

**人工智能综合实训报告**

课题名称： 基于预训练大模型可信自动问答设计实现

班 级： 人工智能202

学 号： 202016034512

姓 名： 郑连驰

指导教师： 刘小明

中原工学院计算机学院

2023年 12 月 22 日

[第1章 问题分析 3](#_Toc155381367)

[1.1可信自动问答的挑战 3](#_Toc155381368)

[1.2预训练大模型在可信自动问答中的应用 4](#_Toc155381369)

[第2章 数据集选择与预处理 5](#_Toc155381370)

[2.1.数据集选择 5](#_Toc155381371)

[2.2.数据预处理 5](#_Toc155381372)

[第3章 模型基本框架 7](#_Toc155381373)

[3.1. 编码 7](#_Toc155381374)

[3.2. 融合 7](#_Toc155381375)

[3.3. 解码 8](#_Toc155381376)

[3.4. 总流程 8](#_Toc155381377)

[第4章 模型搭建与部署 9](#_Toc155381378)

[4.1 模型构建 9](#_Toc155381379)

[4.2 模型部署 14](#_Toc155381380)

[4.2.1环境搭建 14](#_Toc155381381)

[4.2.2模型训练 15](#_Toc155381382)

[4.2.3.flask部署 15](#_Toc155381383)

[4.3 前后端部署实现 17](#_Toc155381384)

[第5章 模型评估 19](#_Toc155381385)

[5.1. 指标评估 19](#_Toc155381386)

[5.1.1 准确率 19](#_Toc155381387)

[5.1.2合理性质量评估 19](#_Toc155381388)

[第6章 总结 20](#_Toc155381389)

[附录 21](#_Toc155381390)

[附录A：源程序代码 21](#_Toc155381391)

# 问题分析

在当今数字化时代，自然语言处理技术的迅速发展为人机交互提供了前所未有的可能性。其中，基于预训练大模型的可信自动问答系统成为人工智能领域的研究热点之一。随着BERT（Bidirectional Encoder Representations from Transformers）、GPT（Generative Pre-trained Transformer）等预训练大模型的崭露头角，自然语言理解的水平得到了质的提升。然而，在解决当前自动问答系统面临的挑战时，近期涌现的T5（Text-To-Text Transfer Transformer）模型及多模态技术为这一领域带来了新的机遇。

T5模型以其独特的“文本到文本”训练框架，将多种自然语言处理任务统一为文本生成问题，实现了更加灵活和通用的语言表示。通过这种方式，T5模型不仅在文本生成任务上表现出色，而且在多模态信息处理中也展现了惊人的效果。多模态技术将图像、语音、视频等不同形式的信息融合到自然语言处理任务中，为自动问答系统增加了更为丰富的输入来源，提高了系统的智能性和适用性。

在实际应用中，这些自动问答系统不仅仅局限于文本输入。多模态场景下，用户可以通过文字、图片、语音等多种形式提出问题，系统也需要能够理解和处理这些多样化的输入。这为企业、机构和服务提供商提供了更多的可能性，例如在虚拟购物助手中，用户可以通过文字描述或上传图片来寻求产品建议；在在线学习平台中，学生可以通过语音提问以获取更直观的解释。

## 1.1可信自动问答的挑战

随着上述技术进步，我们也面临着更为复杂的挑战。如何在多模态场景下设计一个可信的自动问答系统，使其能够有效地处理和融合多样化的信息，成为当前研究的焦点之一。数据偏见、模型解释性、用户误导等问题在多模态环境下可能变得更为显著，因此需要深入研究和创新解决方案。

预训练模型的数据偏见：预训练大模型在训练过程中可能受到特定数据集的影响，导致在某些领域或主题上表现更为出色，而在其他领域可能存在偏见。这可能导致系统在回答问题时产生不准确或倾向性的答案。

可信度的重要性：在自动问答系统中，用户对系统的可信度至关重要。用户需要相信系统能够理解问题，提供准确和可靠的答案。如果系统的可信度不高，用户可能会对系统产生疑虑，影响其使用体验。

## 1.2预训练大模型在可信自动问答中的应用

预训练大模型在可信自动问答中的应用是人工智能领域的一项关键研究方向。这些模型通过在大规模语料库上进行预训练，学得了丰富的语言表示，从而在解决自动问答任务时展现出色的性能。以下是预训练大模型在可信自动问答中的一些主要应用：

文本理解与问题生成：预训练大模型在理解自然语言文本方面表现出色。在自动问答中，这些模型可以准确地理解用户提出的问题，对问题的关键信息进行抽取，以生成准确的回答。此应用旨在提高系统对用户问题的理解能力，使得问答过程更为自然和有效。

语义理解与上下文处理：可信自动问答系统需要更好地处理问题的语义和上下文信息。预训练大模型通过编码上下文信息，能够更好地理解问题的语境，并能够在对话中保持一致性。这对于处理多轮对话、复杂问题和上下文敏感的任务至关重要。

领域自适应：可信自动问答系统在不同领域和专业领域的应用需要模型能够自适应。预训练大模型的通用性使其能够迅速适应新领域，通过微调或领域特定的训练，使系统在特定领域内表现更为出色。

实时问答：预训练大模型的高效性使其适用于实时问答场景。用户可以即时提出问题，系统通过迅速处理并生成答案，实现了实时性需求，例如在线客服场景。

多模态问答：随着多模态技术的发展，预训练大模型也被应用于处理多种形式的输入，如图像、语音和文本的结合。这拓展了自动问答系统的应用范围，使得系统能够更全面地回应用户的不同需求。

