

**大作业结题报告**

课程名称： \_ 高级软件工程

报告题目：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学生姓名： \_\_\_\_\_\_王鸿森 王柯林 王伊梁 王子妍 易士程

班 级：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

指导老师： 罗铁坚

完成时期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

二零二零年十二月十一日

目录

[1 项目背景 3](#_Toc58605253)

[1.1项目意义 3](#_Toc58605254)

[1.2研究现状 3](#_Toc58605255)

[1.3项目目标 5](#_Toc58605256)

[2 需求分析 5](#_Toc58605257)

[2.1项目针对的用户 5](#_Toc58605258)

[2.2项目的具体功能要求 6](#_Toc58605259)

[3 系统设计与实现 6](#_Toc58605260)

[3.1前端模块 6](#_Toc58605261)

[3.2后端模块 8](#_Toc58605262)

[3.2.3 Django通信 9](#_Toc58605263)

[3.3聊天功能模块 12](#_Toc58605264)

[3.3.1文字聊天模块实现 12](#_Toc58605265)

[3.3.2 图片聊天模块实现 14](#_Toc58605266)

[3.4数据库模块 15](#_Toc58605267)

[3.4数据库模块 15](#_Toc58605268)

[3.4.1 redis数据库 15](#_Toc58605269)

[3.4.2 Sqlite3数据库 16](#_Toc58605270)

[4 测试 18](#_Toc58605271)

[4.1单元测试 18](#_Toc58605272)

[4.2 安全测试 19](#_Toc58605273)

[4.3功能模块测试 21](#_Toc58605274)

[4.4数据库安全性测试 22](#_Toc58605275)

[5 项目进行遇到的困难与解决方案 23](#_Toc58605276)

[6 项目协作与部署 24](#_Toc58605277)

[6.1项目协作 24](#_Toc58605278)

[6.2项目部署 25](#_Toc58605279)

[7 项目总结 25](#_Toc58605280)

# 项目背景

## 1.1项目意义

人的需求是有着一个从低级向高级转换的过程，随着现代社会生活的条件的发展，人们得需求逐渐从马斯洛需求层次的低层次向高层次发展。其中归属与爱的需求强调了人与人的联系，这种联系通常能通过交流实现，但是由于工作、学习的繁忙以及时间碎片化趋势，人们无从释放这一需求。

一个自然的想法是通过其他物体来释放这一需求，从而出现了养宠物的盛行。人们也尝试通过机器来践行这一想法。市面上有着众多的聊天机器人如苹果siri、微软小冰等，但是他们也有着各种缺点，比如缺乏外界信息输入单一、模型过重、联想能力弱等。面对目前这一大众日益增长的需求，需要有一个新型的聊天机器人来尝试补全。

作为高级软件工程课的课程设计项目，在这个项目中我们更好地理解了本课程，对于包括但不限于软件工程研究主题、应用场景和需求表达、业务建模和应用框架、设计模式和代码重构、接口方式和前端编程、软件验证与自动测试、安全问题和防御设计、性能评价和优化设计、软件集成部署和运营等领域有了更深的认识。

## 1.2研究现状

聊天机器人(chatbot)由Michael Mauldin在1994年提出，是一个可用于构造一个在线的、文字-文字或文字-语音的，可代替真人进行直接沟通的应用[[1]](#footnote-1)。聊天机器人通常在对话中模拟人参与，通过不断的调试和测试，以期通过图灵测试。聊天机器人常出于实用目的开发，例如用作客服机器人，用于进行简单的售后也可用作虚拟助理一部分(苹果siri、微软小冰等)。聊天室模式的社交机器人是另外一种模式，早期的“Eliza”和 “Parry”等聊天机器人是其中典型代表。

聊天机器人又可以分为开发域的聊天机器人和面向任务的聊天机器人。开放域的聊天机器人面对的用户和用户输入是不固定的，对话是一种闲聊的模式。通常可以通过用户主观体验来衡量对话的质量。可以通过海量FAQ检索以及端到端的方式实现。面向任务的聊天机器人则有明确的对话目的，以完成特定特务为主，实现上有基于规则和基于数据两种。衡量对话质量时，最基本的要求是否能完成特定用户任务。

结果输出

自然语言生成

自然语言理解

用户输入：文字、图片、语音

对话管理

图1. 一个典型聊天机器人框架

目前典型的聊天机器人的实现使用NLP技术，一个大概的聊天机器人框架如图一1所示。其中自然语言理解包含实体识别、目的识别、情感识别、上下文情景指代消解和恢复、据识判断等[[2]](#footnote-2)。自然语言理解使用的方法可以是基于规则的（人工定义语法规则，自然语言理解模块根据表达特定的语义的语法规则解析输入该模块的文本，适合于数据较少的文本解析）和基于统计的（使用大量的数据训练模型去解析输入文本，适合于大多数场景即神经网络）。此外，语法分析、句法分析‘和语义分析也是对自然语言理解关键的基础。Seq to seq模型是现在主流的神经网络对话模型，论文《Sequence-to-Sequence Learning for End-to-End Dialogue System》概述了该模型的发展。Seq to seq由encoder和 decoder两部分组成，标准结构如下：

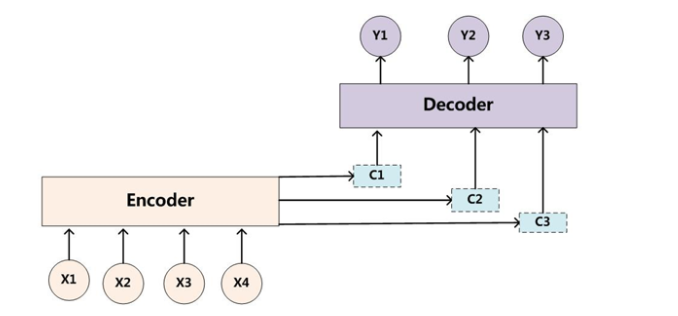


图2

自然语言处理则是确定返回给用户的信息。目前有基于模板、基于检索、基于深度学习的自然语言生成三种方式。基于深度学习的自然语言生成实际上是基于模板，模板由神经网络构成，然后根据深度学习得到的模板生成。

微软小冰是微软在2014年退出的一个完整的、面向全交互的人工智能交互主题框架，包括核心对话引擎、多重交互感官、第三方内容的触发与第一方内容生成，和跨平台的部署解决方案。微软小冰能够实时和多人聊天并学习，根据用户输入会不断晚上自身，微软小冰一个问题是对于信息输入的安全性。恶意用户可以提供诶微软小冰恶意信息，使得微软小冰学习的知识被误导，使得其回答变得暴力、色情等。苹果siri是另外一款受到广泛欢迎的私人助理应用，也具有强大的对话能力。Siri一个强大的地方在于开发域的搜索。Siri能处理常见日常提问，例如天气。

## 1.3项目目标

目前市面上的相关聊天机器人具有庞大、特定输入相关特定平台相关等特点，我们的聊天机器人力求能提供一个对话机器人能够处理多种信息，并能进行正确的回答，该聊天机器人在联网条件下随时可用，满足用户随时随地开始聊天的需求，对于开放域的问题，我们设计的聊天机器人也能够给予回答。

# 需求分析

## 2.1项目针对的用户

项目针对的目标用户是具有网络聊天需求的互联网用户

首先明确的是一个用户显著特征是用户时间的碎片化和用户精神状态的疲惫。目前由于社会经济的发展，存在于各个行业的竞争愈发加剧，为了获得足够的生活资料，人们倾向于付出更多的时间和精力成本。这导致人们难以抽出大段完整时间进行娱乐活动来放松自己，这进而导致了精神状态的疲惫。

针对这些特点，我们项目进行了以下设计：

1. 随时、随地可用

因为繁忙工作和生活压力，用户进行聊天的环境和时间点是不固定的。我们在设计、实现过程中强调了我们的应用可以在联网条件下任何时间、地点、场景下可以正常访问。

1. 聊天可随中断重开

应用设计为可以随时中断访问。现在有大量所谓的利用碎片化时间进行娱乐的应用要求你的碎片化时间有一定的长度。这先让是不符合用户自身需求的。我们的项目打破了这些要求，简化交互流程，用户在聊天过程中可以随时中断离开。用户断开聊天后又可以随时可以重新开始聊天。

