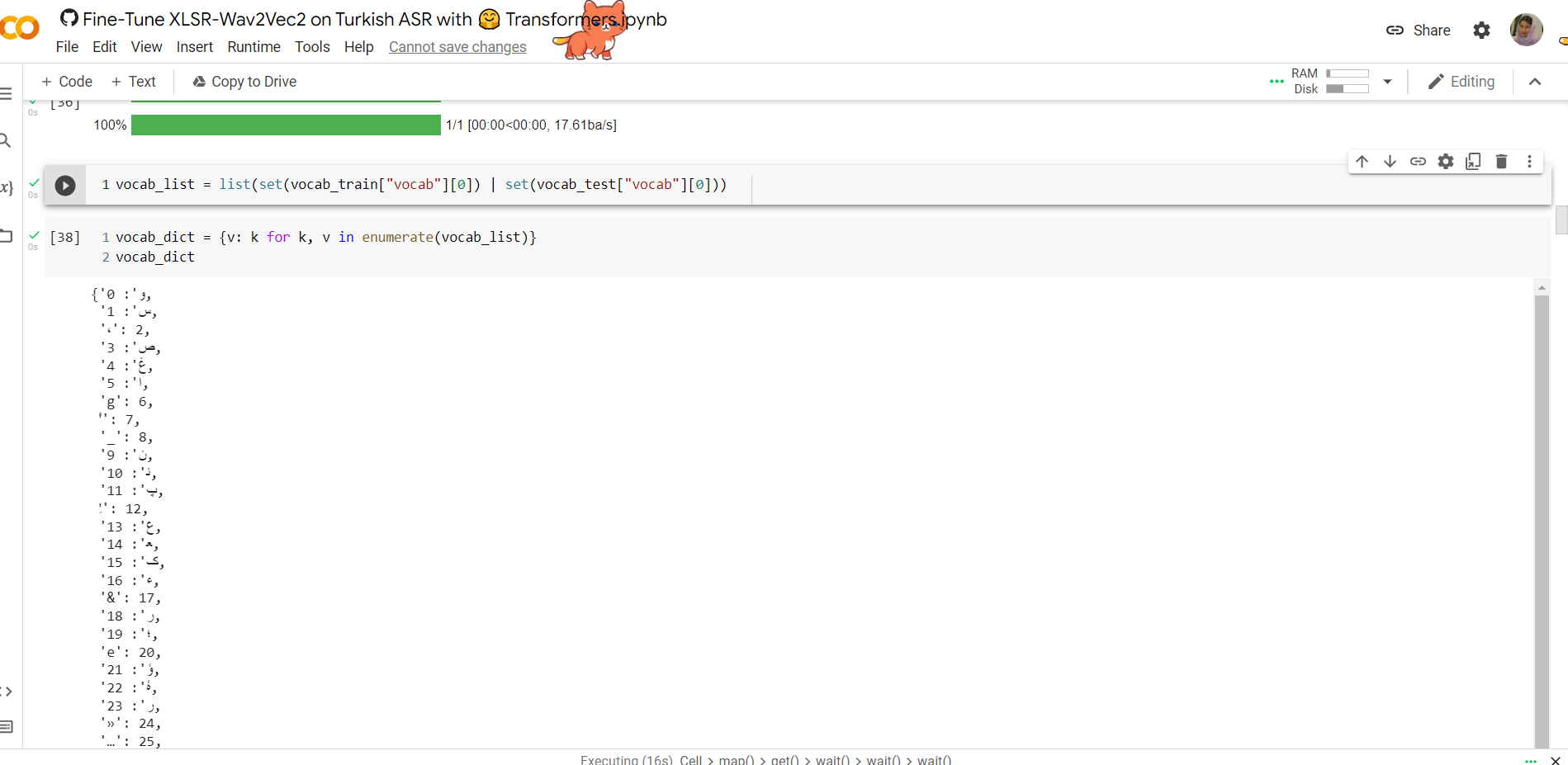
