



به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

مدل‌های مولد عمیق

تمرین شماره دو

نام و نام خانوادگی	کیانا هوشانفر
شماره دانشجویی	۸۱۰۱۰۱۳۶۱
تاریخ ارسال گزارش	

## فهرست گزارش سوالات

سوال ۱.....	۳
سوال ۲.....	۲۶
الف).....	۲۶
ب).....	۳۳
ج).....	۴۳
د).....	۴۷

## سوال ۱

```
!pip install --upgrade transformers
!pip install --upgrade datasets
!pip -qq install bitsandbytes
!pip -qq install peft
```

### - دیتاست

ابتدا کتابخانه های لازم را لود میکنیم، همچنین برای استفاده از مدل این سوال باید احراز هویت هم کنیم:

```
!huggingface-cli login
```

در ابتدا باید مجموعه داده های داده شده را مرتب کنیم و از آن ها conversation و summary را استخراج کنیم:

با استفاده از کدی که در اینترنت پیدا کردم و برای preprocess این دیتاست نوشته شده بود و با تغییر آن دیتاست را به ۳ دسته train-test-val تقسیم کردم و هر کدام را در فایل csv ذخیره کرده ام. کد در لینک زیر قرار داده شده است، ولی برای این سوال بخش هایی از آن را تغییر دادم تا خروجی مورد نظر را بگیرم.

<https://github.com/guyfe/Tweetsumm>

ابتدا دیتاست را لود میکنیم:

```
# Load the dataset from Hugging Face
huggingface_dataset_name = "Salesforce/dialogstudio"
dataset = load_dataset(huggingface_dataset_name, "TweetSumm")
```

خروجی:

```
DatasetDict({
  train: Dataset({
    features: ['original dialog id', 'new dialog id', 'dialog index', 'original dialog info', 'log', 'prompt'],
    num_rows: 879
  })
  validation: Dataset({
    features: ['original dialog id', 'new dialog id', 'dialog index', 'original dialog info', 'log', 'prompt'],
    num_rows: 110
  })
  test: Dataset({
    features: ['original dialog id', 'new dialog id', 'dialog index', 'original dialog info', 'log', 'prompt'],
    num_rows: 110
  })
})
```

مشاهده میکنیم که دیتاست نیاز به تمیز شدن دارد برای اینکار از کدهای زیر استفاده میکنیم:

```
class Turn:
    def __init__(self, is_agent: bool, sentences: List[str]):
        self._is_agent = is_agent
        self.sentences = sentences

    def is_agent(self):
        return self._is_agent

    def get_sentences(self):
        return self.sentences

    def get_json(self) -> str:
        return json.dumps({
            'is_agent': self.is_agent(),
            'sentences': self.sentences
        })

    def __str__(self):
        return ('Agent:\t' if self.is_agent() else 'Customer:\t') + '
'.join(self.sentences)

class Dialog:
    def __init__(self, dialog_id: str, turns: List[Turn]):
        self.dialog_id = dialog_id
        self.turns = turns

    def get_dialog_id(self) -> str:
        return self.dialog_id

    def get_turns(self) -> List[Turn]:
        return self.turns

    def get_json(self) -> str:
        turn_list = [json.loads(turn.get_json()) for turn in self.turns]
        return json.dumps({
            'dialog_id': self.dialog_id,
            'turns': turn_list
        })

    def __str__(self):
        result = self.dialog_id + '\n'
        for turn in self.turns:
            result += '\t' + str(turn) + '\n'
        return result
```

```

class DialogWithSummaries:
    def __init__(self, dialog_id: str, turns: List[Turn],
                  extractive_summaries: List[List[Turn]],
                  abstractive_summaries: List[List[str]]):
        self.dialog = Dialog(dialog_id, turns)
        self.extractive_summaries = extractive_summaries
        self.abstractive_summaries = abstractive_summaries

    def get_dialog(self) -> Dialog:
        return self.dialog

    def get_extractive_summaries(self) -> List[List[Turn]]:
        return self.extractive_summaries

    def get_abstractive_summaries(self) -> List[List[str]]:
        return self.abstractive_summaries

    def get_json(self) -> str:
        dialog = json.loads(self.dialog.get_json())
        extractive_summaries_json = [[json.loads(turn.get_json()) for turn in
summary] for summary in self.extractive_summaries]
        abstractive_summaries_json = self.abstractive_summaries

        return json.dumps({
            'dialog': dialog,
            'summaries': {
                'extractive_summaries': extractive_summaries_json,
                'abstractive_summaries': abstractive_summaries_json
            }
        })

    def __str__(self):
        result = str(self.dialog) + '\nExtractive summaries:\n'
        for cnt, summary in enumerate(self.extractive_summaries):
            result += f'{cnt}:\n' + '\n'.join('\t' + str(turn) for turn in
summary) + '\n'

        result += 'Abstractive summaries:\n'
        for cnt, summary in enumerate(self.abstractive_summaries):
            result += f'{cnt}:\n\t' + ' '.join(summary) + '\n'

        return result

```

این کد پایتون سه کلاس را برای نمایش داده های گفتگوی ساخت یافته تعریف می کند. کلاس Turn یک چرخش واحد در یک مکالمه را نشان می دهد که از طرف یک نماینده یا مشتری و لیست جملات موجود در آن نوبت را ذخیره می کند. کلاس Dialog یک نمونه دیالوگ را با یک شناسه منحصر

به فرد و لیستی از نوبت ها در بر می گیرد. کلاس DialogWithSummaries با ترکیب خلاصه‌های استخراجی و انتزاعی، Dialog را گسترش می‌دهد. شامل روش‌هایی برای تبدیل داده‌ها به فرمت JSON است و نمایش‌های رشته‌ای را برای خوانایی آسان‌تر توسط انسان ارائه می‌دهد.

کد زیر را برای بدست آوردن conversation و summary نوشته شده است:

```
class NewTweetSumProcessor:
    def __init__(self, dataset):
        self.dialogs = [self.process_dialog(dialog_data) for dialog_data in
dataset]

    def process_dialog(self, dialog_data):
        dialog_id = dialog_data['new dialog id']
        original_dialog_info = json.loads(dialog_data['original dialog info'])
        turns = []
        for turn_data in dialog_data['log']:
            turns.extend(self.process_turn(turn_data))
        extractive_summaries =
self.get_summaries(original_dialog_info['summaries']['extractive_summaries'])
        abstractive_summaries =
original_dialog_info['summaries']['abstractive_summaries']
        return DialogWithSummaries(dialog_id, turns, extractive_summaries,
abstractive_summaries)

    def process_turn(self, turn_data):
        turns = []
        # Add user utterance as a customer turn if present
        if 'user utterance' in turn_data and turn_data['user utterance']:
            turns.append(Turn(False, [turn_data['user utterance']]))
        # Add system response as an agent turn if present
        if 'system response' in turn_data and turn_data['system response']:
            turns.append(Turn(True, [turn_data['system response']]))
        return turns

    def get_summaries(self, extractive_summary_data):
        summaries = []
        for summary in extractive_summary_data:
            turns = [Turn(item['is_agent'], item['sentences']) for item in
summary]
            summaries.append(turns)
        return summaries
```

این کد پایتون یک کلاس NewTweetSumProcessor را تعریف می‌کند که مجموعه داده ای از دیالوگ های مربوط به توییت را پردازش می‌کند. با ایجاد فهرستی از نمونه‌های گفتگو DialogWithSummaries با پردازش هر ورودی در مجموعه داده، مقداردهی اولیه می‌شود. روش

process\_dialog اطلاعات مرتبط مانند شناسه گفتگو، نوبت ها و خلاصه ها را از مجموعه داده استخراج می کند. روش process\_turn چرخش های فردی را بر اساس گفته های کاربر و پاسخ های سیستم در داده های گفتگو ایجاد می کند. روش get\_summaries خلاصه های استخراجی را از اطلاعات گفتگوی اصلی استخراج و ساختار می دهد. این کلاس برای تبدیل اطلاعات داده خام به یک قالب ساختاریافته مناسب برای تجزیه و تحلیل یا پردازش بیشتر در زمینه گفتگوهای مرتبط با توییت طراحی شده است.

خروجی این قسمت برای یکی از دیالوگ های دیتاست:

```
TweetSum--test--2
Customer: @AirbnbHelp hi , my Acc was linked to an old number. Now I'm asked to verify my Acc , where a code / call will be sent to my old number. Any way that I can link my Acc to my current number? Pls help
Agent: @637109 Hi there, we are here to help. We will have a specialist contact you about changing your phone number. Thank you.
Customer: @AirbnbHelp Thanks. Hope to get in touch soon
Agent: @637109 That is no problem. Please let us know if you have any further questions in the meantime.

Customer: @AirbnbHelp Hi sorry , is it for my account : __email__
Agent: @637109 Can you please delete this post as it does have personal info in it. We have updated your Case Manager https://t.co/WCQEFGLXC @637109 who will be following up with you shortly. Feel free to DM us anytime with any other questions or concerns 2/2
Customer: @AirbnbHelp Thank you
Agent: @637109 That is no problem. Please do not hesitate to contact us with any further questions. Thank you.

Extractive summaries:
0:
Customer: @AirbnbHelp hi , my Acc was linked to an old number.
Customer: Now I'm asked to verify my Acc , where a code / call will be sent to my old number.
Agent: We will have a specialist contact you about changing your phone number.
Agent: We have updated your Case Manager https://t.co/WCQEFGLXC
1:
Customer: @AirbnbHelp hi , my Acc was linked to an old number.
Agent: @637109 Can you please delete this post as it does have personal info in it.
Agent: We have updated your Case Manager https://t.co/WCQEFGLXC
2:
Customer: Any way that I can link my Acc to my current number?
Agent: We will have a specialist contact you about changing your phone number.
Agent: We have updated your Case Manager https://t.co/WCQEFGLXC

Abstractive summaries:
0:
Customer is asking about the ACC to link to the current number. Agent says that they have updated their case manager.
1:
Customer is enquiring that whether they can change the current number to link the account as they have old number which was linked. Agent states that they have updated to case manager, they will look into it and also to DM for further assistance.
2:
Customer wants to link their account to their new number since its asking verification where a code/call will be sent to their old number. Agent replies that they have updated the Case Manager to resolve this issue.
```

شکل ۱ - خروجی NewTweetSumProcessor

در ادامه داده های train-test-val را به این تابع می دهیم و خروجی ها را در فایل csv ذخیره میکنیم:

```
import pandas as pd

def create_dataframe(processor):
    dialogues = []
```

```

summaries = []

for dialog in processor.dialogs:
    # Concatenate all turns into a single string for the dialogue column
    dialogue_str = '\n'.join(str(turn) for turn in
dialog.get_dialog().get_turns())
    dialogues.append(dialogue_str)
    # Process and concatenate the first abstractive summary for the summary
column
    if dialog.get_abstractive_summaries(): # Check if there are abstractive
summaries
        first_summary = '\n'.join(dialog.get_abstractive_summaries()[0])
        summaries.append(first_summary)
    else:
        summaries.append('') # If no abstractive summary, add an empty
string
    return pd.DataFrame({'dialogue': dialogues, 'summary': summaries})
# Process the dataset
processor = NewTweetSumProcessor(dataset['train']) # or 'test', 'validation'
df = create_dataframe(processor)

# Save the DataFrame to a CSV file if needed
df.to_csv('train_dataset.csv', index=False)

```

ما با این کد conversation و یکی از abstractive summary ها را ذخیره کرده اسم..

خروجی یکی از این دیتاست تمیز شده به شکل زیر است:

```

.....
Example 1
.....
INPUT DIALOGUE:
Customer: Bought these biscuits a couple of weeks ago, only just opened and looked at the best before dat
e... @49975
Agent: @393926 We can't see a picture of the best before date or biscuits attached to your tweets here. Please
can you tweet or DM it to us again? Thanks. https://t.co/FqaAtoiNzs
Customer: @marksandspencer https://t.co/bjTjk67Tv1 @marksandspencer https://t.co/W11bv0UvEa @marksandsp
encer https://t.co/1MgEDf0cfM
Agent: @393926 We'd certainly like to take a closer look into this. Please DM us a picture of your full receipt.
https://t.co/FqaAto1cHU
Customer: @marksandspencer Don't have a receipt as they were bought for my grandparents 3 weeks ago and h
ave only just realised the date on them @marksandspencer The biscuits were purchased at the Marks and Spencer's
Store at the Ricoh Arena, Coventry
Agent: @393926 Did yo use a sparks card on your transaction, Cian?
Customer: @marksandspencer Yes
Agent: @393926 Hi Cian. I'm really sorry to see this, especially when it was such a lovely gesture too! No wor
ries though - we got your back ;) 1/3 @393926 Is there a barcode on the tin anywhere? If there is, could you ta
ke a pic and DM it to us please & also let us know how much they cost? 2/3 @393926 Once we have those, w
e'll explain what happens next. Cheers, Steve :) 3/3
.....
BASELINE HUMAN SUMMARY:
Customer bought a tin of biscuits and the use by date was old.
Wants a photo of the tin and date and how much they cost then they'll explain what happens next.
.....

```

شکل ۲ - مثالی از دیتاست تمیز شده



## - in-context learning

بعد از پیش پردازش های مرحله قبل، با استفاده از prompt مناسب، خروجی از مدل میگیریم. در مرحله اول بصورت zero-shot (بدون مثال) و در مرحله بعد بصورت one-shot (همراه یک مثال برای یادگیری بهتر مدل) کدهای زیر با استفاده از داده های موجود در مجموعه دادگان، prompt های مناسب را ایجاد میکند:

### - zero-shot

```
for i, index in enumerate(example_indices):
    dialogue = df.iloc[index]['dialogue']
    summary = df.iloc[index]['summary']

    prompt = f"""
Summarize the following conversation accurately:{dialogue}
Summary:
    """

    # Input constructed prompt instead of the dialogue.
    inputs = tokenizer(prompt, return_tensors='pt')
    inputs = {key: value.to(model.device) for key, value in inputs.items()} #
Move inputs to the same device as the model

    output = tokenizer.decode(
        model.generate(
            inputs["input_ids"],
            max_length=500,
            num_beams=1,
            temperature=0.8,
        )[0],
        skip_special_tokens=True
    )

    # Splitting the output to get only the summary
    split_output = output.split("Summary:")
    generated_summary = split_output[1].strip() if len(split_output) > 1 else
"Summary not generated properly."

    print(dash_line)
    print('Example ', i + 1)
    print(dash_line)
    print(f'INPUT PROMPT:\n{prompt}')
    print(dash_line)
    print(f'BASELINE HUMAN SUMMARY:\n{summary}')
    print(dash_line)
    print(f'MODEL GENERATION - ZERO SHOT:\n{output}\n')
```

این کد برای هر مثال، با گنجاندن دیالوگ، یک اعلان ایجاد می‌کند و از مدل درخواست می‌کند تا خلاصه‌ای را با استفاده از مدل مشخص شده ایجاد کند. درخواست ورودی نشانه گذاری می‌شود و سپس برای تولید خلاصه به مدل وارد می‌شود. خروجی تولید شده رمزگشایی می‌شود و خلاصه واقعی از مجموعه داده مقایسه می‌شود. نتایج، شامل اعلان ورودی، خلاصه اصلی دیتاست، و خلاصه تولید شده zero-shot مدل، برای تجزیه و تحلیل و ارزیابی چاپ می‌شوند.