当前应用设计强调用户存活率，我们

1. 聊天内容轻松愉悦

我们假定了用户的一般精神状态是疲惫的，这要求我们的聊天机器人能够在聊天过程中缓和用户的精神压力。聊天机器人的输出信息应该是简单的、易于理解的，进一步应该是幽默的。输出信息要求尽量不对用户产生任何理解上的困难

我们的聊天机器人也能满足一定的功能性需求。用户在聊天时，往往有一些实际需求，例如了解当天的天气状况，针对这些开放域要求，我们的聊天机器人应该能给予正确的回答。

1. 操作简单、无任何心智负担

聊天机器人的整体设计（包括UI设计、交互设计、功能设计）应该是十分简洁的。用户能快速的开始聊天，应用和用户的交互除具体聊天外应该是尽量少的。

## 2.2项目的具体功能要求

1. 用户注册、登陆、注销

作为一个web应用，我们队每个用户提供特定服务。通过用户注册信息，我们 能保存特定用户聊天记录，方便用户可以随时随地中断、重开应用。着要求我们提供注册、登陆和注销界面

1. 用户输入信息处理

应用对用户输入信息的类型做以下假设：图片或者文字信息。在传入后端自然语言处理模块之前，我们会对信息做一些简单的预处理。应用界面应有明确的文字输入提示和语言输入提示。

1. 自然语言和图形信息理解

应用对用户传入的文字和图形信息应能正确理解，并给予正确回应。

1. 自然语言生成

在后端自然语言信息和图形信息理解模块生成了相应数据后，应用会根据相应的数据生成相应的回答。

1. 开放域问题回答功能

对于用户在聊天过程中提出的开放域问题，我们应用的应该能正确的生成相应的回复信息。

1. 用户输入信息反馈

针对一般信息输入和开放域问题输入，应用将其发送到服务器端处理时做出分别，服务器响应信息是统一的。客户端应能正确收到服务器响应信息并显示在网页。

# 3 系统设计与实现

## 3.1前端模块

前端部分我们基于python中的Django的框架，使用html、JavaScript以及CSS代码实现了前端页面的展示以及与后端部分的交互。前端部分主要包含两个模块，一个是注册登录模块，另一个是与机器人聊天的模块。

**3.1.1注册登录模块**

前端登录界面的结果如下：



在登录界面中，我们实现了正常的用户通过账号和密码登录的功能，用户能够通过输入已经注册的账号和对应密码进行登录，同时对于未注册的账号或者未正确匹配的账号密码，则会进行报错提示，无法正常登录。

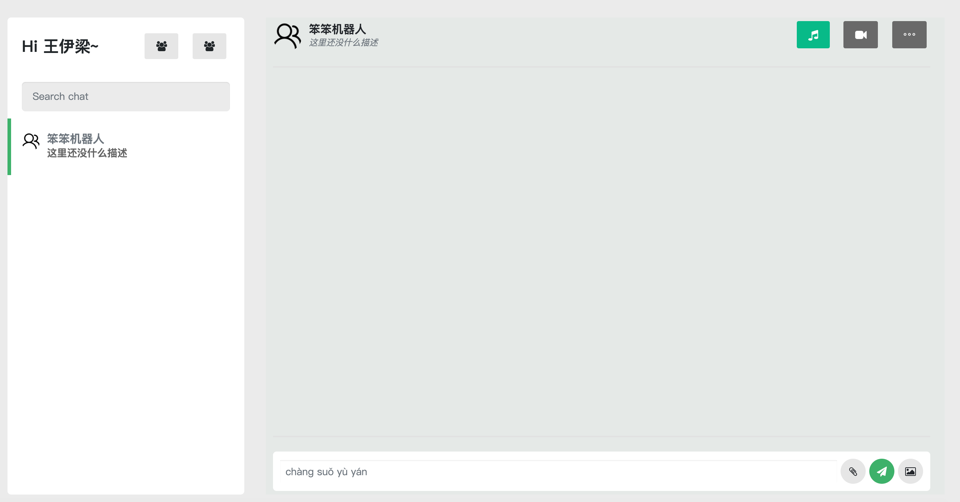
前端注册界面如下：



在注册界面中，我们实现了正常的用户通过账号、邮箱进行账号注册的功能，在注册过程中，我们将对应的用户信息存入数据库中。在注册中，我们办证用户账号以及邮箱唯一，并且需要输入的两次密码完全相同才能允许注册。当用户注册完成后，用户就可以通过对应的账号密码进行登录。

**3.1.2用户聊天模块**

用户与机器人的聊天界面如下：



在用户聊天界面中，用户可以与机器人正常进行聊天。我们在聊天输入框中输入对应的句子，按回车或者右下角发送图标进行发送，前端就会将输入内容展示在聊天框中，并将内容保存至数据库中，同时让后端调用机器学习模型，将对应返回的结果展示到聊天框中。

除此之外，我们在聊天界面中支持了图片聊天的功能。因此用户可以通过右下角的图标上传本地图片，前端会将图片展示到聊天框中。前端同时将图片返回给相应机器学习处理的模块，并将对应返回的结果展示到聊天框中。

## 3.2后端模块

由于聊天分为文字和图片两种，分别由两个模块来实现文字和图片的对话。

**文字部分**

对于文字部分，后端向文字聊天模块发送消息，文字聊天模块处理完之后发送给后端，后端接受之后对聊天信息进行存储，发送给前端展示出来。

请求信息

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| URL | 请求方式 | 请求参数 | 数据类型 | 是否必须 | 说明 |
| 127.0.0.1:2345/ | POST | data\_id | String | 是 | 表示当前请求id |
| question | dict | 是 | dict{"question": "问句"}传入问题 |

返回信息

|  |  |  |
| --- | --- | --- |
| 返回参数 | 数据类型 | 说明 |
| code | int | 状态交互码 |
| chatContent | String | 聊天机器人的问答结果 |

**图片部分**

对于图片部分，在传输过程中，采用图片的base64编码传输信息后端向图片聊天模块发送图片的base64编码，图片聊天模块在处理完之后，将图片的描述信息返回。后端将返回的图片描述信息发送给文字聊天模块处理。

请求信息

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| URL | 请求方式 | 请求参数 | 数据类型 | 是否必须 | 说明 |
| 127.0.0.1：5000/ | POST | data\_id | String | 是 | 表示当前请求id |
| picture | base64 | 是 | Base64格式的图片表示 |

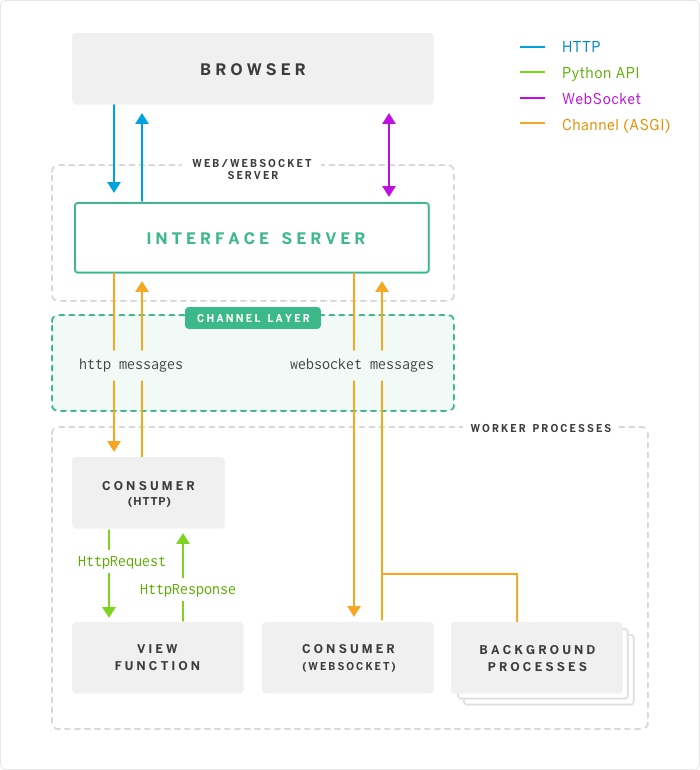
返回信息

|  |  |  |
| --- | --- | --- |
| 返回参数 | 数据类型 | 说明 |
| code | int | 交互状态代码 |
| imgres | String | 针对图片生成的文字标注信息 |

