به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

مدل‌های مولد عمیق

تمرین شماره 3-part1

نام و نام خانوادگی	مهسا ندافی قهنویه
شماره دانشجویی	۸۱۰۱۰۰۴۹۰
تاریخ ارسال گزارش	۱۴۰۲/۱۱/۶

با ارائه گواهی پزشکی به جناب دکتر توسلی پور و آقای فرشاد سنگری مجاز به ارسال تمرین تا این تاریخ هستم.

فهرست گزارش سوالات

سوال ۱ – LLM

در ابتدا مدل و دیتاست خواسته شده را فراخوانی میکنیم:

```
tokenizer = AutoTokenizer.from_pretrained("stabilityai/stablelm-3b-4e1t",
trust_remote_code=True,torch_dtype="auto")
model = AutoModelForCausalLM.from_pretrained("stabilityai/stablelm-3b-
4e1t", trust_remote_code=True,torch_dtype="auto")
dataset = load_dataset("Salesforce/dialogstudio", "TweetSumm")
```

پیش پردازش دیتا شامل حذف علائم خاص می باشد که توسط تابع cleaning انجام شده است سپس با استفاده از conversation_extraction مکالمه بین agent , customer استخراج میشود در نهایت ستون های اضافی پاک شده و با چند بخش زیر جایگزین میشود:

```
      "conversation": conversation_text,
        "summary": summary,
        "text": text,
        "prompt": generate_prompt(conversation_text),
```

که text بصورت زیر تعریف شده است:

```
text = "### Instruction:\nBelow is a conversation between a human and an
agent. Write a summary of the conversation.\n\n### Input:\n" +
conversation_text +  "\n\n### Output:\n" + summary
```

از این بخش برای پارت دوم در finetuning مدل به وسیله Lora استفاده میشود.

بخش اول

۱. Zero-shot

در این بخش با استفاده از کد زیر پرامپت ورودی را تعریف کرده و خلاصه تولید شده توسط مدل را تولید میکنیم در نهایت معیار rough گفته شده را در تمام ۱۱۰ داده تست گزارش میکنیم:

```
results = []
for index in tqdm(range(110)):
    conversation = dataset['test'][index]['conversation']
    summary = dataset['test'][index]['summary']
    prompt = f"""Below is a conversation between a human and an AI agent.
Write a summary of the conversation.
{conversation}
Summary:
"""
    inputs = tokenizer(prompt, return_tensors='pt').to("cuda")
    token = model.generate(
```

```
        **inputs,
        max_new_tokens=47,
        temperature=0.9
        )
    completion_tokens = token[0][inputs['input_ids'].size(1):]
    completion = tokenizer.decode(completion_tokens,
skip_special_tokens=True)
    results.append(rouge.compute(predictions=[completion],
references=[summary]))
average_score = {metric: sum([result[metric].mid.fmeasure for result in
results]) / len(results)
                 for metric in results[0]}
print(average_score)
```

یک نمونه خلاصi تولید شده توسط مدل :

```
Example  109
----------------------------------------------------------------------------------------------------
prompt for zero-shot:
----------------------------------------------------------------------------------------------------
Below is a conversation between a human and an AI agent. Write a summary of the conversation.
Customer: if you have a commercial that says you have 100mbs for 44.99 honor it in all your markets you
provide 100mbs at.
Agent: Good morning. In some areas of our markets our network is not yet able to provide those speeds. We do
not offer a...
Customer: My area offered me 100mb for more money so now that this is the standard and they do offer 100mb
shouldn’t I receive it right away
Agent: I would be happy to take a look. If you upgraded to the 100 plan and are not receiving this we can
investigate. H...
Customer: I have the 60mb plan but I watch many commercials saying they offer 100mb for the same price. My
area has 100mb so shouldn’t I be getting it
Agent: I am sorry, the advertisements are package/product prices for new customers. If you would like to
upgrade that sh...
Customer: So you leave existing customers below your new standards is what your telling me, right?
Agent: Hi William, In order to discuss this further we ask that you please contact our billing specialist
at: 800-892-4357. Thank you.

Summary:

----------------------------------------------------------------------------------------------------
human summary:
Customer having an issue with data speed in his area. Agent updated the customer in some area of their
market network is not yet provided those speed and also informed the customer to contact billing specialist
for further assist.
----------------------------------------------------------------------------------------------------
model output for zero-shot:

The customer is upset that he is not receiving the advertised speeds. The agent explains that the customer
is not receiving the advertised speeds because the customer is not a new customer. The agent then asks the
customer to contact the billing department.

----------------------------------------------------------------------------------------------------
{'rouge1': 0.2682926829268293, 'rouge2': 0.125, 'rougeL': 0.2682926829268293, 'rougeLsum':
0.2682926829268293}
```

شکل ۱ : خلاصه تولید شده توسط **zero-shot**

نتیجه نهایی برای ۱۱۰ داده تست:

100%|██████████| 110/110 [05:53<00:00, 3.21s/it]
{'rouge1': 0.28147042730716154, 'rouge2': 0.08427375021738195,
'rougeL': 0.23239024097682343, 'rougeLsum': 0.22743986580540443}

۱.one-shot

در این بخش یک مثال بصورت ورودی به عنوان پرامپت به مدل داده میشود:

```
results = []
for index in tqdm(range(110)):

    conversation = dataset['test'][index]['conversation']
    summary = dataset['test'][index]['summary']
    if 109 > index:
      prompt = make_prompt([index+1], index)
    else:
      prompt = make_prompt([30], index)
    inputs = tokenizer(prompt, return_tensors='pt').to("cuda")
    token = model.generate(
        **inputs,
        max_new_tokens=50,
        temperature=0.9
        )
    completion_tokens = token[0][inputs['input_ids'].size(1):]
    completion = tokenizer.decode(completion_tokens,
skip_special_tokens=True)
    results.append(rouge.compute(predictions=[completion],
references=[summary]))
average_score = {metric: sum([result[metric].mid.fmeasure for result in
results]) / len(results)
                 for metric in results[0]}
print(average_score)
```

