面向编程领域的问答系统

小组成员:

江子特*
侯兴中**
段宏键†
张枫‡
刘向宇§

中国科学院大学计算机科学与技术学院

* jiangzite19s@ict.ac.cn 201928013229131
** houxingzhong19z@ict.ac.cn 201918013229034
† duanhongjian15@mails.ucas.ac.cn 201928013229033
‡ zhangfeng19s@ict.ac.cn 201928013229130
§ liuxiangyu19@mails.ucas.ac.cn 201948300129011

本工程实现了一个面向编程领域的问答系统[1]。

1. 分工

下面是每个人的分工情况。

江子特:

1. 对输入的句子做单词纠错，在 birkbeck 错误拼写数据集上测试，准确率为 30%。
2. 尝试了对 How 类型的回答。

侯兴中:

1. 对输入的问题进行分类，方便对各类问题用不同方法来解决。
2. 调试和结果测试。

段宏键:

1. 负责文本数据化的处理（数据库建立）。
2. 对用户输入问题与题库中海量问题进行相似度匹配。

张枫:

1. 通过问题的链接地址获取网络用户的答案。
2. 对 Yes/No 问题进行回答。

刘向宇:

1. 对输入问题进行预处理
2. 对 what 定义类问题进行回答

2. 任务定义

作为程序员，在编写程序、实现功能、修改 bug 的时候我们都时常会感到自己知识的匮乏，哪怕在某个领域极其精通的程序人员，由于种种原因，也经常需要借助

1 我们的代码仓库：https://github.com/Jxt1/QA_system

互联网上的知识来帮助自己编程。而搜索引擎提供的内容与方法质量参差不齐，经常需要我们亲自尝试才能判断是否真的 work，这无疑费时又费力。

基于此，我们设计了一个**面向编程领域的问答系统（处理语言为英语），这个系统能接受用户在编程时遇到的各种问题的输入，整合我们问题数据库以及网络上的资源，然后返回问题的答案**。我们称这个系统为无忧编程 1.0 版。无忧编程 1.0 版致力于提升编程人员的工作效率，高效解决大家编程时遇到的问题。

3. 方法描述与实现细节

我们的无忧编程 1.0 版框架如下图所示，其中包含了很多自然语言处理方向的技术：文本分类、文本相似度匹配、文本处理等。按照系统的工作顺序，我们先后实现了**对用户输入的问题预处理、拼写纠错、建立问题数据库、对输入的问题进行相似度匹配、对输入的问题进行分类以及根据类别对答案进行不同的处理、最后返回答案等步骤**。

以下是我们系统的实现框架：

3.1 系统框架

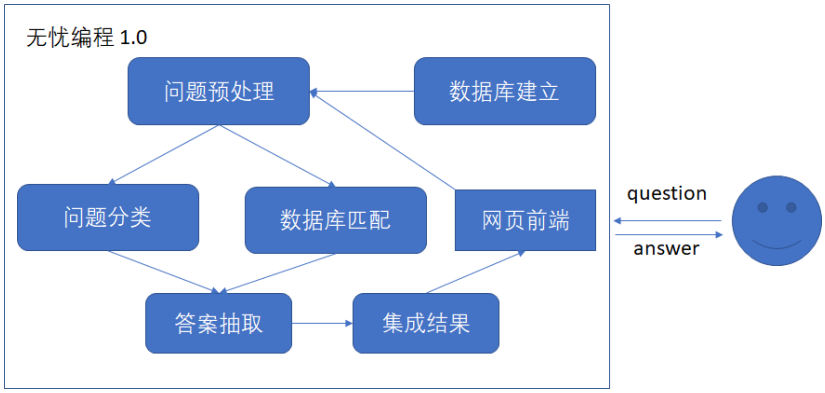

图 1: 系统流程图

3.2 问题预处理

首先，我们在问题分类之前要对问题做一个预处理，以便将问题更容易的分成七类（分类有有助于我们根据不同类别来组织要返回的答案）。

3.2.1 缩写词展开. 对问题的分类需要对问题格式进行统一（如 what’s 要转化成 what is）。因为我们系统的输入是一个有限的问题集，不符合格式的问题不是很多，因此可以用以一个列表一一枚举出不规范问题的种类，然后以另一个列表来枚举出

