毕业设计真实性承诺及指导教师声明

学生毕业设计真实性承诺

本人郑重声明:所提交的毕业设计是本人在指导教师的指导下，独立进行研究工作所取得的成果，内容真实可靠，不存在抄袭、造假等学术不端行为。除设计方案中已经注明引用的内容外，本设计不含其他个人或集体已经发表或撰写过的研究成果。对本设计的研究做出重要贡献的个人和集体，均已在设计文档以明确方式标明。如被发现设计中存在抄袭、造假等学术不端行为，本人愿承担相应的法律责任和一切后果。

学生(签名): 年 月 日

**指导教师关于学生毕业设计真实性审核的声明**

本人郑重声明:已经对学生毕业设计所涉及的内容进行严格审核，确定其成果均由学生在本人指导下取得，对他人设计方案及成果的引用已经明确注明，不存在抄袭、造假等学术不端行为。

指导教师(签名)： 年 月 日

目录

[学生毕业设计真实性承诺 1](#_Toc2111)

[指导教师关于学生毕业设计真实性审核的声明 1](#_Toc1329)

[1. 设计任务 2](#_Toc5901)

[2.1 需求分析 2](#_Toc7963)

[2.1经济的可行性 3](#_Toc14817)

[2.2 技术的可行性 3](#_Toc12146)

[2.3系统功能分析 4](#_Toc22724)

[2.4 功能模块需求分析 4](#_Toc27563)

[3 技术方案 4](#_Toc16274)

[3.1数据库 4](#_Toc12924)

[3 设计内容（过程）与说明 5](#_Toc4234)

[3.1 数据库索引技术 5](#_Toc2537)

[3.2 前端网页实现 7](#_Toc31588)

[2. 用户登录，方便统计每个用户咨询的问题，也方便投放广告。 7](#_Toc1439)

[3.3系统实现 7](#_Toc9373)

[4.0设计总结 7](#_Toc16771)

1. 设计任务

1.1设计背景

问答系统，旨在为用户提供及时、准确的答案，以满足用户的需求。它可以自动识别用户的问题，并从数据库中检索出最佳答案。设计主要包括前端页面，用于接收用户输入的问题，并将答案显示在页面上。

1.2设计目标

1. 提供一个简单易用的前端页面，便于用户输入问题并获取答案。
2. 提供一个安全可靠的系统，保证用户的隐私安全。
3. 提供一个高效可靠的系统，能够快速准确地检索出最佳答案。

1.3设计内容

1. 前端页面应该简洁明了，操作简单，便于用户输入问题。
2. 系统应该能够安全地存储用户的问题和答案，保证用户的隐私安全。
3. 系统应该能够快速准确地检索出最佳答案，以满足用户的需求。

2.设计思路与技术方案

1. 前端页面设计：使用HTML、CSS和JavaScript等前端技术，设计一个简洁明了的页面，便于用户输入问题。
2. 数据库设计：使用MySQL等关系型数据库，存储用户的问题和答案，保证数据的安全性。
3. 检索算法：使用机器学习技术，设计一个检索算法，能够快速准确地检索出最佳答案。

2.1 需求分析

智能问答系统是一种计算机软件，它可以帮助用户解决问题。它的目的是为用户提供快速准确的答案，以满足用户的需求。

在开发智能问答系统之前，需要进行需求分析，以确定系统的功能和性能。首先，要确定系统的用户群体，以及他们的需求。其次，要确定系统的功能，包括收集问题、搜索答案、提供答案等。此外，还需要考虑系统的安全性、可靠性、可扩展性等性能。

最后，要考虑系统的可用性，以确保用户能够轻松访问系统，并获得准确的答案。这需要系统提供易于使用的界面，以及可以满足用户需求的功能。

因此，智能问答系统的需求分析是一项重要的工作，它可以帮助开发者确定系统的功能和性能，以满足用户的需求。

2.1经济的可行性  
 智能问答系统是一种基于计算机技术的自动回答系统，它可以提供快速、准确的回答，从而满足用户的需求。随着技术的发展，智能问答系统已经成为一种普遍的服务，它可以帮助企业提高服务质量，提升客户满意度。 那么，智能问答系统的经济可行性如何？从经济角度来看，智能问答系统的实施可以带来一定的经济效益。首先，智能问答系统可以提高企业的服务质量，提升客户满意度，从而提高企业的市场份额和盈利能力。其次，智能问答系统可以减少企业的人力成本，因为它可以自动处理大量的查询请求，从而节省企业的人力资源。最后，智能问答系统可以帮助企业改善服务流程，提高服务质量，从而降低企业的服务成本。 总之，智能问答系统的经济可行性是可行的。它可以帮助企业提高服务质量，提升客户满意度，减少企业的人力成本，改善服务流程，降低企业的服务成本，从而带来一定的经济效益。

2.2 技术的可行性

随着人工智能技术的发展，智能问答系统已经成为了一种受到广泛关注的技术。它可以提供快速的、准确的答案，从而满足用户的查询需求。智能问答系统的技术可行性是一个值得探讨的话题。

首先，智能问答系统的技术可行性取决于其能够提供准确的答案。这可以通过训练机器学习模型来实现，机器学习模型可以从大量的语料库中学习，从而更好地理解用户的查询内容，并返回准确的答案。

其次，智能问答系统的技术可行性还取决于其能够提供快速的答案。这可以通过构建高效的搜索引擎来实现，搜索引擎可以从大量的数据库中快速检索出相关的答案，从而满足用户的查询需求。