تابع تولید پرامپت:

```
def make_prompt(examples, sample):
    prompt = ''
    for index in examples:
        conversation = dataset['train'][index]['conversation']
        summary = dataset['train'][index]['summary']
        prompt += f"""Below is a conversation between a human and an AI
agent. With a summary of the conversation.
{conversation}
Summary:
```

```
{summary}

"""

    conversation = dataset['test'][sample]['conversation']

    prompt += f"""Below is a conversation between a human and an AI agent.
Write a summary of the conversation.
{conversation}
Summary:
"""
    return prompt
prompt = print(make_prompt([30], 109))
```

نتایج:

```
100%|██████████| 110/110 [06:29<00:00, 3.54s/it]
{'rouge1': 0.3201767403191365, 'rouge2': 0.10116457342954364,
'rougeL': 0.25506544345333143, 'rougeLsum': 0.26571756897339766}
```

همانطور که مشاهده میشود در روش دوم مقادیر معیار ها رشد خوبی داشته است که نشان دهنده آن است که مدل به درستی کار میکند. اما همچنان نتایج قابل استناد نیست چرا که مدل نیاز به instruction finetuning دارد پس به سراغ پارت دوم سوال میرویم.

معیارهای ROUGE یک معیار کمی از کیفیت خلاصه تولید شده را با ارزیابی اینکه چقدر محتوا و ساختار خلاصه مرجع را به تصویر می‌کشد، ارائه می‌کند. نمرات ROUGE بالاتر نشان دهنده تطابق بهتر بین خلاصه های تولید شده و مرجع است که منعکس کننده فرآیند تولید خلاصه با کیفیت بالاتر است. این معیارها به طور گسترده در تحقیق و ارزیابی سیستم های خلاصه سازی متن، ترجمه ماشینی و سایر کارهای تولید زبان طبیعی استفاده می شود.

- ROUGE-1: ROUGE-1 همپوشانی unigram (تک کلمه) را بین خلاصه تولید شده و خلاصه مرجع اندازه گیری می کند
- ROUGE-2: ROUGE-2 مشابه ROUGE-1 است، اما همپوشانی بیگرام ها (جفت کلمات متوالی) بین خلاصه تولید شده و خلاصه مرجع را اندازه گیری می کند.
- ROUGE-L: ROUGE-L بر اساس طولانی ترین دنباله متداول (LCS) کلمات بین خلاصه تولید شده و خلاصه مرجع است. که نشان دهنده طولانی ترین دنباله کلماتی است که در هر دو خلاصه به یک ترتیب ظاهر می شوند.

بخش دوم

LoRA :

تطبیق مدل های زبان بزرگ برای وظایف یا حوزه های خاص به هنگام finetuning اغلب شامل منابع محاسباتی و نیازهای حافظه قابل توجهی است. روش‌های انطباق با رتبه پایین می‌توانند با تقریب پارامترهای مدل با ماتریس‌ها یا تنسور های رتبه پایین‌تر، به رفع این چالش‌ها کمک کنند، در نتیجه بار محاسباتی و حافظه کلی را کاهش می‌دهند.LoRA روشی با فریز کردن وزن‌های از پیش آموزش‌دیده و افزودن ماتریس‌های رتبه پایین به معماری Transformer است. LoRA با تجزیه ماتریس های وزن هر لایه به حاصل ضرب دو ماتریس کوچکتر کار می کند که یکی از آنها ثابت و دیگری قابل آموزش است. این باعث کاهش تعداد پارامترهایی می شود که باید در هنگام تنظیم دقیق به روز شوند و در عین حال قدرت بیان مدل اصلی حفظ می شود.

برخی از مزایای LoRA نسبت به سایر روش های سازگاری عبارتند از:

- کارایی: LoRA بسیار سریع‌تر است و به حافظه کمتری نسبت به روش‌های تنظیم دقیق سنتی نیاز دارد، زیرا تنها بخش کوچکی از پارامترهای اصلی را آموزش می‌دهد. LoRA همچنین از تأخیر استنتاج ناشی از افزودن لایه‌های آداپتور اضافی جلوگیری می‌کند.

- انعطاف پذیری: LoRA امکان انطباق انتخابی بخش های مختلف مدل را با استفاده از ماتریس های کم رتبه با اندازه ها و اشکال مختلف فراهم می کند. LoRA همچنین می تواند برای هر مدل زبانی از پیش آموزش دیده مانند GPT-3، BERT یا DeBERTa اعمال شود.

- عملکرد: LoRA به نتایج قابل مقایسه یا بهتری نسبت به تنظیم دقیق یا تنظیم آداپتور در کارهای مختلف NLP، مانند درک زبان طبیعی، تولید زبان طبیعی و پاسخگویی به سؤالات می‌رسد. LoRA همچنین توانایی تعمیم مدل از پیش آموزش دیده را حفظ می کند و از فراموشی بیش از حد جلوگیری می کند.