# 数据集选择与预处理

## 2.1.数据集选择

ScienceQA 是从小学和高中科学课程中收集的，包含21,208个多模态 多项选择科学问题。在 ScienceQA 的问题中，有10,332个（48.7%）有图像上下文，10,220个（48.2%）有文本上下文，6,532（30.8%）两者兼而有之。大多数问题都附有扎根的讲座（83.9%）和详细的解释 (90.5%). 讲座和讲解提供得出正确答案的一般外部知识和具体原因。综上所述ScienceQA 是第一个对讲座和答案的解释进行注释的大规模多模态数据集。

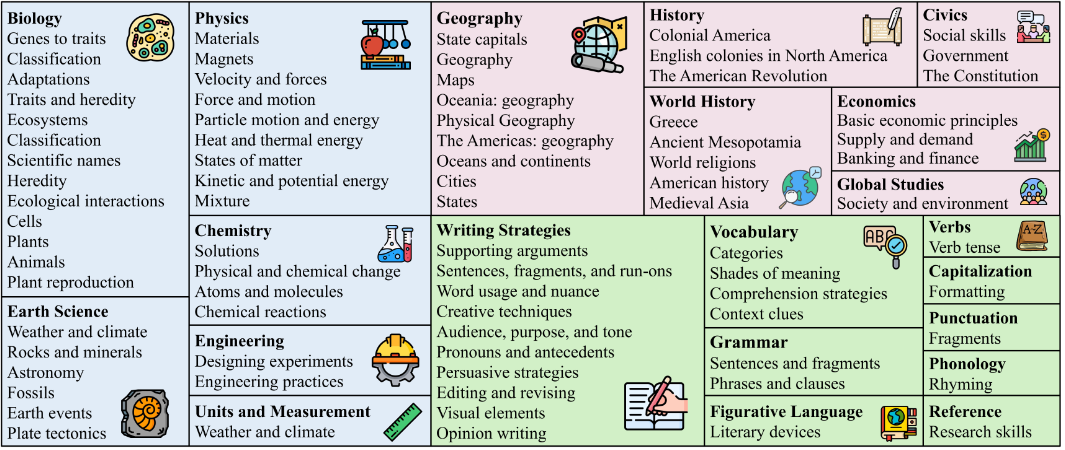
ScienceQA与以前的数据集相比，在自然科学、语言科学和社会科学三个学科中具有更丰富的领域多样性。 每个主题中的问题首先按主题分类 （生物学、物理、化学等），然后按类别（植物、细胞、动物、等），最后通过技能（将水果和蔬菜分类为植物部分，识别非洲国家等）。ScienceQA 包含26个主题、127 个类别和379项技能，涵盖广泛的领域。如图1-1所示

图1-1数据集领域介绍

## 2.2.数据预处理

不同的视觉特征可能会影响模型的性能。使用了四种不同的模型对图像进行提取特征，分别为resnet，cilp，detr，vit，提取特征的维度为（512，2048），（49，2048），（100，256），（145，1024）。提取特征的代码如下所示：

|  |  |
| --- | --- |
| |  | | --- | | vit\_model = timm.create\_model("vit\_large\_patch32\_384", pretrained=True, num\_classes=0)  vit\_model.eval()  config = resolve\_data\_config({}, model=vit\_model)  transform = create\_transform(\*\*config)  with torch.no\_grad():      img = Image.open(input\_image).convert("RGB")      input = transform(img).unsqueeze(0)      feature = vit\_model.forward\_features(input) |   表2-1视觉特征提取代码 |

# 模型基本框架

由于 Multimodal-CoT 任务需要生成推理链并利用视觉特征，我使用了 T5 编码器-解码器架构。

## 3.1. 编码

模型F(X)采用语言和视觉输入，通过以下函数获得文本表示 Hlanguage 和图像特征 Hvision：

= LanguageEncoder()

= · VisionExtractor()

其中 LanguageEncoder(·) 被实现为 Transformer 模型。我使用 Transformer 编码器最后一层的隐藏状态作为语言表示。同时，VisionExtractor(·)用于将输入图像向量化为视觉特征。我应用可学习的投影矩阵将VisionExtractor()

的形状转换为的形状。

## 3.2. 融合

在获得语言和视觉表示后，我使用单头注意力网络将文本标记与图像块相关联，其中query (Q)、key (K) 和value (V) 分别是 、 和 。

其中 dk 与 Hlanguage 的维度相同，因为使用的是单头注意力网络。然后，我应用门控融合机制来融合 和。

其中和都是可学习的参数。

## 3.3. 解码

将 输入解码器以预测目标 Y = Decoder()，其中 Decoder(·) 实现为 Transformer 模型。

## 3.4. 总流程

|  |
| --- |
| **Input:** 文本输入 , 视觉输入  **Output:** 生成推理 R, 推理答案 A  1: 构建输入 X = {, }  2: 生成推理 R = F (X) 使用模型 F (·)  3: 转换图像句子对 Q= LookUp(S, E, F)  4: 增加推理结果到文本输入中 = ◦ R.  5: 构建新的输入 X′= {, }  6: 推理答案 A 通过新的输入, A =F (X′).  7: **procedure** F(X)  8: 编码语言和视觉特征 and ,  9: 通过注意力构建语言和视觉特征之间的交互得到  10: 融合 and 通过门控融合机制得到  11: 将 输入编码器获得输出结果 Y  12: return Y  13: **end procedure** |

# 模型搭建与部署

## 4.1 模型构建

在这一部分，我将详细介绍模型的构建过程。我实现了一个基于T5模型架构的多模态生成模型（T5ForMultimodalGeneration）。以下是模型的主要架构分析：