对应的规范格式，只要出现不规范的格式就用对应的规范格式进行替换。具体实现是先将字符串转换为 list 类型，然后删除不规范的输入格式串，再插入替换后的格式串，最后再将格式转化回字符串类型。

```python
import sys,os

def ques_str_op(ques_str):
    unexpected_form = ["What's","what's","where's","Where's","why's","Why's","When's","when's","How's","how's"]

    correctted_form = ["What is","what is","where is","Where is","why is","Why is","When is","when is","How is" "how
    ques_op = ques_str
    for str in unexpected_form:
        result = str in ques_str
        if result:
            num = unexpected_form.index(str)
            str_new = correctted_form[num]
            pos_str = ques_op.find(str)
            ques_op = add_substr(ques_op, pos_str,str_new)
            ques_op = delete_substr(ques_op,str)
    s = "https://"
    if "/" in ques_op:
        if s in ques_op:
            return ques_op
        else:
            while "/" in ques_op:
                m = ques_op.index('/')
                ques_op = delete_substr(ques_op, '/')
                ques_op = add_substr(ques_op,m," ")
    return ques_op

def add_substr(str,pos_str,sub_str):
    ques_op_list = list(str)
    ques_op_list.insert(pos_str, sub_str)
    ques_op = "".join(ques_op_list)
    return ques_op


def delete_substr(in_str, in_substr):
  start_loc = in_str.find(in_substr)
  in_str, in_substr = list(in_str), list(in_substr)
  [len_str, len_substr] = len(in_str), len(in_substr)
  res_str = in_str[:start_loc]
  for i in range(start_loc + len_substr, len_str):
    res_str.append(in_str[i])
  res = ''.join(res_str)
```

图 2: 预处理部分代码

3.2.2 拼写纠错. 定义英文字母表，大小写共 52 个字母。

1. 对于输入的句子，先进行子串的提取；

通过正则表达式匹配出所有只有字母组成的子串。无法覆盖到如”what’s”, ”It’s”这种缩写的情况。

2. 对于提取到的子串，寻找与之相近的单词；

对一个子串定义 4 种操作：增加、删除、替换、倒转。增加：在单词中某个位置加入字母表中的一个字符；删除：删去子串中某个字符；替换：将子串中某个字符用字母表中的一个字符进行替换；倒转：交换子串中相邻两个字符的位置；

对于一个子串，做 0 次、1 次、2 次上述的操作，得到能在单词表中找到的单词。

3. 通过每个单词的频度，确定出频度最高的单词作为纠正的结果；

3.2.3 拼写纠错测试. 使用了 birkbeck 错误拼写数据集。该数据集有 6136 个单词，一共 36133 个拼写错误单词，收录了英文使用者常见的拼写错误，数据链接：http://www.dcs.bbk.ac.uk/ ROGER/corpora.html。

6136 个单词中，有 248 个单词我们构建的字典中没有收录。最后使用了 35570 个单词作为测试，其中正确的为 10626 个，占 29.9%。

3.3 数据库建立

为了解决程序员可能会提出的问题，我们总结了自身常用的检索方式，比如使用 Google、StackOverflow 以及各种常用语言的官网手册（如 cppreferrnce）。为了满足检索问题的需求，我们的系统目前采用的数据来源于 stackOverflow。它作为一个在计算机领域非常活跃的论坛，上面有着约 18,000,000 条问题，无疑是我们的首选原始数据来源。