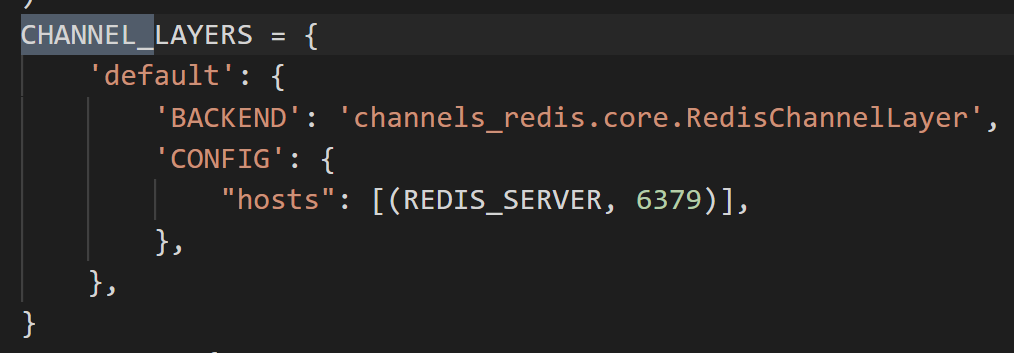
### 3.2.3 Django通信

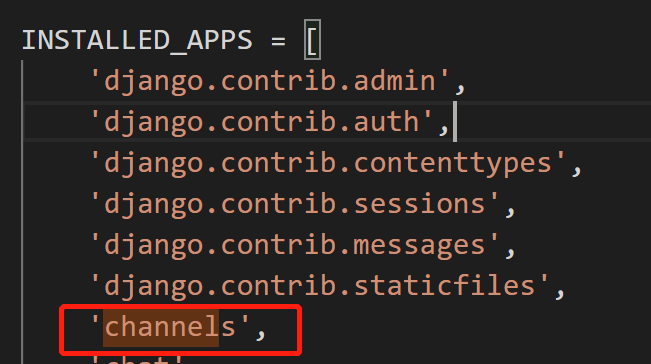
由于本项目涉及到双方之间的对话和讨论，Django本身不支持websocket协议，因此选用Django-channels来实现通信。我们一般使用http发送信息，由客户端发出请求，服务端进行响应，即Request和Response，服务端不能主动发送信息到客户端。而websocket是单个TCP连接上进行的全双工通信的协议，当连接建立之后，双方就建立了持久连接，双方可以随时向对方发送数据。Websocket一个很重要的应用场景就是聊天室，一个浏览器端发出的信息，其他浏览器端都可以同时接受。

在实现过程中，一个服务器可以同时与两个客户端（一个客户端http通信，一个客户端websocket）通信，当在websocket通信时，若需要在接口中http请求另一个服务器，此时不能使用全局变量资源，因为真正部署上了后，取到的应为空，所以尽量使用redis存储，并且部署需要使用都支持http和websocket的Daphne。

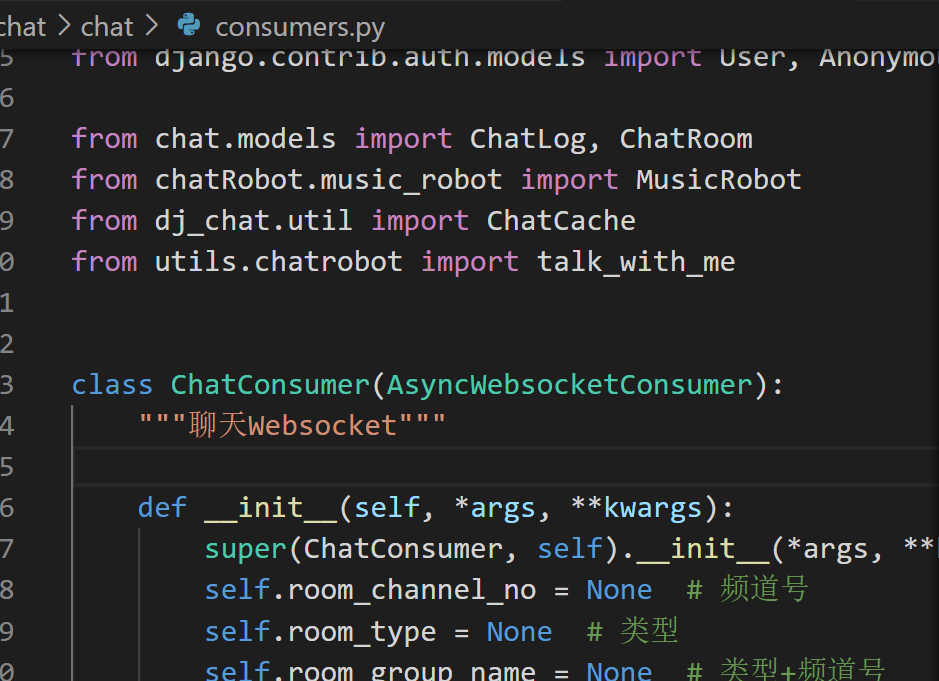


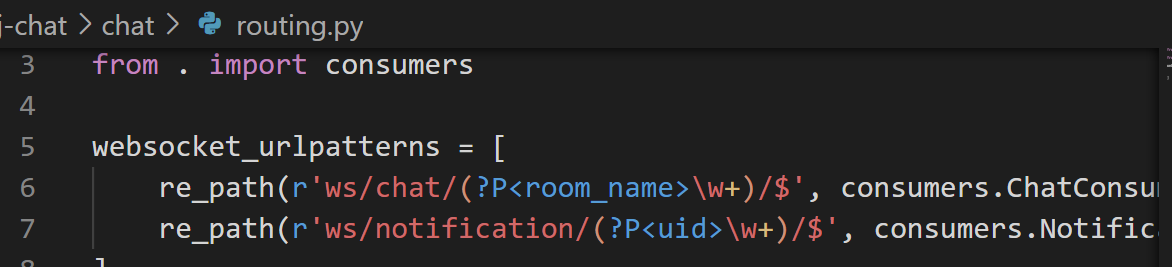
Django channel使用websocket发送消息，基于asgi协议。使用时需要在setting.py进行设置。



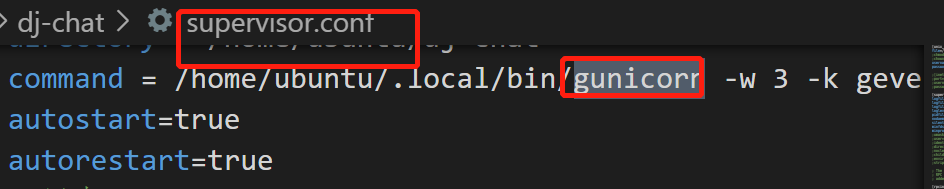


在使用时，在项目目录下新建consumer.py文件，处理聊天时的socket请求，与Django中的view.py类似，并新建routing.py处理url请求，与urls.py类似。





Gunicorn是一个被广泛使用的高性能的Python WSGI UNIX HTTP服务器，在Django中配置，可以适合于高并发的生产环境，用于处理http请求。supervisor是由python实现的一个进程管理工具，可以确保所管理的进程一直运行，当进程出现中断，supervisor会自动进行重启。当web项目存在多个进程需要处理时，方便统一管理，如服务器down机重启时自启动等。



## 3.3聊天功能模块

### 3.3.1文字聊天模块实现

本次实验使用seq2seq模型实现一个chatbot，训练数据来自Cornell电影对话语料库。主要的思路来自于PyTorch官方对于对话聊天的教程。

Cornell电影对话语料库是电影人物的对话数据，它包括：10,292对电影人物(一部电影有多个人物，他们两两之间可能存在对话)的220,579个对话；617部电影的9,035个人物；总共304,713个utterance(utterance是对话中的语音片段，不一定是完整的句子)。这个数据集是比较大并且多样的(diverse)，语言形式、时代和情感都有很多样。这样的数据可以使得我们的chatbot对于不同的输入更加鲁棒(robust)。