1.T5ForMultimodalGeneration类：

继承自T5ForConditionalGeneration类，该类是Hugging Face中Transformers库中的T5模型的一个派生类，用于多模态生成任务。

在forward方法中，模型接收文本输入（input\_ids）、图像输入（image\_ids）以及其他相关参数，并返回生成的文本序列的概率分布（logits）或其他输出，具体取决于任务（训练、生成等）。

包含了一个共享的嵌入层（self.shared），用于将文本和图像输入嵌入到模型的隐藏空间中。包含一个编码器（self.encoder）和一个解码器（self.decoder），分别用于处理输入和生成输出。最后的线性层（self.lm\_head）用于将模型的隐藏状态映射到词汇表上，生成文本。

2.JointEncoder类：

继承自T5Stack类，这是T5模型的一部分，用于处理文本输入。在这里被用作编码器，接收文本和图像输入。

通过在T5编码器的基础上添加自定义的多模态处理层，包括图像嵌入层、多头自注意力层（self.mha\_layer）、门控层（self.gate\_dense）等，实现对文本和图像的联合编码。通过使用门控机制，动态调整文本和图像的权重。

forward方法定义了模型的前向传播逻辑，处理输入数据并返回相应的输出。

在文本输入的基础上，加入了图像输入（image\_ids），并在编码器中进行多模态融合。

使用门控机制将文本和图像的信息融合，以便更好地生成多模态的输出。

prepare\_inputs\_for\_generation方法用于为生成任务准备输入数据，包括解码器的输入（decoder\_input\_ids）、过去的键值状态（past）、注意力掩码（attention\_mask）、是否使用缓存（use\_cache）等。可以接受图像输入（image\_ids），在生成任务中用于融合文本和图像信息。

test\_step方法用于在测试阶段进行模型的评估，生成文本，并与目标文本进行比较，返回预测和目标文本。

总体而言，该模型是一个融合文本和图像信息的多模态生成模型，基于T5模型的架构进行了扩展和定制。通过自定义的多模态编码层，模型能够有效地处理文本和图像的联合信息，用于生成多模态任务的输出。