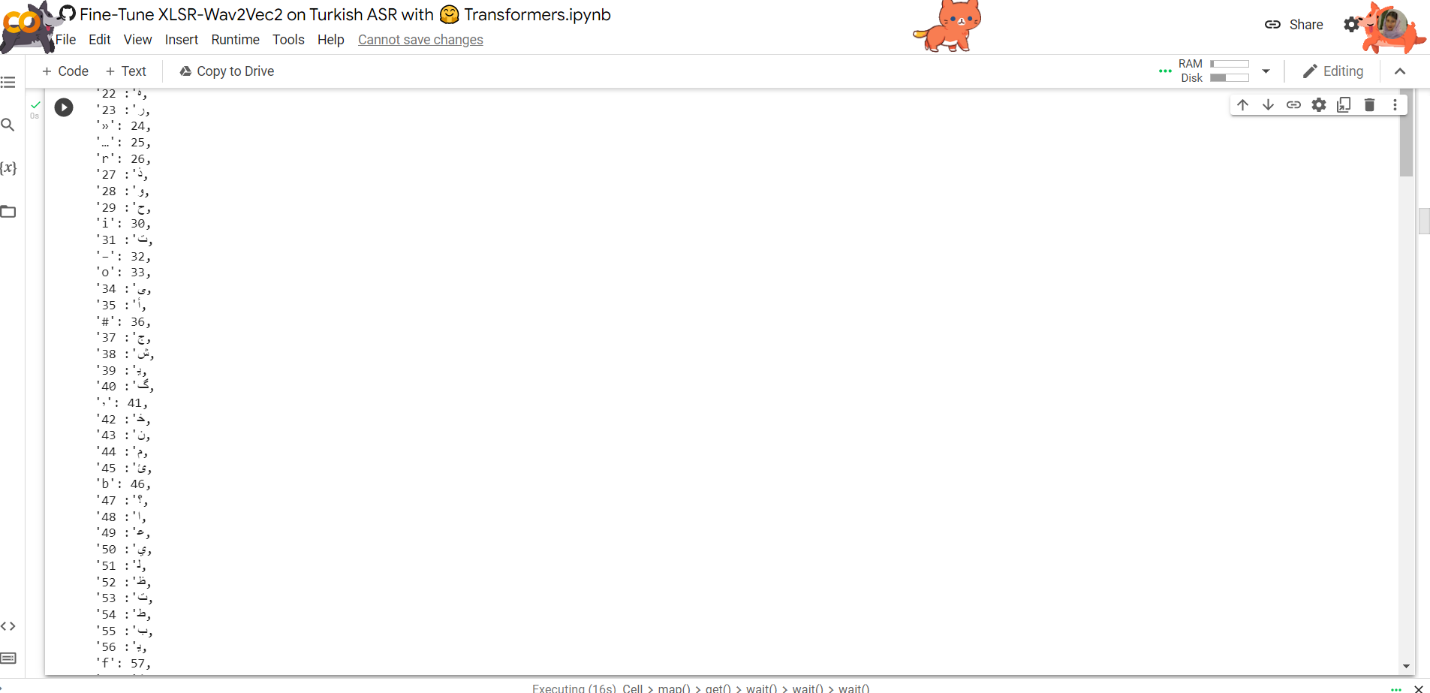
|  |
| --- |
| trainer.train() |

