Intelligent Log System

Team Members:
Lu Qinwen A0261847W
Cai Zimo A0261628B
Hu Zhiqing A0261637B
Yu Hanchun A0261716E
Cai Zheng A0261940H

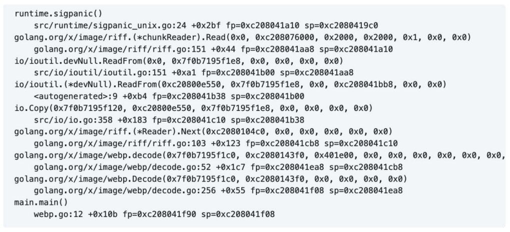
Contents

1. Introduction

1.1 Background

1.1.1 Introduction of Intelligent Logging Systems

The log system is a mechanism for recording and storing runtime information of a software system, which is mainly used to track the running status of the system, diagnose problems, and monitor performance. It usually records log information about applications, operating systems, networks, and databases, such as error messages, warning messages, operation logs, and performance statistics. Back-end developers, operation and maintenance personnel, and even business personnel often use the log system for troubleshooting, performance monitoring, security auditing, and statistical analysis.



```
runtime.sigpanic()
    src/runtime/sigpanic_unix.go:24 +0x2bf fp=0xc208041a10 sp=0xc2080419c0
golang.org/x/image/riff.(*chunkReader).Read(0x0, 0xc208076000, 0x2000, 0x2000, 0x1, 0x0, 0x0)
    golang.org/x/image/riff/riff.go:151 +0x44 fp=0xc208041aa8 sp=0xc208041a10
io/ioutil.devNull.ReadFrom(0x0, 0x7f0b7195f1e8, 0x0, 0x0, 0x0, 0x0)
    src/io/ioutil/ioutil.go:151 +0xa1 fp=0xc208041b00 sp=0xc208041aa8
io/ioutil.(*devNull).ReadFrom(0xc20800e550, 0x7f0b7195f1e8, 0x0, 0xc208041bb8, 0x0, 0x0)
    <autogenerated>:9 +0xb4 fp=0xc208041b38 sp=0xc208041b00
io.Copy(0x7f0b7195f120, 0xc20800e550, 0x7f0b7195f1e8, 0x0, 0x0, 0x0, 0x0)
    src/io/io.go:358 +0x183 fp=0xc208041c10 sp=0xc208041b38
golang.org/x/image/riff.(*Reader).Next(0xc2080104c0, 0x0, 0x0, 0x0, 0x0, 0x0)
    golang.org/x/image/riff/riff.go:103 +0x123 fp=0xc208041cb8 sp=0xc208041c10
golang.org/x/image/webp.decode(0x7f0b7195f1c0, 0xc2080143f0, 0x401e00, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
    golang.org/x/image/webp/decode.go:52 +0x1c7 fp=0xc208041ea8 sp=0xc208041cb8
golang.org/x/image/webp.Decode(0x7f0b7195f1c0, 0xc2080143f0, 0x0, 0x0, 0x0, 0x0)
    golang.org/x/image/webp/decode.go:256 +0x55 fp=0xc208041f08 sp=0xc208041ea8
main.main()
    webp.go:12 +0x10b fp=0xc208041f90 sp=0xc208041f08
```

Fig.1.1 Example of log file

Before the emergence of intelligent logs, the traditional log processing method usually checks log files through manual analysis or scripts. This method becomes more and more difficult and time-consuming when faced with a large amount of complex log data. However, traditional log systems often require developers to spend a long time locating and analyzing code, and then draw conclusions that are not always correct. In addition, the development of some intelligent log systems on the market is still not perfect, and there are still some loopholes in performance and scope of application, so our team proposed an intelligent log system to solve these problems.

1.1.2 Tasks and Functions of the Smart Logging System

The intelligent log system can realize functions such as data collection, processing, storage, search and visualization. Taking ELK as an example, it can collect, filter, and clear various logs such as system logs, website logs, and application system logs, and then store them centrally for real-time retrieval and analysis. These functions are respectively undertaken and realized by the three components of ELK Stack: Elasticsearch, Logstash and Kibana.

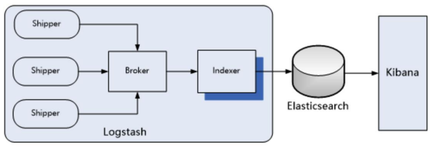


Fig.1.2 Components of the ELK Stack

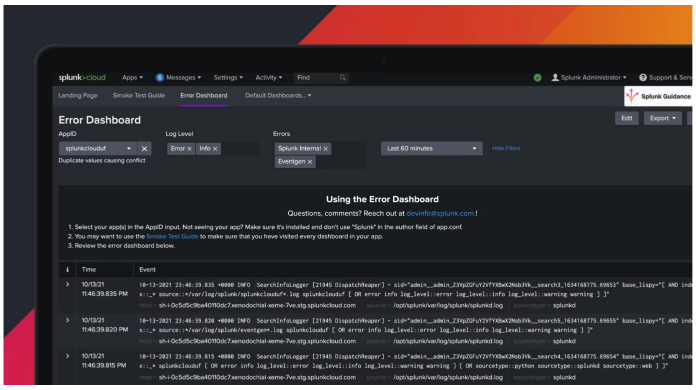
1.2 Business Value

In actual use, the log system is usually used by users with different roles such as developers, operation and maintenance personnel, and business personnel. Developers mainly use the log system to diagnose problems, debug programs, etc.; operation and maintenance personnel mainly use the log system to monitor the system running status, troubleshoot, etc.; business personnel mainly use the log system for data analysis and business decision-making.

1.3 Competitive Analysis

At present, there are mainly three types of intelligent log systems that account for a relatively large market share: ELK, Splunk, and Greylog. They all have varying degrees of pros and cons.

Example for Splunk, which accounts for the largest market share. This is a commercialized intelligent log system, which can summarize, analyze and visualize data from multiple data sources, and provides powerful query and reporting functions. According to market research reports, Splunk has been in the leading position in the intelligent logging system market. This is because Splunk's pricing strategy is usually billed based on the size and usage of data processing and storage. For small and medium-sized enterprises and individual users, the price may be too high and unaffordable. In addition, the threshold for Splunk configuration and use is relatively high. Beginners may need a certain amount of time and learning costs to master its functions and operations.