پیاده سازی:

معماری مدل اصلی:

```
StableLMEpochForCausalLM(
  (model): StableLMEpochModel(
    (embed_tokens): Embedding(50304, 2560)
    (layers): ModuleList(
      (0-31): 32 x DecoderLayer(
        (self_attn): Attention(
          (q_proj): Linear(in_features=2560, out_features=2560, bias=False)
          (k_proj): Linear(in_features=2560, out_features=2560, bias=False)
          (v_proj): Linear(in_features=2560, out_features=2560, bias=False)
          (o_proj): Linear(in_features=2560, out_features=2560, bias=False)
          (rotary_emb): RotaryEmbedding()
        )
        (mlp): MLP(
          (gate_proj): Linear(in_features=2560, out_features=6912, bias=False)
          (up_proj): Linear(in_features=2560, out_features=6912, bias=False)
          (down_proj): Linear(in_features=6912, out_features=2560, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): LayerNorm((2560,), eps=1e-05, elementwise_affine=True)
        (post_attention_layernorm): LayerNorm((2560,), eps=1e-05, elementwise_affine=True)
      )
    )
    (norm): LayerNorm((2560,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=2560, out_features=50304, bias=False)
)
```

شکل ۲: معماری مدل

در ابتدا مدل quantize شده است:

```python
tokenizer = AutoTokenizer.from_pretrained("stabilityai/stablelm-3b-4e1t")
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
)
model = AutoModelForCausalLM.from_pretrained(
    "stabilityai/stablelm-3b-4e1t",
    use_safetensors=True,
    quantization_config=bnb_config,
    trust_remote_code=True,
    device_map="auto",
)
```

کوانتیزاسیون تکنیکی است که اندازه و نیاز به حافظه شبکه های عصبی را با تبدیل وزن ها از انواع داده های با دقت بالاتر (مانند float32) به انواع با دقت کمتر (مانند int8 یا int4) کاهش می دهد.

کلاس BitsAndBytesConfig به شما اجازه می دهد تا پارامترهای مختلفی را برای کوانتیزه کردن مشخص کنید، مانند:

- load_in_4bit: وزن‌های مدل را با دقت ۴ بیت بارگیری کنیم. این باعث کاهش ۸ برابری سایز مدل نسبت به float32 می شود.

- bnb_4bit_quant_type: نوع کوانتیزاسیون مورد استفاده. پیش فرض "fp4" است که مخفف ۴ بیتی نقطه ثابت است. گزینه دیگر "nf4" است که مخفف ۴ بیتی معمولی شناور است، یک نوع داده جدید که برای وزن های توزیع شده معمولی بهینه است.

- bnb_4bit_compute_dtype: نوع داده ای که برای محاسبه استفاده می شود. پیش فرض None است که به معنای همان نوع کوانتیزه کردن است. گزینه دیگر torch.bfloat16 است که یک فرمت ممیز شناور ۱۶ بیتی است که دقت بیشتری نسبت به fp16 حفظ می کند.

تأثیر کوانتیزاسیون بر روی مدل این است که مصرف حافظه و زمان استنتاج را کاهش می‌دهد و در عین حال بیشترین دقت و عملکرد مدل اصلی را حفظ می‌کند. با این حال، کوانتیزه کردن ممکن است برخی از خطاها یا نویز را به دلیل از دست دادن اطلاعات در طول تبدیل ایجاد کند. بنابراین، ارزیابی مدل کوانتیزه شده بر روی وظیفه یا دامنه هدف و مقایسه آن با مدل با دقت کامل بسیار مهم است.

با استفاده از کد زیر میزان حافظه اشغال شده بعد از کوانتیزه کردن مدل را استخراج میکنیم:

```
def calculate_model_size(model):
    model_parameters = filter(lambda p: p.requires_grad,
model.parameters())
    params = sum([np.prod(p.size()) for p in model_parameters])
    return params / (1024 ** 2)  # Convert to megabytes
```

```
Model size before quantization: 2665.9423828125 MB
Model size after quantization: 245.9423828125
MB Model size reduction after quantization: 2420.0 MB
```

اندازه مدل تقریبا ده برابر کاهش داشته است که باعث میشود زمان آموزش مدل به یک سوم کاهش یابد.

در نهایت lora را بر مدل اعمال میکنیم که بر روی لایه Attention اعمال میشود زیرا آنها از نظر محاسباتی گران ترین و گویاترین بخش های مدل هستند:

```
peft_config = LoraConfig(
    task_type=TaskType.CAUSAL_LM,
    r=32,
    lora_alpha=8,
    target_modules=[
        "q_proj",
        "k_proj",
        "v_proj",
        "o_proj"
    ],
    bias="none"
)

model.config.use_cache = False
model = get_peft_model(model, peft_config)
model.print_trainable_parameters()
```

مقدار پارامتر های قابل آموزش:

Rank 16: `trainable params: 10,485,760 || all params: 2,805,928,960 || trainable%: 0.3737001238976485`

Rank 32: `trainable params: 20,971,520 || all params: 2,816,414,720 || trainable%: 0.7446176108609459`

پارامترها بصورت زیر برای هر دو مدل انتخاب میشود:

```
training_arguments = TrainingArguments(
    output_dir="lora-r32-finetuned",
    per_device_train_batch_size=1,
    gradient_accumulation_steps=1,
    per_device_eval_batch_size=1,
    learning_rate=1e-4,
    lr_scheduler_type="cosine",
    save_strategy="epoch",
    evaluation_strategy="epoch",
    logging_steps=1,
    num_train_epochs=5,
    push_to_hub=False
)
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset["train"],
    eval_dataset=dataset["validation"],
```

```
    peft_config=peft_config,
    dataset_text_field="text",
    max_seq_length= 4096,
    tokenizer=tokenizer,
    args=training_arguments,
)
```

در شکل زیر مدل را پس از افزودن لورا مشاهده میکنیم:

```
PeftModelForCausalLM(
  (base_model): LoraModel(
    (model): StableLMEpochForCausalLM(
      (model): StableLMEpochModel(
        (embed_tokens): Embedding(50304, 2560)
        (layers): ModuleList(
          (0-31): 32 x DecoderLayer(
            (self_attn): Attention(
              (q_proj): lora.Linear4bit(
                (base_layer): Linear4bit(in_features=2560, out_features=2560, bias=False)
                (lora_dropout): ModuleDict(
                  (default): Identity()
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=2560, out_features=32, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=32, out_features=2560, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
              (k_proj): lora.Linear4bit(
                (base_layer): Linear4bit(in_features=2560, out_features=2560, bias=False)
                (lora_dropout): ModuleDict(
...
      )
      (lm_head): Linear(in_features=2560, out_features=50304, bias=False)
    )
  )
)
```