最后，智能问答系统的技术可行性还取决于其能够提供可信赖的答案。这可以通过设计可靠的系统架构来实现，系统架构可以确保答案的准确性和可靠性，从而满足用户的查询需求。

总的来说，智能问答系统的技术可行性是一个值得探讨的话题。它可以通过训练机器学习模型、构建高效的搜索引擎以及设计可靠的系统架构来实现，从而满足用户的查询需求。

2.3系统功能分析

智能问答系统是一种自动回答用户提出的问题的计算机系统。它可以根据用户问题的内容，自动搜索知识库，提取相关信息，并进行语义分析，最终给出精准的答案。

智能问答系统的核心功能包括：

1. 知识库搜索：智能问答系统可以根据用户提出的问题，自动搜索知识库，从而获取相关信息。
2. 语义分析：智能问答系统可以根据用户提出的问题，进行语义分析，从而提取出有用的信息，并生成精准的答案。
3. 答案生成：智能问答系统可以根据用户提出的问题，结合知识库搜索和语义分析，最终生成精准的答案。
4. 问题分类：智能问答系统可以根据用户提出的问题，对问题进行分类，从而更好地搜索知识库，提取有用的信息，生成精准的答案。 智能问答系统的功能可以极大地提高用户的查询效率，节省用户的时间，提高用户的满意度。

2.4 功能模块需求分析

智能系统可以根据用户的问答自动的回答问题，它可以帮助用户快速获得所需的信息。智能问答系统的功能模块需求分析是构建一个高效的智能问答系统的重要环节。

首先，智能问答系统需要一个知识库，用于存储系统所需的知识，以便系统能够根据用户提出的问题，快速检索出答案。其次，智能问答系统需要一个自然语言处理模块，用于对用户提出的问题进行语义分析，以便系统能够准确检索出答案。此外，智能问答系统还需要一个智能推理模块，用于帮助系统解决复杂的问题，以及一个结果展示模块，用于将检索到的答案以友好的方式展示给用户。

总之，智能问答系统的功能模块需求分析是构建一个高效的智能问答系统的重要环节，它需要知识库、自然语言处理模块、智能推理模块和结果展示模块等模块的支持。

3 技术方案

3.1数据库

智能问答系统是一种基于机器学习、自然语言处理和数据挖掘等技术的智能化应用，目的是能够在较高准确率的前提下，自动回答用户的问题。

在问答系统数据的处理方面，一般使用的是json格式的数据库。在该数据库中，可以将问题和答案按照一定的格式排列好。如下所示：

|  |
| --- |
| {  "1": {  "question": "您好，我该怎么做才能开通手机银行？",  "answer": "您可以在手机上下载并安装我们银行的手机App，然后按照APP的提示进行绑定设置即可。"  },  "2": {  "question": "请问您们银行支持哪些支付方式？",  "answer": "我们支持支付宝、微信、银联等多种支付方式，您可以根据自己的需求选择。"  },  "3": {  "question": "请问个人网银需要办理手续吗？",  "answer": "需要，您可以到我行任意一家分行携带您的有效身份证件办理申请。"  }  } |

在用户提问方面，我们需要使用自然语言处理技术进行处理。一般采用的是分词技术，将句子分解成一个一个的词汇，然后再与问题库中的所有问题进行比对，计算相似度得分，得分最高的问题对应的答案即是最终的答案。

在答案生成方面，采用的是模板匹配和规则匹配的方式。具体来说，针对一些常见问题，我们可以提前编制好固定的答案模板，如下所示：

|  |
| --- |
| {  "1": {  "type": "绑定手机银行",  "template": "您可以在手机上下载并安装我们银行的手机App，然后按照APP的提示进行绑定设置即可。"  },  "2": {  "type": "支付方式",  "template": "我们支持支付宝、微信、银联等多种支付方式，您可以根据自己的需求选择。"  },  "3": {  "type": "个人网银",  "template": "需要，您可以到我行任意一家分行携带您的有效身份证件办理申请。"  }  } |

在实际应用过程中，通过分析用户提问的问题类型和问题关键词，我们就可以匹配出相应的模板，然后根据模板生成对应的答案。

为了进一步提高问答系统的准确性和智能化水平，我们还可以引入机器学习等人工智能技术。通过训练数据集和算法模型，我们可以使问答系统更加贴近实际应用场景，提高问答的准确性和智能化程度，为用户提供更加便捷高效的服务。

3 设计内容（过程）与说明

* 1. PTuning技术

PTuning是一种深度学习中的技术，它主要是用于解决在跨语言文本分类和命名实体识别任务中遇到的一些困难。PTuning 基于模型预训练的思想，通过对源语言数据和目标语言数据进行预训练来提高模型的泛化能力。在 PTuning 中，源语言预训练模型可以视为一种"老师"，用于指导目标语言预训练模型的学习。经过预训练之后，目标语言预训练模型可以更好地进行语言推理和跨语言任务处理。

PTuning是一种深度学习技术，主要用于解决跨语言文本分类和命名实体识别任务中遇到的问题。跨语言任务通常意味着输入数据可能来自不同语言的数据源，例如在中文领域训练的模型需要应用到英文的语料上，这时就需要解决跨语言的问题。PTuning是一种基于预训练技术的解决方案。

在深度学习中，预训练技术被广泛使用。预训练是指在一个大型数据集上进行的无监督学习，目的是让模型学习到更通用的特征。在深度学习中，通常有两种预训练模式：自监督学习和无监督学习。