Error Dashboard

AppID: splunkclouduf
Duplicate values causing conflict

Log Level: Error, Info

Errors: Splunk Internal, Eventgen

Last 60 minutes

Using the Error Dashboard

Questions, comments? Reach out at devinfo@splunk.com!

1. Select your app(s) in the AppID input. Not seeing your app? Make sure it's installed and don't use "Splunk" in the author field of app.conf.
2. You may want to use the Smoke Test Guide to make sure that you have visited every dashboard in your app.
3. Review the error dashboard below.

```
i  Time           Event
>  10/13/21       10-13-2021 23:46:39.835 +0000 INFO  SearchInfoLogger [21945 DispatchReaper] - sid="admin__admin_Z3VpZGFuY2VfYXBwX2Nsb3Vk__search3_1634168775.69653" base_lispy="[ AND inde
   11:46:39.835 PM x::_* source::*/var/log/splunk/splunkclouduf*.log splunkclouduf [ OR error info log_level::error log_level::info log_level::warning warning ] ]"
                  host = sh-i-0c5d5c9ba40110dc7.xenodochial-xeme-7ve.stg.splunkcloud.com | source = /opt/splunk/var/log/splunk/splunkd.log | sourcetype = splunkd
>  10/13/21       10-13-2021 23:46:39.820 +0000 INFO  SearchInfoLogger [21945 DispatchReaper] - sid="admin__admin_Z3VpZGFuY2VfYXBwX2Nsb3Vk__search5_1634168775.69655" base_lispy="[ AND inde
   11:46:39.820 PM x::_* source::*/var/log/splunk/eventgen*.log splunkclouduf [ OR error info log_level::error log_level::info log_level::warning warning ] ]"
                  host = sh-i-0c5d5c9ba40110dc7.xenodochial-xeme-7ve.stg.splunkcloud.com | source = /opt/splunk/var/log/splunk/splunkd.log | sourcetype = splunkd
>  10/13/21       10-13-2021 23:46:39.815 +0000 INFO  SearchInfoLogger [21945 DispatchReaper] - sid="admin__admin_Z3VpZGFuY2VfYXBwX2Nsb3Vk__search4_1634168775.69654" base_lispy="[ AND inde
   11:46:39.815 PM x::_* splunkclouduf [ OR error info log_level::error log_level::info log_level::warning warning ] [ OR sourcetype::python sourcetype::splunkd sourcetype::web ] ]"
                  host = sh-i-0c5d5c9ba40110dc7.xenodochial-xeme-7ve.stg.splunkcloud.com | source = /opt/splunk/var/log/splunk/splunkd.log | sourcetype = splunkd
```

Fig.1.3 Splunk User Interface

To sum up, we can find that the processing performance and compatibility of the above products for large-scale data are uneven. The common point is that the use is complicated and the functions are complicated. Moreover, these products integrate a lot of complex functions. For a single log retrieval requirement, the requirements for hardware configuration are high, the use price is expensive, and the learning cost is high, which is not suitable for small and medium-sized enterprises and individuals. Therefore, we want to design and develop a more lightweight and easy-to-use intelligent log system, which is convenient for small and medium-sized enterprises and individual programmers.

2. Feature Engineer

2.1 Data Required

The collection of data sets needs to fully consider the feature engineering requirements of the log analysis system. When doing data collection, we need to consciously take the most useful information related to the characteristics of the logs and organize it into a format acceptable to our model.

First, we need to clearly identify the feature engineering requirements of the log analysis system. This includes determining what key information we want to extract from the logs and how to turn it into meaningful features.

During the data collection process, we also pay attention to the integrity and consistency of the data. Ensure that the collected log data can cover all key aspects and provide sufficient sample size to support subsequent analysis and modeling work. Additionally, we need to clean and preprocess the data to remove invalid or duplicate log records and transform them into the usable format we need.

Finally, we want to organize the collected log data into a usable format for the model. This may involve converting the log data into an appropriate feature representation, such as numerical, categorical, or textual features. Depending on the situation, we may need to perform feature engineering, such as feature selection, dimensionality reduction, or construct derived features, to extract useful information for our model.

2.2 Data Resource

Usually, there are the following main ways to obtain the data sets required for transfer learning using large language models:

Open source project logs or public error databases: Error logs for many open source projects are publicly available, and error log data for these projects can be collected and organized. Open source projects on GitHub may expose their bug logs or issue reports, which can be used as training data. And there may be some open source error databases that contain error information from various programming languages and frameworks. These databases may be maintained by the developer community and can be searched to obtain relevant error message data.

Developer communities and forums: Developer communities and forums that participate in discussions and exchanges, such as Stack Overflow, GitHub Issues, etc., provide a wealth of information about Go language errors and solutions. These questions and answer data can be collected for the training of the question answering model.

After investigation, due to the huge amount of data information in the log, it is complex and redundant. Known open source projects have no public datasets available that we can use, which makes our data required tricky.

2.3 Data Mining and Generation

2.3.1 Data Crawling

We use the Scrapy crawler framework to obtain data related to the Go language on Stack Overflow. During crawling, we pay special attention to log-related issues, filtering by adding tags such as [go][log] or [go][error]. Such tags can help us locate issues related to logging and error handling to obtain data relevant to our work.

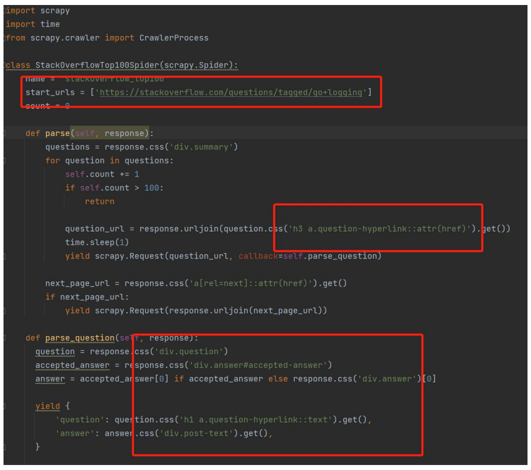
When selecting items to crawl, we selected 100 items based on frequency. This strategy ensures that we get broad data coverage covering different problems and solutions. Such diversity can help us construct a more comprehensive training dataset to improve the robustness and accuracy of our models.