|  |
| --- |
| class T5ForMultimodalGeneration(T5ForConditionalGeneration):      \_keys\_to\_ignore\_on\_load\_missing = [          r"encoder.embed\_tokens.weight",          r"decoder.embed\_tokens.weight",          r"lm\_head.weight",      ]      \_keys\_to\_ignore\_on\_load\_unexpected = [          r"decoder.block.0.layer.1.EncDecAttention.relative\_attention\_bias.weight",      ]      def \_\_init\_\_(self, config: T5Config, patch\_size):          super().\_\_init\_\_(config)  *self*.model\_dim = config.d\_model  *self*.shared = nn.Embedding(config.vocab\_size, config.d\_model)          encoder\_config = copy.deepcopy(config)          encoder\_config.is\_decoder = False          encoder\_config.use\_cache = False          encoder\_config.is\_encoder\_decoder = False  *# self.encoder = T5Stack(encoder\_config, self.shared)*  *self*.encoder = JointEncoder(encoder\_config, *self*.shared, patch\_size)          decoder\_config = copy.deepcopy(config)          decoder\_config.is\_decoder = True          decoder\_config.is\_encoder\_decoder = False          decoder\_config.num\_layers = config.num\_decoder\_layers  *self*.decoder = T5Stack(decoder\_config, *self*.shared)  *self*.lm\_head = nn.Linear(config.d\_model, config.vocab\_size, bias=False)  *# Initialize weights and apply final processing*  *self*.post\_init()  *# Model parallel*  *self*.model\_parallel = False  *self*.device\_map = None      def forward(          self,          input\_ids: Optional[torch.LongTensor] = None,          image\_ids=None,          attention\_mask: Optional[torch.FloatTensor] = None,          decoder\_input\_ids: Optional[torch.LongTensor] = None,          decoder\_attention\_mask: Optional[torch.BoolTensor] = None,          head\_mask: Optional[torch.FloatTensor] = None,          decoder\_head\_mask: Optional[torch.FloatTensor] = None,          cross\_attn\_head\_mask: Optional[torch.Tensor] = None,          encoder\_outputs: Optional[Tuple[Tuple[torch.Tensor]]] = None,          past\_key\_values: Optional[Tuple[Tuple[torch.Tensor]]] = None,          inputs\_embeds: Optional[torch.FloatTensor] = None,          decoder\_inputs\_embeds: Optional[torch.FloatTensor] = None,          labels: Optional[torch.LongTensor] = None,          use\_cache: Optional[bool] = None,          output\_attentions: Optional[bool] = None,          output\_hidden\_states: Optional[bool] = None,          return\_dict: Optional[bool] = None,      ) -> Union[Tuple[torch.FloatTensor], Seq2SeqLMOutput]:          use\_cache = use\_cache if use\_cache is not None else *self*.config.use\_cache          return\_dict = return\_dict if return\_dict is not None else *self*.config.use\_return\_dict  *# FutureWarning: head\_mask was separated into two input args - head\_mask, decoder\_head\_mask*          if head\_mask is not None and decoder\_head\_mask is None:              if *self*.config.num\_layers == *self*.config.num\_decoder\_layers:                  warnings.warn(\_\_HEAD\_MASK\_WARNING\_MSG, FutureWarning)                  decoder\_head\_mask = head\_mask  *# Encode if needed (training, first prediction pass)*          if encoder\_outputs is None:  *# Convert encoder inputs in embeddings if needed*              encoder\_outputs = *self*.encoder(                  input\_ids=input\_ids,                  attention\_mask=attention\_mask,                  inputs\_embeds=inputs\_embeds,                  image\_ids=image\_ids,                  head\_mask=head\_mask,                  output\_attentions=output\_attentions,                  output\_hidden\_states=output\_hidden\_states,                  return\_dict=return\_dict,              )          elif return\_dict and not isinstance(encoder\_outputs, BaseModelOutput):              encoder\_outputs = BaseModelOutput(                  last\_hidden\_state=encoder\_outputs[0],                  hidden\_states=encoder\_outputs[1] if len(encoder\_outputs) > 1 else None,                  attentions=encoder\_outputs[2] if len(encoder\_outputs) > 2 else None,              )          hidden\_states = encoder\_outputs[0]          if *self*.model\_parallel:              torch.cuda.set\_device(*self*.decoder.first\_device)          if labels is not None and decoder\_input\_ids is None and decoder\_inputs\_embeds is None:  *# get decoder inputs from shifting lm labels to the right*              decoder\_input\_ids = *self*.\_shift\_right(labels)  *# Set device for model parallelism*          if *self*.model\_parallel:              torch.cuda.set\_device(*self*.decoder.first\_device)              hidden\_states = hidden\_states.to(*self*.decoder.first\_device)              if decoder\_input\_ids is not None:                  decoder\_input\_ids = decoder\_input\_ids.to(*self*.decoder.first\_device)              if attention\_mask is not None:                  attention\_mask = attention\_mask.to(*self*.decoder.first\_device)              if decoder\_attention\_mask is not None:                  decoder\_attention\_mask = decoder\_attention\_mask.to(*self*.decoder.first\_device)  *# Decode*          decoder\_outputs = *self*.decoder(              input\_ids=decoder\_input\_ids,              attention\_mask=decoder\_attention\_mask,              inputs\_embeds=decoder\_inputs\_embeds,              past\_key\_values=past\_key\_values,              encoder\_hidden\_states=hidden\_states,              encoder\_attention\_mask=attention\_mask,              head\_mask=decoder\_head\_mask,              cross\_attn\_head\_mask=cross\_attn\_head\_mask,              use\_cache=use\_cache,              output\_attentions=output\_attentions,              output\_hidden\_states=output\_hidden\_states,              return\_dict=return\_dict,          )          sequence\_output = decoder\_outputs[0]  *# Set device for model parallelism*          if *self*.model\_parallel:              torch.cuda.set\_device(*self*.encoder.first\_device)  *self*.lm\_head = *self*.lm\_head.to(*self*.encoder.first\_device)              sequence\_output = sequence\_output.to(*self*.lm\_head.weight.device)          if *self*.config.tie\_word\_embeddings:  *# Rescale output before projecting on vocab*  *# See https://github.com/tensorflow/mesh/blob/fa19d69eafc9a482aff0b59ddd96b025c0cb207d/mesh\_tensorflow/transformer/transformer.py#L586*              sequence\_output = sequence\_output \* (*self*.model\_dim\*\*-0.5)          lm\_logits = *self*.lm\_head(sequence\_output)          loss = None          if labels is not None:              loss\_fct = CrossEntropyLoss(ignore\_index=-100)              loss = loss\_fct(lm\_logits.view(-1, lm\_logits.size(-1)), labels.view(-1))  *# TODO(thom): Add z\_loss https://github.com/tensorflow/mesh/blob/fa19d69eafc9a482aff0b59ddd96b025c0cb207d/mesh\_tensorflow/layers.py#L666*          if not return\_dict:              output = (lm\_logits,) + decoder\_outputs[1:] + encoder\_outputs              return ((loss,) + output) if loss is not None else output          return Seq2SeqLMOutput(              loss=loss,              logits=lm\_logits,              past\_key\_values=decoder\_outputs.past\_key\_values,              decoder\_hidden\_states=decoder\_outputs.hidden\_states,              decoder\_attentions=decoder\_outputs.attentions,              cross\_attentions=decoder\_outputs.cross\_attentions,              encoder\_last\_hidden\_state=encoder\_outputs.last\_hidden\_state,              encoder\_hidden\_states=encoder\_outputs.hidden\_states,              encoder\_attentions=encoder\_outputs.attentions,          )      def prepare\_inputs\_for\_generation(          self, decoder\_input\_ids, past=None, attention\_mask=None, use\_cache=None, encoder\_outputs=None, \*\*kwargs      ):  *# cut decoder\_input\_ids if past is used*          if past is not None:              decoder\_input\_ids = decoder\_input\_ids[:, -1:]          output = {              "input\_ids": None,  *# encoder\_outputs is defined. input\_ids not needed*              "encoder\_outputs": encoder\_outputs,              "past\_key\_values": past,              "decoder\_input\_ids": decoder\_input\_ids,              "attention\_mask": attention\_mask,              "use\_cache": use\_cache,  *# change this to avoid caching (presumably for debugging)*          }          if "image\_ids" in kwargs:              output["image\_ids"] = kwargs['image\_ids']          return output        def test\_step(self, tokenizer, batch, \*\*kwargs):          device = next(*self*.parameters()).device          input\_ids = batch['input\_ids'].to(device)          image\_ids = batch['image\_ids'].to(device)          output = *self*.generate(              input\_ids=input\_ids,              image\_ids=image\_ids,              \*\*kwargs          )          generated\_sents = tokenizer.batch\_decode(output, skip\_special\_tokens=True)          targets = tokenizer.batch\_decode(batch['labels'], skip\_special\_tokens=True)          result = {}          result['preds'] = generated\_sents          result['targets'] = targets          return result |