PTuning是一种基于模型预训练的在线模型自适应技术，它利用源语言数据对目标任务的预训练模型进行更新，从而提高模型的泛化能力。在PTuning中，源语言预训练模型可以看作是一种“老师”，用于指导目标语言预训练模型的学习。PTuning大致包含以下步骤：

1. 源语言预训练：在源语言数据集上进行预训练，获得一个预训练模型。

2. 目标语言预训练：在目标语言数据集上进行预训练，用源语言预训练模型作为参数初始化。

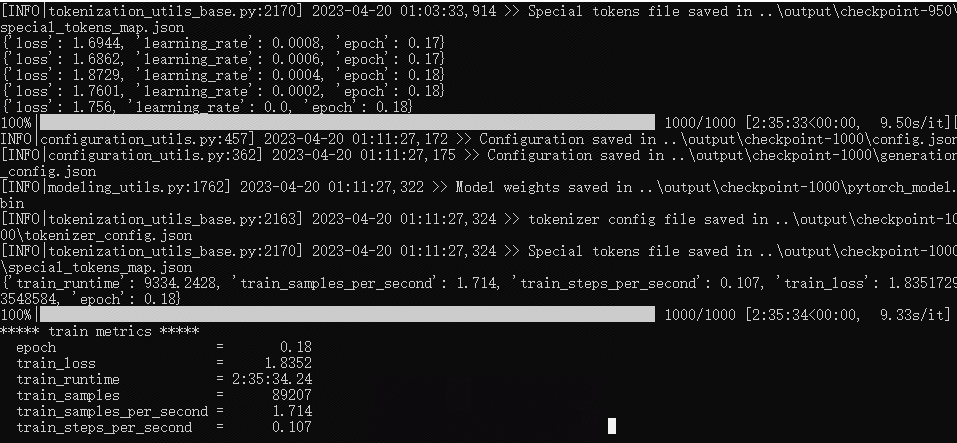
3. PTuning更新：在目标语言数据集上进行训练，同时利用source和target数据对目标任务的预训练模型进行更新。

4. Fine-tuning：在目标任务上进行微调。

与传统的跨语言学习方法不同，PTuning 在源语言和目标语言之间建立了桥梁，利用预训练模型进行参数传递；同时，PTuning还使用了目标语言数据对预训练模型进行修正，强调了模型在目标任务上的适应性。