Based on this, we divide the content of the dataset into three categories:

- Instruction: Crawl by Title: Most titles are the most intuitive questions from users
- Input: relevant content: can be information in the log segment
- Output: Select the answer with the highest praise rate, which has a high probability of being the answer that can finally help users solve the problem

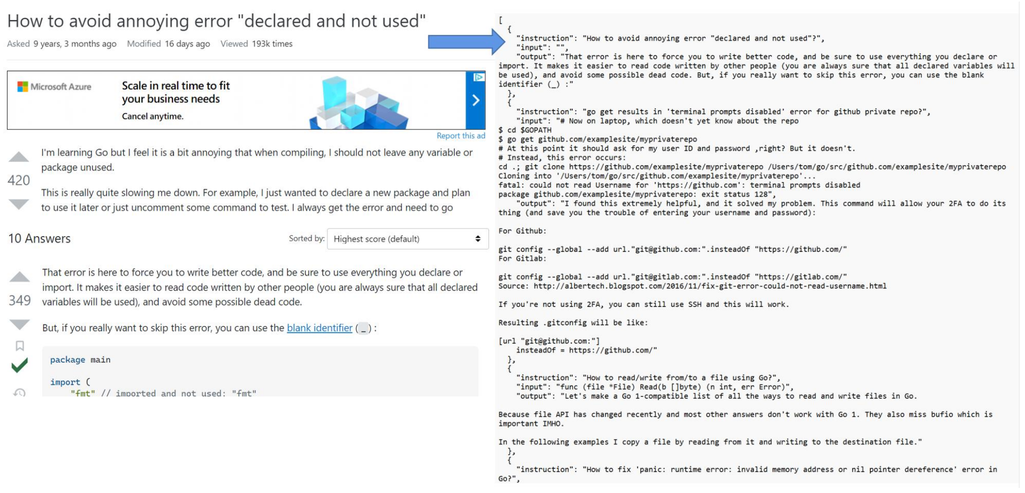
During data collection, we performed several preprocessing steps to ensure the quality of our data. One of these steps is to delete data that is not relevant to the log. We identify and exclude those issues that are not relevant to our work to ensure that our training dataset contains only those related to logging and error handling. And we also manually reviewed the generated data to delete the data that did not match the Q&A information or had poor relevance.

In addition, we also performed redundant identifier removal on the text. By removing redundant identifiers, we can reduce noise and unnecessary information in the input data, thereby improving the effect and efficiency of model training. The crawling result after processing is shown in Figure 2.2.



```
import scrapy
import time
from scrapy.crawler import CrawlerProcess

class StackOverflowTop100Spider(scrapy.Spider):
    name = 'stackoverflow_top100'
    start_urls = ['https://stackoverflow.com/questions/tagged/go+logging']
    count = 0

    def parse(self, response):
        questions = response.css('div.summary')
        for question in questions:
            self.count += 1
            if self.count > 100:
                return

            question_url = response.urljoin(question.css('h3 a.question-hyperlink::attr(href)').get())
            time.sleep(1)
            yield scrapy.Request(question_url, callback=self.parse_question)

        next_page_url = response.css('a[rel=next]::attr(href)').get()
        if next_page_url:
            yield scrapy.Request(response.urljoin(next_page_url))

    def parse_question(self, response):
      question = response.css('div.question')
      accepted_answer = response.css('div.answer#accepted-answer')
      answer = accepted_answer[0] if accepted_answer else response.css('div.answer')[0]

      yield {
          'question': question.css('h1 a.question-hyperlink::text').get(),
          'answer': answer.css('div.post-text').get(),
      }
```

Fig.2.1 Method of crawling


How to avoid annoying error "declared and not used"

Asked 9 years, 3 months ago Modified 16 days ago Viewed 193k times

420

I'm learning Go but I feel it is a bit annoying that when compiling, I should not leave any variable or package unused.

This is really quite slowing me down. For example, I just wanted to declare a new package and plan to use it later or just uncomment some command to test. I always get the error and need to go

10 Answers

349

That error is here to force you to write better code, and be sure to use everything you declare or import. It makes it easier to read code written by other people (you are always sure that all declared variables will be used), and avoid some possible dead code.

But, if you really want to skip this error, you can use the blank identifier (_) :

```
package main

import (
    "fmt" // imported and not used: "fmt"
```

```
[
  {
    "instruction": "How to avoid annoying error "declared and not used"?",
    "input": "",
    "output": "That error is here to force you to write better code, and be sure to use everything you declare or
import. It makes it easier to read code written by other people (you are always sure that all declared variables will
be used), and avoid some possible dead code. But, if you really want to skip this error, you can use the blank
identifier (_) :"
  },
  {
    "instruction": "go get results in 'terminal prompts disabled' error for github private repo?",
    "input": "# Now on laptop, which doesn't yet know about the repo
$ cd $GOPATH
$ go get github.com/examplesite/myprivaterepo
# At this point it should ask for my user ID and password ,right? But it doesn't.
# Instead, this error occurs:
cd .; git clone https://github.com/examplesite/myprivaterepo /Users/tom/go/src/github.com/examplesite/myprivaterepo
Cloning into '/Users/tom/go/src/github.com/examplesite/myprivaterepo'...
fatal: could not read Username for 'https://github.com': terminal prompts disabled
package github.com/examplesite/myprivaterepo: exit status 128",
    "output": "I found this extremely helpful, and it solved my problem. This command will allow your 2FA to do its
thing (and save you the trouble of entering your username and password):

For Github:

git config --global --add url."git@github.com:".insteadOf "https://github.com/"
For Gitlab:

git config --global --add url."git@gitlab.com:".insteadOf "https://gitlab.com/"
Source: http://albertech.blogspot.com/2016/11/fix-git-error-could-not-read-username.html

If you're not using 2FA, you can still use SSH and this will work.

Resulting .gitconfig will be like:

[url "git@github.com:"]
    insteadOf = https://github.com/"
  },
  {
    "instruction": "How to read/write from/to a file using Go?",
    "input": "func (file *File) Read(b []byte) (n int, err Error)",
    "output": "Let's make a Go 1-compatible list of all the ways to read and write files in Go.

Because file API has changed recently and most other answers don't work with Go 1. They also miss bufio which is
important IMHO.

In the following examples I copy a file by reading from it and writing to the destination file."
  },
  {
    "instruction": "How to fix 'panic: runtime error: invalid memory address or nil pointer dereference' error in
Go?",
```

Fig.2.2 Crawling Result

2.3.2 Data Generation

Generating Data Engineering is a tool for creating log error-related question answering datasets. The project is based on the Go language and uses OpenAI's ChatGPT model to generate questions and answers.