شکل ۳:معماری مدل بعد از افزودن **LoRA**

نتایج آموزش:

Rank 16:

```
TrainOutput(global_step=4395,
training_loss=1.929868298626597,
```

```
metrics={'train_runtime': 2024.224,
'train_samples_per_second': 2.171,
'train_steps_per_second': 2.171, 'total_flos':
2.27875588990464e+16, 'train_loss': 1.929868298626597,
'epoch': 5.0})
```

Rank 32:

```
TrainOutput(global_step=4395,
training_loss=1.9214049900486743,
metrics={'train_runtime': 2150.5955,
'train_samples_per_second': 2.044,
'train_steps_per_second': 2.044, 'total_flos':
2.28768123250176e+16, 'train_loss': 1.9214049900486743,
'epoch': 5.0})
```

در ابتدا پارامتر های Finetune شده توسط lora را در مدل آپدیت میکنیم:

```
model = AutoModelForCausalLM.from_pretrained(
  "stabilityai/stablelm-3b-4e1t",
  trust_remote_code=True,
  torch_dtype="auto",
  use_safetensors=True,
).to("cuda")
peft_model = PeftModel.from_pretrained(model, "lora-r32-
finetuned/checkpoint-4395", from_transformers=True)
model = peft_model.merge_and_unload()
```

در نهایت از کد زیر برای مدل finetune شده برای تست ۲ روش zero-shot, one-shot استفاده میکنیم:

```
results_zero_shot = []
results_one_shot = []
for index in tqdm(range(110)):
    model.config.pad_token_id = tokenizer.eos_token_id
    summary = dataset['test'][index]['summary']
    if 109 > index:
      prompt = make_prompt([index+1], index)
    else:
      prompt = make_prompt([30], index)
    inputs = tokenizer(prompt, return_tensors='pt').to("cuda")
    token = model.generate(
        **inputs,
        max_new_tokens=64,
        temperature=0.75,
        pad_token_id=tokenizer.eos_token_id,
        top_p=0.95,
```

```
        do_sample=True,
        )
    completion_tokens = token[0][inputs['input_ids'].size(1):]
    completion = tokenizer.decode(completion_tokens,
skip_special_tokens=True)
    results_one_shot.append(rouge.compute(predictions=[completion],
references=[summary]))
    prompt = dataset['test'][index]['text']
    inputs = tokenizer(prompt, return_tensors='pt').to("cuda")
    token = model.generate(
        **inputs,
        max_new_tokens=64,
        temperature=0.75,
        pad_token_id=tokenizer.eos_token_id,
        top_p=0.95,
        do_sample=True,
        )
    completion_tokens = token[0][inputs['input_ids'].size(1):]
    completion = tokenizer.decode(completion_tokens,
skip_special_tokens=True)
    results_zero_shot.append(rouge.compute(predictions=[completion],
references=[summary]))
average_score_zero_shot = {metric: sum([result[metric].mid.fmeasure for
result in results_zero_shot]) / len(results_zero_shot)
                  for metric in results_zero_shot[0]}
average_score_one_shot = {metric: sum([result[metric].mid.fmeasure for
result in results_one_shot]) / len(results_one_shot)
                  for metric in results_one_shot[0]}
print(f"average score one shot:{average_score_one_shot}\naverage score zero
shot:{average_score_zero_shot}")
```

نتایج:

Rank 16:

100%|[redacted]| 110/110 [13:29<00:00, 7.36s/it]
average score one shot
:{'rouge1': 0.34503884108694416, 'rouge2': 0.12154916425886186,
'rougeL': 0.26681057051662654, 'rougeLsum': 0.2816826578251633}

average score zero shot
:{'rouge1': 0.4091589271547215, 'rouge2': 0.23245091326230136,
'rougeL': 0.3212717036658451, 'rougeLsum': 0.34940292196154804}

Rank 32:

100%|[redacted]| 110/110 [11:19<00:00, 6.18s/it]
average score one shot:
{'rouge1': 0.3681059483218816, 'rouge2': 0.13981877957788952,
'rougeL': 0.2953305409730834, 'rougeLsum': 0.30379438020703853}

average score zero shot:
{'rouge1': 0.4332597143242216, 'rouge2': 0.25816331149057714,
'rougeL': 0.3617806393058581, 'rougeLsum': 0.3747735275398468}

همانطور که مشاهده میشود مقادیر معیار های خواسته شده نسبت به بخش قبل سوال بعد از finetuning پیشرفت چشمگیری داشته است این پیشرفت در Zero-shot بیشتر به چشم میخورد چرا که پرامت ورودی در قسمت آموزش مدل پرامتی بوده است که در zero-shot استفاده میشود همچنین آورد یک مثال میتواند باعث استخراج اطلاعات نادرست توسط مدل شود.همچنین با توجه به اینکه در rank32 پارامتر های بیشتری آموزش دیده است نتایج بهتری به همراه داشته است.

سوال ۲ – Prompt engineering

سوال ۱.

قسمت اول : Introduction

در این قسمت در مورد fine-tune کردن مدلهای زبانی بزرگ از پیش آموزش دیده شده بر instruction های مختلف میپردازد و بر اهمیت generalization بر unseen task تاکید میکند.