总的来说，PTuning 是一种有希望实现跨语言学习技术的较新方法，利用了深度学习技术中的预训练方法来提高模型泛化能力和适应性，具有很好的应用价值。

3.2 试验过程



|  |
| --- |
| import logging  import os  import sys  import json  import numpy as np  from datasets import load\_dataset  import jieba  from rouge\_chinese import Rouge  from nltk.translate.bleu\_score import sentence\_bleu, SmoothingFunction  import torch  import transformers # 导入transformers库，这是一个自然语言处理库  from transformers import ( # 导入transformers库中的一些工具  AutoConfig, # 用于自动配置模型  AutoModel, # 用于自动加载预训练模型  AutoTokenizer, # 用于自动加载预训练分词器  AutoTokenizer, # 用于自动加载预训练分词器  DataCollatorForSeq2Seq, # 用于定义数据读取与处理方式  HfArgumentParser, # 用于解析命令行参数  Seq2SeqTrainingArguments, # 定义Seq2Seq模型的训练参数  set\_seed, # 用于设置随机种子  )  def main():  parser = HfArgumentParser((ModelArguments, DataTrainingArguments, Seq2SeqTrainingArguments))  if len(sys.argv) == 2 and sys.argv[1].endswith(".json"):  #如果我们只向脚本传递一个参数，而且它是指向一个json文件的路径，那么我们可以解析它以获取我们的参数。  model\_args, data\_args, training\_args = parser.parse\_json\_file(json\_file=os.path.abspath(sys.argv[1]))  else:  model\_args, data\_args, training\_args = parser.parse\_args\_into\_dataclasses()  # 设置日志记录  logging.basicConfig(  format="%(asctime)s - %(levelname)s - %(name)s - %(message)s",  datefmt="%m/%d/%Y %H:%M:%S",  handlers=[logging.StreamHandler(sys.stdout)],  )  if training\_args.should\_log:  # training\_args.log\_level 的默认设置是被动的，因此我们在此处将日志级别设为 info 以使用该默认设置。  transformers.utils.logging.set\_verbosity\_info()  log\_level = training\_args.get\_process\_log\_level()  logger.setLevel(log\_level)  # datasets.utils.logging.set\_verbosity(log\_level)  transformers.utils.logging.set\_verbosity(log\_level)  transformers.utils.logging.enable\_default\_handler()  transformers.utils.logging.enable\_explicit\_format()  # 对于每个进程记录小总结：  logger.warning(  f"Process rank: {training\_args.local\_rank}, device: {training\_args.device}, n\_gpu: {training\_args.n\_gpu}"  + f"distributed training: {bool(training\_args.local\_rank != -1)}, 16-bits training: {training\_args.fp16}"  )  logger.info(f"Training/evaluation parameters {training\_args}")  # 在初始化模型之前设置种子。  set\_seed(training\_args.seed)  # 加载数据集  data\_files = {}  if data\_args.train\_file is not None:  data\_files["train"] = data\_args.train\_file  extension = data\_args.train\_file.split(".")[-1]  if data\_args.validation\_file is not None:  data\_files["validation"] = data\_args.validation\_file  extension = data\_args.validation\_file.split(".")[-1]  if data\_args.test\_file is not None:  data\_files["test"] = data\_args.test\_file  extension = data\_args.test\_file.split(".")[-1]  raw\_datasets = load\_dataset(  extension,  data\_files=data\_files,  cache\_dir=model\_args.cache\_dir,  use\_auth\_token=True if model\_args.use\_auth\_token else None,  )  # 加载预先训练好的模型和分词器  config = AutoConfig.from\_pretrained(model\_args.model\_name\_or\_path, trust\_remote\_code=True)  config.pre\_seq\_len = model\_args.pre\_seq\_len  config.prefix\_projection = model\_args.prefix\_projection  tokenizer = AutoTokenizer.from\_pretrained(model\_args.model\_name\_or\_path, trust\_remote\_code=True)  if model\_args.ptuning\_checkpoint is not None:  # 评估  # 加载前缀编码器的额外状态字典  model = AutoModel.from\_pretrained(model\_args.model\_name\_or\_path, config=config, trust\_remote\_code=True)  prefix\_state\_dict = torch.load(os.path.join(model\_args.ptuning\_checkpoint, "pytorch\_model.bin"))  new\_prefix\_state\_dict = {}  for k, v in prefix\_state\_dict.items():  new\_prefix\_state\_dict[k[len("transformer.prefix\_encoder."):]] = v  model.transformer.prefix\_encoder.load\_state\_dict(new\_prefix\_state\_dict)  else:  model = AutoModel.from\_pretrained(model\_args.model\_name\_or\_path, config=config, trust\_remote\_code=True)  if model\_args.quantization\_bit is not None:  print(f"Quantized to {model\_args.quantization\_bit} bit")  model = model.quantize(model\_args.quantization\_bit)  if model\_args.pre\_seq\_len is not None:  # P-tuning v2  model = model.half()  model.transformer.prefix\_encoder.float()  else:  # Finetune  model = model.float()  prefix = data\_args.source\_prefix if data\_args.source\_prefix is not None else ""  # 预处理数据集。  # 我们需要标记输入和目标。  if training\_args.do\_train:  column\_names = raw\_datasets["train"].column\_names  elif training\_args.do\_eval:  column\_names = raw\_datasets["validation"].column\_names  elif training\_args.do\_predict:  column\_names = raw\_datasets["test"].column\_names  else:  logger.info("There is nothing to do. Please pass `do\_train`, `do\_eval` and/or `do\_predict`.")  return  # 获取输入/目标的列名。  prompt\_column = data\_args.prompt\_column  response\_column = data\_args.response\_column  history\_column = data\_args.history\_column    # 临时设置训练max\_target\_length。  max\_target\_length = data\_args.max\_target\_length  def preprocess\_function\_eval(examples):  inputs, targets = [], []  for i in range(len(examples[prompt\_column])):  if examples[prompt\_column][i] and examples[response\_column][i]:  query = examples[prompt\_column][i]  if history\_column is None or len(examples[history\_column][i]) == 0:  prompt = query  else:  prompt = ""  history = examples[history\_column][i]  for i, (old\_query, response) in enumerate(history):  prompt += "[Round {}]\n问：{}\n答：{}\n".format(i, old\_query, response)  prompt += "[Round {}]\n问：{}\n答：".format(len(history), query)  inputs.append(prompt)  targets.append(examples[response\_column][i])  inputs = [prefix + inp for inp in inputs]  model\_inputs = tokenizer(inputs, max\_length=data\_args.max\_source\_length, truncation=True, padding=True)  labels = tokenizer(text\_target=targets, max\_length=max\_target\_length, truncation=True)  if data\_args.ignore\_pad\_token\_for\_loss:  labels["input\_ids"] = [  [(l if l != tokenizer.pad\_token\_id else -100) for l in label] for label in labels["input\_ids"]  ]  model\_inputs["labels"] = labels["input\_ids"]  return model\_inputs  def preprocess\_function\_train(examples):  max\_seq\_length = data\_args.max\_source\_length + data\_args.max\_target\_length  model\_inputs = {  "input\_ids": [],  "labels": [],  }  for i in range(len(examples[prompt\_column])):  if examples[prompt\_column][i] and examples[response\_column][i]:  query, answer = examples[prompt\_column][i], examples[response\_column][i]  if history\_column is None:  prompt = query  else:  prompt = ""  history = examples[history\_column][i]  for i, (old\_query, response) in enumerate(history):  prompt += "[Round {}]\n问：{}\n答：{}\n".format(i, old\_query, response)  prompt += "[Round {}]\n问：{}\n答：".format(len(history), query)  prompt = prefix + prompt  a\_ids = tokenizer.encode(text=prompt, add\_special\_tokens=False)  b\_ids = tokenizer.encode(text=answer, add\_special\_tokens=False)  if len(a\_ids) > data\_args.max\_source\_length - 1:  a\_ids = a\_ids[: data\_args.max\_source\_length - 1]  if len(b\_ids) > data\_args.max\_target\_length - 2:  b\_ids = b\_ids[: data\_args.max\_target\_length - 2]  input\_ids = tokenizer.build\_inputs\_with\_special\_tokens(a\_ids, b\_ids)  context\_length = input\_ids.index(tokenizer.bos\_token\_id)  mask\_position = context\_length - 1  labels = [-100] \* context\_length + input\_ids[mask\_position+1:]    pad\_len = max\_seq\_length - len(input\_ids)  input\_ids = input\_ids + [tokenizer.pad\_token\_id] \* pad\_len  labels = labels + [tokenizer.pad\_token\_id] \* pad\_len  if data\_args.ignore\_pad\_token\_for\_loss:  labels = [(l if l != tokenizer.pad\_token\_id else -100) for l in labels]  