Dataset Goal: The goal of the project is to generate a Log Error Question Answering dataset that includes questions, inputs, and outputs. Each data point includes a Go language common problem (instruction), the specific error log text (input), and the answer to the Go problem (output).

Model preparation: The project uses OpenAI's ChatGPT model, which is a large-scale language model based on the GPT-3.5 architecture. The model learns a wealth of linguistic knowledge during training and is able to generate text in natural language.

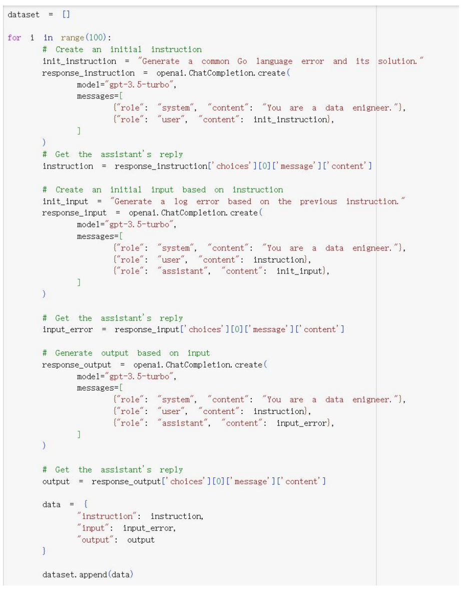
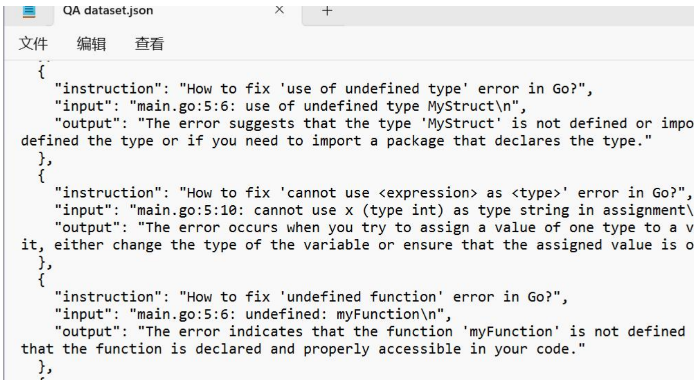

```
dataset = []

for i in range(100):
    # Create an initial instruction
    init_instruction = "Generate a common Go language error and its solution."
    response_instruction = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a data enigneer."},
            {"role": "user", "content": init_instruction},
        ]
    )
    # Get the assistant's reply
    instruction = response_instruction['choices'][0]['message']['content']

    # Create an initial input based on instruction
    init_input = "Generate a log error based on the previous instruction."
    response_input = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a data enigneer."},
            {"role": "user", "content": instruction},
            {"role": "assistant", "content": init_input},
        ]
    )

    # Get the assistant's reply
    input_error = response_input['choices'][0]['message']['content']

    # Generate output based on input
    response_output = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a data enigneer."},
            {"role": "user", "content": instruction},
            {"role": "assistant", "content": input_error},
        ]
    )

    # Get the assistant's reply
    output = response_output['choices'][0]['message']['content']

    data = {
        "instruction": instruction,
        "input": input_error,
        "output": output
    }

    dataset.append(data)
```

Fig.2.3 Method of generation

Data generation process: The process of generating data is automatically completed by the project. The specific steps are as follows: The project first defines a series of common Go language log error problems (instructions), which often appear in the log. Then, the project traverses these problems one by one, and generates a specific error log text (input) for each problem. Next, the project uses the ChatGPT model to take the question and input as input, and generates the corresponding answer (output) by calling the model. Finally, engineering combines the problem, input, and output into a complete data point, and repeats the above steps until the desired number of data points are generated.

Data format: The generated data is output in JSON format. Each data point contains three fields: instruction, input, and output, which represent the content of the question, input, and output, respectively. This structured data format facilitates subsequent processing and analysis.



```
  {
    "instruction": "How to fix 'use of undefined type' error in Go?",
    "input": "main.go:5:6: use of undefined type MyStruct\n",
    "output": "The error suggests that the type 'MyStruct' is not defined or impo
defined the type or if you need to import a package that declares the type."
  },
  {
    "instruction": "How to fix 'cannot use <expression> as <type>' error in Go?",
    "input": "main.go:5:10: cannot use x (type int) as type string in assignment\
    "output": "The error occurs when you try to assign a value of one type to a v
it, either change the type of the variable or ensure that the assigned value is o
  },
  {
    "instruction": "How to fix 'undefined function' error in Go?",
    "input": "main.go:5:6: undefined: myFunction\n",
    "output": "The error indicates that the function 'myFunction' is not defined
that the function is declared and properly accessible in your code."
  },
```

Fig.2.4 Example of generated data

2.4 Data Resource

In order to better fit the data to the model, according to the above mentioned, we used text analysis to extract text and delete redundant identifiers. And the data quality is manually reviewed, and the data with low correlation with the log is deleted. Finally, we integrate the data generated by the two self-collected methods. And divide the data into training sets and test sets to prepare the data for model training.

3. Methods

3.1 Question answering based on Alpaca LoRA

3.1.1 Background

With the emergence of large language models in recent years, such as the GPT, BERT, LLaMa. They have shown powerful performance across various NLP tasks. Therefore, we are trying to study and explore if there's a way to train our own models using LLMs to meet our logging QA requirement.

However, the most effective large language models are not open-source, making them inaccessible to many developers and researchers.

3.1.2 Introduction

Alpaca is an instruction-following model developed by Stanford University, which is fine-tuned from the LLaMA 7B model released by Meta. In the paper, the researchers used the Self-Instruct technique, and generated a dataset containing 52K instruction-following dataset for model fine-tuning. The performance of Alpaca 7B in instruction-following tasks is similar to OpenAI's text-davinci-003 (GPT 3.5), but with a smaller model size and lower training costs.