为了使用方便，将原始数据处理成一个新的文件，这个新文件的每一行都是用TAB分割问题(query)和答案(response)对。

接下来构建词典然后把问答句对加载到内存里。输入是一个句对，每个句子都是词的序列，但是机器学习只能处理数值，因此需要建立词到数字ID的映射。为此，定义了一个Voc类，它会保存词到ID的映射，同时也保存反向的从ID到词的映射。除此之外，它还记录每个词出现的次数，以及总共出现的词的个数。这个类提供addWord方法来增加一个词， addSentence方法来增加句子，也提供方法trim来去除低频的词。有了上面的Voc类我们就可以通过问答句对来构建词典了。

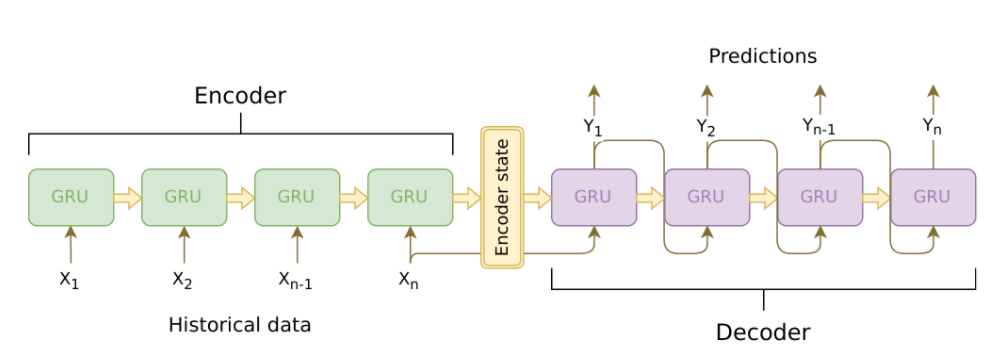
在构建之前首先需要进行一些预处理。需要使用函数unicodeToAscii来把unicode字符变成ascii，比如把à变成a。（这里的代码只是用于处理西方文字，如果是中文，这个函数直接会丢弃掉。）接下来把所有字母变成小写同时丢弃掉字母和常见标点(.!?)之外的所有字符。最后为了训练收敛，我们用函数filterPairs去掉长度超过MAX\_LENGTH的句子(句对)。最终原来共有221282个句对，经过处理后只保留了64271个句对。

截止目前构建了词典，并且对训练数据进行预处理并且滤掉一些句对，但是模型最终用到的是Tensor。最简单的办法是一次处理一个句对，那么上面得到的句对直接就可以使用。但是为了加快训练速度，尤其是重复利用GPU的并行能力，需要一次处理一个batch的数据。

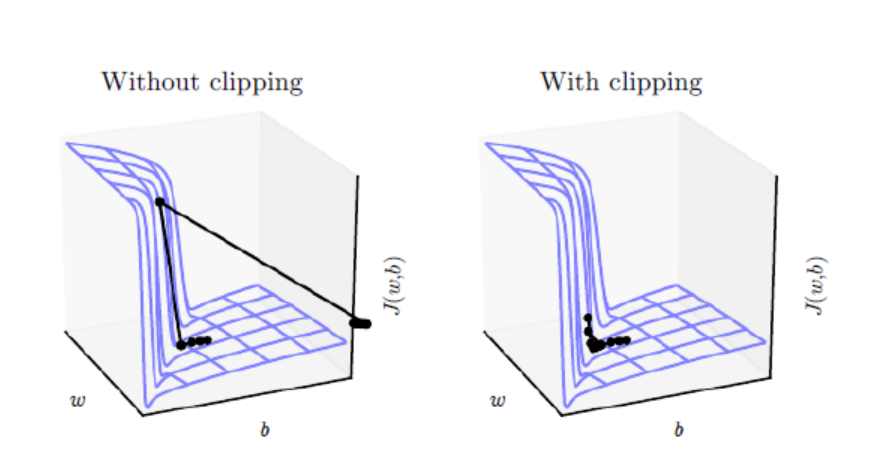
原始的输入通常是batch个list，表示batch个句子，因此自然的表示方法为(batch, max\_length)，这种表示方法第一维是batch，每移动一个下标得到的是一个样本的max\_length个词(包括padding)。因为RNN的依赖关系，在计算t+1时刻必须知道t时刻的结果，因此我们无法用多个核同时计算一个样本的forward。但是不同样本之间是没有依赖关系的，因此我们可以在根据t时刻batch样本的当前状态计算batch个样本的输出和新状态，然后再计算t+2时刻，…。为了便于GPU一次取出t时刻的batch个数据，我们通常把输入从(batch, max\_length)变成(max\_length, batch)，这样使得t时刻的batch个数据在内存(显存)中是连续的，从而读取效率更高。这个过程如下图所示，原始输入的大小是(batch=6, max\_length=4)，转置之后变成(4,6)。这样某个时刻的6个样本数据在内存中是连续的。

使用到了一些工具函数，例如：inputVar(), outputVar(), batch2TrainData()来实现上述处理。

接着进入核心的定义模型功能，seq2seq模型的输入是一个变长的序列，而输出也是一个变长的序列。而且这两个序列的长度并不相同。使用RNN来处理变长的序列，Sutskever等人的论文发现通过使用两个RNN可以解决这类问题。这类问题的输入和输出都是变长的而且长度不一样，包括问答系统、机器翻译、自动摘要等等都可以使用seq2seq模型来解决。其中一个RNN作为Encode（本实验使用多层的Gated Recurrent Unit(GRU)作为Encoder r），它把变长的输入序列编码成一个固定长度的context向量，这个向量包含了输入句子的语义。而第二个RNN作为Decoder，初始隐状态是Encoder的输出context向量，输入是(表示句子开始的特殊Token)，然后用RNN计算第一个时刻的输出，接着用第一个时刻的输出和隐状态计算第二个时刻的输出和新的隐状态，...，直到某个时刻输出特殊的(表示句子结束的特殊Token)或者长度超过一个阈值。Seq2Seq模型如下图所示。

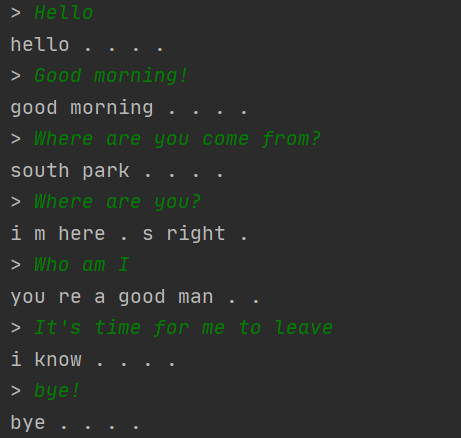


接着进行训练，先定义loss函数的计算，主要是计算mask的loss，使用交叉熵进行计算，实现交叉熵用到了gather函数。函数train实现一个batch数据的训练，训练的时候我们会限制Decoder的输出，使得Decoder的输出长度和”真实”答案一样长。但是我们在训练的时候如果让Decoder自行输出，那么收敛可能会比较慢，因为Decoder在t时刻的输入来自t-1时刻的输出。如果前面预测错了，那么后面很可能都会错下去。用到了梯度裁剪(gradient clipping)的技巧，个技巧通常是为了防止梯度爆炸(exploding gradient)，它把参数限制在一个范围之内，从而可以避免梯度的梯度过大或者出现NaN等问题。效果如下图所示：



最终训练部分的操作步骤为：  
 1) 把整个batch的输入传入encoder 2) 把decoder的输入设置为特殊的，初始隐状态设置为encoder最后时刻的隐状态 3) decoder每次处理一个时刻的forward计算 4) 如果是teacher forcing，把上个时刻的"正确的"词作为当前输入，否则用上一个时刻的输出作为当前时刻的输入 5) 计算loss 6) 反向计算梯度 7) 对梯度进行裁剪 8) 更新模型(包括encoder和decoder)参数