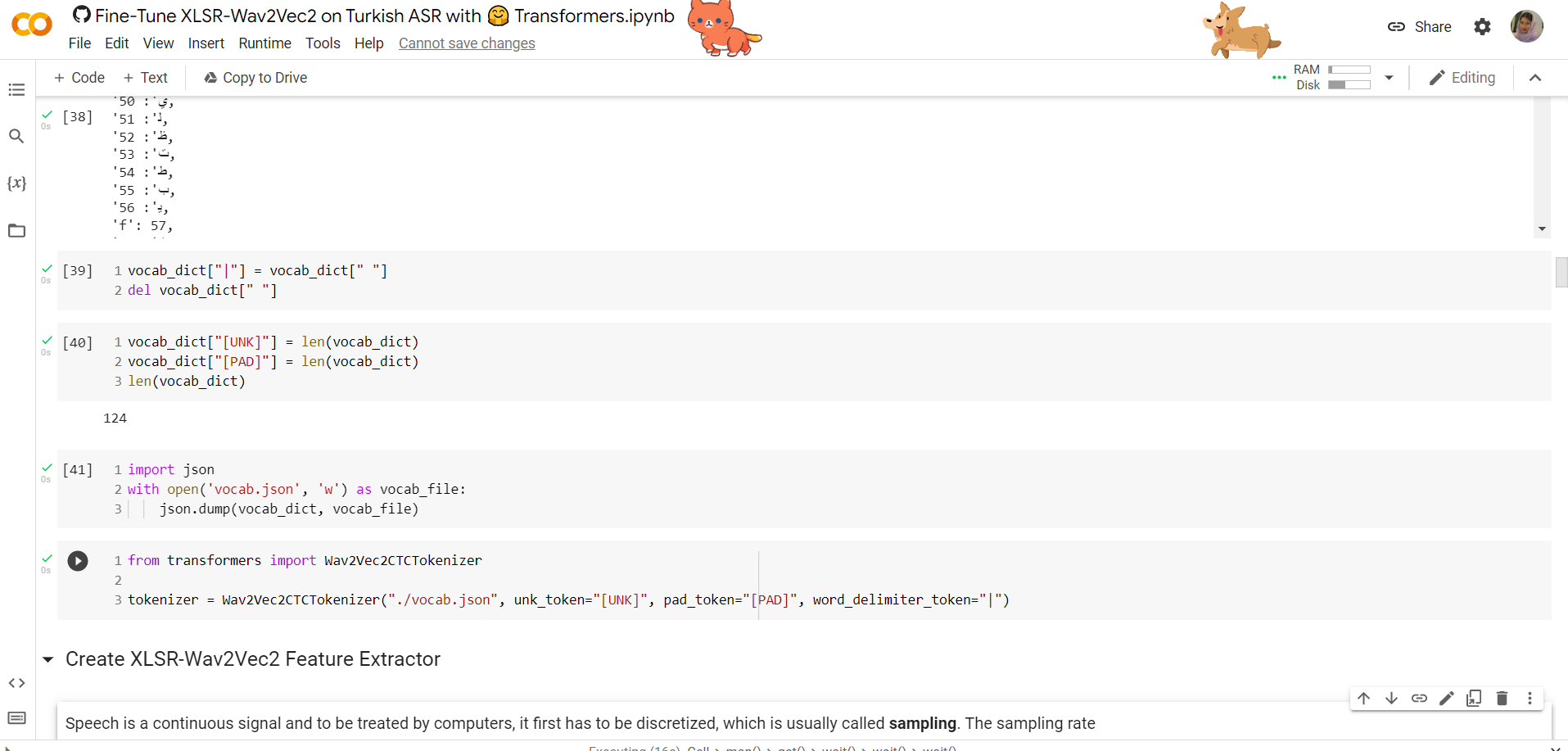