خروجی:

#### BASELINE HUMAN SUMMARY:

Customer bought a tin of biscuits and the use by date was old.

Wants a photo of the tin and date and how much they cost then they'll explain what happens next.

#### MODEL GENERATION - ZERO SHOT:

The customer bought biscuits from Marks and Spencer and opened them to find that the best before date was after the date of purchase. The customer tweeted the store to complain. The store responded by asking for a picture of the receipt and the best before date. The customer responded by providing a picture of the receipt and the best before date. The store responded by asking for a picture of the barcode on the tin. The customer responded by providing a picture of the barcode on

مشاهده میکنیم که خروجی مدل مناسب است.

- One-shot

در این قسمت یک مثال نیز در prompt می‌آوریم:

```
def one_shot_prompt(example=20, summarize=10):
    prompt = ''
    dialogue = df.iloc[example]['dialogue']
    summary = df.iloc[example]['summary']
    prompt += f"Summarize this conversation
accurately:\n\n{dialogue}\n\nSummary:{summary}\n\n"
    dialogue = df.iloc[summarize]['dialogue']
    prompt += f"Summarize this conversation
accurately:\n\n{dialogue}\n\nSummary:"

    return prompt
one_shot_prompt = one_shot_prompt()

print(one_shot_prompt)
```

این کد تابع one\_shot\_prompt را تعریف می‌کند که یک دستور برای یک کار خلاصه‌سازی one-shot می‌سازد. برای استخراج دیالوگ و اطلاعات خلاصه از یک «df» (DataFrame) به دو شاخص «مثال»

و «خلاصه» نیاز است. تابع با الحاق اطلاعات گفتگو و خلاصه برای هر دو مثال مشخص شده یک درخواست را تشکیل می دهد. سپس دستور به دست آمده چاپ می شود. علاوه بر این، تابع با شاخص های پیش فرض فراخوانی می شود (۲۰ برای «مثال» و ۱۰ برای «خلاصه»)، و اعلان تولید شده چاپ می شود. خروجی در فایل کد هست. (برای طولانی نشدن گزارش خروجی را در این قسمت نیاوردم)

در ادامه پرامتی که ساختیم را به مدل می دهیم (همانند قسمت قبل):

```
summary = df.iloc[10]['summary']

inputs = tokenizer(one_shot_prompt, return_tensors='pt')
inputs = {key: value.to(model.device) for key, value in inputs.items()}

# Existing code to generate the summary
output = tokenizer.decode(
    model.generate(
        inputs["input_ids"],
        max_length=900,
        num_beams=5,
        temperature=0.7,
        no_repeat_ngram_size=2
    )[0],
    skip_special_tokens=True
)

print(dash_line)
print(f'BASELINE HUMAN SUMMARY:\n{summary}\n')
print(dash_line)
print(f'MODEL GENERATION - ONE SHOT:\n{output}')
```

خروجی:

- BASELINE HUMAN SUMMARY:

Customer bought a tin of biscuits and the use by date was old. Wants a photo of the tin and date and how much they cost then they'll explain what happens next.

- MODEL GENERATION - ZERO SHOT:

The customer bought biscuits from Marks and Spencer and opened them to find that the best before date was after the date of purchase. The customer tweeted the store to complain. The store responded by asking for a picture of the receipt and the best before date. The customer responded by providing a picture of the receipt and the best before date. The store responded by asking for a picture of the barcode on the tin. The customer responded by providing a picture of the barcode on

## MODEL GENERATION - ONE SHOT:

Marks and Spencers customer is not happy with the expiry date of a biscuit pack he/she bought. The agent is trying to get more information from the customer.

مشاهده میکنیم که در این حالت، خلاصه ی بهتری به ما داده شده است.

دلایل متعددی وجود دارد که چرا درخواست های oneshot ممکن است در سناریوهای خاص پاسخ های بهتری را ارائه دهند:

۱. وضوح متن: اعلان های oneshot اغلب زمینه واضح و خاصی را برای مدل فراهم می کنند. زمانی که اطلاعات مورد نیاز برای یک پاسخ مربوطه مختصر و در یک اعلان موجود باشد، می تواند سودمند باشد. ابهام را کاهش می دهد و تمرکز مدل را بر جزئیات مربوطه آسان تر می کند.

۲. تعریف وظیفه: اگر وظیفه یا سوالی که برای مدل مطرح می شود به خوبی تعریف شده باشد و بتوان آن را به اندازه کافی در یک اعلان بیان کرد، رویکرد oneshot ممکن است موثرتر باشد. این ارتباط کار با مدل را بدون ایجاد پیچیدگی غیر ضروری ساده می کند.

۳. اجتناب از ابهام: اعلان های گسترده تر ممکن است باعث ایجاد ابهام یا چندین تفسیر احتمالی شود. درخواست های oneshot دامنه ورودی را محدود می کند، احتمال سوء تفاهم مدل از قصد کاربر و ایجاد پاسخ های مرتبط کمتر را کاهش می دهد.

۴. الگوهای داده های آموزشی: اگر مدل بر روی مجموعه داده ای آموزش داده شده باشد که در آن اعلان های oneshot رایج بوده یا الگوهایی را نشان می دهد که به نفع چنین ورودی است، ممکن است در طول استنتاج به ورودی های مشابه تعمیم داده شود.

در نتیجه در این حالت oneshot به جواب بهتری رسیده ایم. معیار rouge هم همین مسئله را نشان می دهد که در قسمت آخر آورده شده است.

## Finetune with LoRA -

در گام دوم، مدل را finetune میکنیم:

LoRAها (تطبیق های رتبه پایین Low-Rank Adaptation) فایل های کوچکتری هستند (از ۱ مگابایت تا ۲۰۰ مگابایت) که آنها را با مدل های Stable Diffusion موجود ترکیب می کنیم تا مفاهیم جدیدی را به مدل های خود معرفی کنید تا مدل بتواند این مفاهیم را تولید کند. این مفاهیم جدید به طور

کلی تحت ۱ از ۲ دسته قرار می گیرند: موضوعات یا سبک ها. PEFT که مخفف Parameter-Efficient Fine-Tuning است چارچوبی شامل پیاده سازیهای مختلفی از fine-tuning، از جمله LoRa ارائه می دهد. کارایی پارامتر: تنظیم دقیق مدل های بزرگ به طور سنتی نیازمند به روز رسانی تمام پارامترهای مدل است. این فرآیند از نظر محاسباتی گران است و ذخیره سازی فشرده است. با این حال، LoRa امکان تنظیم دقیق مدل را تنها با به روزرسانی بخش کوچکی از پارامترهای مدل فراهم می کند.

ماتریس های رتبه پایین: LoRa با معرفی ماتریس های رتبه پایین که تغییرات ماتریس های وزن مدل را تقریبی می کند، کار می کند. به جای به روز رسانی کل ماتریس وزن (که در مدل های بزرگ بسیار بزرگ است)، LoRa ماتریس های کوچکتر و با رتبه پایین را به روز می کند. سپس از این ماتریس های با رتبه پایین برای تقریب تغییرات در ماتریس های وزن واقعی در طول تمرین استفاده می شود.

حفظ پارامترهای اصلی: در LoRa، پارامترهای اصلی مدل از پیش آموزش دیده ثابت نگه داشته می شوند. فقط ماتریس های اضافی با رتبه پایین آموزش داده می شوند. این بدان معنی است که دانش مدل پایه حفظ می شود، در حالی که هنوز به مدل اجازه می دهد تا با وظایف یا مجموعه داده های خاص سازگار شود.

کارایی و مقیاس پذیری: از آنجایی که LoRa فقط تعداد کمی از پارامترها را آموزش می دهد، از نظر منابع محاسباتی و استفاده از حافظه در مقایسه با روش های تنظیم دقیق سنتی بسیار کارآمدتر است. این آن را برای تنظیم دقیق مدل های بزرگ در وظایف یا مجموعه داده های خاص بدون نیاز به منابع گسترده مقیاس پذیرتر می کند.

برنامه ها: LoRa را می توان در برنامه های مختلفی که نیاز به سفارشی سازی یا انطباق مدل دارند، مانند درک زبان خاص دامنه، سیستم های توصیه شخصی یا تولید زبان خاص کار استفاده کرد.

ابتدا مدل را لود میکنیم:

```
tokenizer = AutoTokenizer.from_pretrained("stabilityai/stablelm-3b-4e1t",
use_auth_token=True)
model = AutoModelForCausalLM.from_pretrained("stabilityai/stablelm-3b-4e1t",
torch_dtype=torch.bfloat16, use_auth_token=True, trust_remote_code=True)
```

مدل را برای اینکه به محدودیت سخت افزار نخوریم torch.bfloat16 لود کردیم. و دیتاست را به شکل زیر لود خواهیم زد:

```

dataset = load_dataset('csv', data_files={
    'train': 'train_dataset.csv',
    'validation': 'validation_dataset.csv',
    'test': 'test_dataset.csv'
})

# Add [PAD] as the padding token
tokenizer.add_special_tokens({'pad_token': '[PAD]'})

def tokenize_function(examples):
    start_prompt = 'Summarize this conversation accurately:\n\n'
    end_prompt = '\n\nSummary: '
    prompts = [start_prompt + dialogue + end_prompt for dialogue in
examples["dialogue"]]
    # Tokenizing the prompts and the summaries
    tokenized_inputs = tokenizer(prompts, padding="max_length", truncation=True,
max_length=1024, return_tensors="pt")
    tokenized_labels = tokenizer(examples["summary"], padding="max_length",
truncation=True, max_length=1024, return_tensors="pt")
    # Explicitly convert input_ids to torch.long
    examples['input_ids'] = tokenized_inputs.input_ids.to(torch.long)
    examples['labels'] = tokenized_labels.input_ids
    return examples
# Apply the function to the dataset
tokenized_datasets = dataset.map(tokenize_function, batched=True)

```

این کد با افزودن یک توکن padding PAD یک توکنایزر را گسترش می دهد. سپس یک تابع توکن سازی به نام tokenize\_function تعریف می کند که مجموعه ای از دیالوگ ها و خلاصه های مربوط به آنها را می گیرد. این تابع با اضافه کردن نشانه های شروع و پایان خاص، دستورهای را برای هر گفتگو ایجاد می کند. با استفاده از توکنایزر پیکربندی شده، اعلان ها و خلاصه ها را نشانه گذاری می کند و از حداکثر طول ۱۰۲۴ توکن با padding و truncation اطمینان حاصل می کند. ورودی ها و برچسب های نشانه سازی شده به ترتیب در مجموعه داده ها تحت کلیدهای input\_ids و labels ذخیره می شوند. علاوه بر این، input\_id ها به صراحت به نوع torch.long تبدیل می شوند. این تابع با استفاده از تابع «نقشه» بر روی کل مجموعه داده اعمال می شود، و در نتیجه یک مجموعه داده نشانه گذاری شده مناسب برای آموزش یک مدل زبان در وظایف خلاصه سازی گفتگو ایجاد می شود.

در کل با تابع بالا دیتاست را tokenize میکنیم.

قبل از اعمال کردن LoRA مشاهده میکنیم که تمام پارامترهای مدل قابل ترین شدن هستند حدود

۳ میلیارد پارامتر

trainable model parameters: 2795443200  
all model parameters: 2795443200  
percentage of trainable model parameters: 100.00%

اگر  $r$  را برابر ۱۶ بگیریم:

```
lora_r = 16
lora_alpha = 64
lora_dropout = 0.1
lora_target_modules = [
    "q_proj",
    "v_proj"
]

peft_config = LoraConfig(
    r=lora_r,
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    target_modules=lora_target_modules,
    bias="none",
    task_type="CAUSAL_LM",
)

peft_model1 = get_peft_model(model, peft_config)
number_of_trainable_model_parameters(peft_model1)
```

trainable model parameters: 5242880  
all model parameters: 2800686080  
percentage of trainable model parameters: 0.19%

اگر  $r$  را برابر ۳۲ بگیریم:

```
lora_r = 32
lora_alpha = 64
lora_dropout = 0.1
lora_target_modules = [
    "q_proj",
    "v_proj"
]

peft_config = LoraConfig(
    r=lora_r,
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    target_modules=lora_target_modules,
    bias="none",
    task_type="CAUSAL_LM",
)
```

```
)

peft_model2 = get_peft_model(model, peft_config)
number_of_trainable_model_parameters(peft_model2)
```

trainable model parameters: 10485760  
 all model parameters: 2805928960  
 percentage of trainable model parameters: 0.37%

مشاهده میکنیم که با اعمال LoRA تعداد پارامترهای مدل قابل ترین به قدر قابل توجهی کمتر شده است. هرچقدر پارامتر rank کمتر باشد تعداد پارامترهای قابل ترین هم کمتر است. پارامتر  $r$  رتبه ماتریس های به روزرسانی را مشخص میکند. پارامتر  $\text{lora\_alpha}$  نرخ یادگیری برای ماتریس های به روزرسانی را کنترل میکند. پارامتر  $\text{target\_modules}$  لایه های LLM را مشخص میکند. پارامتر  $\text{lora\_dropout}$  dropout را به ماتریس های به روزرسانی اعمال میکند. پارامتر  $\text{bias}$  مشخص می کند که آیا ماتریس ها دارای یک عبارت بایاس باشند. پارامتر  $\text{task\_type}$  نوع وظیفه ای را مشخص میکند که LLM برای آن fine-tune می شود.