model\_inputs["input\_ids"].append(input\_ids)  model\_inputs["labels"].append(labels)  return model\_inputs    def print\_dataset\_example(example):  print("input\_ids",example["input\_ids"])  print("inputs", tokenizer.decode(example["input\_ids"]))  print("label\_ids", example["labels"])  print("labels", tokenizer.decode(example["labels"]))  if training\_args.do\_train:  if "train" not in raw\_datasets:  raise ValueError("--do\_train requires a train dataset")  train\_dataset = raw\_datasets["train"]  if data\_args.max\_train\_samples is not None:  max\_train\_samples = min(len(train\_dataset), data\_args.max\_train\_samples)  train\_dataset = train\_dataset.select(range(max\_train\_samples))  with training\_args.main\_process\_first(desc="train dataset map pre-processing"):  train\_dataset = train\_dataset.map(  preprocess\_function\_train,  batched=True,  num\_proc=data\_args.preprocessing\_num\_workers,  remove\_columns=column\_names,  load\_from\_cache\_file=not data\_args.overwrite\_cache,  desc="Running tokenizer on train dataset",  )  print\_dataset\_example(train\_dataset[0])  if training\_args.do\_eval:  max\_target\_length = data\_args.val\_max\_target\_length  if "validation" not in raw\_datasets:  raise ValueError("--do\_eval requires a validation dataset")  eval\_dataset = raw\_datasets["validation"]  if data\_args.max\_eval\_samples is not None:  max\_eval\_samples = min(len(eval\_dataset), data\_args.max\_eval\_samples)  eval\_dataset = eval\_dataset.select(range(max\_eval\_samples))  with training\_args.main\_process\_first(desc="validation dataset map pre-processing"):  eval\_dataset = eval\_dataset.map(  preprocess\_function\_eval,  batched=True,  num\_proc=data\_args.preprocessing\_num\_workers,  remove\_columns=column\_names,  load\_from\_cache\_file=not data\_args.overwrite\_cache,  desc="Running tokenizer on validation dataset",  )  print\_dataset\_example(eval\_dataset[0])  if training\_args.do\_predict:  max\_target\_length = data\_args.val\_max\_target\_length  if "test" not in raw\_datasets:  raise ValueError("--do\_predict requires a test dataset")  predict\_dataset = raw\_datasets["test"]  if data\_args.max\_predict\_samples is not None:  max\_predict\_samples = min(len(predict\_dataset), data\_args.max\_predict\_samples)  predict\_dataset = predict\_dataset.select(range(max\_predict\_samples))  with training\_args.main\_process\_first(desc="prediction dataset map pre-processing"):  predict\_dataset = predict\_dataset.map(  preprocess\_function\_eval,  batched=True,  num\_proc=data\_args.preprocessing\_num\_workers,  remove\_columns=column\_names,  load\_from\_cache\_file=not data\_args.overwrite\_cache,  desc="Running tokenizer on prediction dataset",  )  print\_dataset\_example(predict\_dataset[0])  # Data collator  label\_pad\_token\_id = -100 if data\_args.ignore\_pad\_token\_for\_loss else tokenizer.pad\_token\_id  data\_collator = DataCollatorForSeq2Seq(  tokenizer,  model=model,  label\_pad\_token\_id=label\_pad\_token\_id,  pad\_to\_multiple\_of=None,  padding=False  )  # Metric  def compute\_metrics(eval\_preds):  preds, labels = eval\_preds  if isinstance(preds, tuple):  preds = preds[0]  decoded\_preds = tokenizer.batch\_decode(preds, skip\_special\_tokens=True)  if data\_args.ignore\_pad\_token\_for\_loss:  # 替换标签中的 -100，因为我们无法解码它们。  labels = np.where(labels != -100, labels, tokenizer.pad\_token\_id)  decoded\_labels = tokenizer.batch\_decode(labels, skip\_special\_tokens=True)  score\_dict = {  "rouge-1": [],  "rouge-2": [],  "rouge-l": [],  "bleu-4": []  }  for pred, label in zip(decoded\_preds, decoded\_labels):  hypothesis = list(jieba.cut(pred))  reference = list(jieba.cut(label))  rouge = Rouge()  scores = rouge.get\_scores(' '.join(hypothesis) , ' '.join(reference))  result = scores[0]    for k, v in result.items():  score\_dict[k].append(round(v["f"] \* 100, 4))  bleu\_score = sentence\_bleu([list(label)], list(pred), smoothing\_function=SmoothingFunction().method3)  score\_dict["bleu-4"].append(round(bleu\_score \* 100, 4))  for k, v in score\_dict.items():  score\_dict[k] = float(np.mean(v))  return score\_dict  # 覆盖 Seq2SeqTrainer 的解码参数  training\_args.generation\_max\_length = (  training\_args.generation\_max\_length  if training\_args.generation\_max\_length is not None  else data\_args.val\_max\_target\_length  )  training\_args.generation\_num\_beams = (  data\_args.num\_beams if data\_args.num\_beams is not None else training\_args.generation\_num\_beams  )  # 初始化我们的训练器  trainer = Seq2SeqTrainer(  model=model,  args=training\_args,  train\_dataset=train\_dataset if training\_args.do\_train else None,  eval\_dataset=eval\_dataset if training\_args.do\_eval else None,  tokenizer=tokenizer,  data\_collator=data\_collator,  compute\_metrics=compute\_metrics if training\_args.predict\_with\_generate else None,  )  # Training  if training\_args.do\_train:  checkpoint = None  if training\_args.resume\_from\_checkpoint is not None:  checkpoint = training\_args.resume\_from\_checkpoint  # elif last\_checkpoint is not None:  # checkpoint = last\_checkpoint  model.gradient\_checkpointing\_enable()  model.enable\_input\_require\_grads()  train\_result = trainer.train(resume\_from\_checkpoint=checkpoint)  # trainer.save\_model() # Saves the tokenizer too for easy upload  metrics = train\_result.metrics  max\_train\_samples = (  data\_args.max\_train\_samples if data\_args.max\_train\_samples is not None else len(train\_dataset)  )  metrics["train\_samples"] = min(max\_train\_samples, len(train\_dataset))  trainer.log\_metrics("train", metrics)  trainer.save\_metrics("train", metrics)  trainer.save\_state()  # Evaluation  results = {}  if training\_args.do\_eval:  logger.info("\*\*\* Evaluate \*\*\*")  metrics = trainer.evaluate(metric\_key\_prefix="eval", do\_sample=True, top\_p=0.7, max\_length=512, temperature=0.95)  max\_eval\_samples = data\_args.max\_eval\_samples if data\_args.max\_eval\_samples is not None else len(eval\_dataset)  metrics["eval\_samples"] = min(max\_eval\_samples, len(eval\_dataset))  trainer.log\_metrics("eval", metrics)  trainer.save\_metrics("eval", metrics)  if training\_args.do\_predict:  logger.info("\*\*\* Predict \*\*\*")  predict\_results = trainer.predict(predict\_dataset, metric\_key\_prefix="predict", max\_length=512, do\_sample=True, top\_p=0.7, temperature=0.95)  metrics = predict\_results.metrics  max\_predict\_samples = (  data\_args.max\_predict\_samples if data\_args.max\_predict\_samples is not None else len(predict\_dataset)  )  metrics["predict\_samples"] = min(max\_predict\_samples, len(predict\_dataset))  trainer.log\_metrics("predict", metrics)  trainer.save\_metrics("predict", metrics)  if trainer.is\_world\_process\_zero():  if training\_args.predict\_with\_generate:  predictions = tokenizer.batch\_decode(  predict\_results.predictions, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True  )  predictions = [pred.strip() for pred in predictions]  labels = tokenizer.batch\_decode(  predict\_results.label\_ids, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True  )  labels = [label.strip() for label in labels]  output\_prediction\_file = os.path.join(training\_args.output\_dir, "generated\_predictions.txt")  with open(output\_prediction\_file, "w", encoding="utf-8") as writer:  for p, l in zip(predictions, labels):  res = json.dumps({"labels": l, "predict": p}, ensure\_ascii=False)  writer.write(f"{res}\n")  return results |