表4-1基础模型代码

## 4.2 模型部署

在整个模型构建、训练和部署的过程中，我遇到了一系列挑战。以下是我在这些阶段所遇到的困难以及克服方法：

### 4.2.1环境搭建

在搭建环境时，我使用了虚拟环境，确保项目的依赖项不与系统中其他项目的依赖项冲突。使用包管理工具如pip，并利用requirements.txt文件记录所有依赖项及其版本，有助于保持环境的一致性。此外，如果使用 GPU 进行训练，确保CUDA和cuDNN版本与PyTorch兼容。

|  |
| --- |
| huggingface-hub>=0.4.0  numpy==1.23.2  openai==0.23.0  pandas==1.4.3  rouge==1.0.1  sentence-transformers==2.2.2  transformers==4.30.0  nltk==3.6.6  evaluate==0.4.0  rouge==1.0.1  rouge\_score==0.1.2  rich>=13.3.2 |

表4-2主要依赖库

### 4.2.2模型训练

模型微调多达 20 个 epoch，学习率为 5e-5。最大输入序列长度为 512。基础模型和大模型的batch\_size大小分别为 16 和 8。我的实验在1个RTX 3090 32G GPU上运行。

### 4.2.3.flask部署

Flask是一个轻量级的Web应用框架，其简洁、灵活的设计使得部署变得相对容易。下面是使用Flask部署Web应用的一些优点：

1. 简易性： Flask是一个简单而直观的框架，易于学习和使用。其代码结构清晰，开发者能够快速上手，迅速搭建出一个基本的Web应用。
2. 灵活性： Flask的设计理念是“micro”框架，它提供了最基本的工具，但允许开发者根据需求选择性地集成其他组件。这种灵活性使得开发者可以根据项目的具体需求进行定制，而不被强加于固定的结构或组件。
3. 轻量级： 由于其轻量级的特性，Flask在内存和计算资源上的消耗相对较低。这对于部署在资源有限的环境中（比如云服务器或嵌入式设备）的应用来说是一个优势。
4. 适用于小型项目： 对于小型项目或原型开发，Flask是一个理想的选择，因为它允许开发者快速构建并迭代，而无需过多复杂性。

总的来说，Flask的优点在于其简单、灵活、轻量级的特性，使得它成为构建小到中型Web应用的理想框架，特别是在初创公司或小团队的项目中。以下是flask后端代码

|  |
| --- |
| import torch,os  from flask\_cors import CORS  from flask import Flask, request, jsonify  from transformers import AutoTokenizer, T5ForConditionalGeneration  from model import  T5ForMultimodalGeneration  import numpy as np  import timm  from PIL import Image  from timm.data import resolve\_data\_config  from timm.data.transforms\_factory import create\_transform  le\_tokenizer = AutoTokenizer.from\_pretrained("models/rationale")  le\_model = T5ForConditionalGeneration.from\_pretrained("models/rationale")  ans\_tokenizer = AutoTokenizer.from\_pretrained("models/answer")  ans\_model = T5ForMultimodalGeneration.from\_pretrained("models/answer", patch\_size=(145, 1024))  vit\_model = timm.create\_model("vit\_large\_patch32\_384", pretrained=True, num\_classes=0)  vit\_model.eval()  app = Flask(\_\_name\_\_)  CORS(app)  def extract\_features(input\_image):      config = resolve\_data\_config({}, model=vit\_model)      transform = create\_transform(\*\*config)      with torch.no\_grad():          img = Image.open(input\_image).convert("RGB")          input = transform(img).unsqueeze(0)          feature = vit\_model.forward\_features(input)      return feature  def get\_le(prompt):      prompt = prompt + "\nSolution:"      input\_ids = le\_tokenizer(prompt, return\_tensors="pt").input\_ids      outputs = le\_model.generate(input\_ids,max\_length=512)      le = le\_tokenizer.decode(outputs[0], skip\_special\_tokens=True)      return le  def get\_ans(le,prompt,have\_pic=False,input\_image=None):      prompt = prompt + le + "\nAnswer:"      input\_ids = ans\_tokenizer(prompt, return\_tensors="pt").input\_ids      if have\_pic:          image\_ids = extract\_features(input\_image)      else:          image\_ids = np.zeros((145, 1024))      image\_ids = torch.FloatTensor(image\_ids).squeeze()      image\_ids = image\_ids.unsqueeze(0)      outputs = ans\_model.generate(input\_ids,image\_ids=image\_ids,max\_length=512)      ans = ans\_tokenizer.decode(outputs[0], skip\_special\_tokens=True)      return ans    @app.route('/chat', methods=['GET'])  def chat():      R={}      prompt = str(request.args["message"])      have\_pic=False      filename = "image.png"      input\_image=None      if os.path.exists(filename):          have\_pic = True          input\_image=filename        le = get\_le(prompt)      R["le"]=le      ans = get\_ans(le,prompt,have\_pic,input\_image)      R["ans"]=ans      return jsonify(R)  if \_\_name\_\_ == "\_\_main\_\_":  app.run(port=80,debug = False) |

表4-3 flask后端代码

## 4.3 前后端部署实现

我前端使用了Vue,综合后端flask部署在jeecg上作为一个菜单，实现功能为：1.上传图片作为问答的内容 2.进行prompt对模型进行提问对话。以下是测试样例：

1. Question: Which of these states is farthest north?

Context: An aerial view of a painting of a forest.

Options: (A) West Virginia (B) Louisiana (C) Arizona (D) Oklahoma

answer：A

测试结果（测试样例包含一张图片）如图4-1所示。



图4-1 sample1

2.Question: How long does it take to bake lasagna in the oven?

Context: Select the better estimate.