لایه هایی که تحت تاثیر اثرات Lora قرار می گیرند، همان لایه هایی هستند که در  $\text{lora\_target\_modules}$  مشخص شده اند. در این حالت،  $\text{q\_proj}$  و  $\text{v\_proj}$  بخشی از مکانیسم توجه در لایه های ترانسفورماتور هستند.  $\text{q\_proj}$  به لایه query projection layer و  $\text{v\_proj}$  به لایه value projection layer اشاره دارد. این لایه ها در محاسبه attention scores در ترانسفورماتورها نقش دارند و این لایه ها قابل ترین هستند.

Parameter: base\_model.model.model.layers.0.self\_attn.q\_proj.lora\_A.default.weight, Requires Gradient: True  
 Parameter: base\_model.model.model.layers.0.self\_attn.q\_proj.lora\_B.default.weight, Requires Gradient: True  
 Parameter: base\_model.model.model.layers.0.self\_attn.k\_proj.weight, Requires Gradient: False  
 Parameter: base\_model.model.model.layers.0.self\_attn.v\_proj.base\_layer.weight, Requires Gradient: False  
 Parameter: base\_model.model.model.layers.0.self\_attn.v\_proj.lora\_A.default.weight, Requires Gradient: True  
 Parameter: base\_model.model.model.layers.0.self\_attn.v\_proj.lora\_B.default.weight, Requires Gradient: True  
 Parameter: base\_model.model.model.layers.0.self\_attn.o\_proj.weight, Requires Gradient: False  
 Parameter: base\_model.model.model.layers.0.mlp.gate\_proj.weight, Requires Gradient: False

میبینیم که این لایه ها Requires Gradient: True است.



برای فایل تیون کردن به شکل زیر عمل میکنیم:

```
peft_training_args = TrainingArguments(
    output_dir='./results_model1',
    num_train_epochs=2,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    warmup_steps=2,
    max_steps=7,
    weight_decay=0.01,
    logging_dir='./logs',
)
peft_trainer1 = Trainer(
    model=peft_model1,
    args=peft_training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"]
)
peft_training_args = TrainingArguments(
    output_dir='./results_model2',
    # num_train_epochs=2,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    warmup_steps=2,
    max_steps=7,
    weight_decay=0.01,
    logging_dir='./logs2',
)
peft_trainer2 = Trainer(
    model=peft_model2,
    args=peft_training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"]
)
```

این کد پیکربندی آموزشی را برای فرآیند PEFT تنظیم می کند و آموزش یک مدل PEFT را با استفاده از کتابخانه Hugging Face Transformers آغاز می کند. کلاس TrainingArguments برای تعریف تنظیمات آموزشی مختلف، مانند فهرست خروجی برای ذخیره نتایج مدل، اندازه های دسته ای برای آموزش و ارزیابی، مراحل گرم کردن، حداکثر مراحل تمرین، کاهش وزن، و فهرست ورود به سیستم استفاده می شود. متعاقباً، یک نمونه (peft\_trainer1) Trainer با مدل PEFT مشخص شده peft\_model1 آرگومان های آموزشی peft\_training\_args و مجموعه داده های توکن شده برای آموزش و اعتبار سنجی ایجاد می شود. در نهایت، روش train فراخوانده می شود تا فرآیند آموزش را آغاز کند، جایی که مدل بر اساس مجموعه داده های ارائه شده و پیکربندی آموزشی تنظیم می شود. این کد برای تنظیم دقیق یک

مدل PEFT در یک کار خاص با استفاده از آرگومان های آموزشی تعریف شده و مجموعه داده طراحی شده است.

با دستور زیر مدل ها را ترین میکنیم:

```
peft_trainer1.train()
peft_trainer2.train()
```

کلاس Trainer در کتابخانه Hugging Face Transformers یک رابط سطح بالا است که آموزش و ارزیابی مدل های یادگیری ماشین را تسهیل می کند. پیچیدگی مدیریت حلقه های آموزشی، بهینه سازی و معیارهای ارزیابی را از بین می برد و راهی راحت و کارآمد برای آموزش مدل ها ارائه می کند. در کد ارائه شده، نمونه peft\_trainer1 از کلاس Trainer برای آموزش یک مدل PEFT استفاده شده است. متد train() سپس فراخوانی می شود تا فرآیند آموزش را آغاز کند. این روش حلقه آموزشی را برای تعداد معین دوره یا مراحل اجرا می کند و مدل را بر اساس مجموعه داده آموزشی ارائه شده و پارامترهای پیکربندی تنظیم می کند. اساساً، peft\_trainer1.train() مسئول آموزش مدل PEFT و به روز رسانی پارامترهای آن برای بهینه سازی عملکرد در کار مشخص شده است.

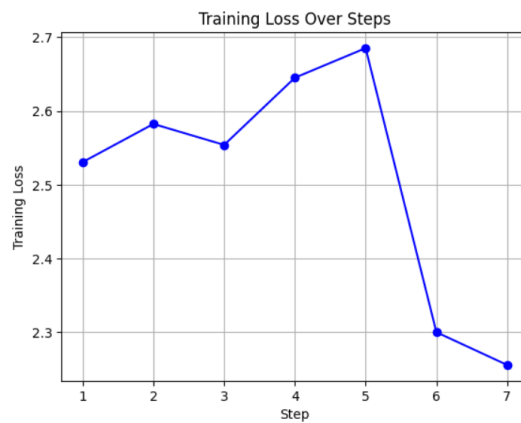
خروجی بخش ترین دو مدل:

مدل اول:

Step	Training Loss
1	2.530700
2	2.582400
3	2.554000
4	2.645100
5	2.685200
6	2.300100
7	2.255700

: TrainOutput(global\_step=7, training\_loss=2.5076094695499966, metrics={'train\_runtime': 9.783, 'train\_samples\_per\_second': 1.431, 'train\_steps\_per\_second': 0.716, 'total\_flos': 229826824765440.0, 'train\_loss': 2.5076094695499966, 'epoch': 0.02})

شکل ۳ - مقادیر loss

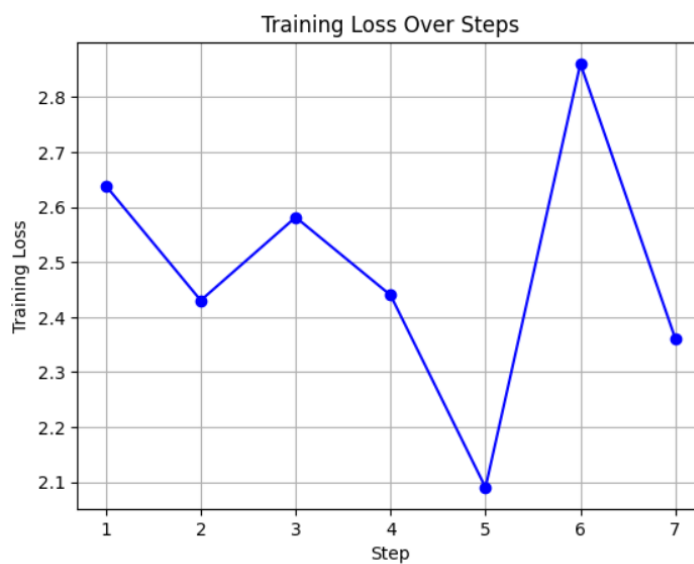


شکل ۴ - نمودار loss

مدل دوم:

Step	Training Loss
1	2.538800
2	2.430200
3	2.581400
4	2.439700
5	2.090300
6	2.860400
7	2.360800

TrainOutput(global\_step=7, training\_loss=2.405596405192485, metrics={'train\_runtime': 21.1729, 'train\_samples\_per\_second': 0.661, 'train\_steps\_per\_second': 0.331, 'total\_flos': 95461740573760.0, 'train\_loss': 2.405596405192485, 'epoch': 0.02})



شکل ۵ - نمودار loss

نمودارهایی که بدست آوردیم، کاهشی بوده اند، برای اینکه ببینیم مدل به خوبی finetune شده است یا نه کد زیر را اجرا میکنیم:

ابتدا مدل هایی که در قسمت های قبل استفاده کردیم را لود می کنیم و خروجی مدل ها را برای مدل اصلی oneshot – zeroshot و مدل finetune شده به ازای ۱۰۰ مقدار از داده های تست بدست می آوریم.

```
tokenizer = AutoTokenizer.from_pretrained("stabilityai/stablelm-3b-4e1t",
use_auth_token=True)
model = AutoModelForCausalLM.from_pretrained("stabilityai/stablelm-3b-4e1t",
torch_dtype=torch.bfloat16,use_auth_token=True, trust_remote_code=True)
model.to("cuda:0")
model.eval()
lora_r = 16
lora_alpha = 64
lora_dropout = 0.1
lora_target_modules = [
    "q_proj",
    "v_proj"
]
peft_config = LoraConfig(
    r=lora_r,
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    target_modules=lora_target_modules,
    bias="none",
    task_type="CAUSAL_LM",
)
peft_model1 = get_peft_model(model, peft_config)
number_of_trainable_model_parameters(peft_model1)
peft_model1 = PeftModel.from_pretrained(peft_model1,
                                        './results_model1/',
                                        torch_dtype=torch.bfloat16,
                                        is_trainable=False)
peft_model2 = PeftModel.from_pretrained(peft_model2,
                                        './results_model2/',
                                        torch_dtype=torch.bfloat16,
                                        is_trainable=False)

peft_model1.to("cuda:0")
peft_model1.eval()
peft_model2.to("cuda:0")
peft_model2.eval()
print("peft_model loaded")
```

این کد یک مدل زبان و یک مدل PEFT را برای وظایف مدل سازی علی زبان با استفاده از کتابخانه Transformers Hugging Face و کتابخانه PEFT مقاردهی می کند. مدل زبان، یک مدل StableLM،

و توکنایزر مربوط به آن از مدل از پیش آموزش دیده stabilityai/stablelm-3b-4e1t بارگیری می‌شوند. سپس مدل PEFT با استفاده از یک شی LoraConfig پیکربندی می‌شود و پارامترهایی مشخص می‌کند. تابع get\_peft\_model برای به دست آوردن یک مدل PEFT finetune شده peft\_model1 بر اساس پیکربندی مشخص شده و مدل زبان اصلی استفاده می‌شود. تابع number\_of\_trainable\_model\_parameters تعداد پارامترهای قابل آموزش را در مدل PEFT محاسبه و چاپ می‌کند. متعاقباً، مدل PEFT از پیش آموزش دیده با استفاده از روش PeftModel.from\_pretrained بارگیری می‌شود، فهرست دایرکتوری، dtype را مشخص می‌کند و آن را به عنوان غیرقابل آموزش تنظیم می‌کند. این فرآیند برای مدل PEFT دیگر peft\_model2 تکرار می‌شود. در نهایت، مدل‌های PEFT به GPU منتقل می‌شوند، روی حالت ارزیابی تنظیم می‌شوند و یک پیام تأیید چاپ می‌شود. به طور کلی، کد یک مدل PEFT را برای مدل سازی زبان علی در بالای یک مدل زبانی از پیش آموزش دیده تنظیم و بارگذاری می‌کند.

```

dialogues = dataset['test'][0:100]['dialogue']
human_baseline_summaries = dataset['test'][0:100]['summary']

model_summaries_zeroshot = []
model_summaries_oneshot = []
peft_model1_summaries = []
peft_model2_summaries = []

generation_config = GenerationConfig(
    max_length=512,
    num_beams=1,
    pad_token_id=tokenizer.eos_token_id,
    temperature=0.7,
)

for idx, dialogue in enumerate(dialogues):
    prompt = f"""
Summarize the following conversation accurately:{dialogue}
Summary:
"""

    input_ids = tokenizer(prompt, return_tensors="pt")
    inputs = {key: value.to("cuda:0") for key, value in input_ids.items()}

    input_ids1 = tokenizer(one_shot_prompt, return_tensors="pt")
    inputs1 = {key: value.to("cuda:0") for key, value in input_ids1.items()}

    human_baseline_text_output = human_baseline_summaries[idx]

```

```

    model_outputs = model.generate(input_ids=inputs["input_ids"],
generation_config=generation_config)
    model_outputs_text_output = tokenizer.decode(model_outputs[0],
skip_special_tokens=True)
    last_summary_part_model_outputs = original_model_text_output.split('Summary:
')[-1]
    model_outputs1 = model.generate(input_ids=inputs1["input_ids"],
generation_config=generation_config)
    model_outputs_text_output1 = tokenizer.decode(model_outputs1[0],
skip_special_tokens=True)
    last_summary_part_model_outputs1 =
original_model_text_output1.split('Summary: ')[-1]

    peft_model_outputs1 = peft_model1.generate(input_ids=inputs["input_ids"],
generation_config=generation_config)
    peft_model_text_output1 = tokenizer.decode(peft_model_outputs1[0],
skip_special_tokens=True)
    last_summary_part1 = peft_model_text_output1.split('Summary: ')[-1]

    peft_model_outputs2 = peft_model2.generate(input_ids=inputs["input_ids"],
generation_config=generation_config)
    peft_model_text_output2 = tokenizer.decode(peft_model_outputs2[0],
skip_special_tokens=True)
    last_summary_part2 = peft_model_text_output2.split('Summary: ')[-1]

    model_summaries_zeroshot.append(last_summary_part_model_outputs)
    model_summaries_oneshot.append(last_summary_part_model_outputs1)

    peft_model_summaries1.append(last_summary_part1)
    peft_model_summaries2.append(last_summary_part2)

zipped_summaries = list(zip(human_baseline_summaries, model_summaries_zeroshot,
peft_model1_summaries, peft_model2_summaries))

df = pd.DataFrame(zipped_summaries, columns = ['human_baseline_summaries',
'model_zero-shot', 'model_one-shot',
'peft_model1_summaries', 'peft_model2_summaries'])
df.head()

```

این کد عملکرد خلاصه سازی مدل های مختلف را با ایجاد خلاصه برای مجموعه ای از دیالوگ ها مقایسه می کند. از کتابخانه Transformers Hugging Face استفاده می کند و یک پیکربندی را برای وظایف تولید متن تعریف می کند. دیالوگ ها به عنوان دستور پردازش می شوند، و مدل ها، از جمله یک مدل پایه، یک مدل finetune شده peft\_model1 و مدل دیگر peft\_model2، با استفاده از این اعلان ها

خلاصه‌هایی را تولید می‌کنند. خلاصه‌های به دست آمده استخراج شده و در لیست‌هایی برای هر مدل سازماندهی می‌شوند. سپس کد یک DataFrame ایجاد می‌کند، خلاصه‌های نوشته شده توسط انسان، خروجی‌های مدل zeroshot، خروجی‌های مدل oneshot، و خروجی‌های مدل finetune را در ستون‌ها گروه‌بندی می‌کند. این DataFrame به عنوان یک نمایش ساختار یافته از نتایج خلاصه‌سازی برای تجزیه و تحلیل و مقایسه عمل می‌کند. به طور کلی، کد قابلیت‌های خلاصه‌سازی مدل‌های مختلف را بر روی مجموعه‌ای از دیالوگ‌ها ارزیابی می‌کند و نتایج را در قالب جدولی برای ارزیابی بیشتر ارائه می‌کند. آرگومان max\_length تعداد حداکثر طول‌های تولید شده را مشخص می‌کند. آرگومان temperature میزان تصادفی بودن متن تولید شده را کنترل می‌کند.