我们以爬虫的方法（spider/stackoverflow_spider.py）获得了数据。

从 stackoverflow 上抓下来的包的格式如下图：

```
1 Why is processing a sorted array faster than processing an unsorted array?
https://stackoverflow.com/questions/11227809/
why-is-processing-a-sorted-array-faster-than-processing-an-unsorted-array
['java', 'c\\xc3\\xa7\\xc3\\xa7', 'performance', 'optimization', 'branch-prediction']
23680
2 How do I undo the most recent local commits in Git?
https://stackoverflow.com/questions/927358/how-do-i-undo-the-most-recent-local-commits-in-git
['git', 'version-control', 'git-commit', 'undo', 'pre-commit']
20351
3 How do I delete a Git branch locally and remotely?
https://stackoverflow.com/questions/2003505/how-do-i-delete-a-git-branch-locally-and-remotely
['git', 'version-control', 'git-branch', 'git-push', 'git-remote']
16344
4 What is the difference between 'git pull' and 'git fetch'?
https://stackoverflow.com/questions/292357/
what-is-the-difference-between-git-pull-and-git-fetch
['git', 'version-control', 'git-pull', 'git-fetch']
11458
5 What is the correct JSON content type?
https://stackoverflow.com/questions/477816/what-is-the-correct-json-content-type
['json', 'http-headers', 'content-type']
9972
6 What does the “yield” keyword do?
https://stackoverflow.com/questions/231767/what-does-the-yield-keyword-do
```

图 3: 原始数据的格式

原始数据中每个问题有 4 行，分别为问题行、链接行、问题标签行以及 “赞数”。考虑到硬盘容量的大小，我们并未将完整的答案爬下来，只是给出了答案的链接来快速索引答案。

对于原始数据的处理，我们使用的 python 脚本都在 preprocess 文件夹中。

1. 首先我们把原始数据中的问题行提取出来，并对标点与特殊符号的处理，形成一个纯的 question.txt 文件。（process_question.py）

2. 统计所有（去掉重用词外）词及其词频（用 dictory 处理）。（count_word.py）
3. 给每个词定义一个 id（与词频有关，词频越高 id 越小）。（word2id.py）
4. 根据词的 id，把每个问题转化成一个排序好的 list，list 添加额外的最后一位来保存问题的原序号，便于寻找问题对应的链接。（question2id.py）
5. 所有句子的 list 组成一个大的排好序的 list（最后的结果类似于二维向量从左到右递增排序，从上到下递增排序的形式），便于对用户输入的问题做检索。（question_sort.py）

至此，我们完成了文本的数据化过程，建立起了一百四十万问题集合的数据库（data/listofsentence.json）以及文本中词到数字的转化映射（data/word2id.json）。

3.4 相似度匹配

在对用户输入的问题与数据库中的问题进行相似度匹配的过程中，我们分别尝试并实现了编辑距离（LibSimilarity/process_edit_distance.py）、Jaccard 系数（LibSimilarity/process_Jaccard_index.py）、TFIDF（LibSimilarity/similarity_TFIDF.py）以及 word2vex（LibSimilarity/word2vec）等方法来计算相似度。但我们的问题数据库中有一百多万的问题，基于计算余弦距离然后一一匹配的相似度计算与匹配需要耗费大量时间，所以我们最后的实现中没有采用这些方法。

我们认为我们的问答系统面向编程这个特定领域，故如果用户输入问题中的关键字能和我们数据库中问题的关键字匹配到一起，我们就能认为这两个问题相似。如当用户输入问题为

How do I redirect to another webpage?

时，我们仅需抽取出关键字

redirect、another、webpage

以及知道问题的类型为 how，就能从问题数据库中找到与此输入问题最为相似的问题。

基于上述思想，我们实现了一个**在用户输入问题与百万级问题数据库之间快速匹配并计算相似度的算法**。（similarity_main.py）此算法采用二分检索在排好序的问题数据库上（listofsentence.json）寻找与用户输入问题最相似的匹配，并把问题库中匹配到的问题的链接以及相似度返回，用以后续的处理。