迭代训练后即可进行语言对话，效果如下图所示。



### 3.3.2 图片聊天模块实现

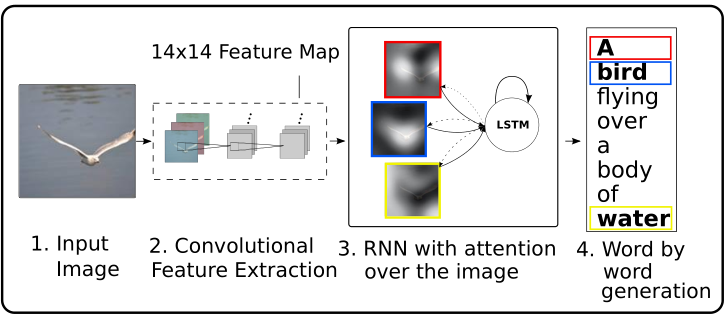
图片聊天模块的主要功能为图片标注，该功能的模型实现主要基于《Show, attend and tell: Neural image caption generation with visual attention》论文提出的思路和方法。模型采用encoder — decoder架构，训练数据集为MS COCO，模型的主要流程为如下：

1、Image Caption（图片描述）模型中，输入为图片数据(第一步）

2、经过CNN进行卷积提取图片特征信息最终形成图片的特征图信息（第二步，属于encoder过程）

3、attention对提取的特征图进行加强与抑制，作为后续进入LSTM模型的输入数据，不同时刻的attention数据会受到上一时刻LSTM模型输出数据而有所调整

4、LSTM模型最终输出文本信息（第四部分）



图表 1

在encoder阶段，模型使用CNN来提取 L 个 D 维的特征vector作为注释向量，每一个都对应图像的一个区域，如下式。



在decoder阶段，解码阶段用LSTM网络生成caption，集合为下式，其中C是句子长度，K是词表大小，y是各个词的one-hot编码所构成的集合。



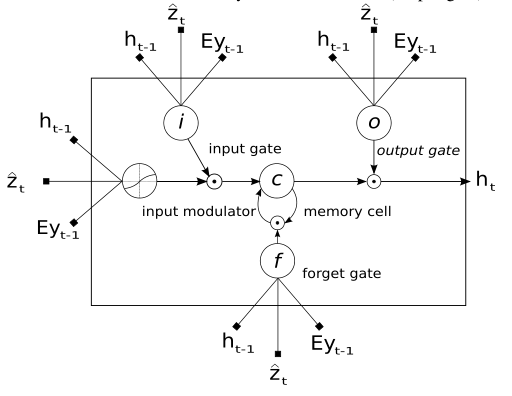
LSTM模型结构如图2所示：

图片聊天模块具体运行如下：

1、首先启动服务，通过接口与前端进行交互，不断检查请求并从前端获取到输入图片的数据。数据格式为base64。

2、将base64格式数据转换为jpg图片格式，之后运行模型对该输入图片进行图片标注。

3、将图片标注结果通过接口返回给前端，完成一次请求。



图表 2 LSTM模型架构

## 3.4数据库模块

## 3.4数据库模块

数据库使用Django自带的sqlite3数据库和redis数据库，两种数据库各司其职。

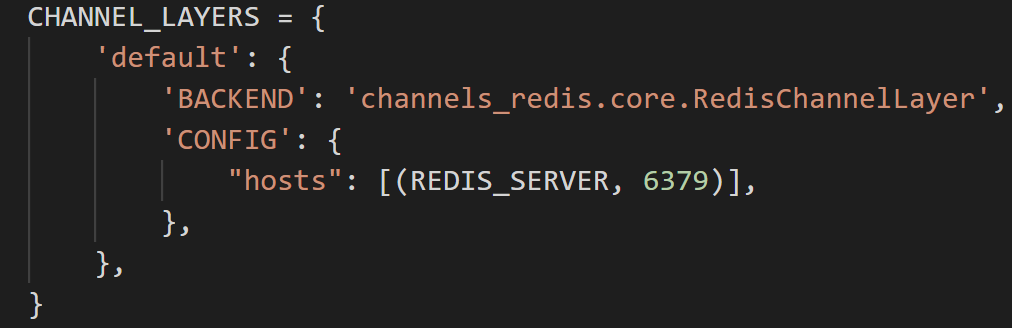
### 3.4.1 redis数据库

redis数据库是一个高性能的key-value数据库，该数据库的使用是为了做服务器缓存。对于数据非经常更新，但是需要经常从硬盘中读取数据，就会浪费服务器资源，拖慢响应速度或者数据更新频率很快，服务器负担比较大，而且若要将这些数据保存到数据库中，我们需要额外建立一张对应的表存储数据。在Django中建立表是要建立一个模型，但是处理起来问题调试麻烦，开发时间长久，因此使用redis内存数据库，将数据写入内存中，再进行读写。可以有效减少计算量，提高服务器响应速度。

而redis的功能非常强大，不单单是可以做缓存，还可以做排行榜、计算器、限速器、好友关系、简单消息队列、session服务器。在日常的项目开发中，当我们的数据库大起来之后，后台向数据库查找速度会变得很慢，这时候我们需要缓存来帮助我们提高浏览器的工作效率。而我们在项目中使用Django-channels也需要使用redis来进行通信。

若要在django数据库中使用redis数据库作为缓存，需要在django的setting.py文件中声明缓存，对redis缓存的连接池类型、连接池最大连接数、时间等进行设置





### 3.4.2 Sqlite3数据库

Sqlite3数据库是一种普遍使用的关系数据库，也是Django默认使用的数据库，用来对用户信息、聊天信息等进行存储。下面将介绍本项目中使用的数据库表。

聊天室表

当我们每次和一个朋友或者机器人对话时，就开了一个聊天室。聊天室表存储了这个多人聊天的一些信息。具体字段信息如下表所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段描述 | 数据类型 | 是否主键 | 是否可空 |
| id | 聊天室id（自增） | integer | 是 | 否 |
| room\_name | 房间名称 | varchar(64) | 否 | 否 |
| room\_description | 房间描述 | varchar(255) | 否 | 否 |
| img\_path | 头像地址 | varchar(255) | 否 | 否 |
| channel\_no | 房间频道号（unique） | varchar(8) | 否 | 否 |
| max\_number | 允许最大人数 | integer | 否 | 否 |
| ordering | 置顶权 | integer | 否 | 否 |
| data\_created | 创建实践 | datetime | 否 | 否 |
| date\_modified | 最新修改时间 | datetime | 否 | 否 |
| chat\_type | 房间类型 | varchar(16) | 否 | 否 |

Userprofile表

Userprofile表是介绍用户的基本信息，具体字段如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段描述 | 数据类型 | 是否主键 | 是否可空 |
| id | 表id（自增） | integer | 是 | 否 |
| nick\_name | 昵称 | varchar(64) | 否 | 否 |
| signature | 个性签名 | varchar(255) | 否 | 是 |
| unicode\_id | 唯一ID Unique | integer | 否 | 否 |
| user\_id | Unique 外键 | integer | 否 | 否 |
| city | 城市 | varchar(64) | 否 | 是 |
| qq\_number | QQ号码 | varchar(16) | 否 | 是 |
| is\_use\_qq\_img | 是否使用QQ头像 | bool | 否 | 否 |
| date\_created | 创建日期 | datetime | 否 | 否 |
| date\_modified | 修改日期 | datetime | 否 | 否 |
| img\_path | 头像地址 | varchar(255) | 否 | 否 |

Chatlog表

Chatlog表是聊天记录表，该表记录了聊天的发送者、接收者、时间等信息，具体字段如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段描述 | 数据类型 | 是否主键 | 是否可空 |
| id | 表id（自增） | Integer | 是 | 否 |
| chat\_datetime | 接收时间 | datetime | 否 | 是 |
| content | 聊天内容 | text | 否 | 是 |
| msg\_type | 消息类型 | varchar(16) | 否 | 是 |
| said\_together | 历史记录 | varchar(64) | 否 | 是 |
| status | 消息状态 | varchar(16) | 否 | 否 |
| said\_to\_id | 接收者 | integer | 否 | 是 |
| said\_to\_room\_id | 接收房间 | integer | 否 | 是 |
| who\_said\_id | 发送者 | integer | 否 | 否 |