Options: (A) 33 minutes (B) 33 hours

answer：A

测试结果如图4-2所示。



图4-2 sample2

# 模型评估

## 5.1. 指标评估

### 5.1.1 准确率

在eval集上，我们评估了模型的整体准确率。对于eval集中的每个问题，模型预测的答案与实际答案相比较，并计算模型的准确率。同时分别计算自然科学、社会科学、语言科学等不同学科的准确率。根据文本、图像的存在与否，计算包含文本、包含图像、无上下文的准确率。根据年级划分，计算不同年级范围内的准确率。其中对于推理答案模型准确率为0.8594671068144305。

### 5.1.2合理性质量评估

在eval集上，使用BLEU（Bilingual Evaluation Understudy）计算1-gram和4-gram的得分，衡量模型生成的理由与参考答案的相似度。使用Rouge-L计算得分，评估生成的理由的长文本相似性。利用预训练的Sentence Transformer模型（MiniLM-L6-v2）计算生成理由与参考答案的相似性得分。对于得到推理过程模型，其中eval\_rouge1: 93.7021,eval\_rouge2: 81.2144,eval\_rougeL: 91.4566,eval\_rougeLsum: 93.24。

# 总结

通过此次实训，我深入了解了自然语言处理技术中的可信自动问答系统过程，特别是关注了基于预训练大模型和多模态技术的应用。通过对BERT、GPT、T5等模型的研究，我了解了它们在文本理解，多模态等场景下的优势和挑战。

我学习了T5模型的独特训练框架，将多种自然语言处理任务统一为文本生成问题，实现了更加灵活和通用的语言表示。同时，多模态技术的应用使得系统能够处理图像、语音、文本等多种形式的信息，提升了系统的全面性和用户体验。

在模型的搭建与部署过程中，我了解了T5编码器-解码器架构的基本原理，并通过自定义的多模态编码层实现了对文本和图像信息的联合处理。模型的部署中，我选择使用Flask框架，充分利用其简易性、灵活性和轻量级的特点，使得部署过程相对容易。

在模型训练阶段，我遇到了环境搭建、依赖库管理等一系列挑战，通过使用虚拟环境、包管理工具和详细记录依赖项的方式，成功解决了这些问题。同时，我在Flask框架的前后端部署中，运用Vue实现了一个简单而实用的用户界面，使得模型能够方便地接收用户的问题并生成相应的回答。

在模型评估方面，我通过准确率、BLEU、Rouge等指标对模型性能进行了全面评估。对于推理答案的生成，我还使用了Sentence Transformer模型进行相似性评估，增强了模型在合理性质量上的评估。

通过这次实训，我不仅深入了解了自然语言处理技术的前沿发展，还学到了模型搭建、训练和部署的实际操作技能。我提升了对多模态场景下可信自动问答系统设计的认识。在今后，我将继续深入了解基于预训练大模型的可信自动问答，努力提升自己在人工智能领域的应用能力和创新能力。同时，我计划不断跟踪自然语言处理技术的最新进展，保持对行业动态的敏感性，为解决实际问题提供更加有效的解决方案。