این مقاله شامل مقیاس‌پذیری instruction finetuning ها با توجه به وظایف و اندازه مدل ارزیابی می‌کند، و ادعا می‌کند سهم مثبتی در عملکرد داشته باشد و پتانسیل برای مقیاس‌بندی بیشتر را نشان می‌دهد. همچنین افزایش توانایی های استدلال از طریق finetuning، به ویژه در ارزیابی chain of thought (CoT) مشهود است. معرفی Flan-PaLM، یک مدل با۵۴۰ میلیارد پارامتر با وظایف finetuning و داده‌های CoT، نتایج پیشرفته‌ای را در معیارهای مختلف به نمایش می‌گذارد که نشان‌دهنده بهبود استدلال و توانایی‌های چند زبانه در مقایسه با مدل‌های قبلی است. Flan-PaLM در کارهایی مانند درک زبان چندوظیفه ای عظیم (MMLU)، TyDiQA تک شات ، و استدلال حسابی در زبان هایی که کمتر نشان داده شده اند، همراه با پیشرفت هایی در معیارهای ارزیابی هوش مصنوعی مسئول نشان می دهد. علاوه بر این، instruction finetuning برای انواع مدل‌های Flan-T5 توانایی‌های قوی zero-shot, few-shot و CoT را نشان می‌دهد که از مدلهای قبلی مانند T5,PaLM در برخی وظایف خاص عملکرد بهتری دارد.

قسمت دوم : Flan Finetuning

این قسمت Flan را معرفی می‌کند، یک روش finetuning برای مدل‌های زبان، که با پیشوندهایی مانند Flan-PaLM نشان داده می‌شود، و اثربخشی آن را در اندازه‌ها و معماری‌های مختلف مدل بررسی می‌کند.

بخش finetuning data شامل مخلوط داده‌های finetuning، از جمله Muffin، T0-SF، NIV2 و CoT است که در مجموع ۱۸۳۶ وظیفه finetuning را شامل می‌شود. این مقاله یک ترکیب CoT را معرفی می‌کند که بر وظایف استدلال متمرکز شده است و 9 مجموعه داده با annotation های CoT نوشته شده توسط انسان است. هر وظیفه با حدود ده الگوی instruction همراه است تا یادگیری مدل را هدایت کند.

بخش Finetuning procedure در مورد instruction finetuning در خانواده‌های مدل‌های مختلف، شامل T5، PaLM، و U-PaLM، با مدل‌هایی از Flan-T5-small تا PaLM و U-PaLM،

که اندازه‌های پارامترهای مختلف را در بر می‌گیرند، اعمال می‌شود. یک روش آموزشی یکنواخت در تمام مدل‌ها، با تغییرات در هایپر پارامترها مانند learning rate, batch size, dropout و finetuning steps اجرا می‌شود.علیرغم منابع محاسباتی گسترده مورد نیاز برای آموزش، مقدار محاسبات مورد استفاده برای finetuning نسبت به مرحله pre-training به طور قابل توجهی کمتر است. به عنوان مثال، تنها ۰.۲ درصد از محاسبات pre-training برای finetuning دستورات Flan-PaLM 540B استفاده می شود، که نمونه ای از تخصیص منابع کارآمد است.

در قسمت Evaluation protocol تمرکز بر ارزیابی عملکرد Flan-PaLM در مورد وظایفی است که بخشی از داده های finetuning نیستند. به جای استفاده از مجموعه‌های ارزیابی با همپوشانی داده‌های آموزشی، این مقاله از معیارهای چالش‌برانگیز استفاده می‌کند از جمله MMLU، BBH، TyDiQA و MGSM. مثلا با MMLU و BBH هر دو پیش‌بینی مستقیم و CoT را ارزیابی می‌کنند.ارزیابی شامل responsible AI است که اطمینان میدهد مدل به دستور العمل های اخلاقی پایبند باشد.

قسمت سوم: Scaling to 540B parameters and 1.8K tasks

این مطالعه تأثیر اندازه مدل مقیاس‌بندی و تعداد وظایف finetuning را بر عملکرد در کارهای انجام‌شده بررسی می‌کند.نتایج نشان می‌دهد که instruction finetuningهای چند وظیفه‌ای به طور قابل‌توجهی عملکرد را در مقایسه با عدم finetuning در تمام اندازه‌های مدل افزایش می‌دهد، با افزایش از ۹.۴٪ تا ۱۵.۵٪. افزایش تعداد وظایف finetuning عملکرد را بهبود می بخشد، به ویژه تا ۲۸۲ کار و بعد از آن بازده کاهش می یابد. اگرچه افزایش بیشتر اندازه مدل میتواند مزیت هایی به همراه داشت باشد اما افزودن تعداد بیشتری وظایف Finetuning ممکن است در مدلهای بزرگ فقط پیشرفت اندکی را به همراه داشته باشد.

قسمت چهارم:Finetuning with chain-of-thought annotations

در این بخش به بررسی افزودن داده های CoT در فرایند finetuning مدلها با هدف بهبود عملکرد نقطه checkpoint در ارزیابی‌های مختلف، از جمله multi-step reasoning در کنار وظایف سنتی NLP میپردازد. در ابتدا، این مطالعه نشان می دهد که ادغام 9 مجموعه داده CoT به طور قابل توجهی توانایی های استدلال را افزایش می دهد، مدل Flan-PaLM بهتر از مدل های قبلی در چندین معیار عمل میکند. این بیشتر اهمیت CoT finetuning را با آزمایش‌های ablation بررسی می‌کند، و نشان می‌دهد که حذف مجموعه داده‌های CoT توانایی‌های استدلال را کاهش می‌دهد، و بر ضرورت نمونه‌های CoT در حفظ و بهبود عملکرد مدل در ارزیابی‌ها تاکید می‌کند. علاوه بر این، CoT finetuning استدلال

Zero-shot را امکان پذیر می کند، به ویژه در عملکرد بهبود یافته مدل های Flan-PaLM در وظایف چالش برانگیز مانند کارهای big-bench مشهود است، و اثربخشی استدلال CoT فعال شده توسط عبارات خاص مانند Let’s think step by step را نشان می دهد.با وجود برخی محدودیت ها مانند عدم برتری نسبت به مدلهای تخصصی در وظایف الگوریتمی اضافه کردن annotation های CoT یک عامل مهم در توانایی مدل برای استدلال تعمیم دهی نشان داده شده است.