Talklog表

Talklog是说说表，talk可以类比到微信发朋友圈和QQ发说说，在表中也记录说说的阅读量和点赞量，具体字段如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段描述 | 数据类型 | 是否主键 | 是否可空 |
| id | 表id（自增） | integer | 是 | 否 |
| content | 说说内容 | text | 否 | 是 |
| star | 点赞量 | integer | 否 | 否 |
| reading | 阅读量 | integer | 否 | 否 |
| profile\_id | 说说id | integer | 否 | 否 |
| date\_created | 创建时间 | datetime | 否 | 否 |
| date\_modified | 修改时间 | datetime | 否 | 否 |

History表

History表记录了聊天室的历史开启的具体信息，具体字段如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段描述 | 数据类型 | 是否主键 | 是否可空 |
| id | 表id（自增） | integer | 是 | 否 |
| ip | IP地址 | varchar(64) | 否 | 是 |
| city | 城市 | varchar(64) | 否 | 是 |
| device | 设备 | varchar(64) | 否 | 是 |
| date\_created | 创建日期 | datetime | 否 | 否 |
| count | 访问量 | integer | 否 | 否 |

auth\_user表

user表是在用户注册时就建立的一个表，里面包含了用户名、密码、邮箱等信息。为了安全，密码采用sha256存储，比md5和sha1更加安全可靠。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段描述 | 数据类型 | 是否主键 | 是否可空 |
| id | 表id（自增） | integer | 是 | 否 |
| password | 密码 | varchar(128) | 否 | 否 |
| last\_login | 最近登录时间 | datetime | 否 | 是 |
| is\_superuser | 是不是管理员 | bool | 否 | 否 |
| username | 用户名（unique） | varchar(150) | 否 | 否 |
| last\_name | 姓氏 | varchar(150) | 否 | 否 |
| email | 邮箱 | varchar(254) | 否 | 否 |
| is\_staff | 是不是员工 | bool | 否 | 否 |
| is\_active | 是否在线 | bool | 否 | 否 |
| date\_joined | 注册时间 | datetime | 否 | 否 |
| first\_name | 名字 | varchar(150) | 否 | 否 |

# 测试

## 4.1单元测试

单元测试是指对软件中最小可测试单元进行正确性检查和测试。本应用基于python语言开发，基本测试单元包括函数和类，此外用于本应用是一个网络应用，我们将对外暴露的API也作为一个基本的测试单元。单元测试能有效的确保代码质量并改善代码设计并确保代码重构不会引入新的问题。

Python中最基本的单元测试工具是unittest，使用该库可以对各个测试单元进行充分的安全性测试。对于应用中的网络请求我们使用mock进行替代。

## 4.2 安全测试

测试总体上分为单元测试和总体安全测试。安全测试使用360开发的webscan工具进行[[3]](#footnote-3)。

OWASP(Open Web Application Security Project)每年会发布一个web应用安全性测试的调研报告，其中会总结当年威胁性最大的十种漏洞，为web应用安全性测试提供指南。2020年OWASP总结了以下十大威胁[[4]](#footnote-4)：

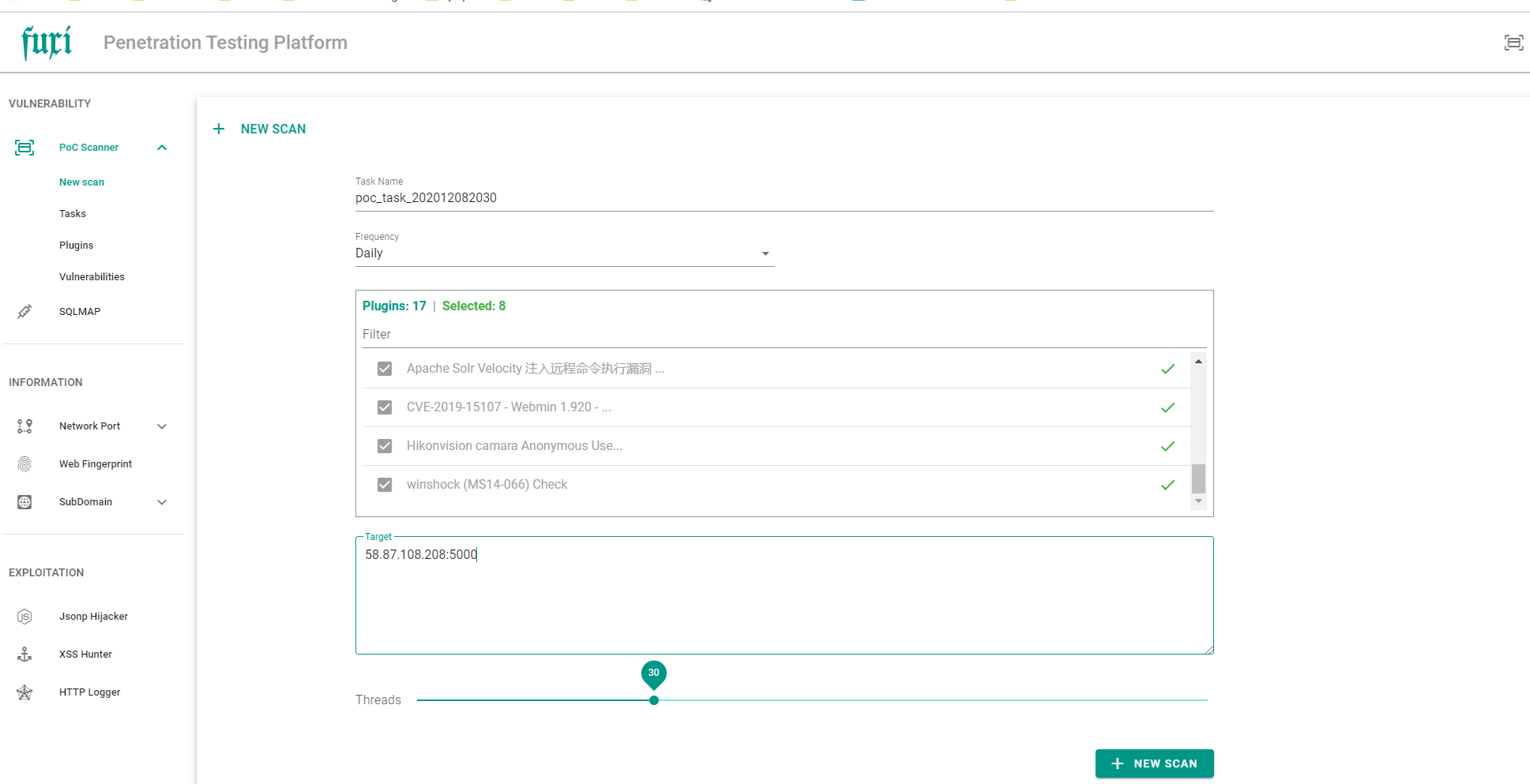
* + - 1. 注入漏洞。包含SQL、NOSQL、OS、LADP注入。后端命令或者查询生成程序不正确的处理不可信数据就可能发生
      2. 认证功能失效。应用授权认定或者会话管理功能不正确实现。
      3. 敏感数据泄露。敏感数据会在没有额外保护的情况下可能会被无意识泄露，例如没有采取加密技术，并且与浏览器进行交换时没有采取相应的预防措施。
      4. XML外部实体(XXE)。应用实现中XML解析器配置错误，无法限制外部实体解析。
      5. 访问控制失效。对于授权用户的允许进行的动作没有进行合适限制，导致攻击者可以访问未授权的资源、功能。
      6. 安全配置错误。此类错误是最严重的可见问题。包括操作系统、防火墙、中间件、底层链接库等的安全配置出错都有可能导致此类漏洞。
      7. 跨站脚本攻击(XSS)。当使用用户通过可插入HTML或者JavaScript甲苯的浏览器API提供的数据更新一个现有的web应用页面或者不进行合理的验证和转义就将一些不可信数据插入web页面。
      8. 不安全的反序列化。这类错误易导致远程代码执行、注入、提权等攻击。
      9. 使用了具有漏洞的组件。Web应用本身没有漏洞但是，但是运行在相同权限的应用模块有漏洞，攻击者可以通过暴露的攻击面直接利用该组件进行攻击。
      10. 日志记录不充分。日志和监视的不足，再加上事件响应的缺失或无效集成，使攻击者可以进一步攻击系统，并实现攻击的持久化，以转向更多系统以及篡改，提取或破坏数据。 大多数研究表明，发现大量持续时间超过200天以上的攻击，通常是由外部而不是内部流程或监视发现的。