3.1.3 Data Generation: Self-Instruct

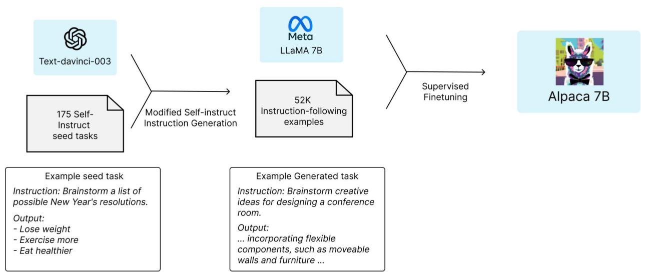

Text-davinci-003
175 Self-Instruct seed tasks
Modified Self-instruct Instruction Generation
Meta LLaMA 7B
52K Instruction-following examples
Supervised Finetuning
Alpaca 7B
Example seed task
Instruction: Brainstorm a list of possible New Year's resolutions.
Output:
- Lose weight
- Exercise more
- Eat healthier
Example Generated task
Instruction: Brainstorm creative ideas for designing a conference room.
Output:
... incorporating flexible components, such as moveable walls and furniture ...

Fig.3.1 Alpaca overview

Self-Instruct is an iterative bootstrapping algorithm used to generate a vast amount of instruction data through the model itself, thereby improving the model's ability to follow natural language instructions. In the Alpaca model, the authors employed this method to generate a large amount of data, which is capable of guiding model training from limited data.

The author adopted the Self-Instruct technique to GPT-3.5, successfully generated a dataset containing 52K instruction-following examples. Then, they used this dataset to fine-tune the LLaMA 7B model and achieve optimal instruction-following performance.

3.1.4 Alpaca-LoRA

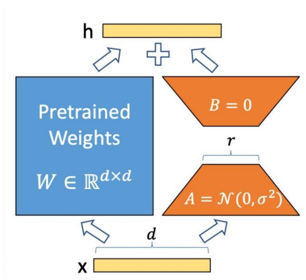

h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

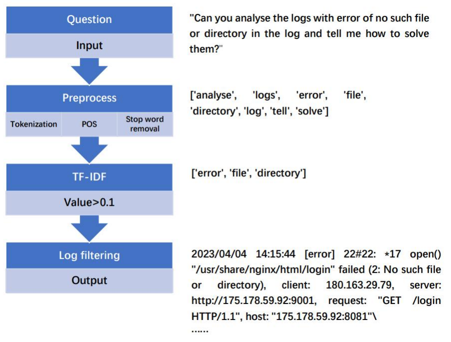
Fig.3.2 Low rank adaptation structure

The model we adopted is Alpaca-LoRA. Compared to Alpaca, Alpaca-LoRA employs

LoRA technique, adding extra network layers to the original LLaMA model and training only the parameters of these new layers while freezing the rest. This significantly reduces the number of trainable parameters for downstream tasks, allowing users to complete fine-tuning a 7B model with a consumer-level GPU, and it is very likely that the required fine-tuning can be done on Google Colab.

3.2 Log filtering based on Log Text Matching

Due to the problems in the existing model: the amount of log text is too large, the model reasoning takes a long time, which affects the performance of the model, and some irrelevant log text needs to be filtered. So on this basis, we used nlp-based log text matching for optimization.



"Can you analyse the logs with error of no such file or directory in the log and tell me how to solve them?"

['analyse', 'logs', 'error', 'file', 'directory', 'log', 'tell', 'solve']

['error', 'file', 'directory']

```
2023/04/04 14:15:44 [error] 22#22: *17 open() "/usr/share/nginx/html/login" failed (2: No such file or directory), client: 180.163.29.79, server: http://175.178.59.92:9001, request: "GET /login HTTP/1.1", host: "175.178.59.92:8081"\
......
```

Fig.3.3 Process of log filtering

3.2.1 Question preprocessing

The goal of this step is to preprocess the text of the question, remove redundant information. This step uses the following methods:

1. Word tokenization: The function of this step is to decompose the whole problem into tokens, so that each token can be processed separately in the future. In this code, the word_tokenize function of NLTK is used to realize it.

2. Part-of-speech tags: The function of this step is to assign parts of speech to the words or vocabulary in the text, so as to facilitate the screening of the required keywords in the token. Generally speaking, the most informative keywords in the question are nouns and verbs, so in this project, the part of speech in the question is selected as ['NN', 'NNS', 'NNP', 'NNPS', 'VB ', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'] words and vocabulary. In this code, the pos_tag function of NLTK is used.

3. Remove stopwords: The function of this step is to delete stop words. Stop words refer to words that appear in large numbers in the text but have little influence on the meaning of the text, and have no substantial help to the search function of this project. Therefore, these words are deleted during the text processing of this project to improve the accuracy of log matching speed and efficiency. Using NLTK's English stopwords library in this code helps us remove stopwords.

3.2.2 TF-IDF

The goal of this step is to extract the core keywords from the keywords obtained in the previous step, and further screen out keywords related to the text. The main method used in this step is TF-IDF. This algorithm calculates the TF-IDF value of each word in the keyword list in the log, and screens out keywords with a TF-IDF value greater than the threshold as core keywords. In this algorithm, the threshold is set to 0.1. This code is implemented using the related functions of TfidfVectorizer in the sklearn library.

3.2.3 Log core keyword matching

The goal of this step is to filter out the part of the log that contains core keywords. After reading a log, if the log contains at least one core keyword, keep the log and count the number of occurrences of the core keyword, otherwise discard the log. Finally, all logs containing core keywords will be our output results.

4. System Design

4.1 User Interface

This is the user interface of the intelligent log system designed in this paper. The interface includes five functions: upload, extraction, report, analyze, and answer, as shown in Fig.4.1.

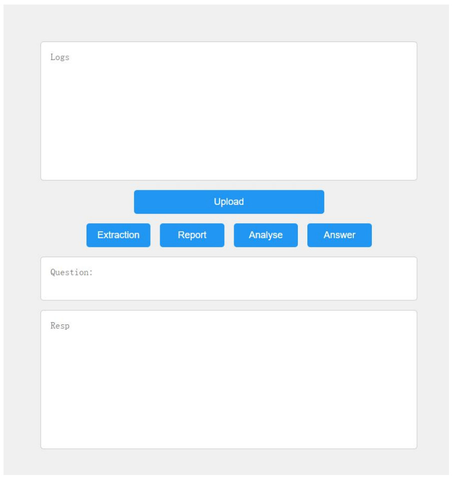
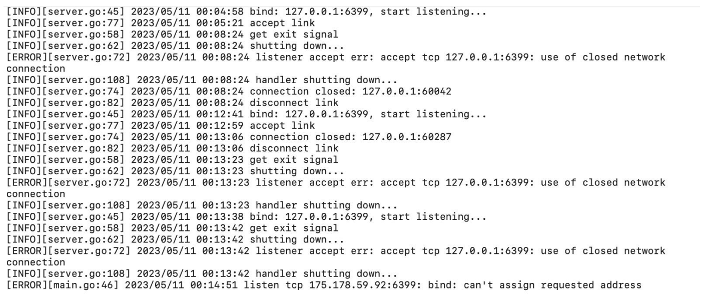