مثالی از خروجی:

human_baseline_summaries	model_zero-shot	model_one-shot	peft_model1_summaries	peft_model2_summaries
\n\n Customer is complaining about parcel s...	\n\n The customer is upset that the deliver...	\n\n "Customer enquired about his Iphone an...	\n\n The user is unhappy with the parcel se...	\n\n User is complaining about the bad serv...
human_baseline_summaries	model_zero-shot	model_one-shot	peft_model1_summaries	peft_model2_summaries
Customer is complaining about parcel service of delivery drivers. Agent is asking to provide more details. Customer is frustrated that the delivery drivers are lazy and delivery is made on time. Agent updated that package will be deliver on time and asked customer to DM for further tracking.	The customer is upset that the delivery driver did not deliver the package to the customer. The customer is also upset that the delivery driver did not leave a note.	Customer enquired about his Iphone and Apple watch which is not showing his any steps/activity and health activities. Agent is asking to move to DM and look into it.	The user is unhappy with the parcel service of the company and is asking the company to not leave him feeling this way. He is asking the company to provide more details so that he can get his parcel delivered.	User is complaining about the bad service from the delivery driver. The system is providing the service details.

شکل ۶ - خروجی

برای مقایسه‌ی روش‌ها از معیار ROUGE استفاده می‌کنیم:

```
import evaluate
rouge = evaluate.load('rouge')
```

کد ارائه شده یک ماژول به نام evaluate را وارد می‌کند و به طور خاص مولفه rouge را از آن بارگیری می‌کند. مولفه ROUGE، معیاری که معمولاً در پردازش زبان طبیعی و ترجمه ماشینی برای ارزیابی کیفیت خلاصه‌های متن از طریق مقایسه آنها با خلاصه‌های مرجع مورد استفاده قرار می‌گیرد. به نظر می‌رسد ماژول "ارزیابی" یک ماژول سفارشی یا بخشی از یک چارچوب ارزیابی بزرگتر است که شامل

پیاده سازی هایی برای معیارهای مختلف است. با بارگذاری مولفه rouge از این مازول، کد قصد استفاده از متریک ROUGE را برای ارزیابی عملکرد مدل های خلاصه سازی متن پیشنهاد می کند. باشد.

Recall-Oriented Understudy for Gisting Evaluation مجموعه ای از معیارهای مورد استفاده برای ارزیابی خودکار متن تولید شده توسط ماشین، به ویژه در زمینه خلاصه سازی متن و ترجمه ماشینی است. کیفیت یک خلاصه تولید شده را با مقایسه آن با یک یا چند خلاصه مرجع ارزیابی می کند. معیارهای ROUGE معیار کمی از کیفیت متن تولید شده را ارائه می کند و به محققان و متخصصان کمک می کند تا عملکرد مدل های خود را به طور عینی ارزیابی کنند. این معیارها به طور گسترده در جامعه پردازش زبان طبیعی برای مقایسه سیستم های خلاصه سازی مختلف و ارزیابی اثربخشی آنها در گرفتن اطلاعات مهم از اسناد منبع استفاده می شود.

ROUGE شامل معیارهای مختلفی است:

#### ۱. ROUGE-N (N-gram Overlap)

همپوشانی n-گرم (دنباله های پیوسته از n کلمه) را بین خلاصه تولید شده و خلاصه مرجع اندازه گیری می کند. ROUGE-1، ROUGE-2 و ROUGE-3 به ترتیب به همپوشانی unigram، bigram و trigram اشاره دارند. در زمینه خلاصه سازی متن، ROUGE-2 همزمانی جفت های کلمه مجاور را در نظر می گیرد و میزان تشابه را در انتخاب و ترتیب عبارات دو کلمه ای بین خلاصه تولید شده و خلاصه مرجع به دست می آورد.

#### ۲. ROUGE-L (Longest Common Subsequence)

طولانی ترین دنباله مشترک کلمات را بین خلاصه های تولید شده و مرجع محاسبه می کند. طول دنباله متداول عادی شده با تعداد کل کلمات در خلاصه مرجع را در نظر می گیرد.

#### ۳. ROUGE-W (Weighted Overlap)

همپوشانی عبارات وزنی را در خلاصه ها در نظر می گیرد و به کلمات کلیدی اهمیت بیشتری می دهد.



نمرات ROUGE نشان دهنده این است که خلاصه های تولید شده تا چه حد با خلاصه های اصلی مطابقت دارند. نمرات بالاتر نشان دهنده مطابقت بهتر است. برای مثال اگر این معیار ۰,۴ باشد، به این معناست که خلاصه های تولید شده به طور متوسط ۴۰ درصد از کل کلمات خلاصه های اصلی را شامل می شوند.

خروجی نهایی این معیار به صورت زیر است:

جدول ۱ - مقایسه نتایج

prompt	Rouge1	Rouge2	RougeL
ZeroShot original model	۰,۲۱۱۸۸۱۰۱۶۷۸۷۵۷	۰,۰۷۵۳۸۹۷۵۱۳۱۲۰۳۷	۰,۱۵۱۵۸۰۰۶۹۲۰۳۲۴۲
OneShot original model	۰,۲۷۲۲۶۱۳۶۹۸۶۲۱۶۳	۰,۰۶۹۰۹۴۲۷۲۱۴۷۳۴۴	۰,۱۷۵۴۴۹۷۸۱۷۷۸۲۶۰
Finetune model_1	۰,۲۸۷۴۹۷۶۴۰۹۲۱۲۴۵	۰,۰۹۹۷۰۲۰۲۲۵۲۱۱۳۰	۰,۲۰۴۶۵۰۶۲۷۶۵۶۴۱۹
Finetune model_2	۰,۳۰۱۲۱۶۶۹۹۵۰۵۶۹۷	۰,۱۰۸۳۹۸۹۱۴۱۵۳۳۳۰	۰,۲۱۲۴۷۰۲۸۹۳۰۹۴۸۵

این نتایج نشان میدهد که آموزش مدل بر روی داده های Fine-Tuning کیفیت خلاصه های تولید شده را بهبود میبخشد. خلاصه ها مرتبط تر و تکرار کمتری دارند و طولانی ترین زیر دنباله مشترک بیشتری با خلاصه های اصلی دارند. همچنین استفاده از oneshot برای مدل زبانی prompt باعث بهبود بسیار زیاد عملکرد مدل می شود.

```
MODEL_zeroshot:
{'rouge1': 0.27536307411265154, 'rouge2': 0.07397794309342791, 'rougeL': 0.1745382837665075, 'rougeLsum': 0.199271106488691}
MODEL_oneshot:
{'rouge1': 0.27226136986216337, 'rouge2': 0.06909427214734443, 'rougeL': 0.17544978177826087, 'rougeLsum': 0.1971415170385126}
PEFT MODEL with r=16:
{'rouge1': 0.28749764092124547, 'rouge2': 0.09970202252113097, 'rougeL': 0.20465062765641967, 'rougeLsum': 0.180644881330203}
None
PEFT MODEL with r=32:
{'rouge1': 0.30121669950569796, 'rouge2': 0.10839891415333017, 'rougeL': 0.21247028930948508, 'rougeLsum': 0.1939035231697902}
```

شکل ۷ - معیار مقایسه مدل ها

#### - بخش ۱:

بخش ۱ مقاله بر مقیاس‌بندی مدل‌های زبانی instruction-finetuned تمرکز دارد. این اثر مقیاس‌بندی را از نظر اندازه مدل و تعداد وظایف finetuning بر عملکرد بررسی می‌کند. این مطالعه نشان می‌دهد که تنظیم دقیق دستورالعمل با اندازه مدل و task number به خوبی مقیاس می‌شود، که نشان می‌دهد تحقیقات آینده باید این جنبه‌ها را بیشتر مقیاس‌بندی کنند. علاوه بر این، تأثیر finetuning بر وظایف استدلال را بررسی می‌کند و نشان می‌دهد که ترکیب داده‌های زنجیره‌ای از فکر CoT در تنظیم دقیق، عملکرد را در ارزیابی‌های مختلف به طور قابل توجهی افزایش می‌دهد. این نشان دهنده ضرورت داده‌های CoT برای حفظ توانایی‌های استدلال در مدل‌های زبانی است. Flan-PaLM، یک مدل با پارامتر ۵۴۰B، معرفی شده است که شامل ۱,۸ هزار وظایف تنظیم دقیق و داده‌های CoT است که منجر به عملکرد برتر در معیارها و وظایف چند زبانه می‌شود. علاوه بر این، مدل‌های Flan-T5 توسعه یافته‌اند که پیشرفت‌های قابل توجهی را در ارزیابی‌های zero-shot, few-shot, and CoT نشان می‌دهند و از مدل‌های قبلی مانند T5 و حتی PalM در کارهای خاص بهتر عمل می‌کنند.

#### - بخش ۲:

بخش ۲ مقاله به بررسی اثرات مقیاس‌بندی مدل و تنوع وظایف در مدل‌های زبانی instruction-finetuned می‌پردازد. این نشان می‌دهد که افزایش اندازه مدل و تنوع instruction-finetuned به طور همزمان عملکرد مدل را افزایش می‌دهد. این بخش به‌ویژه تأثیر مثبت ترکیب داده‌های CoT را در طول finetuning نشان می‌دهد، که به طور قابل توجهی قابلیت‌های استدلال مدل را بهبود می‌بخشد. این بخش تأکید می‌کند که تنظیم دقیق دستورالعمل، زمانی که با داده‌های CoT همراه شود، منجر به پیشرفت‌های قابل توجهی در معیارها و ارزیابی‌های مختلف می‌شود که پتانسیل این روش‌ها را در پیشبرد مهارت مدل زبانی نشان می‌دهد.

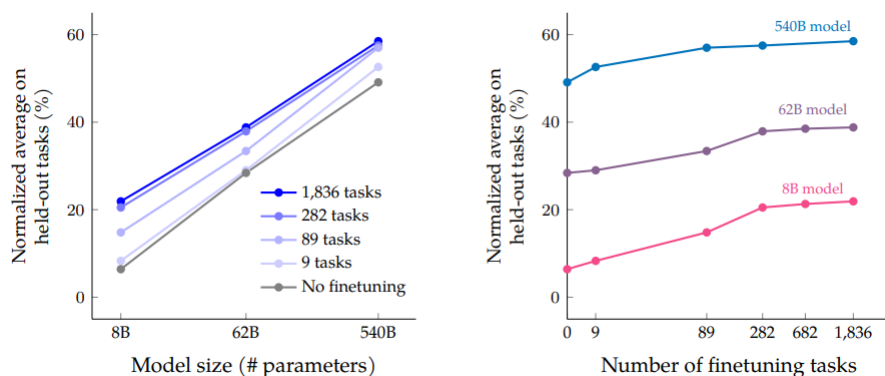
#### ۲,۲ Fine-tuning Procedure:

در بخش ۲,۲ مقاله، نویسندگان به روش finetuning مدل‌های زبان خود، به طور خاص بر جنبه‌های مقیاس‌بندی تمرکز می‌کنند. آنها آزمایشاتی را در سه اندازه مختلف مدل PALM انجام دادند: پارامترهای

۸B، ۶۲B و ۵۴۰B. این مقیاس برای درک تأثیر اندازه مدل بر عملکرد آن بود. در کنار اندازه مدل، آنها همچنین تعداد وظایف finetuning را متنوع کردند، از چند تا ۱۸۳۶ کار. آزمایش‌ها با هدف مشاهده تأثیر مقیاس‌گذاری مدل و تنوع وظایف بر توانایی مدل برای تعمیم و انجام وظایف جدید و نادیده‌شده انجام شد.

۲,۳ پروتکل ارزیابی:

بخش ۲,۳، با عنوان "پروتکل ارزیابی"، بینشی در مورد روش استفاده شده برای ارزیابی این مدل‌ها ارائه می‌دهد. نویسندگان از چندین مجموعه معیار برای ارزیابی استفاده کردند، از جمله MMLU، BBH، TYDIQA و MGSM، شامل انواع کارها و زبانها. این رویکرد جامع برای اطمینان از ارزیابی قوی از قابلیت‌های مدل‌ها بود. یکی از جنبه‌های کلیدی این بخش شکل ۱ است که رفتار مقیاس‌بندی finetuning دستورالعمل چند وظیفه‌ای را در رابطه با اندازه مدل و تعداد وظایف تنظیم دقیق نشان می‌دهد. مدل‌ها در طیفی از وظایف finetuning، از هیچ‌کدام تا ۱۸۳۶ مورد تجزیه و تحلیل قرار گرفتند که به درک دقیق این امکان را می‌دهد که چگونه افزایش تعداد وظایف بر عملکرد مدل تأثیر می‌گذارد. محورهای X در شکل از نظر لگاریتمی مقیاس‌بندی شده‌اند تا طیف گسترده‌ای از پارامترها و وظایف را در خود جای دهند. این دو بخش، تلاش‌های جامع انجام شده در finetuning و ارزیابی مدل‌های زبان بزرگ را برجسته می‌کنند، و بر اهمیت اندازه مدل و تنوع وظایف در افزایش عملکرد و تعمیم‌پذیری تأکید می‌کنند.