这十大威胁，我们会在设计和测试时充分考虑并予以解决。但是解决了这十大威胁并不意味着我们的应用没有任何漏洞，但是我们尽可能缩小应用的攻击面。

对于注入漏洞，我们通过两种方式进行限制：django对于前端传回数据的过滤和转义；前端通过javascript代码进行验证。针对认证功能失效的问题，我们通过应用系统的设计进行避免并利用先进的密码算法来缩减攻击面。加密技术也可以解决敏感数据泄露的问题，应用敏感信息主要有两种：用户账户密码信息、用户聊天记录。用户账户信息通过加密传输，密文、加盐存储等等方式保护，用户聊天记录的保护本地不保护，云端保存。聊天记录传输采用公钥加密方式传递分组加密密钥，实际传递不用分组加密算法。加密算法不采用DES等加密强度低的算法。XML外部实体注入攻击使用和注入攻击相似的防御手段，通过过滤用户提交的XML数据防御。访问控制失效和安全配置错误问题都可以通过仔细配置操作系统、防火墙的安全功能避免。跨站脚本攻击本质上就是诸如攻击，解决方式是类似的。不安全的反序列化可能由多种原因产生，针对web传输中的序列化安全问题可以通过JWT(JSON WEB TOKEN)完整性验证解决，针对应用中的序列化问题除了完整性验证还必须避免使用有漏洞的序列化组件。日志记录则需要仔细设计应用日志，使用功能完善的日志库，在应用重要节点都使用日志记录，并定期检查日志。

我们对于应用安全漏洞的防治分为三个方面，预防、发现和攻击拦截。前两者通过伏羲网络安全检测工具实现，后者通过Selinux实现。

伏羲fuxi-sanner是一个开源的web应用网络安全检测工具[[5]](#footnote-5)。该工具能自动定时的扫描网络应用中的各种漏洞，通过在该工具中创建定时扫描任务，将安全测试、漏洞发现持续集成到了应用开发中。



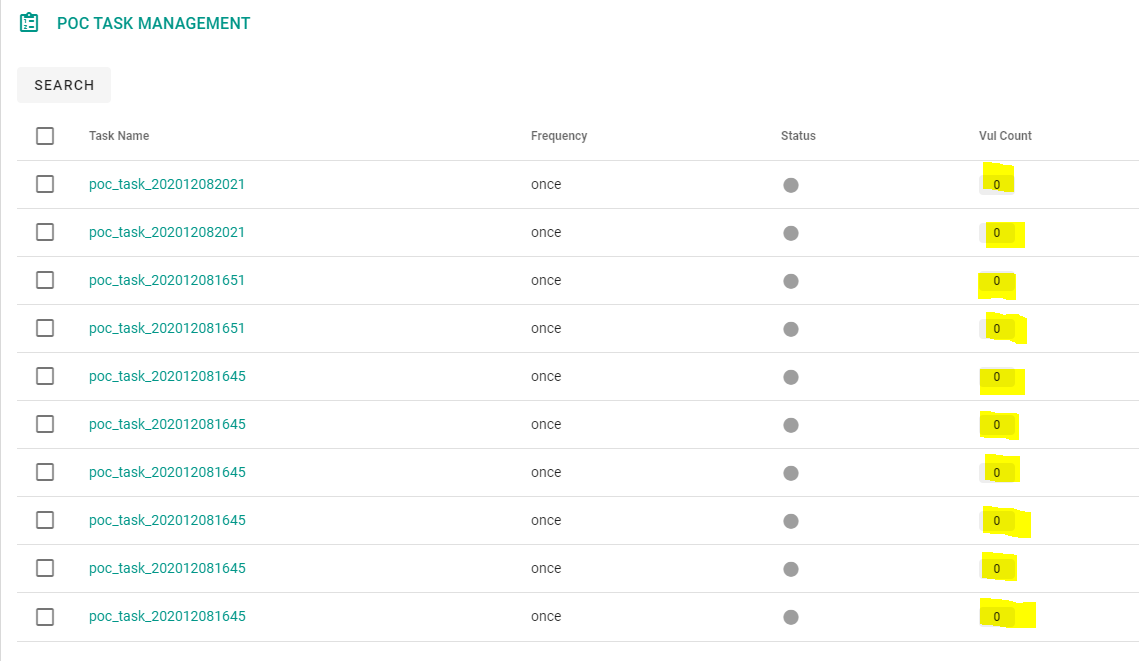
图表 3在伏羲漏洞检测工具中创建定时检查任务

在该应用的后台，我们可以检查漏洞检查情况，目前应用中没有发现任何严重的漏洞，如图 所示。

伏羲漏洞检测框架通过爬虫模拟用户行为并进行定向注入进行检测，能够发现上面列举的OWASP十大漏洞中的绝大多数漏洞。

伏羲漏洞检测工具的作用是

Selinux全称强制访问控制安全系统，是linux历史上最杰出的内核信息安全系统，是linux



图表 4伏羲漏洞检测工具报告

安全模块中来说，Selinux的功能是最全面的，测试也是最充分。即使对于未知的漏洞，Selinux依然有很好得防护效果。

## 4.3功能模块测试

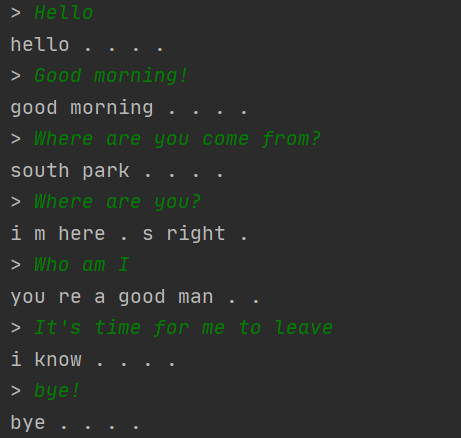
这里的功能模块主要是指两个神经网络模块。

神经网络分为两块：第一文本信息自然语言处理模块，第二图像信息处理模块。两个神经网络是采用相同的测试思路，通过构造特定的测试数据集，然后通过比较神经网络输出和正确答案得到准确率。



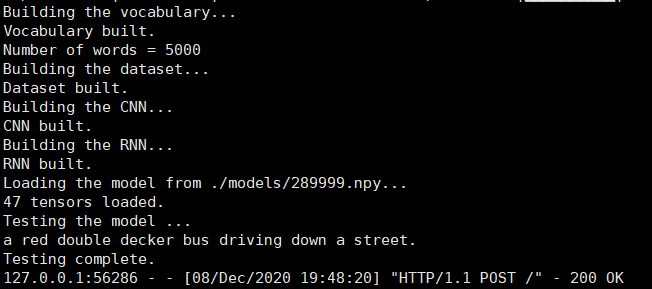
图表 5图片信息理解模块

如上图所示是我们应用的图片信息神经网络的一个输出示例，该模块正确识别图片中的环境信息以及各项物体，精确到了各种颜色。本应用采用的模型具有良好的效果，其BLEU指标达到了目前世界先进水平，在各个数据得分均在67分左右。