# 附录

## 附录A：源程序代码

|  |
| --- |
| import os  import numpy as np  import torch  import os  import re  import json  import argparse  import random  from transformers import AutoTokenizer, DataCollatorForSeq2Seq, Seq2SeqTrainingArguments, Seq2SeqTrainer, T5ForConditionalGeneration  from model import T5ForMultimodalGeneration  from utils\_data import img\_shape, load\_data\_std, load\_data\_img, ScienceQADatasetStd, ScienceQADatasetImg  from utils\_prompt import \*  from utils\_evaluate import get\_scores  from rich.table import Column, Table  from rich import box  from rich.console import Console  console = Console(record=True)  import nltk  import evaluate  def parse\_args():  parser = argparse.ArgumentParser()  parser.add\_argument('--data\_root', type=str, default='data')  parser.add\_argument('--output\_dir', type=str, default='experiments')  parser.add\_argument('--model', type=str, default='allenai/unifiedqa-t5-base')  parser.add\_argument('--options', type=list, default=["A", "B", "C", "D", "E"])  parser.add\_argument('--epoch', type=int, default=20)  parser.add\_argument('--lr', type=float, default=5e-5)  parser.add\_argument('--bs', type=int, default=16)  parser.add\_argument('--input\_len', type=int, default=512)  parser.add\_argument('--output\_len', type=int, default=64)  parser.add\_argument('--eval\_bs', type=int, default=16)  parser.add\_argument('--eval\_acc', type=int, default=None, help='evaluate accumulation step')  parser.add\_argument('--train\_split', type=str, default='train', choices=['train', 'trainval', 'minitrain'])  parser.add\_argument('--val\_split', type=str, default='val', choices=['test', 'val', 'minival'])  parser.add\_argument('--test\_split', type=str, default='test', choices=['test', 'minitest'])    parser.add\_argument('--use\_generate', action='store\_true', help='only for baseline to improve inference speed')  parser.add\_argument('--final\_eval', action='store\_true', help='only evaluate the model at the final epoch')  parser.add\_argument('--user\_msg', type=str, default="baseline", help='experiment type in the save\_dir')  parser.add\_argument('--img\_type', type=str, default=None, choices=['detr', 'clip', 'resnet','vit'], help='type of image features')  parser.add\_argument('--eval\_le', type=str, default=None, help='generated rationale for the dev set')  parser.add\_argument('--test\_le', type=str, default=None, help='generated rationale for the test set')  parser.add\_argument('--evaluate\_dir', type=str, default=None, help='the directory of model for evaluation')  parser.add\_argument('--caption\_file', type=str, default='data/captions.json')  parser.add\_argument('--use\_caption', action='store\_true', help='use image captions or not')  parser.add\_argument('--prompt\_format', type=str, default='QCM-A', help='prompt format template',  choices=['QCM-A', 'QCM-E', 'QCM-LE', 'QCMG-A', 'QCM-LEA', 'QCM-ALE'])  parser.add\_argument('--seed', type=int, default=42, help='random seed')  args = parser.parse\_args()  return args    def T5Trainer(  dataframe, args,  ):  torch.manual\_seed(args.seed) # pytorch random seed  np.random.seed(args.seed) # numpy random seed  torch.backends.cudnn.deterministic = True    if args.evaluate\_dir is not None:  args.model = args.evaluate\_dir  tokenizer = AutoTokenizer.from\_pretrained(args.model)  console.log(f"""[Model]: Loading {args.model}...\n""")  console.log(f"[Data]: Reading data...\n")  problems = dataframe['problems']  qids = dataframe['qids']  train\_qids = qids['train']  test\_qids = qids['test']  val\_qids = qids['val']    if args.evaluate\_dir is not None:  save\_dir = args.evaluate\_dir  else:  model\_name = args.model.replace("/","-")  gpu\_count = torch.cuda.device\_count()  save\_dir = f"{args.output\_dir}/{args.user\_msg}\_{model\_name}\_{args.img\_type}\_{args.prompt\_format}\_lr{args.lr}\_bs{args.bs \* gpu\_count}\_op{args.output\_len}\_ep{args.epoch}"  if not os.path.exists(save\_dir):  os.mkdir(save\_dir)  print(save\_dir)  if args.img\_type is not None:  patch\_size = img\_shape[args.img\_type]  model = T5ForMultimodalGeneration.from\_pretrained(args.model, patch\_size=patch\_size)  name\_maps = dataframe['name\_maps']  image\_features = dataframe['image\_features']  train\_set = ScienceQADatasetImg(  problems,  train\_qids,  name\_maps,  tokenizer,  args.input\_len,  args.output\_len,  args,  image\_features,  )  eval\_set = ScienceQADatasetImg(  problems,  val\_qids,  name\_maps,  tokenizer,  args.input\_len,  args.output\_len,  args,  image\_features,  args.eval\_le,  )  test\_set = ScienceQADatasetImg(  problems,  test\_qids,  name\_maps,  tokenizer,  args.input\_len,  args.output\_len,  args,  image\_features,  args.test\_le,  )  else:  model = T5ForConditionalGeneration.from\_pretrained(args.model)  train\_set = ScienceQADatasetStd(  problems,  train\_qids,  tokenizer,  args.input\_len,  args.output\_len,  args,  )  eval\_set = ScienceQADatasetStd(  problems,  val\_qids,  tokenizer,  args.input\_len,  args.output\_len,  args,  args.eval\_le,  )    test\_set = ScienceQADatasetStd(  problems,  test\_qids,  tokenizer,  args.input\_len,  args.output\_len,  args,  args.test\_le,  )  datacollator = DataCollatorForSeq2Seq(tokenizer)  print("model parameters: ", model.num\_parameters())  def extract\_ans(ans):  pattern = re.compile(r'The answer is \(([A-Z])\)')  res = pattern.findall(ans)    if len(res) == 1:  answer = res[0] # 'A', 'B', ...  else:  answer = "FAILED"  return answer  # accuracy for answer inference  def compute\_metrics\_acc(eval\_preds):  if args.use\_generate:  preds, targets = eval\_preds  if isinstance(preds, tuple):  preds = preds[0]  else:  preds = eval\_preds.predictions[0]  targets = eval\_preds.label\_ids  preds = preds.argmax(axis=2)  preds = tokenizer.batch\_decode(preds, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True)  targets = tokenizer.batch\_decode(targets, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True)  correct = 0  assert len(preds) == len(targets)  for idx, pred in enumerate(preds):  reference = targets[idx]  reference = extract\_ans(reference)  extract\_pred = extract\_ans(pred)  best\_option = extract\_pred  if reference == best\_option:  correct +=1  return {'accuracy': 1.0\*correct/len(targets)}    # rougel for rationale generation  metric = evaluate.load("rouge")  def postprocess\_text(preds, labels):  preds = [pred.strip() for pred in preds]  labels = [label.strip() for label in labels]  preds = ["\n".join(nltk.sent\_tokenize(pred)) for