3.5 问题分类

对于输入的问题，首先要判断问题是属于哪一类的，我们为了方便对于问题的处理，将问题按照回答的格式进行分类。主要使用了关键词匹配和词性标注的方法。

对 stackoverflow 上问题进行分析，可以将问题分为以下几类：

3.5.1 How 型问题. How 型问题就是提问者提问某个事情需要怎么做或者说某个问题需要怎么解决。Stackoverflow 上的很多 How 型问题都是以 "how" 关键词来进行提问的。当然，有一部分的 How 型问题是没有疑问词的，直接以祈使句的形式来表达问题。如 **"Catching multiple exception types in one catch block"**，这一问题没有任何疑问词，所以如果要判断是否是 How 型问题，还需要对没有疑问词的问题进行词性标注。在处理这类问题时，我们将问题化简，可以注意到这类问题都是以一个动词开始的祈使句，所以在判断时，只需要判断第一个词的是否是动词即可。这里我们使用了 NLTK 这一自然语言处理工具来进行词性判断。由于 NTLK 的词性标注效果足够，所以我们直接使用了 NTLK 的词性标注函数。

3.5.2 Yes or No 型问题. Yes or No 型问题则是判断某一陈述是否正确或者某一操作是否能够实现某种功能。其一般关键词为 **"Can"**（如 "Can (a==1 && a==2 && a==3) ever evaluate to true?"）、**"Is there"**（"Is there a way to instantiate a NSManaged Object without inserting it?"）、**"Is"**（"Is it possible to build a JPA entity by extending a POJO?"）、**"Are"**（"Are there any languages that compile to Bash?"）、**"Do"**（"Do conditional statements slow down shaders?"）、**"Does"**（"Does C# 8 support the .NET Framework?"）等，使用关键字匹配的方法来确定 Yes or No 问题。将 Yes or No 型问题单独归为一类，会极大地方便这类问题的后续处理。

3.5.3 Why 型问题. Why 型问题主要是解释某一个行为，我们称之为动作解释型问题。其一般表现形式为 **"why"**（如 "Why is processing a sorted array faster than processing an unsorted array"）、**"reason"**（"What is the reason for having '//' in Python?"）等，这一类问题使用关键字匹配来归类。

3.5.4 Compare 型问题. Compare 型问题是对两种事物进行比较。可以观察到，stackoverflow 上很多问题都是关于两种东西孰优孰劣或者是比较两个东西之间的相同或不同之处。其一般关键词为 **"compare"**（如 "Compare two files in Visual Studio"）、**"difference"**（"Differences between socket.io and websockets"）、**"vs"**（"Static vs class functions/variables in Swift classes?"）等。这类问题可以近似地看做是两个名词解释型问题，我们对于这类问题的识别同样是关键词匹配。

3.5.5 What do 型问题. What do 型问题就是名词解释型问题，这与 Why 型问题即动作解释型问题是不一样的。Stackoverflow 上有一部分问题是以 **"What do ... do"**（"What does Function.prototype.toMethod() do"）和 **"What do ... mean"**（"What does PuLP LpStatus=Undefined actually mean"）的形式出现的。这类问题一般是问某个函数或者某个句子等的含义，我们将其归类为名词解释型问题，即 What do 型问题。由于这类问题的关键字较为明显，所以使用关键词匹配法来归类这类问题。

3.5.6 What 型问题. What 型问题即是名词问答型问题，这类问题一般只需要回答一个名词即可。其问题有以下几种：

1. **"What"** 型，如 "What is the scope of variables in JavaScript?"。
2. **"Which"** 型，如 "Which characters are valid in CSS class names/selectors?"。
3. **"Who"** 型，如 "Who architected / designed C++'s IOStreams?"。
4. **"Where"** 型，如 "Firebase messaging, where to get Server Key?"。
5. **"When"** 型，如 "When should I use std::thread::detach?"。

这类问题带有明显的疑问词，所以根据关键词就可以判断这类问题。

对于 what 这种定义类问题，起初想到的解决方式是爬取网站加上本地检索。但是想这种程序类的小众问题，一般的网站是很难有非常标准的答案的。所以我们第一步就是先在本地的数据库中查找，找到答案之后，做一些字符串的处理返回答案内容，若找不到就去 wiki 中扒取，把问题处理后只保留一个 keyword，对其进行搜索。将搜索到结果的第一句话进行扒取，字符串处理后返回。