شکل ۸ - Scaling behavior of multi-task instruction finetuning with respect to model size and

number of finetuning tasks

نکات اصلی این بخش:

۱. اندازه‌های مدل مقیاس‌بندی: آزمایش‌ها بر روی سه اندازه مختلف از مدل Palm انجام شد: پارامترهای ۸B، ۶۲B و ۵۴۰B، نتایج نشان داد که تنظیم دقیق instruction-finetuned به طور قابل

توجهی عملکرد را در هر سه اندازه مدل بهبود می بخشد. افزایش عملکرد از ۹,۴٪ تا ۱۵,۵٪ در مقایسه با مدل های بدون finetuning متغیر بود.

۲. افزایش تعداد وظایف finetuning: مقاله گزارش می دهد که افزایش تعداد Fine-tuning Tasks منجر به بهبود عملکرد می شود. با این حال، اکثر این بهبود با حداکثر ۲۸۲ کار مشاهده می شود. فراتر از این، gain کمتر است. دو دلیل بالقوه پیشنهاد شده است:

- وظایف اضافی ممکن است به اندازه کافی متنوع نباشند تا دانش جدیدی به مدل ارائه دهند.

- مزایای multi-task instruction fine-tuning در درجه اول ممکن است ناشی از یادگیری مدل برای بیان بهتر دانشی باشد که از قبل از آموزش دارد. با توجه به اینکه داده های قبل از آموزش شامل ۷۸۰ میلیارد توکن است، در حالی که تنظیم دقیق دستورالعمل تنها از ۱,۴ میلیارد توکن (۰,۲٪ از توکن های قبل از آموزش) استفاده می کند، این توضیح قابل قبولی است.

۳. اثر افزایش مقیاس مدل: افزایش مقیاس مدل با یک مرتبه بزرگی (از ۸B به ۶۲B یا از ۶۲B به ۵۴۰B) به طور قابل ملاحظه ای عملکرد را برای مدل های finetuning و بدون finetuning بهبود می بخشد. با این حال، این مقاله به پیچیدگی در تعیین اینکه آیا تنظیم دقیق دستورالعمل برای مدل های کوچکتر یا بزرگتر سودمندتر است، اشاره می کند.

۴. این مقاله نشان می دهد که مقیاس بندی بیشتر اندازه مدل می تواند منجر به افزایش عملکرد قابل توجهی شود، اگرچه این ممکن است چالش برانگیز باشد. به طور مشابه، افزایش تعداد وظایف finetuning انتظار می رود عملکرد را بهبود بخشد، اما احتمالاً فقط به صورت تدریجی. نویسندگان نتیجه می گیرند که کار آینده باید بر روی instruction fine-tuning مقیاس بندی تمرکز کند تا عملکرد مدل را بهبود بخشد

### - بخش ۳:

در این بخش، نویسندگان به طور گسترده در مورد تأثیرات مقیاس بندی اندازه مدل و تعداد وظایف finetuning بر عملکرد مدل های زبان بحث می کنند. نکات کلیدی این بخش:

۱. مقیاس سازی اندازه مدل: این تحقیق شامل آزمایش هایی با سه اندازه مدل PaLM: ۸ میلیارد (8B)، ۶۲ میلیارد (۶۲B) و ۵۴۰ میلیارد (۵۴۰B) پارامتر بود. این مقیاس بندی با هدف درک چگونگی تأثیر افزایش اندازه مدل بر عملکرد آن انجام شد.

۲. مقیاس‌سازی شماره کار: تعداد وظایف finetuning نیز مقیاس‌بندی شد، از ترکیبی با وظایف کمتر تا یکی با وظایف بیشتر شروع شد. مخلوط‌های کاری خاص مورد استفاده CoT, Muffin, T0-SF و NIV2 بودند.

۳. تأثیر Multi-task Instruction Fine-tuning: برای هر سه اندازه مدل، Multi-task Instruction Fine-tuning به طور قابل توجهی عملکرد را در مقایسه با مدل‌های بدون finetuning بهبود داد. افزایش عملکرد از ۹,۴٪ تا ۱۵,۵٪ متغیر بود. این بهبود اثربخشی تنظیم دقیق در افزایش قابلیت‌های مدل را برجسته می‌کند.

۴. اثر افزایش وظایف finetuning: افزایش تعداد وظایف finetuning به طور کلی به عملکرد بهتر منجر شد، با بیشترین پیشرفت تا ۲۸۲ کار مشاهده شد. فراتر از این، دستاوردها کمتر بود. این می‌تواند به دلیل وظایف اضافی فاقد تنوع باشد یا به این دلیل که مدل‌ها در درجه اول از بیان بهتر دانش از قبل موجود به جای یادگیری اطلاعات جدید سود می‌برند. توضیح اخیر با توجه به اختلاف زیاد در میزان داده‌های مورد استفاده برای پیش‌آموزش (۷۸۰ میلیارد توکن) در مقابل تنظیم دقیق (۱,۴ میلیارد توکن) قابل قبول است.

۵. اهمیت افزایش مقیاس مدل: افزایش مقیاس مدل با یک مرتبه بزرگی (مانند از ۸B به ۶۲B یا از ۶۲B به ۵۴۰B) عملکرد قابل ملاحظه‌ای را برای مدل‌های finetuning و غیر finetuning بهبود بخشید. با این حال، این مقاله به پیچیدگی در تعیین اینکه آیا finetuning دستورالعمل برای مدل‌های کوچکتر یا بزرگتر در مقایسه با مدل‌های بدون finetuning مفیدتر است، اشاره می‌کند.

۶. این مقاله نشان می‌دهد که افزایش بیشتر اندازه مدل می‌تواند دستاوردهای عملکرد قابل توجهی داشته باشد، اگرچه ممکن است چالش برانگیز باشد. به طور مشابه، افزایش تعداد وظایف finetuning می‌تواند عملکرد را بهبود بخشد، اما احتمالاً فقط به صورت تدریجی. این تحقیق حاکی از آن است که کار آینده باید بر روی instruction fine-tuning مقیاس‌بندی تمرکز کند تا عملکرد مدل را بهبود بخشد.

#### - بخش ۴:

در بخش ۴ مقاله، نویسندگان بر تأثیر finetuning مدل‌های زبان با استفاده از حاشیه نویسی‌های زنجیره‌ای از فکر CoT تمرکز می‌کنند. این بخش تجزیه و تحلیل دقیقی از چگونگی تأثیر تنظیم دقیق CoT بر عملکرد مدل در وظایف مختلف و اندازه‌های مدل ارائه می‌دهد.

#### ۴,۱ Fine-tuning on Chain-of-Thought Improves Reasoning on Held-Out Tasks

این بخش نشان می‌دهد که چگونه Fine-tuning on Chain-of-Thought توانایی‌های استدلال را افزایش می‌دهد. گنجاندن تنها ۹ مجموعه داده با CoT در مخلوط finetuning منجر به بهبود عملکرد در معیارهای مختلف ارزیابی نگهداری شده شد. مدل Flan-PaLM که با CoT به خوبی تنظیم شده است، از مدل‌های قبلی در این معیارها بهتر عمل می‌کند و اثربخشی حاشیه‌نویسی CoT را در تقویت مهارت‌های استدلال در مدل‌های زبانی نشان می‌دهد.

#### ۴,۲ Some Chain-of-Thought Data is Needed to Maintain Reasoning Ability

نویسندگان همچنین در مورد لزوم داده‌های CoT در finetuning برای حفظ توانایی استدلال بحث می‌کنند. این مقاله نشان می‌دهد که finetuning دستورالعمل بدون CoT در واقع می‌تواند قابلیت‌های استدلال مدل را کاهش دهد. با این حال، شامل حتی تعداد کمی (نه) از مجموعه داده‌های CoT به طور قابل توجهی عملکرد را در تمام ارزیابی‌ها بهبود می‌بخشد. این یافته بر ارزش داده‌های CoT در حفظ و تقویت توانایی استدلال مدل‌های زبان تأکید می‌کند.

#### ۴,۳ Unlocking Zero-Shot Reasoning

در نهایت، این مقاله بررسی می‌کند که چگونه این روش استدلال zero-shot را ممکن می‌سازد. مدل Flan-PaLM، هنگامی که با "let's think step-by-step," خواسته شد، عملکرد بهتری را در وظایف چالش برانگیز BIG-Bench نشان داد. این نشان می‌دهد که تنظیم CoT نه تنها عملکرد در کارهای استاندارد را بهبود می‌بخشد، بلکه پتانسیل توانایی‌های استدلال zero-shot را در مدل‌های زبانی باز می‌کند.

نکات دیگر این بخش:

۱. متریک ارزیابی: معیار ارزیابی استفاده شده دقت few-shot است (تطابق دقیق)، که در آن یک میانگین وزنی برای همه کارها گرفته می‌شود. این مقاله میانگین نرمال شده چندین معیار، از جمله MMLU-direct، MMLU-CoT، BBH-direct، BBH-CoT، TyDiQA و MGSM را گزارش می‌کند. برای اطمینان از ارزیابی منصفانه از قابلیت‌های تعمیم مدل، این معیارها حذف شده‌اند، به این معنی که در داده‌های تنظیم دقیق گنجانده نشده‌اند.

۲. ترکیبات و وظایف finetuning: نویسندگان با مخلوط های finetuning مختلف (Muffin, CoT) NIV2, T0-SF و تعداد متفاوتی از کارها آزمایش کردند. مدل ها با این مخلوط ها تنظیم شدند و سپس برای بهبود عملکردشان ارزیابی شدند.

۳. مقایسه عملکرد در میان مدل های مختلف: این مقاله مقایسه ای از عملکرد مدل قبل و بعد از finetuning در سه اندازه مدل ارائه می دهد: ۸ B, ۶۲ B و ۵۴۰ B برای هر اندازه مدل، مقاله معیارهای عملکرد را هم با و هم بدون finetuning، و در تعداد مختلف وظایف تنظیم دقیق فهرست می کند.

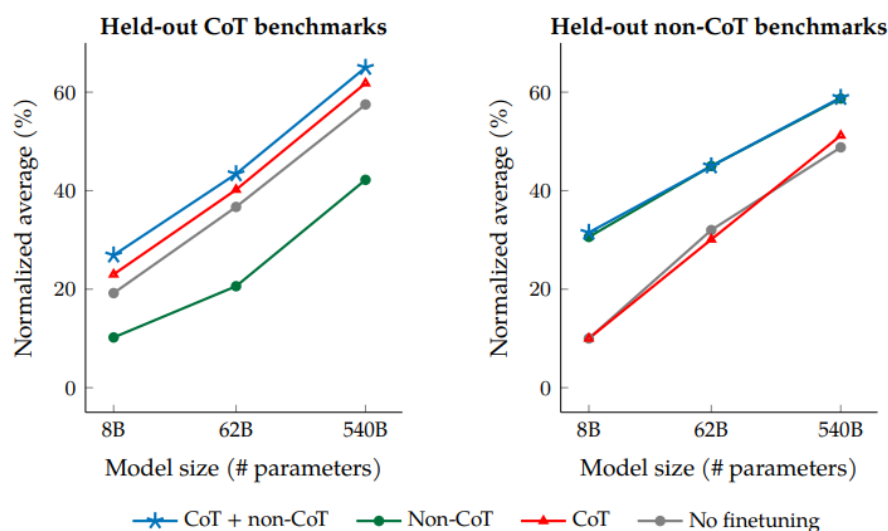
۴. تأثیر Chain-of-Thought Fine-tuning: نتایج نشان می دهد که Chain-of-Thought Fine-tuning منجر به بهبود عملکرد در همه اندازه های مدل می شود. به عنوان مثال، مدل ۸B با تنظیم دقیق با ۱۸۳۶ کار، از ۶,۴ درصد به ۲۱,۹ درصد در عملکرد متوسط نرمال شده بهبود یافته است. روندهای مشابهی در مدل های ۶۲B و ۵۴۰B مشاهده می شود، اگرچه اعداد دقیق متفاوت است.

۵. اهمیت تنوع وظایف: با افزایش تعداد وظایف finetuning، بهبود در عملکرد قابل توجه است، که نشان دهنده اهمیت تنوع کار در افزایش توانایی های حل مسئله مدل ها است. با این حال، بیشترین سود با تعداد کمتری از وظایف مشاهده می شود، که نشان دهنده کاهش بازده با تعداد بسیار زیاد وظایف است. این بخش شواهد قوی برای اثربخشی Chain-of-Thought Fine-tuning در مدل های زبان بزرگ ارائه می کند و پتانسیل آن را برای افزایش قابل توجه عملکرد مدل در طیف وسیعی از وظایف و اندازه های مدل نشان می دهد.

## - بخش ۷:

در این بخش، نویسندگان به جنبه های مختلف یافته های خود و پیامدهای آنها می پردازند. آنها تصدیق می کنند که در حالی که Flan-PaLM در همه معیارهای ارزیابی از مدل PaLM بهتر عمل می کند، در مقایسه با مدل های تخصصی خاص به نتایج پیشرفته SOTA دست نمی یابد. برای مثال، در کارهایی که نیاز به دستکاری نمادین دارند (BBH-algo)، Flan-PaLM حتی با Chain-of-Thought Fine-tuning و self-consistency (SC) از کد-davinci-002 بهتر عمل نمی کند. آنها همچنین خاطرنشان می کنند که علیرغم پیشرفت های قابل توجه در TyDiQA one-shot، Flan-PaLM به سطح عملکرد ByT5 تنظیم شده در مجموعه آموزشی TyDiQA نمی رسد.

این مقاله تاکید می کند که finetuning داده های CoT برای حفظ توانایی های استدلال بسیار مهم است. شکل ۲ در این مقاله نشان می دهد که finetuning در داده های CoT و غیر CoT عملکرد ارزیابی ها را در مقایسه با finetuning فقط در یک نوع بهبود می بخشد. finetuning برخی از نمونه های CoT بسیار مهم است، زیرا finetuning فقط در غیر CoT می تواند عملکرد وظایف CoT را به میزان قابل توجهی کاهش دهد. این یافته در تضاد با مطالعات قبلی است که نشان می دهند finetuning دستورالعمل عملکرد در کارهای نادیده را بهبود می بخشد. نویسندگان پیشنهاد می کنند که finetuning دستورالعمل، وظایف نادیده را هنگامی که در الگوی محرکی مشابه وظایف finetuning (غیر CoT یا CoT) قرار دارند، افزایش می دهد، که نشان دهنده نیاز به هر دو نوع داده در finetuning است.