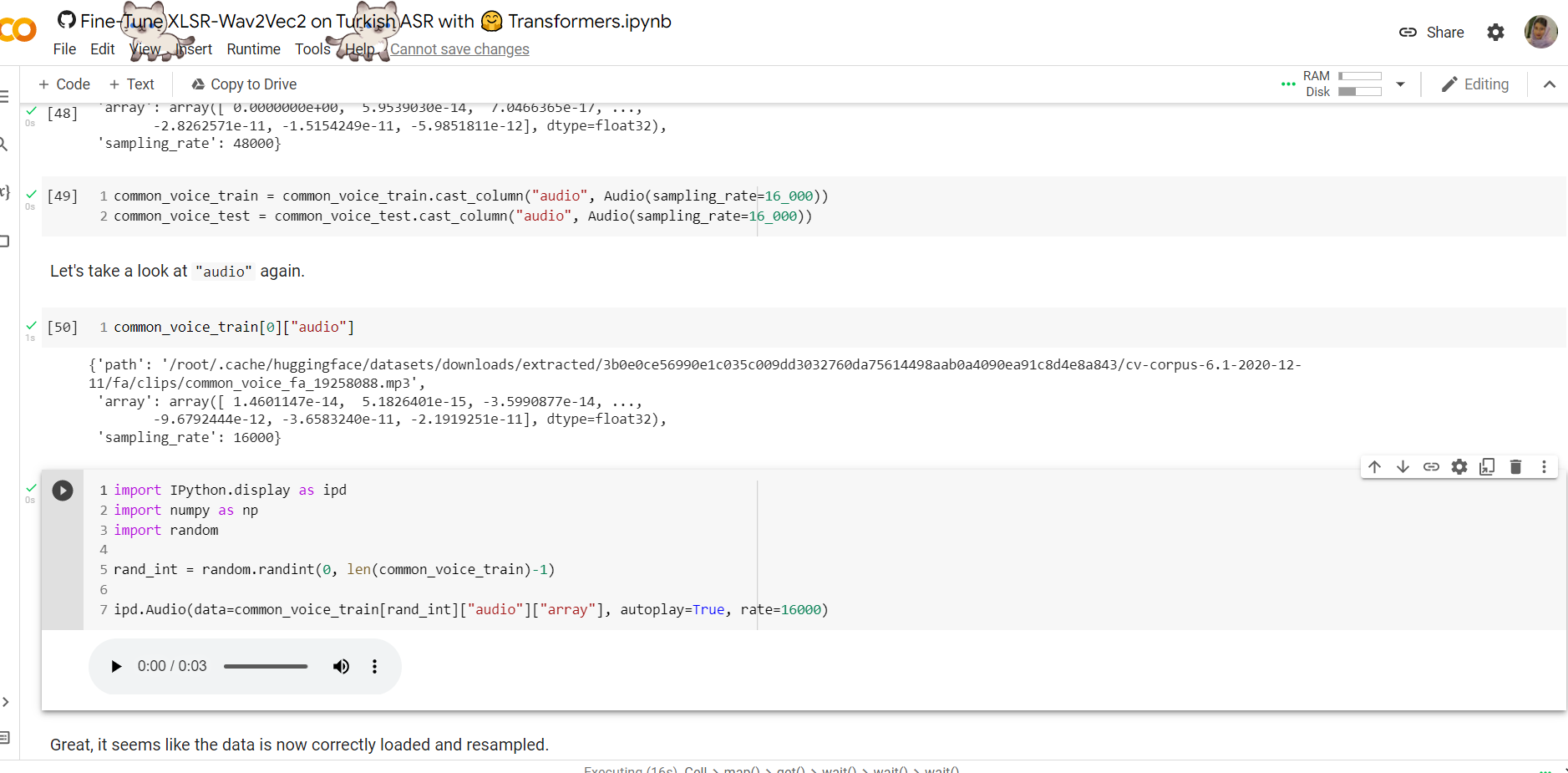
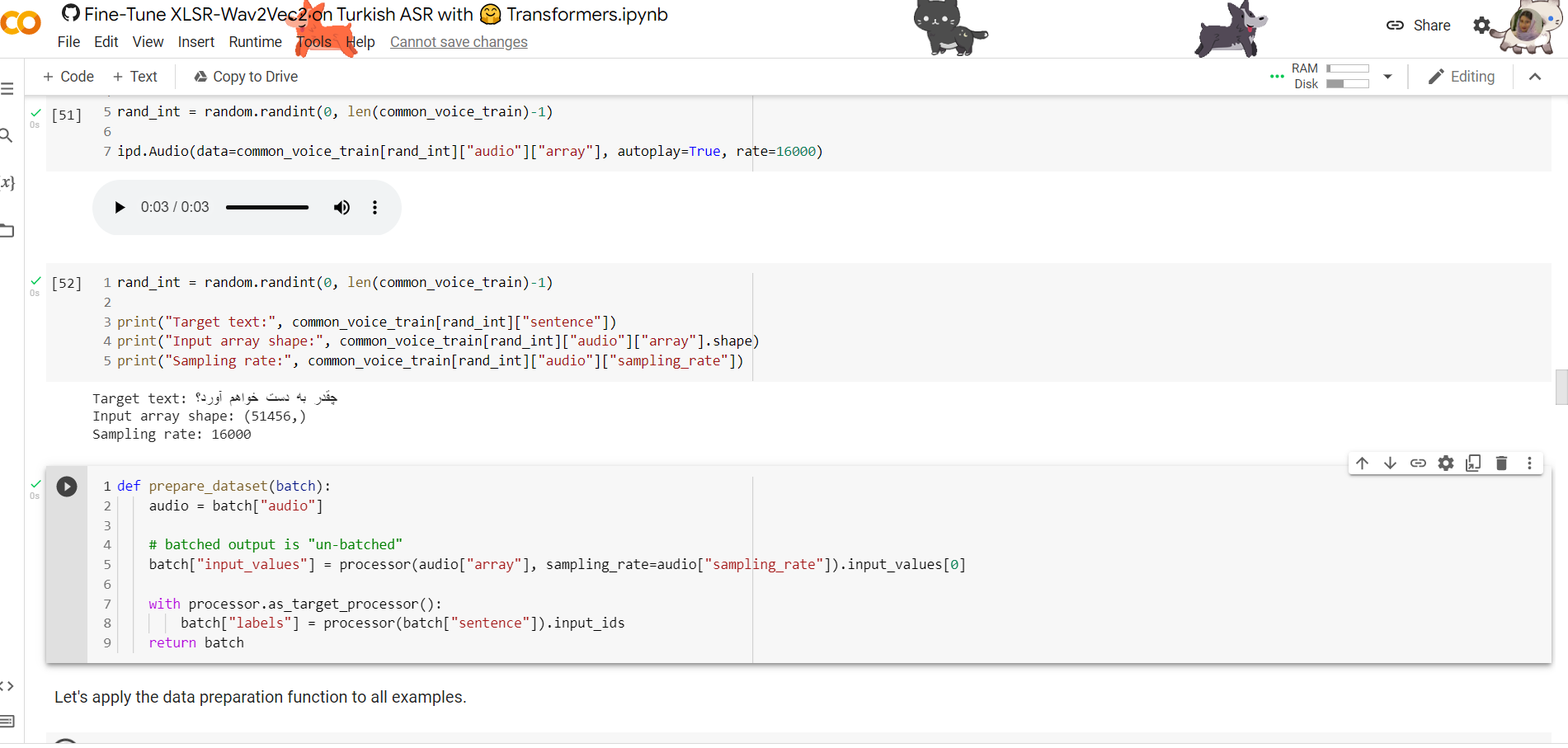