3.5.7 others 问题. 我们将不属于上面 6 类的问题分类到 others 中，在接下来的问题处理使用不同的方法对这类问题进行处理。

其中，在进行问题分类时，尤其是关键词平匹配的环节，可能某个句子中拥有不同类型问题的关键字（如 "Is there any reason to use RabbitMQ over Kafka?"，这一句既包含了 Yes or No 型问题中 "Is there" 关键字，也包含了 Why 型问题中 "reason" 关键字，然而这是一个 Why 型问题），所以我们需要对关键字匹配的顺序进行优先级排序，从而得到较好的分类结果。经过我们的测试，将这 6 个类型关键词的优先级排序如下：Compare 型问题、Why 型问题、How 型问题、Yes or No 型问题、What do 型问题、What 型问题、others 问题，在这样的优先级排序下，我们可以将问题较为高效准确地分为以上 7 类问题，从而为后面的答案抽取提供了便利。

3.6 答案抽取

3.6.1 获取社区用户答案. 在实现的问答系统中，抓取了相关社区网站的问题目录和问题所对应的链接地址。通过访问这些链接地址，从对应问题回答的页面中获取最佳的社区用户答案。这一部分的功能在 get_answers.py/get_best_answer_on_url 函数中实现。

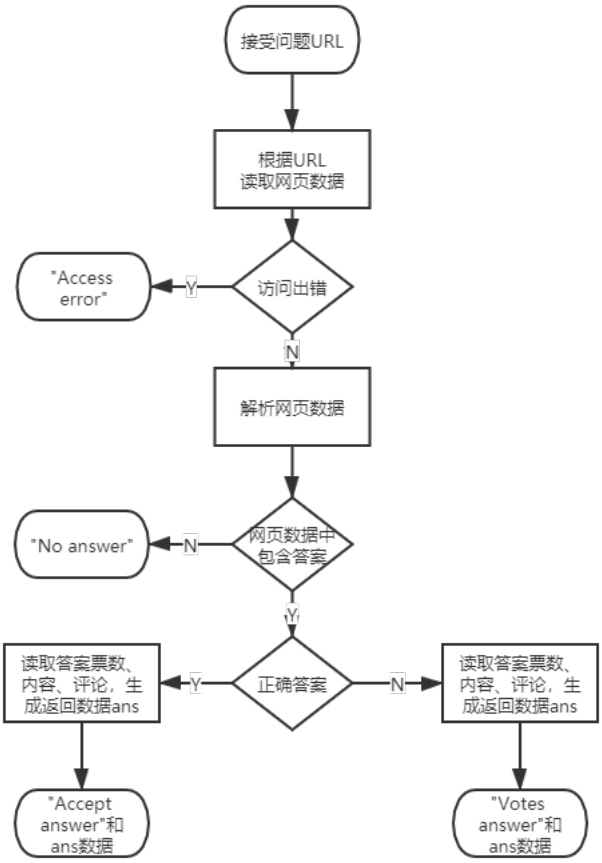

图 4: 获取社区用户答案流程

具体流程如下：

1. 使用 requests 库通过 URL 读取网页数据 html_data。
2. 使用 BeautifulSoup 库解析 html_data。
3. 判断网页数据中是否包含答案，并判断是正确答案还是票数最多的答案。

4. 根据 html 的标签 <div class=” js-vote-count grid–cell fc-black-500 fs-title grid fd-column ai-center” > 获取答案的投票数量、<div class=” post-text” > 获取答案的内容、<span class=” comment-copy” > 获取答案的评论。

5. 处理获取到的 html 数据，去除其中的 html 标签，识别出答案中的代码部分，生成返回的数据 (票数, 答案 list(答案, 代码), 评论 list(评论 1，评论 2,···))