قسمت هفتم: Discussion

این مقاله، instruction finetuning را با مقیاس‌بندی تعداد وظایف finetuning، مقیاس‌بندی اندازه مدل و استفاده از CoT مورد بررسی قرار داده است. مدل‌های تنظیم‌شده با instructionهای حاصل، عملکرد بهبود یافته‌ای را در ارزیابی‌های مختلف، از جمله ارزیابی‌های few-shot، zero-shot، و CoT نشان می‌دهند. CoT finetuning برای تقویت توانایی های استدلال بسیار مهم است، با finetuning مشترک در داده های غیر CoT و CoT که باعث بهبود عملکرد قابل توجهی می شود در حالی که تطبیق پذیری در ارزیابی ها حفظ می شود. علاوه بر این، این مطالعه عمومیت instruction finetuning را در معماری‌های مدل، اندازه‌ها و اهداف پیش‌آموزشی مختلف نشان می‌دهد و سازگاری آن با سایر تکنیک‌های انطباق مدل را برجسته می‌کند. علاوه بر این، instruction finetuning، قابلیت استفاده را افزایش می‌دهد و آسیب‌های احتمالی را کاهش می‌دهد، به‌ویژه برای کارهای پیچیده مانند استدلال و توضیح، و رفع نگرانی‌های مربوط به استفاده از زبان toxic. نکته قابل توجه، ثابت شده است که finetuning instruction نسبتاً کارآمدی محاسباتی دارد، و بهبود عملکرد را با کسری کوچک از محاسبات قبل از آموزش ارائه می دهد. در نتیجه، شواهد ارائه شده بر اثربخشی instruction finetuning در بهبود عملکرد مدل در معیارهای ارزیابی مختلف تأکید می‌کند و پذیرش آن را برای تقریباً همه مدل‌های زبانی از پیش آموزش‌دیده توصیه می‌کند.

سوال ۲.

در این بخش از API مدل flan-t4-large استفاده کرده ایم:

```
model = T5ForConditionalGeneration.from_pretrained('google/flan-t5-large')
tokenizer = T5Tokenizer.from_pretrained('google/flan-t5-large')
```

سپس دیتاست گفته شده را فراخوانی کرده و از دیتای validation برای تست مدل و گزارش سه روش خواسته شده استفاده کرده ایم:

```
dataset = load_dataset("tasksource/bigbench", "sports_understanding")
test_data = dataset['validation']
```

یک نمونه از داده ها در زیر آورده شده است:

```
---------------------------------------------------------------------------------------------------
Example  1
---------------------------------------------------------------------------------------------------
INPUT:
Determine whether the following statement or statements are plausible or implausible:
Statement: Victor Wanyama took a left footed shot
Plausible/implausible?
---------------------------------------------------------------------------------------------------
Target:
['plausible']
---------------------------------------------------------------------------------------------------
```

شکل **4**: یک نمونه از دیتاست

برای Answer only فقط input را به عنوان پرامت به مدل میدهیم برای 3-shot سه نمونه مثال از داده های آموزش در پرامپت آورده شده است که در شکل زیر یک نمونه از آن را میبینیم:

```
def make_prompt(examples, sample):
    prompt = ''
    for index in examples:
        inputs = dataset['train'][index]['inputs']
        targets = dataset['train'][index]['targets'][0]
        prompt += f"""
inputs:
{inputs}
answer:
{targets}
"""

    inputs = dataset['validation'][sample]['inputs']

    prompt += f"""
inputs:
{inputs}
answer:
"""
    return prompt
prompt = print(make_prompt([30, 110, 150], 10))
```

```
inputs:
Determine whether the following statement or statements are plausible or implausible:
Statement: Chris Godwin walked to first base
Plausible/implausible?
answer:
implausible

inputs:
Determine whether the following statement or statements are plausible or implausible:
Statement: Tristan Jarry dunked the ball
Plausible/implausible?
answer:
implausible

inputs:
Determine whether the following statement or statements are plausible or implausible:
Statement: Nelson Cruz was out at first
Plausible/implausible?
answer:
plausible

inputs:
Determine whether the following statement or statements are plausible or implausible:
Statement: Igor Shesterkin launched a hail mary
Plausible/implausible?
answer:
```

شکل **5**: پرامت تولید شده برای **3-shot**

کد بخش 3-shot به شکل زیر پیاده شده است و برای هر تست از مثال های مختلفی از داده های آموزش استفاده میکند:

```
prompts = []
targets = []
outputs = []
#To-do the problem of is using train data as prompt
for i, data in enumerate(dataset['validation']):
    if 191> i >5 :
      prompt = make_prompt([i+5, i+4, i-5], i)
    else:
      prompt = make_prompt([30, 110, 150], i)
    input = tokenizer.encode(prompt, return_tensors="pt").to(device)
    output = tokenizer.decode(model.generate(input,max_new_tokens=50)[0],
skip_special_tokens=True)

    prompts.append(prompt)
    targets.append(data["targets"][0])
    outputs.append(output)

df_3_shot = pd.DataFrame({
```

```
    'prompt': prompts,
    'target': targets,
    'model_output': outputs
})

df_3_shot.to_csv('results_3_shot.csv', index=False)
accuracy_3_shot = accuracy_score(targets,outputs)
print(accuracy_3_shot)
```