pred in preds]  labels = ["\n".join(nltk.sent\_tokenize(label)) for label in labels]  return preds, labels  def compute\_metrics\_rougel(eval\_preds):  if args.use\_generate:  preds, targets = eval\_preds  if isinstance(preds, tuple):  preds = preds[0]  else:  preds = eval\_preds.predictions[0]  targets = eval\_preds.label\_ids  preds = preds.argmax(axis=2)  preds = tokenizer.batch\_decode(preds, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True)  targets = tokenizer.batch\_decode(targets, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True)  decoded\_preds, decoded\_labels = postprocess\_text(preds, targets)  result = metric.compute(predictions=decoded\_preds, references=decoded\_labels, use\_stemmer=True)  result = {k: round(v \* 100, 4) for k, v in result.items()}  prediction\_lens = [np.count\_nonzero(pred != tokenizer.pad\_token\_id) for pred in preds]  result["gen\_len"] = np.mean(prediction\_lens)  return result  # only use the last model for evaluation to save time  if args.final\_eval:  training\_args = Seq2SeqTrainingArguments(  save\_dir,  do\_train=True if args.evaluate\_dir is None else False,  do\_eval=False,  evaluation\_strategy="no",  logging\_strategy="steps",  save\_strategy="epoch",  save\_total\_limit = 2,  learning\_rate= args.lr,  eval\_accumulation\_steps=args.eval\_acc,  per\_device\_train\_batch\_size=args.bs,  per\_device\_eval\_batch\_size=args.eval\_bs,  weight\_decay=0.01,  num\_train\_epochs=args.epoch,  predict\_with\_generate=args.use\_generate,  generation\_max\_length=args.output\_len,  report\_to="none",  )  # evaluate at each epoch  else:  training\_args = Seq2SeqTrainingArguments(  save\_dir,  do\_train=True if args.evaluate\_dir is None else False,  do\_eval=True,  evaluation\_strategy="epoch",  logging\_strategy="steps",  save\_strategy="epoch",  save\_total\_limit = 2,  learning\_rate= args.lr,  eval\_accumulation\_steps=args.eval\_acc,  per\_device\_train\_batch\_size=args.bs,  per\_device\_eval\_batch\_size=args.eval\_bs,  weight\_decay=0.01,  num\_train\_epochs=args.epoch,  metric\_for\_best\_model="accuracy" if args.prompt\_format == "QCMG-A" or args.prompt\_format == "QCM-A" else "rougeL",  predict\_with\_generate=args.use\_generate,  generation\_max\_length=args.output\_len,  load\_best\_model\_at\_end=True,  report\_to="none",  )  trainer = Seq2SeqTrainer(  model=model,  args=training\_args,  train\_dataset=train\_set,  eval\_dataset=eval\_set,  data\_collator=datacollator,  tokenizer=tokenizer,  compute\_metrics = compute\_metrics\_acc if args.prompt\_format == "QCMG-A" or args.prompt\_format == "QCM-A" else compute\_metrics\_rougel  )  if args.evaluate\_dir is None:  trainer.train()  trainer.save\_model(save\_dir)    metrics = trainer.evaluate(eval\_dataset = test\_set, max\_length=args.output\_len)  trainer.log\_metrics("test", metrics)  trainer.save\_metrics("test", metrics)  predict\_results = trainer.predict(test\_dataset=test\_set, max\_length=args.output\_len)  if trainer.is\_world\_process\_zero():  if args.use\_generate:  preds, targets = predict\_results.predictions, predict\_results.label\_ids  else:  preds = predict\_results.predictions[0]  targets = predict\_results.label\_ids  preds = preds.argmax(axis=2)  preds = tokenizer.batch\_decode(  preds, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True  )  targets = tokenizer.batch\_decode(  targets, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True  )  results\_ans = {}  results\_rationale = {}  results\_reference = {}    num\_fail = 0  for idx, qid in enumerate(test\_qids):  pred = preds[int(idx)]  ref = targets[int(idx)]  extract\_pred = extract\_ans(pred)  if extract\_pred != "FAILED":  if extract\_pred in args.options:  extract\_pred = args.options.index(extract\_pred)  else:  extract\_pred = random.choice(range(0,len(args.options)))  else:  num\_fail += 1  extract\_pred = random.choice(range(len(args.options))) # random choose one option  results\_ans[str(qid)] = extract\_pred  results\_rationale[str(qid)] = pred  results\_reference[str(qid)] = ref  scores = get\_scores(results\_ans, results\_rationale, results\_reference, os.path.join(args.data\_root, "scienceqa/problems.json"))  preds = [pred.strip() for pred in preds]  output\_data = {  "num\_fail": num\_fail,  "scores": scores,  "preds": preds,  "labels": targets}  output\_prediction\_file = os.path.join(save\_dir,"predictions\_ans\_test.json")  with open(output\_prediction\_file, "w") as writer:  writer.write(json.dumps(output\_data, indent=4))    # generate the rationale for the eval set  if args.prompt\_format == "QCM-LE" or args.prompt\_format == "QCM-E":  torch.cuda.empty\_cache()  del predict\_results, preds, targets  predict\_results = trainer.predict(test\_dataset=eval\_set, max\_length=args.output\_len)  if trainer.is\_world\_process\_zero():  if args.use\_generate:  preds, targets = predict\_results.predictions, predict\_results.label\_ids  else:  preds = predict\_results.predictions[0]  targets = predict\_results.label\_ids  preds = preds.argmax(axis=2)  preds = tokenizer.batch\_decode(  preds, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True  )  targets = tokenizer.batch\_decode(  targets, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True  )  preds = [pred.strip() for pred in preds]  output\_data = {"preds": preds,  "labels": targets}  output\_prediction\_file = os.path.join(save\_dir,"predictions\_ans\_eval.json")  with open(output\_prediction\_file, "w") as writer:  writer.write(json.dumps(output\_data, indent=4))    if \_\_name\_\_ == '\_\_main\_\_':  # training logger to log training progress  training\_logger = Table(  Column("Epoch", justify="center"),  Column("Steps", justify="center"),  Column("Loss", justify="center"),  title="Training Status",  pad\_edge=False,  box=box.ASCII,  )    args = parse\_args()  print("args",args)  print('====Input Arguments====')  print(json.dumps(vars(args), indent=2, sort\_keys=False))  random.seed(args.seed)    if not os.path.exists(args.output\_dir):  os.mkdir(args.output\_dir)  if args.img\_type is not None:  problems, qids, name\_maps, image\_features = load\_data\_img(args) # probelms, test question ids, shot example ids  dataframe = {'problems':problems, 'qids':qids, 'name\_maps': name\_maps, 'image\_features': image\_features}  else:  problems, qids = load\_data\_std(args) # probelms, test question ids, shot example ids  dataframe = {'problems':problems, 'qids':qids}  T5Trainer(  dataframe=dataframe,  args = args  ) |