شکل ۹ - finetuning on non-CoT and CoT data

علاوه بر این، این مقاله قابلیت finetuning دستورالعمل در داده های CoT را برای فعال کردن استدلال zero-shot مورد بحث قرار می دهد. این را عملکرد بهبود یافته Flan-PaLM در معیار BBH وظایف چالش برانگیز BIG-Bench هنگام استفاده از استدلال CoT که با عبارت "let's think step-by-step" فعال می شود، نشان می دهد. این در تضاد با PalM بدون finetuning است، که CoT موثر برای حل این مشکلات ایجاد نمی کند.

در پایان، بخش بحث پیچیدگی ها و تفاوت های ظریف عملکرد مدل را در رابطه با روش های finetuning، به ویژه نقش CoT در افزایش توانایی های استدلال و باز کردن قابلیت های جدید مانند استدلال zero-shot نشان می دهد. نویسندگان بر اهمیت داده های zero-shot متنوع برای حفظ و بهبود عملکرد مدل در طیف وسیعی از وظایف تأکید می کنند.



## ب)

ابتدا کتابخانه های مورد نیاز را دانلود می کنیم:

```
!pip install --q datasets
!pip install --q sentencepiece
```

و در ادامه دیتاست و مدل را نیز به شکل زیر لود می کنیم، همچنین از ما خواسته شده است که از API ایم مدل استفاده کنیم که آن را نیز در ادامه لود میکنیم: (در این سوال هم از خروجی API استفاده کردیم و هم از خروجی مدل(مدل را لود کرده و از آن خروجی گرفتیم.))

```
from datasets import load_dataset

# Load the dataset
dataset = load_dataset("lukaemon/bbh", "sports_understanding")
test_dataset = dataset['test']

# Load the dataset
dataset = load_dataset("tasksource/bigbench", "sports_understanding")
train_dataset = dataset['train']
```

هم دیتاست تست را لود کردیم و برای مثال های 3shot از دیتاست ترین استفاده کردیم.

لود مدل:

```
import pandas as pd
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer
from sklearn.metrics import accuracy_score
from transformers import T5Tokenizer, T5ForConditionalGeneration
from tqdm import tqdm

# Initialize the tokenizer and model
tokenizer = T5Tokenizer.from_pretrained("google/flan-t5-large")
model = T5ForConditionalGeneration.from_pretrained("google/flan-t5-large")
```

لود API مدل:

```
import requests
from getpass import getpass

token = getpass()
def make_query(data_payload):
    api_endpoint = "https://api-inference.huggingface.co/models/google/flan-t5-large"
    api_token = token
    headers = {"Authorization": f"Bearer {api_token}"}
```

```

json_data = {
    "inputs": data_payload,
    "parameters": {"max_length": 128},
    "wait_for_model": True
}

api_response = requests.post(api_endpoint, headers=headers, json=json_data)
return api_response.json()

```

این کد برای تعامل با Hugging Face Model Inference API استفاده می‌کند. هدف ایجاد یک پرس و جو برای مدل مشخص شده Hugging Face، در این مورد، google/flan-t5-large برای تولید متن است. تابع getpass برای وارد کردن ایمن یک توکن API استفاده می‌شود. تابع make\_query یک data\_payload را به عنوان ورودی می‌گیرد، که معمولاً متن یا داده‌ای را نشان می‌دهد که مدل باید خروجی برای آن تولید کند. این تابع یک بار JSON را با داده‌های ورودی می‌سازد، پارامترهایی مانند حداکثر طول خروجی تولید شده را روی ۱۲۸ تنظیم می‌کند و شامل گزینه‌ای برای صبر کردن برای پایان پردازش مدل است. سپس بار ساخته شده به عنوان یک درخواست POST به نقطه پایانی API مشخص شده با توکن مجوز ارائه شده ارسال می‌شود. در نهایت، تابع، پاسخ JSON را از API، حاوی خروجی تولید شده یا اطلاعات مرتبط از مدل، برمی‌گرداند.

- Answer only

در این حالت به شکل زیر prompt را به مدل و یا api می‌دهیم، در هر رو حالت به یک جواب میرسیم:

:Api

```

predictions = []

for question in tqdm(X_test):
    formatted_input = f"Given the statement: {question} Is the statement true or false? Please respond 'yes' for true and 'no' for false."
    api_response = make_query(formatted_input)

    # Check if the API response contains the expected output
    if "error" in api_response:
        print(f"Error for input {formatted_input}: {api_response['error']}")
        continue

    raw_prediction = api_response[0]['generated_text']
    prediction = "yes" if "yes" in raw_prediction.lower() else "no"
    predictions.append(prediction)

```

```
# Rest of your code for saving results and plotting remains the same
results_df = pd.DataFrame({'Input': X_test, 'Actual Target': y_test, 'Predicted':
predictions})
results_df.to_csv('model_answer_only_api.csv', index=False)

accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy}")

yes_count = predictions.count('yes')
no_count = predictions.count('no')

plt.figure(figsize=(6, 4))
plt.bar(['Yes', 'No'], [yes_count, no_count], color=['blue', 'red'])
plt.ylabel('Count')
plt.title('Count of Yes/No Predictions for answer only prompt')
plt.show()
```

این قطعه کد روی مجموعه‌ای از سؤالات `X_test` تکرار می‌شود و هر سؤال را به‌عنوان یک عبارت برای تأیید درستی یا نادرستی قالب‌بندی می‌کند. سپس این سؤالات فرمت‌شده را برای ایجاد پاسخ‌ها به یک API خارجی `make_query` ارسال می‌کند. پاسخ API برای استخراج متن تولید شده پردازش می‌شود و بر اساس وجود کلمه "بله" در پاسخ، پیش‌بینی مدل به دو دسته "بله" یا "خیر" طبقه‌بندی می‌شود. پیش‌بینی‌ها در یک لیست ذخیره می‌شوند و کد به ذخیره نتایج در یک فایل CSV و محاسبه دقت مدل با مقایسه پیش‌بینی‌های آن با اهداف واقعی `y_test` ادامه می‌دهد. در نهایت، دقت را چاپ می‌کند، پیش‌بینی‌های «بله» و «نه» را شمارش می‌کند و توزیع را با استفاده از نمودار میله‌ای به تصویر می‌کشد. این کد برای کار با یک API طراحی شده است که پاسخ‌هایی را برای عبارات ارائه شده به عنوان ورودی تولید می‌کند و پیش‌بینی‌هایی را بر اساس وجود کلمه "بله" در متن تولید شده استخراج می‌کند. در این حالت `answer only` است، در نتیجه فقط سوال‌ها را به عنوان `prompt` به مدل می‌دهیم.

کد مربوط به استفاده از مدل:

```
predictions = []
for question in tqdm(X_test):
    formatted_input = f"Given the Question: {question} Is the Question true or false? Please respond 'yes' for true and 'no' for false."
    inputs = tokenizer(formatted_input, return_tensors="pt", padding=True, truncation=True, max_length=512)
    outputs = model.generate(inputs['input_ids'], max_length=128)
    raw_prediction = tokenizer.decode(outputs[0], skip_special_tokens=True)
    # Post-process to get a yes/no answer
```

```

prediction = "yes" if "yes" in raw_prediction.lower() else "no"
predictions.append(prediction)

# Save predictions and actual targets to a CSV file
results_df = pd.DataFrame({'Input': X_test, 'Actual Target': y_test, 'Predicted':
predictions})
results_df.to_csv('model_answer_only.csv', index=False)

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy}")

yes_count = predictions.count('yes')
no_count = predictions.count('no')

# Create a bar plot for the counts of 'yes' and 'no' predictions
plt.figure(figsize=(6, 4))
plt.bar(['Yes', 'No'], [yes_count, no_count], color=['blue', 'red'])
plt.ylabel('Count')
plt.title('Count of Yes/No Predictions for answer only prompt')
plt.show()

```

خروجی این قسمت:

**Accuracy: 0.584**



شکل ۱۰ - barplot مربوط به جواب های قسمت answer only

برای این روش ابتدا باید ۳ تا مثال را از داده های train بصورت رندوم انتخاب کنیم و در ابتدای پرامپ به مدل بدیم، اینکار را به روش زیر انجام میدهیم:

```
# Select three random examples from the training set
example_indices = random.sample(range(len(dataset["train"])), 3)
examples = [dataset["train"][i] for i in example_indices]
```

مثال های زیر انتخاب شده اند:

```
{'inputs': 'Determine whether the following statement or statements are plausible or implausible:\nStatement: Daniele De Rossi got a base hit\nPlausible/implausible?',
'targets': ['implausible'],
'multiple_choice_targets': ['plausible', 'implausible'],
'multiple_choice_scores': [0, 1],
'idx': 192},
{'inputs': 'Determine whether the following statement or statements are plausible or implausible:\nStatement: Brent Burns drove into the paint\nPlausible/implausible?',
'targets': ['implausible'],
'multiple_choice_targets': ['plausible', 'implausible'],
'multiple_choice_scores': [0, 1],
'idx': 615},
{'inputs': 'Determine whether the following statement or statements are plausible or implausible:\nStatement: DeMar DeRozan was called for the goal tend\nPlausible/implausible?',
'targets': ['plausible'],
'multiple_choice_targets': ['plausible', 'implausible'],
'multiple_choice_scores': [1, 0],
'idx': 175}]
```

دقت این حالت وابسته است به مثال هایی که در ابتدای پرامپ آورده ایم، ممکن است با آوردن بعضی از مثال ها دقت این روش کمتر از حالت قبل شود.

```
model.to(device)
predictions = []

for i in tqdm(range(len(test_dataset))):
    prompt = ""
    for example in examples:
        statement = example['inputs'].replace('\n', ' ')
        answer = ["yes" if label == "plausible" else "no" for label
in example['targets']][0]
        prompt += f"Statement: {statement} Answer with 'yes' or 'no': {answer}\n"
```

```

    # Add the current test question to the prompt with instruction to answer with
    'yes' or 'no'
    question = test_dataset[i]['input']
    prompt += f"Given the statement: {question} Is the statement true or false?
    Please respond 'yes' for true and 'no' for false."

    api_response = make_query(prompt)

    # Check if the API response contains the expected output
    if "error" in api_response:
        print(f"Error for input {formatted_input}: {api_response['error']}")
        continue

    # Assuming the API response is correctly formatted
    raw_prediction = api_response[0]['generated_text']
    prediction = "yes" if "yes" in raw_prediction.lower() else "no"
    predictions.append(prediction)

target_labels = [test_dataset[i]['target'] for i in range(len(test_dataset))]

results_df = pd.DataFrame({'Input': [test_dataset[i]['input'] for i in
range(len(test_dataset))],
                           'Actual Target': target_labels,
                           'Predicted': predictions})

results_df.to_csv('3_shot_api.csv', index=False)

accuracy = accuracy_score(target_labels, predictions)
print(f"Accuracy: {accuracy}")

yes_count = predictions.count('yes')
no_count = predictions.count('no')

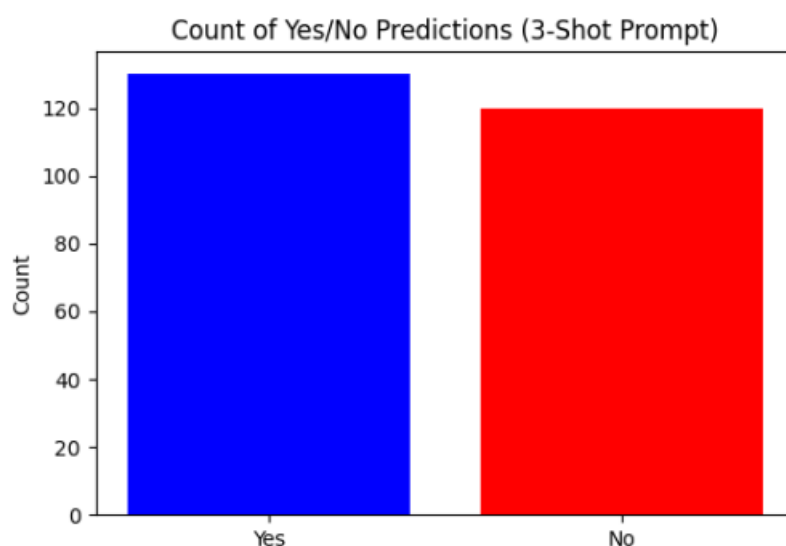
plt.figure(figsize=(6, 4))
plt.bar(['Yes', 'No'], [yes_count, no_count], color=['blue', 'red'])
plt.ylabel('Count')
plt.title('Count of Yes/No Predictions (3-Shot Prompt)')
plt.show()

```

این کد پیش‌بینی‌هایی را برای مجموعه‌ای از سؤالات test با استفاده از یک استراتژی 3-shot ایجاد می‌کند. بر روی مجموعه داده آزمایشی تکرار می‌شود، و با ترکیب سه عبارت اضافی با پاسخ‌های متناظر آن‌ها، یک اعلان برای هر سؤال ایجاد می‌کند و زمینه‌ای را برای مدل تشکیل می‌دهد. از مدل خواسته می‌شود که به سوال داده شده با "بله" یا "خیر" پاسخ دهد. اعلان‌ها، شامل متن و سوال تست، سپس به یک API

خارجی `make_query` ارسال می‌شوند و پاسخ‌های مدل برای استخراج پیش‌بینی‌ها پردازش می‌شوند. برچسب‌های هدف واقعی و پیش‌بینی‌های مدل در یک `DataFrame` ذخیره می‌شوند، که سپس به عنوان یک فایل `CSV` ذخیره می‌شود. دقت مدل با مقایسه پیش‌بینی‌های آن با برچسب‌های هدف واقعی محاسبه می‌شود و نمودار میله‌ای که تعداد پیش‌بینی‌های «بله» و «خیر» را تجسم می‌کند، تولید می‌شود. خروجی این قسمت:

**Accuracy: 0.54**



شکل ۱۱ – `barplot` مربوط به جواب‌های قسمت `3-shot`

مشاهده می‌کنیم که دقت کمتر از حالت قبل شده است.