در نهایت برای پیاده سازی chain of thought از دیتای شماره ۱۵ داده آموزش استفاده کرده و مراحل مختلف فکر کردن را به مدل گفته ایم و از جمله Let’s think step by step نیز استفاده کرده ایم:

```
def make_prompt(sample):
    prompt = ''
    prompt += f"""
Answer the following Plausible/implausible question by reasoning step-by-
step.
inputs:
{dataset['train'][10]['inputs']}
answer:
Let’s think step by step
No one can hit net.So the answer is {dataset['train'][10]['targets'][0]}
"""
    inputs = dataset['validation'][sample]['inputs']
    prompt += f"""
Answer the following Plausible/implausible question by reasoning step-by-
step.
inputs:
{inputs}
answer:
Let’s think step by step

"""
    return prompt
prompt = print(make_prompt(12))
```

Answer the following Plausible/implausible question by reasoning step-by-step.

inputs: Determine whether the following statement or statements are plausible or implausible: Statement: Baker Mayfield hit nothing but net Plausible/implausible?

answer: Let’s think step by step No one can hit net.So the answer is implausible

Answer the following Plausible/implausible question by reasoning step-by-step.

inputs: Determine whether the following statement or statements are plausible or implausible: Statement: Malcolm Brogdon ran out of bounds
Plausible/implausible?

answer: Let’s think step by step

*** اینکار را بدون آوردن مثال فقط با آوردن جمله let’s think step by step نیز انجام دادیم و دقت مدل حدود یک درصدکاهش داشته است.پرامت این روش نیز در قسمت زیر آورده شده است:

```
inputs = dataset['validation'][sample]['inputs']
prompt += f"""
inputs:
{inputs}
Let’s think step by step
answer:

"""
```

نتایج:

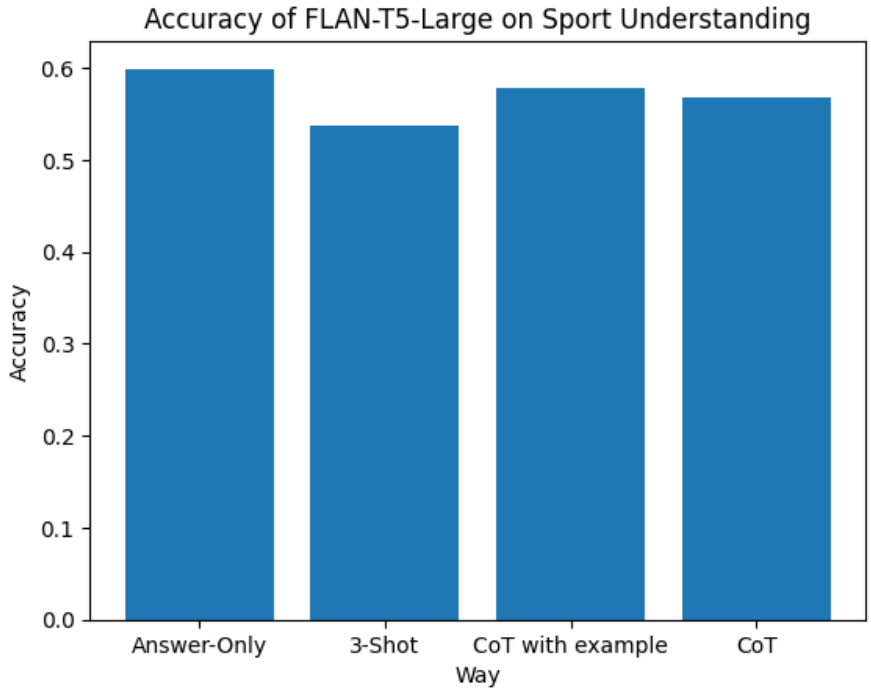
Answer_only = 0.5989

3-shot = 0.5380

CoT = 0.5685

CoT with example = 0.5786

نمودار میله ای مربوط به هر ۴ روش به شکل زیر است:



شکل **6**:نمودار میله ای برای مقایسه ۳ روش خواسته شده

همانطور که مشخص است آوردن ۳ نمونه در پرامت باعث به اشتباه افتادن مدل شده است این در حالی است که بدون آوردن هیچ مثالی مدل به خوبی عمل میکند که میتواند نشان دهنده این باشد که آوردن نمونه باعث پرت شدن مدل از سوال اصلی میشود و ممکن است نمونه ها ربطی به سوال نداشته باشند در روش CoT با آوردن یک نمونه و گفتن روش حل به مدل حدود یک درصد عملکرد آن بهتر شده است در حالیس که بدون آوردن مدل نیز عملکرد آن قابل قبول بوده است باید دقت شود که تمامی مثال های آورده شده در پرامت ها مربوط به دیتاست آموزش بوده است.

۴ فایل csv تولید شده به همراه تمامی کد پیوست شده است.

سوال ۳.