Fig.4.1 ILS User Interface

When using our intelligent log system, the first button users use is: Upload, which is used to upload and preprocess the log content that needs to be analyzed. When uploading, the input log limit is 512 tokens, and the system will automatically truncate the input content if it exceeds the character limit. Preprocessing is used to filter the input content, which has been described in detail in the data preprocessing stage. Here is an example of the input log content that we used to demonstrate the functionality:

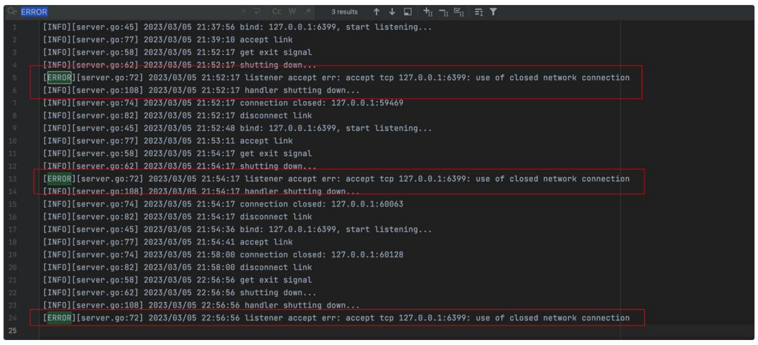


```
[INFO][server.go:45] 2023/05/11 00:04:58 bind: 127.0.0.1:6399, start listening...
[INFO][server.go:77] 2023/05/11 00:05:21 accept link
[INFO][server.go:58] 2023/05/11 00:08:24 get exit signal
[INFO][server.go:62] 2023/05/11 00:08:24 shutting down...
[ERROR][server.go:72] 2023/05/11 00:08:24 listener accept err: accept tcp 127.0.0.1:6399: use of closed network
connection
[INFO][server.go:108] 2023/05/11 00:08:24 handler shutting down...
[INFO][server.go:74] 2023/05/11 00:08:24 connection closed: 127.0.0.1:60042
[INFO][server.go:82] 2023/05/11 00:08:24 disconnect link
[INFO][server.go:45] 2023/05/11 00:12:41 bind: 127.0.0.1:6399, start listening...
[INFO][server.go:77] 2023/05/11 00:12:59 accept link
[INFO][server.go:74] 2023/05/11 00:13:06 connection closed: 127.0.0.1:60287
[INFO][server.go:82] 2023/05/11 00:13:06 disconnect link
[INFO][server.go:58] 2023/05/11 00:13:23 get exit signal
[INFO][server.go:62] 2023/05/11 00:13:23 shutting down...
[ERROR][server.go:72] 2023/05/11 00:13:23 listener accept err: accept tcp 127.0.0.1:6399: use of closed network
connection
[INFO][server.go:108] 2023/05/11 00:13:23 handler shutting down...
[INFO][server.go:45] 2023/05/11 00:13:38 bind: 127.0.0.1:6399, start listening...
[INFO][server.go:58] 2023/05/11 00:13:42 get exit signal
[INFO][server.go:62] 2023/05/11 00:13:42 shutting down...
[ERROR][server.go:72] 2023/05/11 00:13:42 listener accept err: accept tcp 127.0.0.1:6399: use of closed network
connection
[INFO][server.go:108] 2023/05/11 00:13:42 handler shutting down...
[ERROR][main.go:46] 2023/05/11 00:14:51 listen tcp 175.178.59.92:6399: bind: can't assign requested address
```

Fig.4.2 Input log file

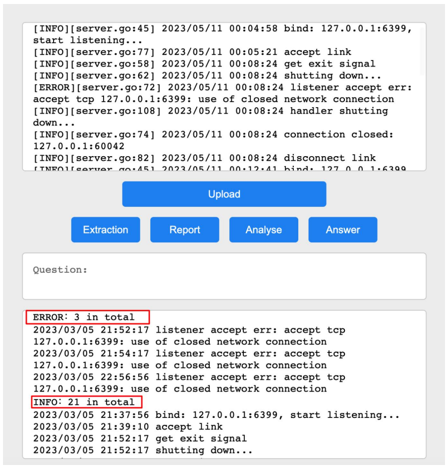
After the upload word limit and filtering process, the following four steps of Extraction, Report, Analyze and Answer can be carried out.

Extraction part: the error message (EEROR) in the general log is scattered in a large amount of normal information (INFO), and it takes a lot of browsing time to manually find the error message. So, we designed the extraction function. Complete line extraction for the filtered content, extract error lines (ERROR) and normal information lines (INFO), and the input and output are as follows:



```
[INFO][server.go:45] 2023/03/05 21:37:56 bind: 127.0.0.1:6399, start listening...
[INFO][server.go:77] 2023/03/05 21:39:10 accept link
[INFO][server.go:58] 2023/03/05 21:52:17 get exit signal
[INFO][server.go:62] 2023/03/05 21:52:17 shutting down...
[ERROR][server.go:72] 2023/03/05 21:52:17 listener accept err: accept tcp 127.0.0.1:6399: use of closed network connection
[INFO][server.go:108] 2023/03/05 21:52:17 handler shutting down...
[INFO][server.go:74] 2023/03/05 21:52:17 connection closed: 127.0.0.1:59469
[INFO][server.go:82] 2023/03/05 21:52:17 disconnect link
[INFO][server.go:45] 2023/03/05 21:52:48 bind: 127.0.0.1:6399, start listening...
[INFO][server.go:77] 2023/03/05 21:53:11 accept link
[INFO][server.go:58] 2023/03/05 21:54:17 get exit signal
[INFO][server.go:62] 2023/03/05 21:54:17 shutting down...
[ERROR][server.go:72] 2023/03/05 21:54:17 listener accept err: accept tcp 127.0.0.1:6399: use of closed network connection
[INFO][server.go:108] 2023/03/05 21:54:17 handler shutting down...
[INFO][server.go:74] 2023/03/05 21:54:17 connection closed: 127.0.0.1:60063
[INFO][server.go:82] 2023/03/05 21:54:17 disconnect link
[INFO][server.go:45] 2023/03/05 21:54:36 bind: 127.0.0.1:6399, start listening...
[INFO][server.go:77] 2023/03/05 21:54:41 accept link
[INFO][server.go:74] 2023/03/05 21:58:00 connection closed: 127.0.0.1:60128
[INFO][server.go:82] 2023/03/05 21:58:00 disconnect link
[INFO][server.go:58] 2023/03/05 22:56:56 get exit signal
[INFO][server.go:62] 2023/03/05 22:56:56 shutting down...
[INFO][server.go:108] 2023/03/05 22:56:56 handler shutting down...
[ERROR][server.go:72] 2023/03/05 22:56:56 listener accept err: accept tcp 127.0.0.1:6399: use of closed network connection
```