COT -

در پرامپ این بخش Answer the following yes/no question by reasoning step-by-step را

اضافه میکنیم که مرحله به مرحله به جواب نهایی برسد.

```
import re
from tqdm import tqdm

def extract_yes_no(answer):
    # Define patterns that typically indicate a yes or no answer
    yes_pattern = re.compile(r"\b(yes|yeah|sure|correct|indeed|right|true)\b",
re.IGNORECASE)
    no_pattern = re.compile(r"\b(no|nah|nope|false|incorrect|wrong|not)\b",
re.IGNORECASE)
    # Check for 'yes' or 'no' in the answer
    if yes_pattern.search(answer):
        return "yes"
    elif no_pattern.search(answer):
        return "no"
    else:
        return "uncertain"

predictions = []
for question in tqdm(X_test):
    cot_input = f"Answer the following yes/no question by reasoning step-by-step:
{question}"
    api_response = make_query(cot_input)
    # Check if the API response contains the expected output
    if "error" in api_response:
        print(f"Error for input {formatted_input}: {api_response['error']}")
        continue
    raw_prediction = api_response[0]['generated_text']
    # Extract yes/no from the prediction
    yes_no_prediction = extract_yes_no(raw_prediction)
    predictions.append(yes_no_prediction)

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
# Save predictions and actual targets to a CSV file
results_df = pd.DataFrame({'Input': X_test, 'Actual Target': y_test, 'Predicted':
predictions})
results_df.to_csv('cot_api.csv', index=False)
# Count the occurrences of 'yes' and 'no'
yes_count = predictions.count('yes')
no_count = predictions.count('no')
# Create a bar plot for the counts of 'yes' and 'no' predictions
plt.figure(figsize=(6, 4))
```



```
plt.bar(['Yes', 'No'], [yes_count, no_count], color=['blue', 'red'])
plt.ylabel('Count')
plt.title('Count of Yes/No Predictions with COT Prompting')
plt.show()
print(f"Accuracy: {accuracy}")
```

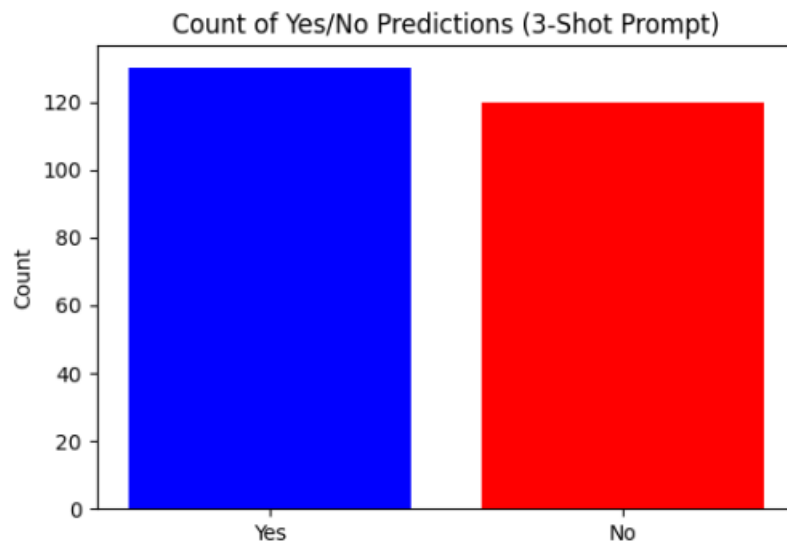
این کد از COT (chain of thought) برای به دست آوردن پیش بینی های بله/خیر برای مجموعه ای از سوالات آزمون استفاده می کند. از طریق سؤالات  $X_{test}$  تکرار می شود، برای هر سؤال یک اعلان ایجاد می کند که به مدل دستور می دهد یک پاسخ بله/خیر مستدل ارائه کند، و سپس یک API خارجی `make_query` را با درخواست تولید شده درخواست می کند. پاسخ های API پردازش می شوند و یک تابع `extract_yes_no` برای دسته بندی پیش بینی ها به عنوان «بله»، «نه» یا «نامشخص» بر اساس وجود الگوهای خاص در متن تولید شده استفاده می شود (نمونه خروجی در شکل ۵ آورده شده است و می بینیم که کلمه `yes/no` را از متن استخراج کرده است). کد، دقت مدل را با مقایسه پیش بینی های آن با برجسب های هدف واقعی  $y_{test}$  محاسبه می کند و نتایج، از جمله سوالات ورودی، اهداف واقعی و پیش بینی های مدل، در یک فایل CSV ذخیره می شوند. علاوه بر این، نمودار میله ای تعداد پیش بینی های «بله» و «خیر» را به تصویر می کشد.

Input	Actual Target	outout	Predicted
Is the following sentence plausible? "Delon Wright went for it on fourth down."	no	Delon Wright is a football player and not a football player. Therefore, the final answer is no.	no
Is the following sentence plausible? "Malcolm Brogdon banked the shot in."	yes	Malcolm Brogdon is a professional basketball player who has banked shots in the past. Therefore, the final answer is yes.	yes
Is the following sentence plausible? "Michael Conforto comitted a three second violation."	no	Michael Conforto is a football player who has never committed a violation. Therefore, the final answer is no.	no
Is the following sentence plausible? "Jakub Vrana hit a walkoff homer."	no	Jakub Vrana is a Czech professional baseball player. Therefore, the final answer is yes.	yes
Is the following sentence plausible? "Tom Wilson scored a penalty kick."	no	Tom Wilson is a football player and a penalty kick is a kick that is scored. Therefore, the final answer is yes.	yes
Is the following sentence plausible? "Aleksander Barkov passed the puck."	yes	Aleksander Barkov is a professional ice hockey player. Therefore, the final answer is yes.	yes
Is the following sentence plausible? "Malik Beasley comitted a three second violation."	yes	Malik Beasley is a professional basketball player who has never committed a violation. Therefore, the final answer is no.	no
Is the following sentence plausible? "Draymond Green threw a touchdown."	no	Draymond Green is a football player and a quarterback. Therefore, the final answer is yes.	yes
Is the following sentence plausible? "T.Y. Hilton threw a touchdown in the AFC divisional round."	yes	T.Y. Hilton was a quarterback in the NFL and threw a touchdown in the AFC divisional round. Therefore, the final answer is yes.	yes
Is the following sentence plausible? "Robert Woods converted the first down."	yes	Robert Woods is a football player and not a football player. Therefore, the final answer is no.	no

شکل ۱۲ - نمونه از خروجی این بخش

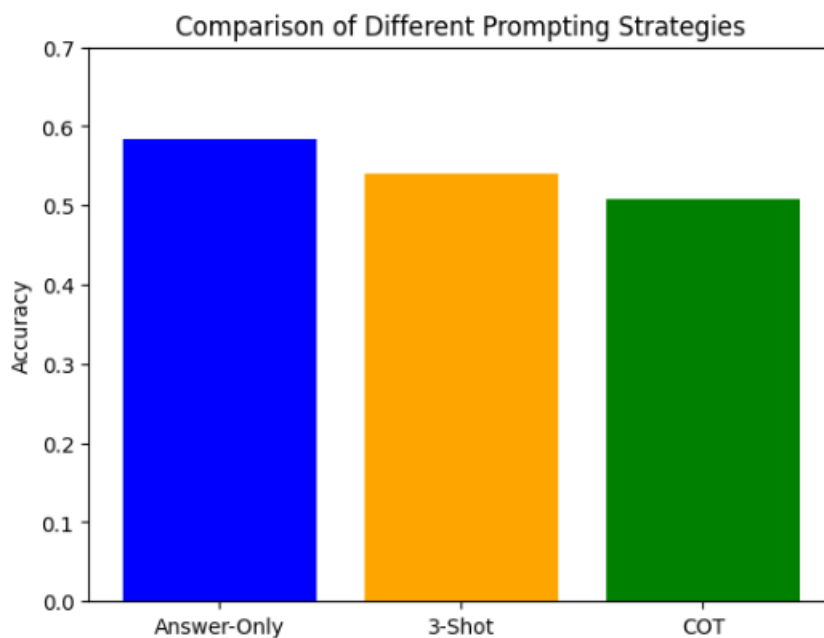
خروجی این قسمت:

**Accuracy: 0.508**



شکل ۱۳ - barplot مربوط به جواب های قسمت COT

مشاهده میکنیم که دقت کمتر از حالت قبل شده است.



شکل ۱۴ - مقایسه ۳ روش در این سوال

مشاهده میکنیم که در روش اول به دقت بالاتری رسیده ایم، علت اینکه در 3shot به دقت خوبی نرسیدیم به این دلیل باشد که مثال هایی که بصورت رندوم انتخاب شده اند، مناسب نبودند.

## ج)

این مقاله "Active-Prompt" را معرفی می‌کند، روشی که از معیارهای عدم قطعیت برای انتخاب سودمندترین سوالات برای annotation در یک مجموعه داده استفاده می‌کند، با هدف افزایش عملکرد مدل‌های زبان بزرگ (LLM) در وظایف استدلالی پیچیده. این شامل مراحل مانند تخمین عدم قطعیت، انتخاب سوال، annotation و استنتاج است. معیارهای مورد استفاده برای تخمین عدم قطعیت شامل disagreement، آنتروپی، واریانس و self-confidence است. این روش عملکرد برتر را در مقایسه با سایر مدل‌های پایه در مجموعه داده‌های متعدد و تکلیف استدلال نشان می‌دهد.

روش‌ها و معیارهای مورد استفاده در مقاله عبارتند از:

مواد و روش‌ها:

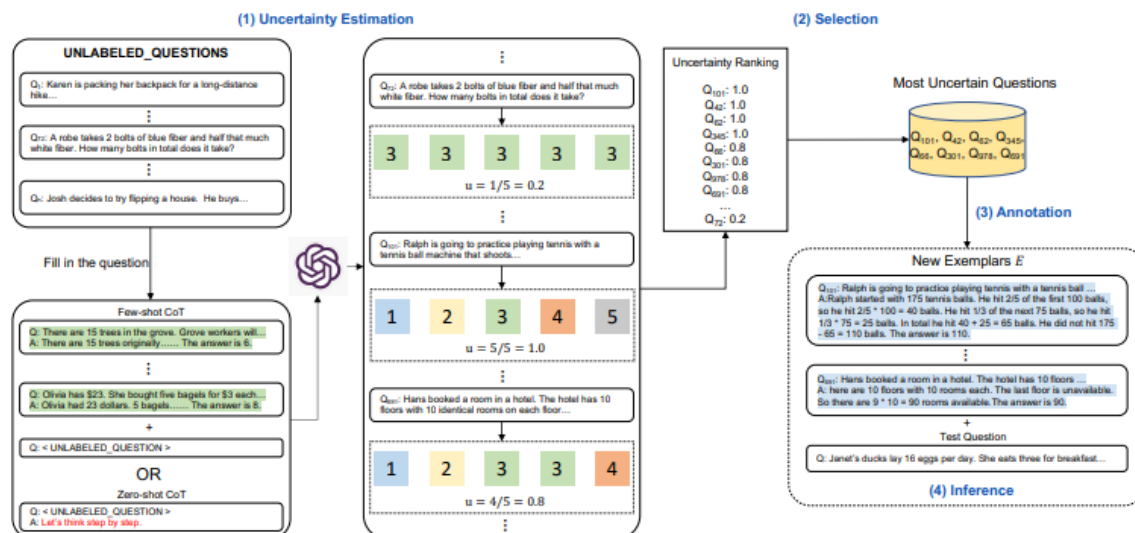
۱. Active-Prompt: تکنیکی برای انتخاب فعالانه سوالات مفید برای annotation به منظور بهبود عملکرد مدل‌های زبان بزرگ LLM در وظایف استدلالی پیچیده. معیارهای برآورد عدم قطعیت:

۱. اختلاف: تنوع در پاسخ‌های مدل‌های مختلف یا اجراهای مختلف یک مدل را اندازه‌گیری می‌کند.

۲. آنتروپی: عدم قطعیت در پیش‌بینی‌های مدل را با ارزیابی گسترش توزیع احتمال بر نتایج احتمالی محاسبه می‌کند.

۳. واریانس: تغییرپذیری در پیش‌بینی‌های انجام شده توسط مدل در اجراهای مختلف یا مدل‌های فرعی مختلف را ارزیابی می‌کند.

۴. self-confidence: اعتماد خود مدل به پیش‌بینی‌هایش را ارزیابی می‌کند، معمولاً بر اساس احتمالی که به پاسخ یا نتیجه انتخاب شده اختصاص داده شده است.



شکل ۱۵ - مدل سیستم

در تصور بالا تصاویری از رویکرد پیشنهادی در مقاله است. چهار مرحله وجود دارد. (۱) تخمین عدم قطعیت: با یا بدون چند زنجیره افکار cot نوشته شده توسط انسان، مدل زبان بزرگ  $k$  در این تصویر  $k=5$  را بارها جستجو می کند تا پاسخ های ممکن را با مراحل میانی برای مجموعه ای از سوالات آموزشی ایجاد کند. سپس عدم قطعیت  $u$  را بر اساس پاسخ های  $k$  از طریق یک متریک عدم قطعیت محاسبه می کند (در این تصویر از اختلاف نظر استفاده می کند). (۲) انتخاب: با توجه به عدم قطعیت، نامشخص ترین سوالات را برای حاشیه نویسی انتخاب می شود. (۳) حاشیه نویسی: انسان ها را برای حاشیه نویسی سوالات انتخاب شده درگیر می کنند. (۴) استنتاج: هر سؤال را با نمونه های مشروح جدید استنباط می شود.

رویکرد گفته شده در بالا بر اساس تعداد کمی نمونه گرفته شده few-shot از تنظیمات معمولی دستور دادن به مدل توسعه یافته است. هدف این رویکرد با داشتن یک داده آموزشی بدون برچسب Der و  $m$  داده تست، که هر کدام شامل یک سوال بدون پاسخ یا گام استدلال هستند، تنها  $n$  سوال از مجموعه داده آموزشی را به عنوان نمونه های few-shot انتخاب کند. این سوالات سپس به عنوان annotation با ساختن یک مجموعه نمونه جدید شامل گام های استدلال و پاسخ  $a$  مورد استفاده قرار می گیرند. سپس از مدل  $E$  برای دستور دادن به تمام داده های آزمایشی Dte استفاده می شود و پیش بینی ها را دریافت می کند.