6. 返回状态值 Accept answer/Votes answer 和返回数据

3.6.2 How 型问题的回答. 对于 How 类型的回答，最好的结果显示为一个执行序列。**目标**是将顺序有条理的列出来，但是在实现中，做不到对篇章进行分析，并整理出逻辑。只是简单对句子做分割，然后按照原始句子的顺序进行罗列。

对原始结果，首先进行句子分割。由于原始的 html 格式对 code 的标签对一句话的影响，使得我们在句子分割的时候需要判断 code 是出于句子中还是在句子外，具体情况如下图。

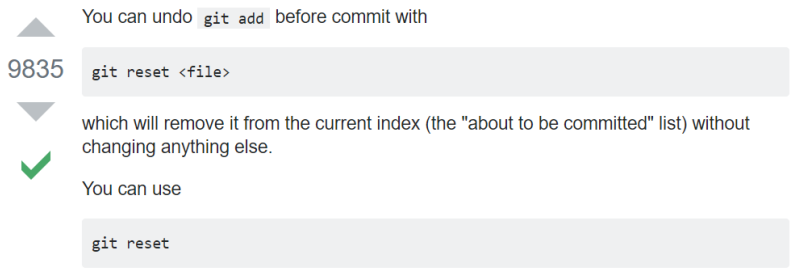

图 5: how 类型的社区回答例子

在该部分，只是简单的对每个句子进行标号。

句子分割。以英文中的句号 “.” 为分割符号，对句子进行划分。在这里，需要区分代码中的 “.” 对结果的影响。

冗余句子的删除。另外观察到，一个句子中如果每个词都不在问题中出现，很有可能是冗余的句子。因此，对于这种情况的句子，我们直接删除掉。

3.6.3 Yes/No 型问题的答案. 针对 Yes/No 问题主要目标是能够判断出问题的正确与否，对于部分问题可以进一步的给出解析或示例。这一部分的主要是根据访问网页获取的原始答案，进行进一步的信息检索，然后返回简洁有效的答案。具体功能在 AnswerExtractor/AE2.py/answer 和 handle_yes_or_no 函数中实现。

流程如下：

1. 判断原始的答案中是否有 Yes 或 No。如果包含 No，则直接返回 No；如果包含 Yes，则返回 Yes 并检索后面的答案，返回必要的信息。
2. 针对没有包含 Yes 或 No 的答案，判断原始的答案中是否给出了例子或解释。若给出了例子或解释，则说明该问题答案大概率属于 Yes 类型，返回 Yes 和例子或必要的说明；否则返回 No。

3.7 集成结果

考虑到系统中各个模块可能存在的误差。

比如分类的结果不一定正确，使得最终返回的结果不能完全按照预测类别作为结果。同时，即便预测正确了，我们的模型有时候也不能正确的回答问题。

因此，该模块会综合每类回答的情况，给出返回给用户的结果。直接通过每一类的概率选择出比值最大的类别的答案。

4. 数据集

4.1 stackoverflow

写网络爬虫，由于爬虫的速度慢，来不及将 Stackoverflow 上的所有数据爬下来，只有 7.77% 的数据，共 1447200 条。

4.2 停用词

由于 Python 的 sklearn.feature_extraction.stop_words 中提供了较为完备的停用词（ENGLISH_STOP_WORDS），故我们处理问题数据以及用户输入数据时都使用了 sklearn 模块提供的停用词。

5. 基准系统

我们尝试寻找关于编程领域的问答系统的基准系统，但此领域并无特别好的基准系统。我们也尝试了 MIT 的 START[2] 系统的，发现它对问题的回答并不好。

6. 实验设置与结果展示

下面分别展示了 6 种问题回答的效果，其中 How 型问题的问句中有一个单词 number 被误写为了 nuber，可以看到答案中是按照 number 来回答的，在对问题的