* + 1. 数据库的模型设计

智能问答系统的数据库模型是一种抽象的数据库模型，它是一种抽象的数据库模型，它可以描述智能问答系统中的数据结构和数据关系，以及数据的存储和检索方式。

* + 1. 数据库表结构设计

智能问答系统的数据库模型由三个部分组成：数据结构、数据关系和数据存储和检索方式。

（1）数据结构：数据结构是指智能问答系统中的数据的组织形式，它可以描述数据的类型、结构和属性。

（2）数据关系：数据关系是指智能问答系统中的数据之间的关系，它可以描述数据之间的联系，以及数据之间的关联性。

1. 数据存储和检索方式：数据存储和检索方式是指智能问答系统中的数据存储和检索方式，它可以描述数据的存储方式，以及数据的检索方式。

3.1.3模型设计的原则

智能问答系统的数据库模型的设计原则包括：

（1）简单性原则：数据库模型的设计应该尽可能简单，以便于理解和维护。

（2）完整性原则：数据库模型的设计应该尽可能完整，以便于满足用户的需求。

（3）可扩展性原则：数据库模型的设计应该尽可能可扩展，以便于满足用户的未来需求。

1. 可维护性原则：数据库模型的设计应该尽可能可维护，以便于满足用户的长期需求。
   1. 前端网页实现

智能问答系统的前端网页实现主要包括前端界面、后端服务器和数据库三部分。前端界面是用户与系统进行交互的界面，它可以提供用户友好的界面，使用户能够轻松地使用系统。后端服务器是系统的核心，它负责处理用户的请求，并将结果返回给用户。数据库是系统的存储空间，它存储着系统的知识库，以及用户的历史查询记录等信息。