در این بخش، توضیح داده می شود که چگونه  $n$  سوال بی اطمینان ترین از مجموعه داده آموزشی را انتخاب و annotation می کند. برای انتخاب چند سوال از یک مجموعه داده بزرگ، نیاز به یک روش بدون نظارت

وجود دارد. مطالعات نشان می‌دهند که کاهش عدم اطمینان مدل به بهبود عملکرد آن کمک می‌کند. در نتیجه، عدم اطمینان مدل زبان بزرگ را به عنوان یک معیار برای انتخاب داده‌ها معرفی می‌کنیم. در قالب مفهوم COT، ابتدا مدل زبان بزرگ را  $k$  بار جلو می‌فرستیم تا پاسخ برای هر سوال به دست آید. سپس عدم اطمینان یک سوال با استفاده از چندین روش مختلف اندازه‌گیری می‌شود. سپس با توجه به عدم اطمینان هر سوال یک رتبه بندی عدم اطمینان ایجاد میشود و سپس  $n$  سوال با بی اطمینان بالاتر برای annotation انتخاب میشوند اگر بیش از  $n$  سوال با بیشترین عدم اطمینان وجود داشته باشد به صورت تصادفی  $n$  سوال از آنها را انتخاب می‌شود این  $n$  سوال سپس با زنجیره های اندیشه و پاسخ ها annotation میشوند.

در این مقاله، نویسندگان بر ارزیابی عملکرد روش Active-Prompt با استفاده از مجموعه داده‌های مختلف که انواع مختلفی از وظایف استدلال را نشان می‌دهند، تمرکز می‌کنند. آنها همچنین از یک معیار ارزیابی خاص برای اندازه گیری عملکرد مدل استفاده می‌کنند. در اینجا جزئیات است:

مجموعه داده ها:

مجموعه داده‌های مورد استفاده در این مطالعه برای آزمایش قابلیت‌های استدلال مدل در حوزه‌های مختلف طراحی شده‌اند:

۱. GSM8K ریاضی مدرسه ابتدایی : مجموعه داده ای حاوی مسائل مربوط به کلمات ریاضی معمول در سطح مدرسه، با تاکید بر استدلال حسابی.

۲. ASDiv: مجموعه ای از مسائل حسابی متنوع: دارای مجموعه ای از مسائل حسابی است که هر کدام به شکل مجزایی از استدلال یا مفهوم ریاضی نیاز دارند.

۳. SVAMP : در هم ریختن انواع مسائل حسابی: بر مسائل حسابی تمرکز می‌کند، به طور خاص برای آزمایش استحکام در برابر تغییرات در ارائه یا ساختار مسئله طراحی شده است.

۴. CSQA CommonsenseQA : مجموعه داده ای با هدف ارزیابی استدلال عقل سلیم مدل، که در آن سوالات بر اساس دانش و سناریوهای روزمره است.

۵. StrategyQA: مجموعه داده ای که مدل را به چالش می‌کشد تا به سؤالاتی پاسخ دهد که به یک فرآیند استدلال چند هاپ نیاز دارند، که اغلب شامل درک استراتژیک از روابط یا مفاهیم ضمنی است.

۶. مجموعه داده استدلال نمادین: مجموعه داده ای طراحی شده برای ارزیابی قابلیت های مدل در رسیدگی به وظایفی که مستلزم استدلال نمادین است، مانند درک و دستکاری نمادها یا مفاهیم انتزاعی.

معیار ارزیابی:

۱. دقت تطابق دقیق exact match accuracy: این معیار اولیه است که برای ارزیابی عملکرد مدل استفاده می شود. نسبت پاسخ های مدل را که دقیقاً با پاسخ صحیح مطابقت دارد اندازه گیری می کند. این یک معیار دقیق است زیرا نیاز دارد که خروجی مدل دقیقاً مشابه پاسخ مرجع باشد، بدون هیچ انحرافی.

استفاده از این مجموعه داده های متنوع تضمین می کند که عملکرد مدل به طور دقیق در طیف وسیعی از انواع استدلال و پیچیدگی ها آزمایش می شود. متریک دقت تطابق دقیق، ارزیابی واضح و سرراستی از همسویی پاسخ های مدل با پاسخ های مورد انتظار را ارائه می دهد و سنجش اثربخشی روش Active-Prompt و قابلیت های استدلال مدل را آسان تر می کند

در این مقاله طبق آزمایشها بر روی سه نوع مجموعه داده انجام شدند:

۱ استدلال حسابی شامل مجموعه های داده: ( GSM8K -ASDiv- SingleEq - SVAMP -

۲. استدلال عقلانیت عمومی این شامل مجموعه های داده CSQA - StrategyQA

۳. استدلال نمادین: وظیفه آزمایش در قابلیت اتصال آخرین حروف کلمات در یک نام ارزیابی شد

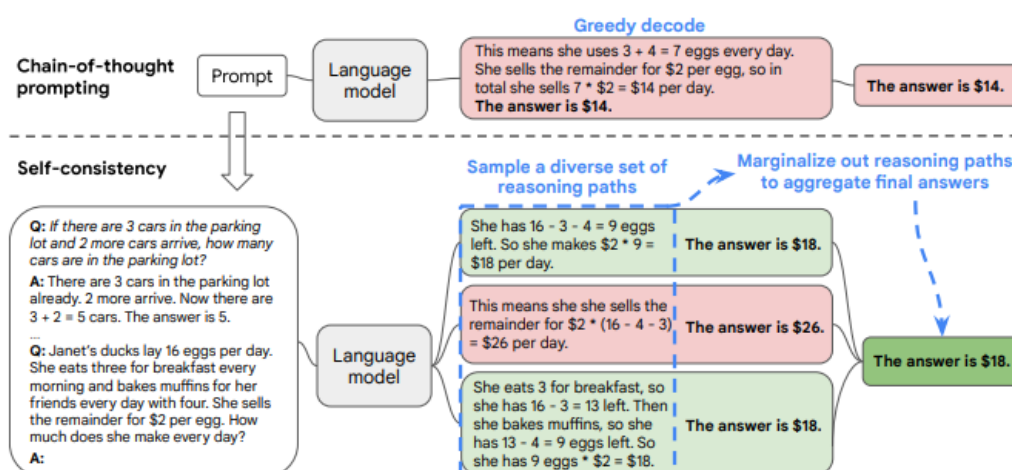
baseline مورد استفاده در مقاله عبارتند از:

۱. زنجیره فکر CoT: نمونه نوشته شده توسط انسان را با یک سری مراحل استدلال میانی ارائه می دهد.

۲. سازگاری با خود SC: نسخه بهبود یافته CoT، نمونه برداری از مسیرهای استدلال مختلف و انتخاب رایج ترین پاسخ.

۳. Auto-CoT: به طور خودکار نمونه ها را با خوشه بندی و تولید منطق با درخواست صفر شات می سازد.

۴. Random-CoT: شبیه به Active-Prompt است اما به جای استفاده از معیارهای عدم قطعیت پیشنهادی، به طور تصادفی سؤالات را برای حاشیه نویسی نمونه برداری می کند.



شکل ۱۶ - مدل مقاله

این مقاله «سازگاری خود» را به عنوان یک استراتژی رمزگشایی جدید برای بهبود استدلال زنجیره‌ای از فکر در مدل‌های زبانی پیشنهاد می‌کند. با نمونه‌برداری از مسیرهای استدلال متنوع و انتخاب سازگارترین پاسخ، عملکرد بهتری را از رمزگشایی حریصانه سنتی بهتر می‌کند. این روش سودمند است زیرا نیازی به آموزش یا حاشیه‌نویسی اضافی ندارد، با مدل‌های از پیش آموزش‌دیده کار می‌کند، و به طور قابل توجهی عملکرد را در معیارهای مختلف محاسباتی حسابی و عقلانی افزایش می‌دهد.

این مقاله روشی به نام **self-consistency** را برای تقویت استدلال زنجیره‌ای فکر **COT** در مدل‌های زبانی معرفی می‌کند. در اینجا نحوه بهبود **COT** و چرایی بهتر بودن آن نسبت به رویکردهای قبلی آورده شده است:

- چگونه **self-consistency** کار می‌کند:

۱. مسیرهای استدلال متنوع: مدل به جای تکیه بر یک مسیر استدلالی واحد، چندین مسیر استدلالی را برای یک مسئله معین ایجاد می‌کند.
۲. ارزیابی سازگاری: سپس این مسیرها را ارزیابی می‌کند (با استفاده از میانگین گیری) و منسجم‌ترین پاسخ را در میان آنها شناسایی می‌کند.

مزایا نسبت به روش‌های قبلی:

۱. استحکام در برابر خطاها: روش‌های سنتی معمولاً یک زنجیره فکری ایجاد می‌کنند و بر اساس آن پاسخ نهایی را انتخاب می‌کنند. اگر این زنجیره استدلال ناقص باشد، احتمالاً پاسخ نهایی نادرست است.

در مقابل، خود سازگاری زنجیره‌های استدلالی متعددی را ایجاد می‌کند و خطر تکیه بر یک زنجیره واحد و بالقوه اشتباه را کاهش می‌دهد.

۲. بدون نیاز به آموزش اضافی: رویکرد **self-consistency** را می‌توان برای مدل‌های از پیش آموزش دیده بدون نیاز به آموزش بیشتر یا تنظیم دقیق اعمال کرد. این آن را به یک راه حل همه کاره و مقرون به صرفه برای افزایش قابلیت‌های استدلال تبدیل می‌کند.

۳. عملکرد بهبود یافته در معیارها: این مقاله نشان می‌دهد که روش **self-consistency** به طور قابل توجهی بهتر از روش‌های سنتی در معیارهای مختلف حسابی و استدلال عقل سلیم عمل می‌کند. این نشان می‌دهد که نه تنها یک پیشرفت نظری را فراهم می‌کند، بلکه مزایای عملی را در کارهای دنیای واقعی نیز به همراه دارد.

۴. تطبیق پذیری: با مدل‌های از پیش آموزش دیده موجود سازگار است، و آن را به یک رویکرد انعطاف پذیر تبدیل می‌کند که می‌تواند به راحتی در تنظیمات مختلف بدون نیاز به تنظیمات خاص مدل ادغام شود.

با معرفی مسیرهای استدلال متعدد و یک بررسی سازگاری، خودسازگاری استحکام و قابلیت اطمینان استدلال **COT** را در مدل‌های زبانی افزایش می‌دهد، و پیشرفت قابل توجهی را نسبت به رویکردهای استدلال تک مسیری قبلی نشان می‌دهد.

#### • خود سازگاری:

خود سازگاری یک استراتژی رمزگشایی است که هدف آن افزایش قابلیت‌های استدلال مدل‌های زبانی است. این کار را با ایجاد چندین مسیر استدلال برای یک سوال مشخص و سپس انتخاب سازگارترین پاسخ در بین این مسیرها انجام می‌دهد. این رویکرد مبتنی بر این ایده است که در حالی که مسیرهای استدلال فردی ممکن است حاوی خطا باشند، پاسخ صحیح احتمالاً به طور مداوم در چندین تلاش استدلال مستقل ظاهر می‌شود.

#### • مراحل مربوط به خودسازگاری:

۱. تولید مسیرهای استدلال: برای یک سوال معین، مدل چندین مسیر استدلال ایجاد می‌کند. هر مسیر نشان دهنده دنباله‌ای از افکار یا مراحل است که به یک پاسخ منتهی می‌شود. برخلاف روش‌های سنتی که یک مسیر واحد را ایجاد می‌کنند، هدف خودسازگاری تنوع در مسیرهای استدلال برای کشف طیف وسیع‌تری از راه‌حل‌های ممکن است.



۲. ارزیابی سازگاری: هنگامی که مسیرهای استدلالی متعددی ایجاد شد، مدل سازگاری پاسخ های به دست آمده از این مسیرها را ارزیابی می کند. فرض اساسی این است که پاسخ صحیح بیشتر در مسیرهای استدلالی مختلف تکرار می شود، حتی اگر برخی از مسیرها منجر به پاسخ های نادرست شوند.

۳. انتخاب منسجم ترین پاسخ: مرحله نهایی شامل انتخاب پاسخی است که در مسیرهای استدلال تولید شده به طور مداوم ظاهر می شود. این پاسخ به عنوان پاسخ نهایی مدل به سوال در نظر گرفته می شود.

• چرا خودسازگاری موثر است:

۱. کاهش خطاهای مسیر فردی: این روش با تکیه بر مسیرهای استدلال متعدد، تأثیر خطاهایی را که ممکن است در یک زنجیره استدلال رخ دهد کاهش می دهد. اگر یک مسیر به دلیل سوء تفاهم یا اشتباه در استدلال به پاسخ نادرست منتهی شود، مسیرهای دیگر ممکن است همچنان به پاسخ صحیح منتهی شود و اطمینان حاصل شود که پاسخ صحیح می تواند از طریق سازگاری شناسایی شود.

۲. رفتار شبیه گروه مهار: این رویکرد شبیه روش های گروهی در یادگیری ماشین است، که در آن مدل های متعدد (یا در این مورد، مسیرهای استدلال متعدد) برای به دست آوردن یک نتیجه قابل اعتمادتر استفاده می شوند. تنوع در مسیرهای استدلال تضمین می کند که تصمیم نهایی مدل کاملاً آگاهانه و در برابر ناهنجاری های مسیر فردی استوار است.

۳. سازگاری با مدل های از پیش آموزش دیده: خودسازگاری نیازی به آموزش مجدد مدل یا داده های حاشیه نویسی اضافی ندارد. از قابلیت های موجود مدل های از پیش آموزش دیده استفاده می کند و یک استراتژی رمزگشایی جدید را برای بهبود عملکرد در وظایف استدلال اعمال می کند.

خودسازگاری با ایجاد مسیرهای استدلالی متعدد و انتخاب سازگارترین پاسخ از میان آنها، استدلال زنجیره فکر را بهبود می بخشد. این رویکرد نه تنها مدل را در برابر خطاهای فردی قوی تر می کند، بلکه عملکرد آن را در کارهای استدلالی پیچیده بدون نیاز به آموزش یا داده های اضافی افزایش می دهد.