2 http://start.csail.mit.edu/index.php

处理过程中已经对拼写进行纠错。others 问题的回答则是从 stackoverflow 上搜索最为适合的答案，将其作为我们的回答输出。

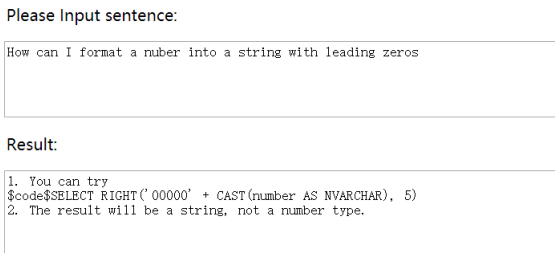

图 6: How 型问题的回答

图 7: Yes or No 型问题的回答

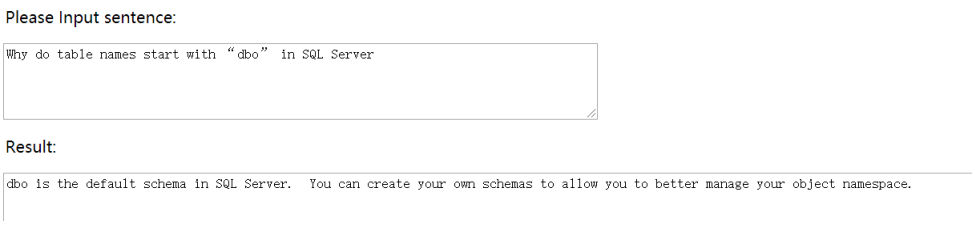

图 8: Why 型问题的回答

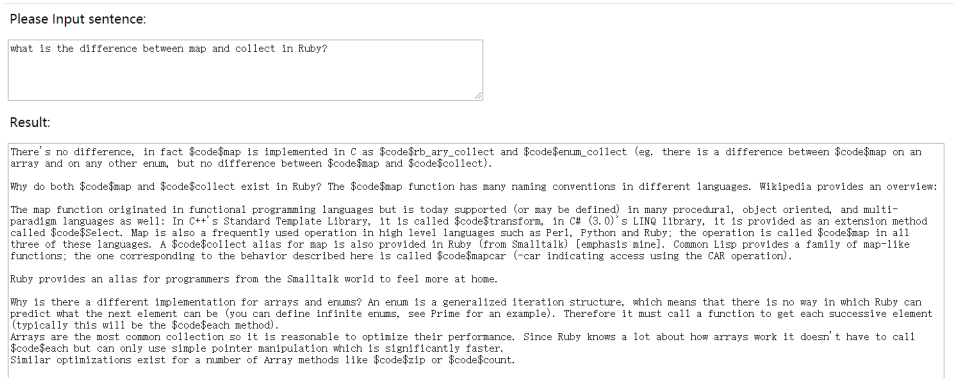

图 9: Compare 型问题的回答

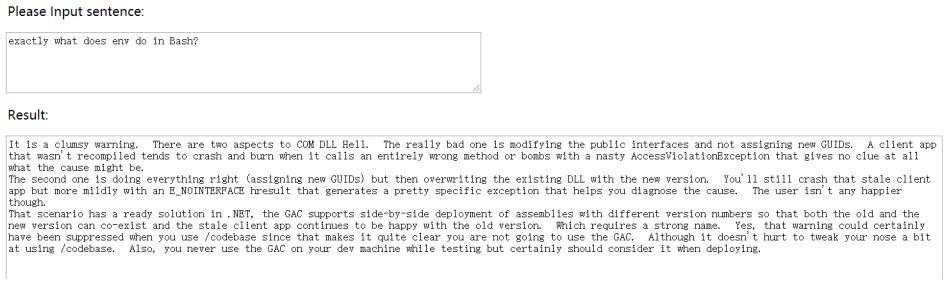

图 10: What do 型问题的回答

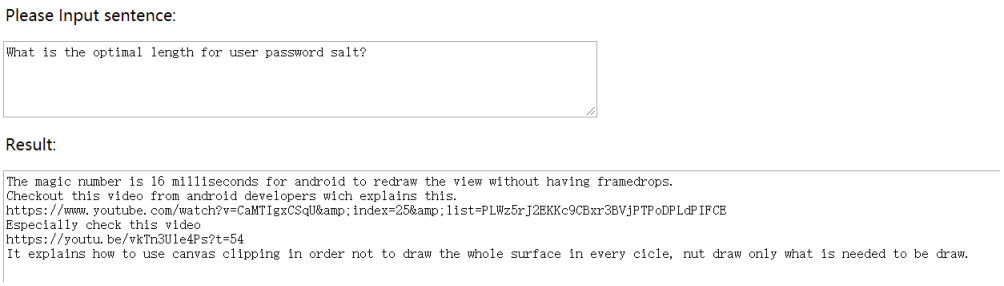

图 11: What 型问题的回答

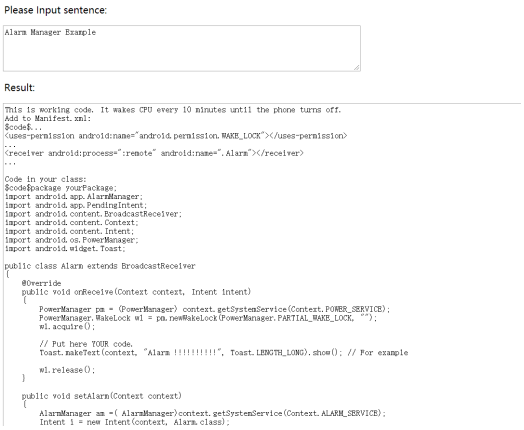

图 12: others 问题的回答

7. 实验分析与总结

我们的问答系统目前已经能够很好的 work，但我们在构建这个系统的时候遇到了许多问题，到目前为止其中仍然有很多的点值得进一步的优化，以下是我们对整个系统构建过程中遇到难点的分析以及可能的优化方向。

7.1 难点及未来优化方向

7.1.1 拼写纠错. 一个是不在词典中的单词，难以处理。同时，可能会把代码中的缩写改掉。对于拼写中的缩写的错误难以看出。

7.1.2 相似度匹配. 由于我们的系统只使了句子的关键词来对用户输入的问题和问题数据库中的问题进行匹配，所以匹配的效果并不固定。下一步我们会引入深度学习的方法，完成对语义更深入的解析，再此基础上进行更精确的相似度的匹配。