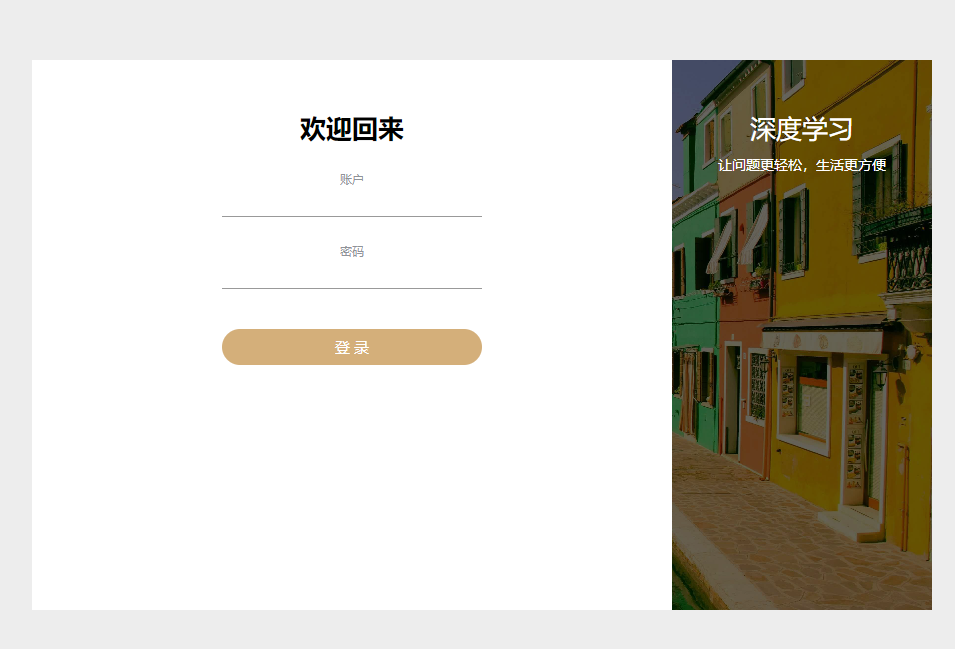
3.2.1前端UI设计

主页：

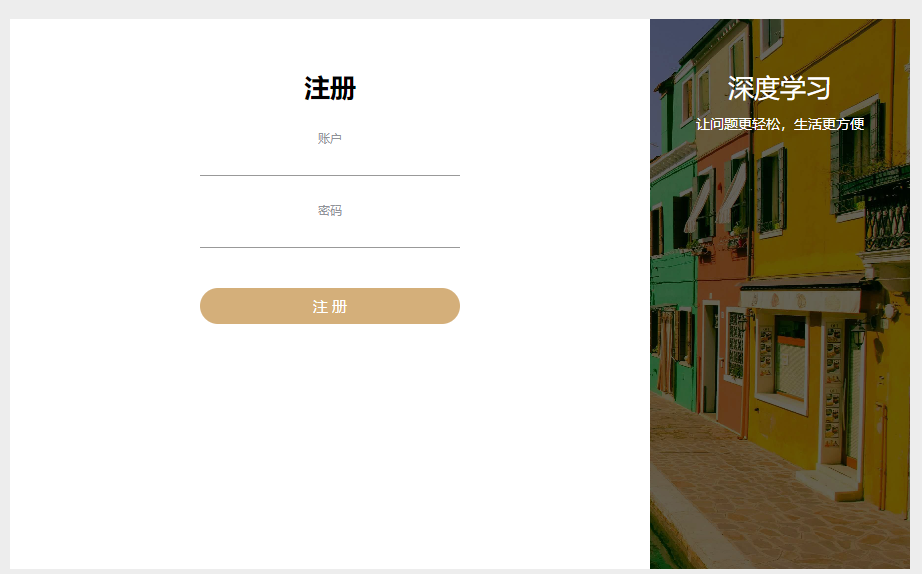
主要是对项目的介绍，对智能问答的大概介绍，下面是对网站前台的界面



登录：



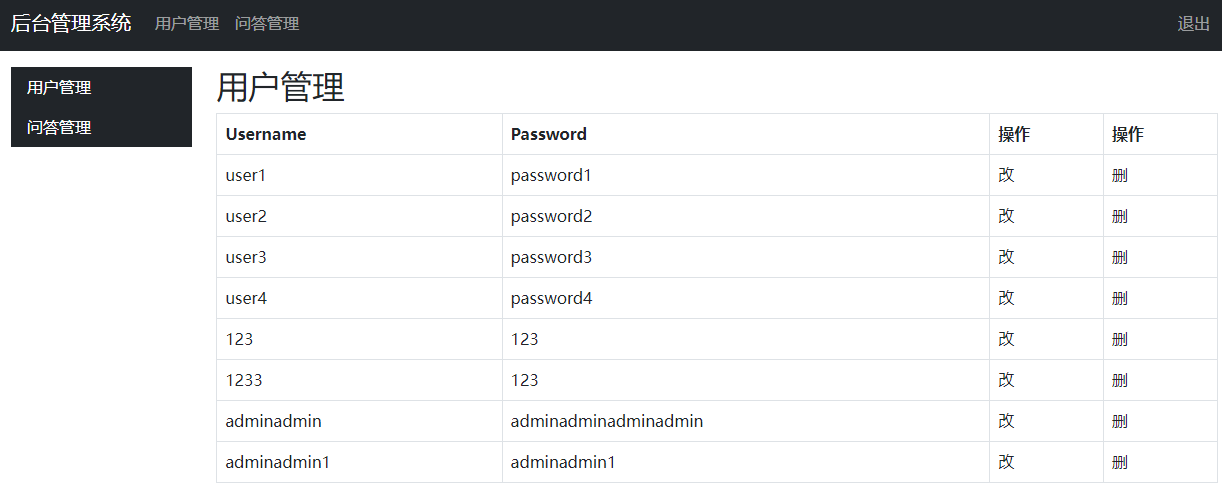
注册：

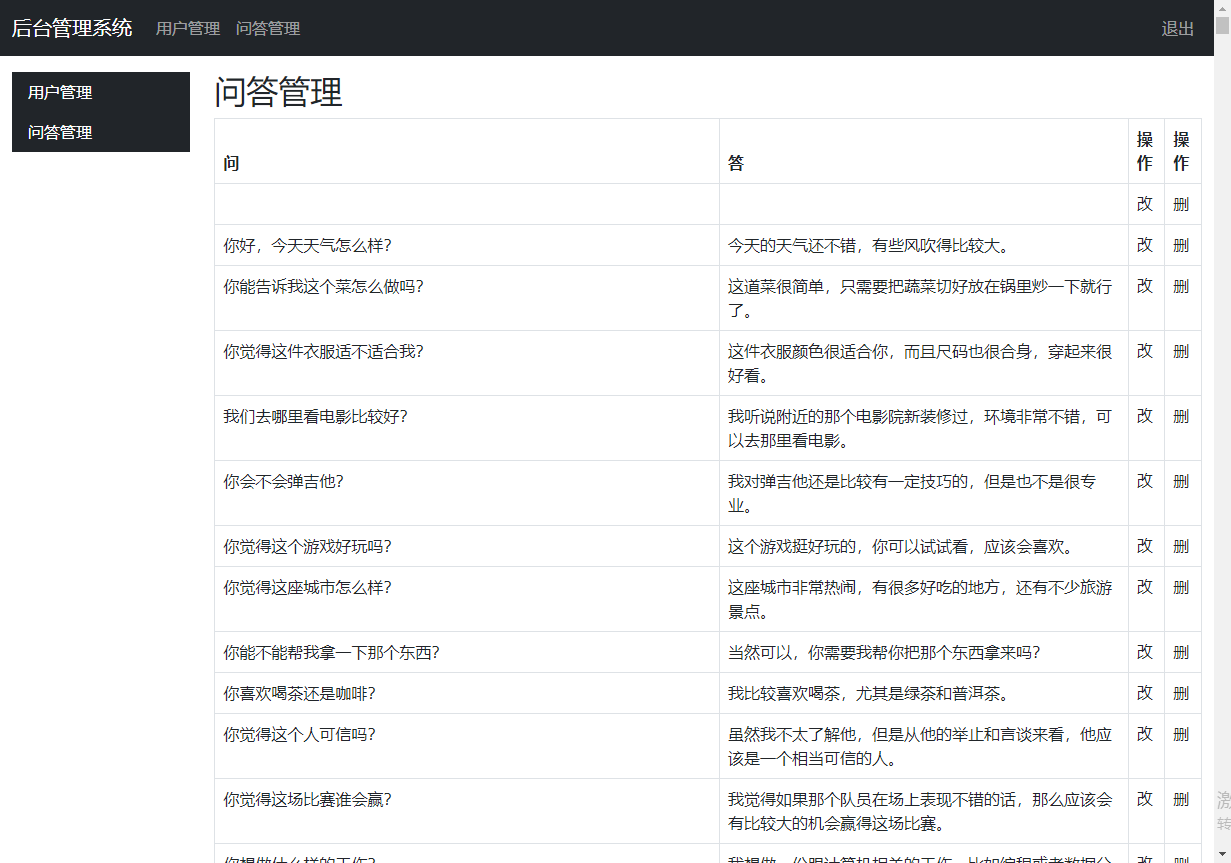


网站管理员登录：



用户管理：



问答管理：

智能问答系统的前端网页实现主要提供以下功能：

1. 智能问答：系统可以根据用户的输入，自动检索知识库，并返回相关答案。
2. 用户登录，方便统计每个用户咨询的问题，也方便投放广告。
3. 网站后台：方便统计用户，客户咨询的的问题

3.3系统实现

智能问答系统的前端网页实现主要包括以下几个步骤：

1. 用户界面设计：首先，需要设计用户界面，使用户能够轻松地使用系统。

2. 数据库设计：其次，需要设计数据库，存储系统的知识库，以及用户的历史查询记录等信息。

3. 系统功能实现：然后，需要实现系统的功能，包括智能问答、历史查询和用户反馈等功能。

4. 系统测试：最后，需要对系统进行测试，以确保系统的正确性和可靠性。

4.0设计总结

智能问答是一种自动回答用户提出的问题的技术，它可以帮助用户快速获得所需的信息，提高用户体验。智能问答的设计包括以下几个方面：

1. 问题分析：首先要分析用户提出的问题，把问题分解成一个个可以处理的子问题，以便更好地理解用户的意图。

2. 知识库构建：构建一个完整的知识库，包括问题的分类、答案的提取、答案的排序等，以便更好地回答用户的问题。

3. 答案提取：根据用户提出的问题，从知识库中提取出最合适的答案，以满足用户的需求。

4. 答案排序：根据用户提出的问题，对答案进行排序，以便更好地满足用户的需求。

5. 结果展示：将提取出的答案以友好的方式展示给用户，以提高用户体验。

通过以上几个方面的设计，可以有效地提高智能问答的准确性和可用性，从而更好地满足用户的需求。

后续工作展望：

1. 智能问答的设计可以更加精细化，比如可以添加更多的知识库，更多的问题分析算法，以及更多的答案排序算法，以提高智能问答的准确性和可用性。

2. 可以添加更多的自然语言处理技术，比如语义分析、语义推理等，以更好地理解用户的意图，提高智能问答的准确性。

1. 可以添加更多的智能化技术，比如机器学习、深度学习等，以更好地提高智能问答的准确性和可用性。

参考文献

1.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. OpenAI Blog. https://blog.openai.com/language-unsupervised/

2.

Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.