****

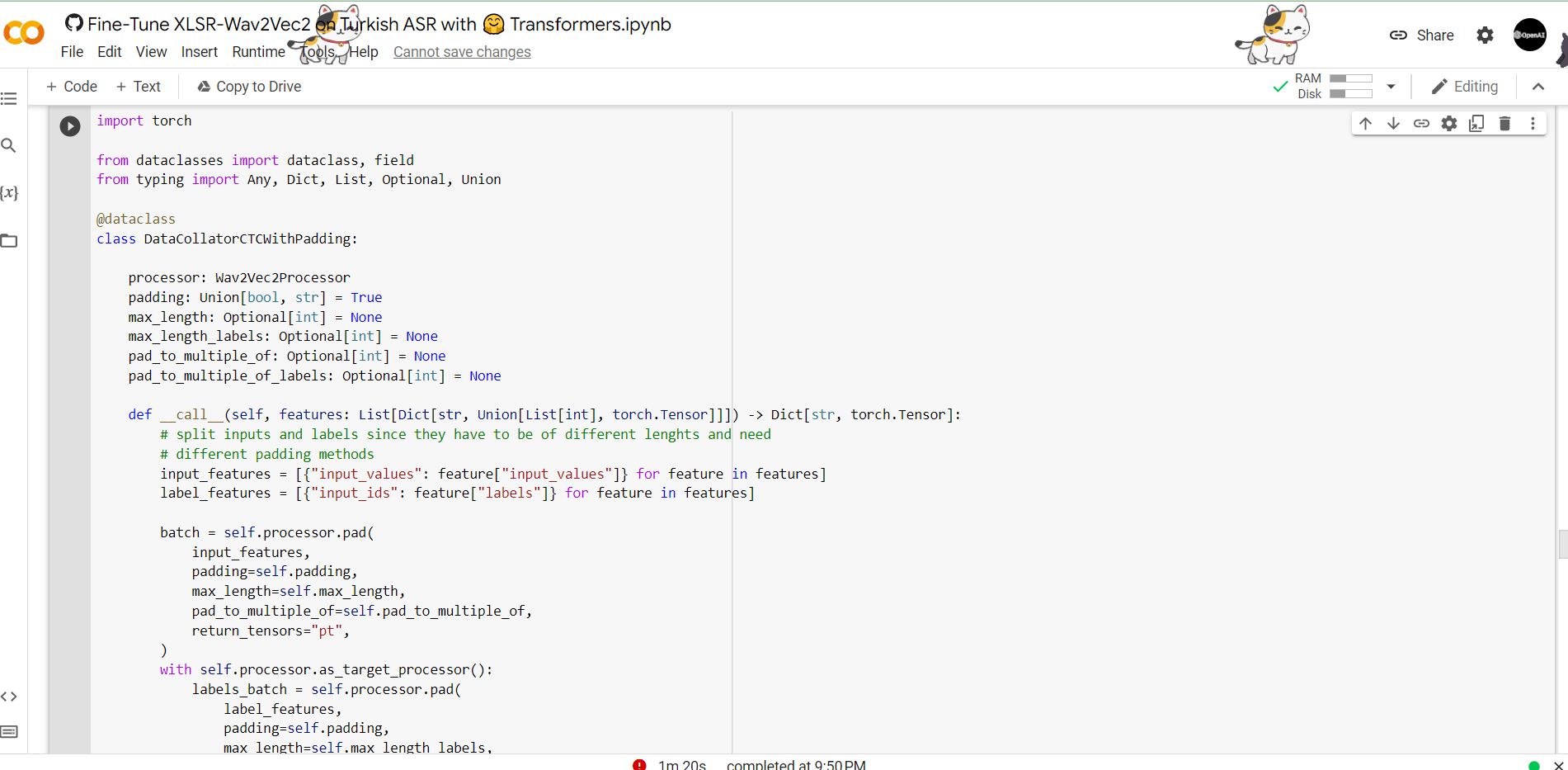