图表 6神经网络对话

同时我们也对各个模块进行了模块间的连接测试。模块间通过restful API进行连接。



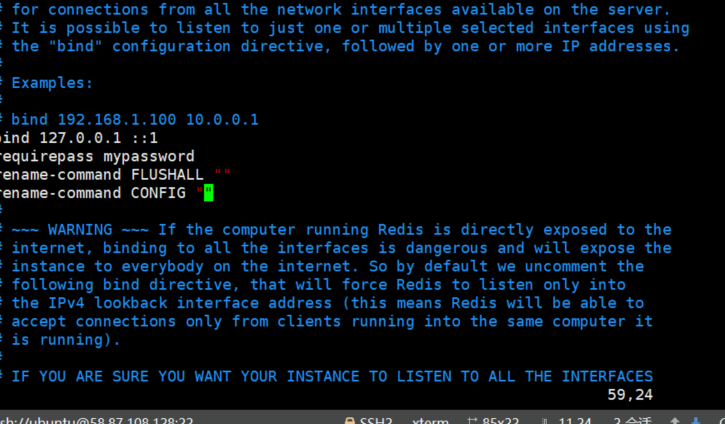
图表 7 功能模块连接测试

## 4.4数据库安全性测试

我们的数据库采用了redis，redis是一个高性能的键值数据库，但是redis设计的时候只是考虑在可信环境的可信机器访问，在互联网环境下，可能有攻击者能够通过TCP直接连接，redis数据库的安全防护主要通过安全配置实现。

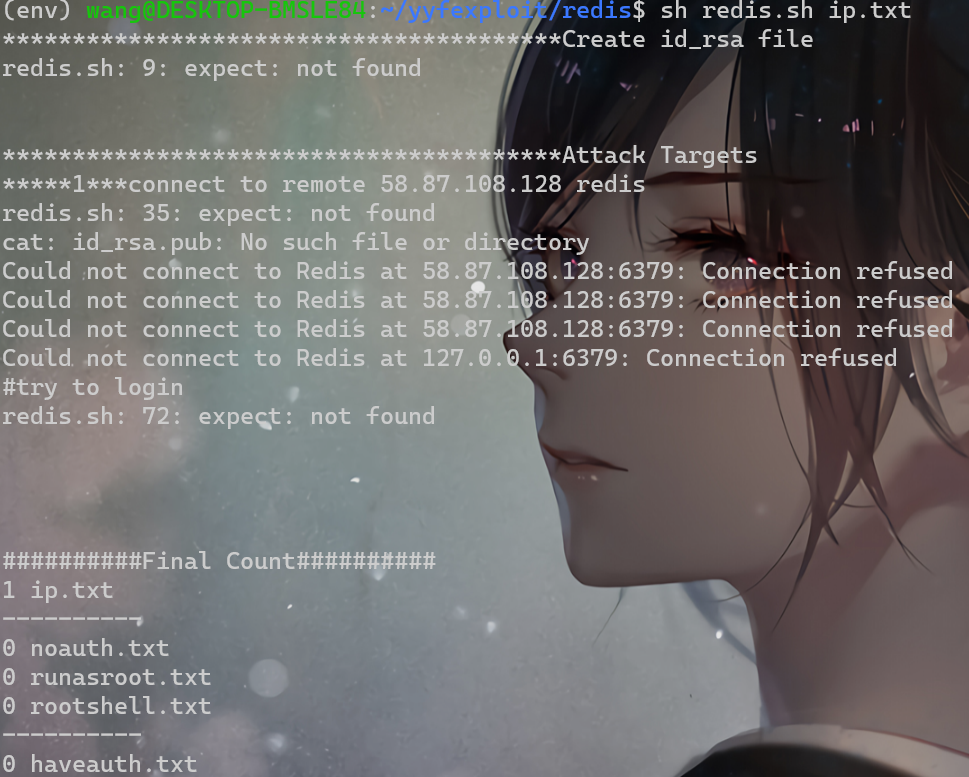
我们主要考虑了以下redis安全配置：

1. 禁用远程修改DB文件
2. 使用低权限运行redis服务
3. 为redis添加多重密码验证
4. 禁用外网连接redis



图表 8redis安全配置

我们通过yyfexploit工具（redis漏洞exploit工具）对服务器部署的redis数据库安全设置进行了安全测试[[6]](#footnote-6)，图中我们可以看出，我们无法通过外部网络连接到服务器的redis数据库实例中，也就无法进行渗透。



# 项目进行遇到的困难与解决方案

数据源。神经网络模型的效果本质上是由数据决定的，不好的数据源往往不能训练出满足功能需求的神经网络模型。在训练seq2seq模型时，我们在选择尝试了目前可以获得大量中文对话数据集包括不限于：华语电影对话数据集、小黄鸡语料等，发现效果均不是很好。反之对于能找到的英文语料，反而后期能训练处一个比较令人满意的效果。

云部署。应用的云端部署尤其是部署到租赁的云计算厂商提供的云服务器时，由于与本地环境的巨大差异尤其是需要自己管理许多底层依赖库的依赖关系导致部署出现了一些困难通过查阅云计算厂商提供相关文档和网络上相关博客，并通过大胆实验，我们成功克服了相关困难，将应用成功部署到了云上。

# 6 项目协作与部署

## 6.1项目协作

Git github 文档

项目分工如下

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 完成时间点 | 王鸿森 | 王伊梁 | 王柯林 | 易士程 | 王子妍 |
| 需求分析 | 2020.10.9 | √ | √ | √ | √ | √ |
| 功能分析 | 2020.10.9 | √ | √ | √ | √ | √ |
| 原型分析 | 2020.10.9 | √ | √ | √ | √ | √ |
| 前端实现 | 2020.10.30 |  |  |  | √ |  |
| 后端实现 | 2020.11.15 |  |  |  | √ |  |
| 数据库实现 | 2020.11.15 |  |  | √ |  |  |
| 对话神经网络心功能实现 | 2020.11.15 |  | √ |  |  | √ |
| 测试 | 2020.11.30 | √ | √ | √ | √ | √ |
| 部署 | 2020.12.4 | √ |  |  |  |  |
| 结题报告撰写 | 2020.12.5 | √ | √ | √ | √ | √ |

项目协作基于git和github。代码托管网址为：<https://github.com/WangHongSen/ChatBot>。

项目总体开发是具有敏捷的，在完成一个系统demo后进行了快速迭代，这个过程中通过微信群进行交流并定期线下面对面交流。

项目过程中我们确定了以下规范：

1. git同步文件前后端、数据操作逻辑分离，基于MVC框架实现
2. 实行规范化的git commit push merge，过程中保持良好的log
3. 项目后端以及核心功能实现使用python开发，开发过程中使用谷歌python开发规范
4. [[7]](#footnote-7)项目前端采用JavaScript和HTML、CSS技术，遵从谷歌前端开发规范[[8]](#footnote-8)
5. 测试驱动开发，编码同时进行单元测试

## 6.2项目部署

项目部署在腾讯云服务器（X86\_64架构）

操作系统： linux (内核版本4.15.0-118-generic)

操作系统：Ubuntu 18.04.4 LTS (Bionic Beaver)

基础支撑软件版本：cpython 3..6.9

django2.24

pytorch 1.50+cpu

tensorflow 1.4

nginx 2.14

硬件环境：

CPU: Intel(R) Xeon(R) CPU E5-26xx v4

RAM: 2GB

硬盘: 50G SSD

访问网址：http://58.87.108.128:5000

代码托管网址：https://github.com/WangHongSen/ChatBot

# 项目总结

我们基于深度学习技术设计完成了一个在线的聊天机器人应用，在联网条件下，该机器人嫩能够实时和用户交流，机器人能够记忆用户输入，并能处理多种形式的输入。针对特定信息，我们的实现的机器人能够给与正确的回答。同时我们机器人能够处理开放域问题。项目进行了完整的测试并进行了部署，在实际场合中进行了测试。

1. https://en.wikipedia.org/wiki/Chatbot [↑](#footnote-ref-1)
2. https://blog.csdn.net/qq\_38796548/article/details/105456435 [↑](#footnote-ref-2)
3. https://webscan.360.cn/ [↑](#footnote-ref-3)
4. https://owasp.org/www-project-top-ten/ [↑](#footnote-ref-4)
5. https://github.com/jeffzh3ng/fuxi [↑](#footnote-ref-5)
6. https://github.com/qingxp9/yyfexploit [↑](#footnote-ref-6)
7. https://google.github.io/styleguide/pyguide.html [↑](#footnote-ref-7)
8. https://zh-google-styleguide.readthedocs.io/en/latest/ [↑](#footnote-ref-8)