Fig.4.3 ERROR in log file



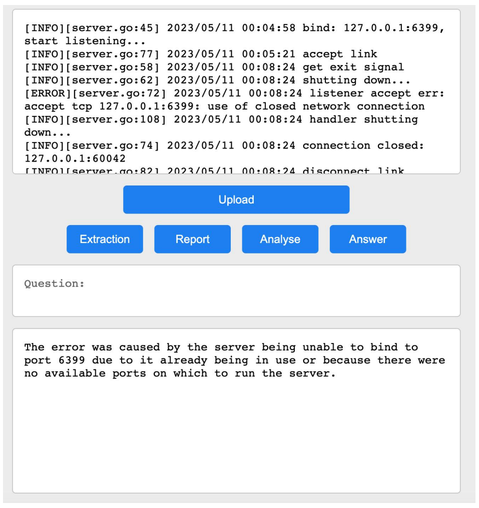
```
[INFO][server.go:45] 2023/05/11 00:04:58 bind: 127.0.0.1:6399,
start listening...
[INFO][server.go:77] 2023/05/11 00:05:21 accept link
[INFO][server.go:58] 2023/05/11 00:08:24 get exit signal
[INFO][server.go:62] 2023/05/11 00:08:24 shutting down...
[ERROR][server.go:72] 2023/05/11 00:08:24 listener accept err:
accept tcp 127.0.0.1:6399: use of closed network connection
[INFO][server.go:108] 2023/05/11 00:08:24 handler shutting
down...
[INFO][server.go:74] 2023/05/11 00:08:24 connection closed:
127.0.0.1:60042
[INFO][server.go:82] 2023/05/11 00:08:24 disconnect link
[INFO][server.go:45] 2023/05/11 00:12:41 bind: 127.0.0.1:6399
```

```
ERROR: 3 in total
2023/03/05 21:52:17 listener accept err: accept tcp
127.0.0.1:6399: use of closed network connection
2023/03/05 21:54:17 listener accept err: accept tcp
127.0.0.1:6399: use of closed network connection
2023/03/05 22:56:56 listener accept err: accept tcp
127.0.0.1:6399: use of closed network connection
INFO: 21 in total
2023/03/05 21:37:56 bind: 127.0.0.1:6399, start listening...
2023/03/05 21:39:10 accept link
2023/03/05 21:52:17 get exit signal
2023/03/05 21:52:17 shutting down...
```

Fig.4.4 Extraction output

The output result is shown in the figure above. The system extracts the log lines of ERROR and INFO respectively, and the user can intuitively see a total of 3 lines of log error information. The specific content of the corresponding error is listener accept err: the user has used the closed network connection, and the time period of the error. Through this function, especially when faced with a large amount of log information, users can quickly locate the error message and the specific error content. Users can choose to manually research the error content, or analyze and summarize the cause of the error through our Analyze function.

Analyze section: The system automatically studies all erroneous lines and analyzes them, giving possible explanations. As shown below:



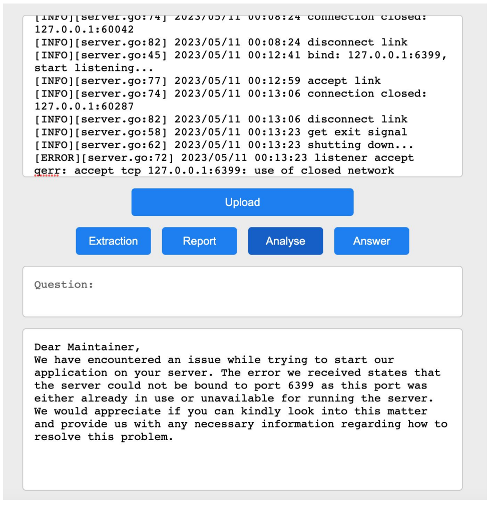
```
[INFO][server.go:45] 2023/05/11 00:04:58 bind: 127.0.0.1:6399,
start listening...
[INFO][server.go:77] 2023/05/11 00:05:21 accept link
[INFO][server.go:58] 2023/05/11 00:08:24 get exit signal
[INFO][server.go:62] 2023/05/11 00:08:24 shutting down...
[ERROR][server.go:72] 2023/05/11 00:08:24 listener accept err:
accept tcp 127.0.0.1:6399: use of closed network connection
[INFO][server.go:108] 2023/05/11 00:08:24 handler shutting
down...
[INFO][server.go:74] 2023/05/11 00:08:24 connection closed:
127.0.0.1:60042
[INFO][server.go:82] 2023/05/11 00:08:24 disconnect link
```

```
The error was caused by the server being unable to bind to
port 6399 due to it already being in use or because there were
no available ports on which to run the server.
```

Fig.4.5 Analyse output

In this output, the system gives two possible reasons: the server being ubalbe to bind to port 6399 due to it already being in use or there were no available ports to which to run the server. This is the above 3 Possible explanations are given after the summary and analysis of each ERROR line.

Report part: The function is to generate a more general text report and remind developers to pay attention in time by email. Since the data set of the model contains email-related text training data, we will also consider sending the generated report content to the corresponding developer in email format.