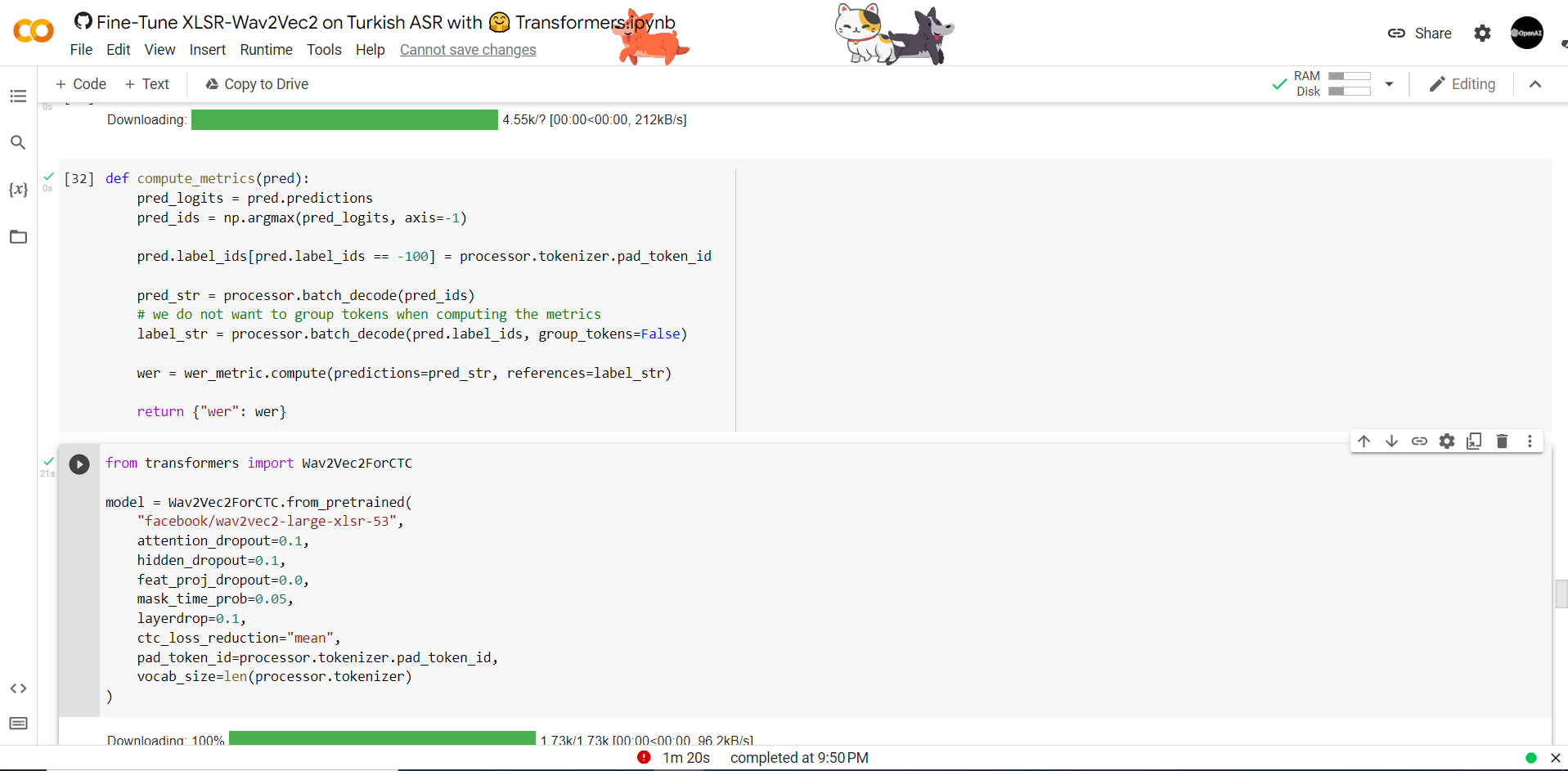
**خروجی**