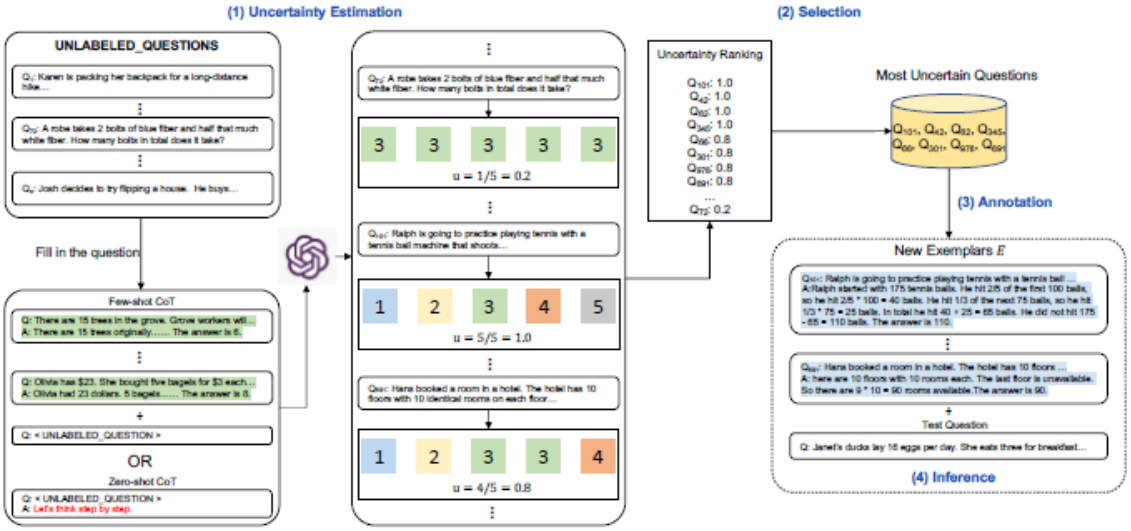
به تعداد l داده آموزش و m داده تست بصورت سوال بدون جواب یا مراحل استدلال در نظر گرفته شده است هدف این است که n سوال از داده های آموزش را به عنوان few-shot به همراه جواب و مراحل استدلال به عنوان E در نظر میگیریم. سپس از E به عنوان پرامپت برای تمام داده های تست استفاده میکینم و پیش بینی جواب به این دسته را بدست می آوریم حال سوال این است که n داده را چطور انتخاب کنیم؟

Uncertainty estimation : برای شناسایی سوالات با بیشترین عدم اطمینان در پیش بینی های مدل با معیارهایی از جمله self-confidence , disagreement ، واریانس و آنتروپی مورد بررسی قرار

میگیرد. با یا بدون چند CoT نوشته شده توسط انسان، LLM را K بار جستجو میکنند تا پاسخ های ممکن را با مراحل میانی برای مجموعه ای از سوالات آموزشی ایجاد کنند. سپس عدم قطعیت u را بر اساس پاسخ های k از طریق یک متریک عدم قطعیت محاسبه می کنند.

Selection and annotation : بر اساس مقدار عدم اطمینان تخمین زده شده در مرحله قبل سوالات رتبه بندی شده و نا مشخص ترین آنها برای annotation انسانی انتخاب میشوند.

Inference : نمونه های annotation در مرحله قبل از n داده آموزش به عنوان Few-shot در پرامپت به مدل داده میشود تا هر سؤال را با نمونه های مشروح جدید استنباط شود.



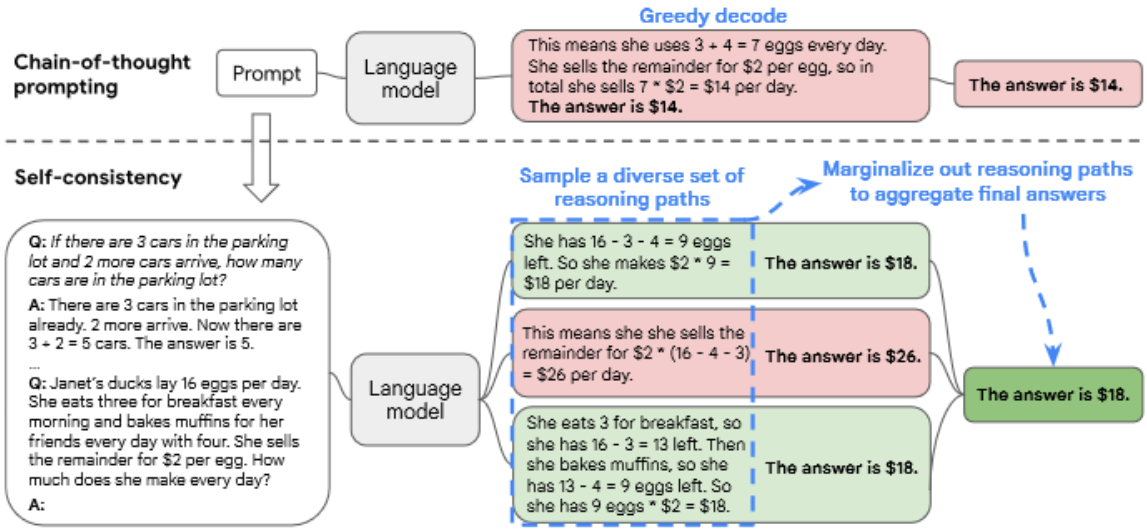
شکل 7:active-prompt

سوال ۴.

در این مقاله، ما یک استراتژی decoding جدید، self-consistency، برای جایگزینی naive greedy decoding مورد استفاده در پرامپت CoT، پیشنهاد می‌شود. ابتدا مجموعه متنوعی از مسیرهای استدلال را نمونه برداری می کند، به جای اینکه فقط از مسیر greedy استفاده کند، و سپس با به حاشیه راندن مسیرهای استدلال نمونه، سازگارترین پاسخ را انتخاب می کند. این روش شامل مراحل زیر است:

یک مدل LLM را با استفاده از تکنیک CoT پرامپت کرده سپس "greedy decoding " در پرامپت CoT با نمونه برداری از LLM، Decoding جایگزین میشود به نحوی که مجموعه متنوعی از مسیرهای استدلال تولید میشود. در انتها با انتخاب سازگارترین پاسخ در مجموعه پاسخ نهایی، آنها aggregate میشوند .

این روش نسبت به CoT موثرتر است چراکه محدودیت های greedy decoding را پشت سر گذاشته و از بی ثباتی های نمونه برداری بصرت تکی جلوگیری میکند که منجذ به نتایج استدلال دقیق تر و قابلب اعتمادتر میشود. سازگاری با خود این شهود را تحت تأثیر قرار می دهد که یک مسئله استدلال پیچیده معمولاً چندین روش مختلف تفکر را می پذیرد که منجر به پاسخ صحیح منحصر به فرد آن می شود.



شکل 8: **self-consistency**