7.1.3 问题分类. 目前的问题分类是基于关键词的分类方法，这种方法的局限性较高，不太适用于普遍性的问题分类，而且问题需要比较严格的格式。对于词性标注部分来说，NLTK 的标注准确率尚可，但还达不到足够的高度。如果需要更高精确度的问题分类，则需要更细粒度的问题分类，并使用深度学习模型对爬下来的数据集进行训练，这样一来，标注训练集又成为了下一个难题。

7.1.4 获取社区用户答案. 因网络质量问题，访问某些社区时会出现错误，难以正确返回社区用户的答案，对系统输出答案的准确率造成了影响。下一步可以从多个数据库以及网页源头来获取原始的答案材料然后进行加工，以便对用户提出的问题进行更好更全面的回答。

7.1.5 How 类型回答. 需要对篇章中单独的句子逻辑进行理解，这个问题以目前查到的资料来讲，仍没一个好的解决方案。

同时对于计算机中的问题，篇章中容易出现代码块以及程序中的一些描述方式，这使得对理解句子间的关系变得更加复杂。

7.1.6 Yes/No 型问题回答. 没有完善的答案知识库，对于问题的答案缺少有效的手段进行简化，系统输出的部分答案比较臃肿，并且目前的回答结果多数依赖于社区用户答案，缺少足够的判别能力。可以进一步获取多个社区的信息和搜寻网络上其他数据库，构建一个完善的知识库，能够使用句法分析、篇章分析对问题答案进行更多的分析；构建问答系统的知识图谱，提高系统的判断能力和问题回答的准确率。

8. 展望：继续进化

在完善面向编程的问答系统之后，我们可以考虑更加通用的问答系统的实现。目前的想法是要么把多个面向单领域的问答系统整合为多领域的全能问答系统，要么构建新的系统，使其能对各个领域的问题进行回答。