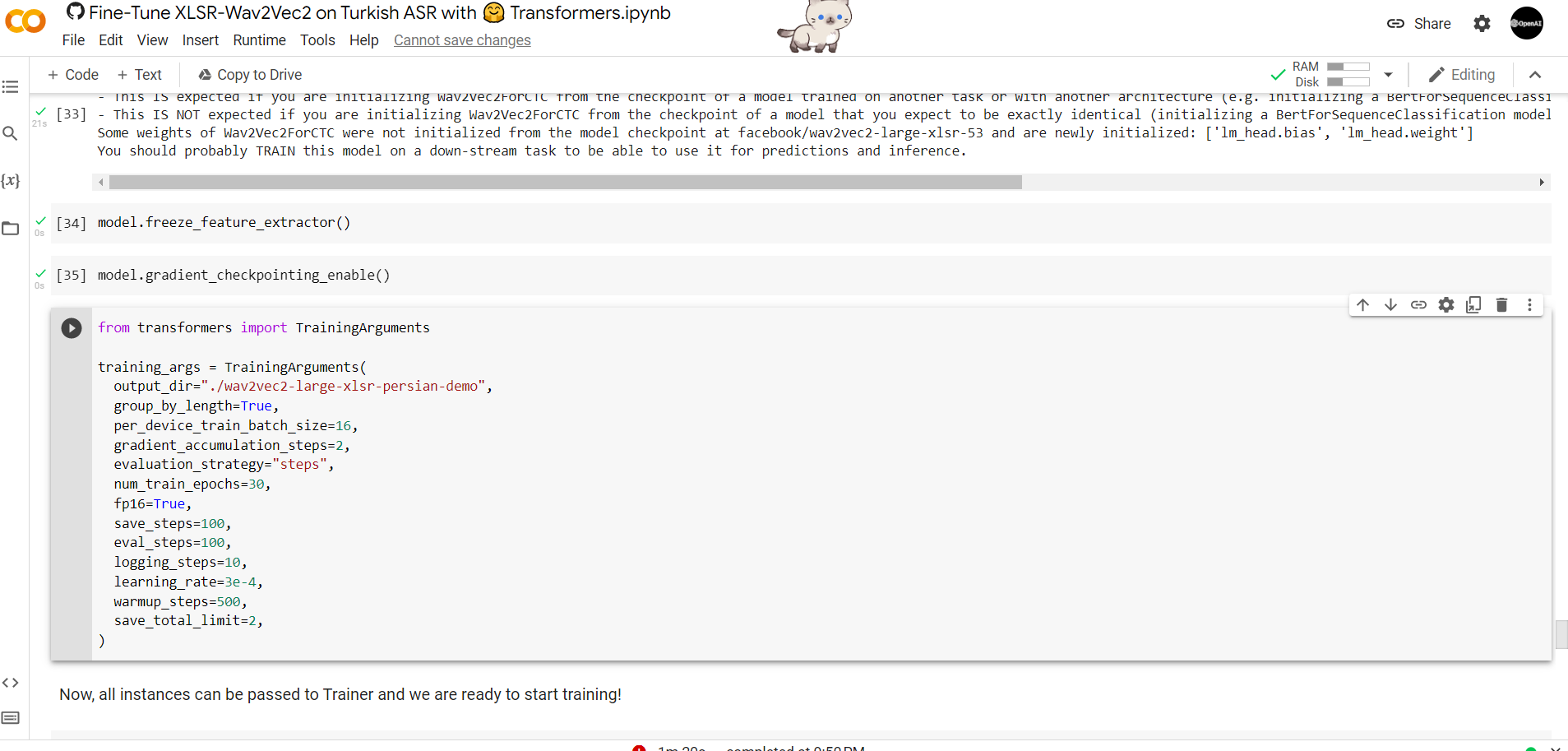
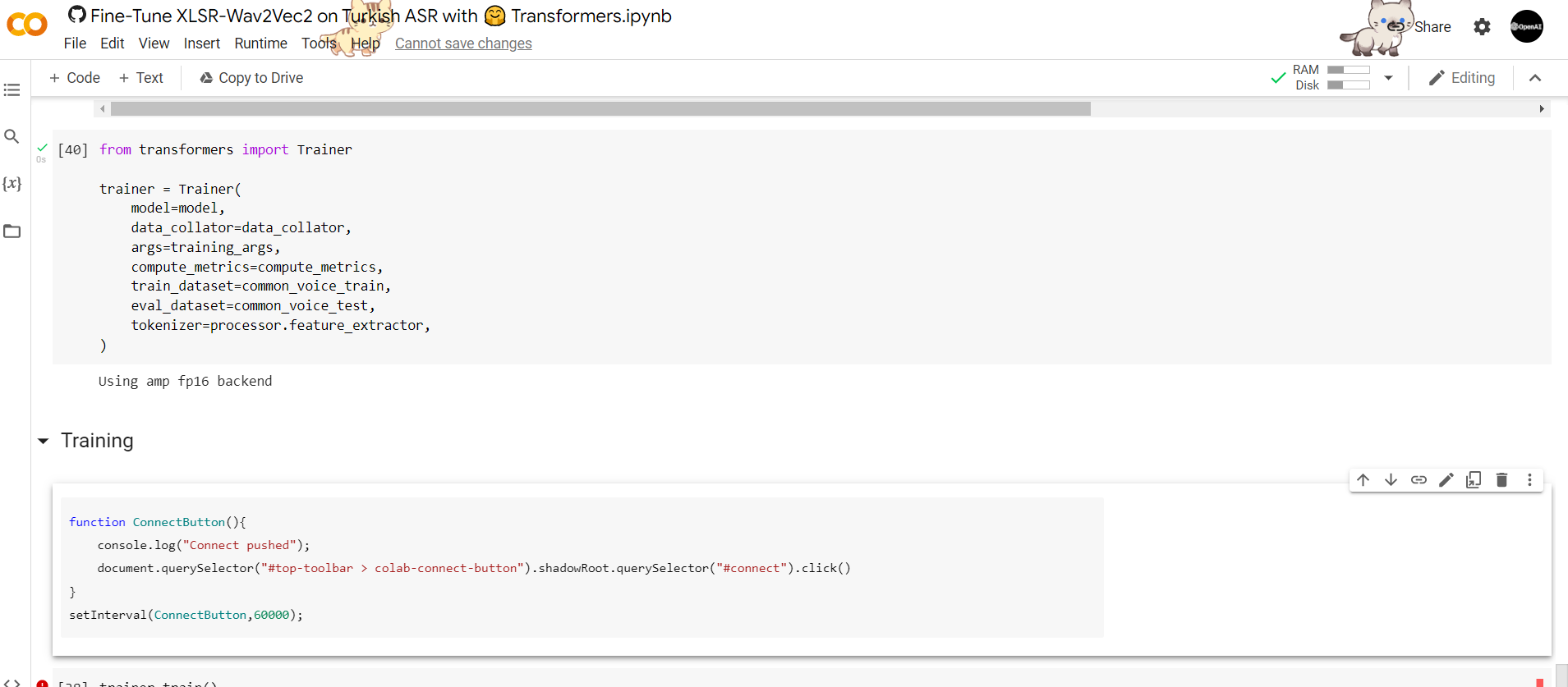
**اجرا در کولب**

****

****

****

****

****