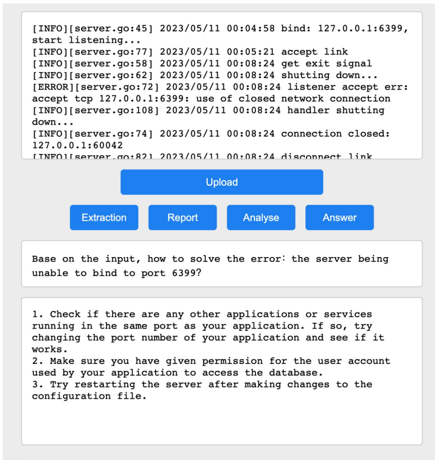
```
[INFO][server.go:74] 2023/05/11 00:08:24 connection closed:
127.0.0.1:60042
[INFO][server.go:82] 2023/05/11 00:08:24 disconnect link
[INFO][server.go:45] 2023/05/11 00:12:41 bind: 127.0.0.1:6399,
start listening...
[INFO][server.go:77] 2023/05/11 00:12:59 accept link
[INFO][server.go:74] 2023/05/11 00:13:06 connection closed:
127.0.0.1:60287
[INFO][server.go:82] 2023/05/11 00:13:06 disconnect link
[INFO][server.go:58] 2023/05/11 00:13:23 get exit signal
[INFO][server.go:62] 2023/05/11 00:13:23 shutting down...
[ERROR][server.go:72] 2023/05/11 00:13:23 listener accept
err: accept tcp 127.0.0.1:6399: use of closed network
```

```
Dear Maintainer,
We have encountered an issue while trying to start our
application on your server. The error we received states that
the server could not be bound to port 6399 as this port was
either already in use or unavailable for running the server.
We would appreciate if you can kindly look into this matter
and provide us with any necessary information regarding how to
resolve this problem.
```

Fig.4.6 Report output

The above output results summarize the problems in the log (have trouble starting the application on the server), the possible reasons (not bound to port 6399 or port already in use), and invite the management personnel to deal with the above problems in time, and provide more information.

Answer section: We found that the existing functions can only realize error line location and error cause analysis, so we introduced the interactive mode of dialogue to allow users to understand the log system more deeply. Users can ask the logging system questions like: What are the possible solutions to this problem? Seek system solutions and more explanations for specific problems.



```
[INFO][server.go:45] 2023/05/11 00:04:58 bind: 127.0.0.1:6399,
start listening...
[INFO][server.go:77] 2023/05/11 00:05:21 accept link
[INFO][server.go:58] 2023/05/11 00:08:24 get exit signal
[INFO][server.go:62] 2023/05/11 00:08:24 shutting down...
[ERROR][server.go:72] 2023/05/11 00:08:24 listener accept err:
accept tcp 127.0.0.1:6399: use of closed network connection
[INFO][server.go:108] 2023/05/11 00:08:24 handler shutting
down...
[INFO][server.go:74] 2023/05/11 00:08:24 connection closed:
127.0.0.1:60042
[INFO][server.go:82] 2023/05/11 00:08:24 disconnect link
```

```
Base on the input, how to solve the error: the server being
unable to bind to port 6399?
```

```
1. Check if there are any other applications or services
running in the same port as your application. If so, try
changing the port number of your application and see if it
works.
2. Make sure you have given permission for the user account
used by your application to access the database.
3. Try restarting the server after making changes to the
configuration file.
```

Fig.4.7 Answer output

In the figure above, the user asks the system for possible solutions to the problem, and the system gives a reply (Check if there are any other applications or services running in the same port as your application. If so, try changing the port number of your application and see if it works. and try restarting the server after making changes to the configuration file) Based on past experience, these solutions are valuable.

4.2 Implementation Details

ILS employs an approach that combines natural language processing (NLP) techniques and machine learning algorithms to extract, analyze and understand complex data obtained from various logs. Below we will introduce the workflow of the system in detail:

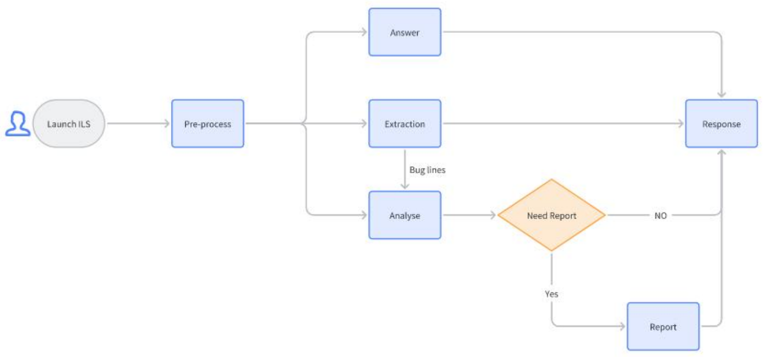
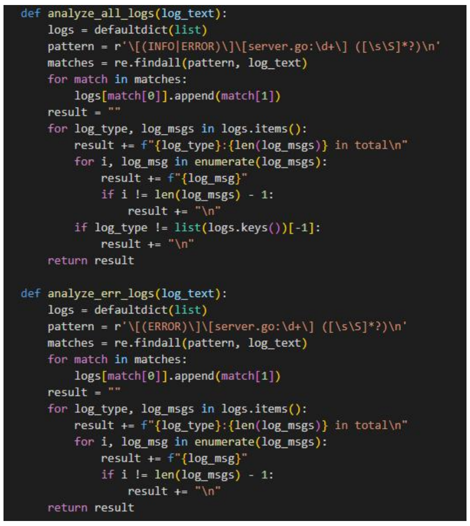


Fig.4.8 ILS workflow

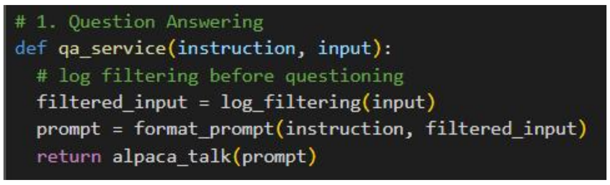
Extraction: This function extracts preset fields, including ERROR and INFO, through rule matching, such as regular matching and string matching.



```python
def analyze_all_logs(log_text):
    logs = defaultdict(list)
    pattern = r'\[(INFO|ERROR)\]\[server.go:\d+\] ([\s\S]*?)\n'
    matches = re.findall(pattern, log_text)
    for match in matches:
        logs[match[0]].append(match[1])
    result = ""
    for log_type, log_msgs in logs.items():
        result += f"{log_type}:{len(log_msgs)} in total\n"
        for i, log_msg in enumerate(log_msgs):
            result += f"{log_msg}"
            if i != len(log_msgs) - 1:
                result += "\n"
        if log_type != list(logs.keys())[-1]:
            result += "\n"
    return result

def analyze_err_logs(log_text):
    logs = defaultdict(list)
    pattern = r'\[(ERROR)\]\[server.go:\d+\] ([\s\S]*?)\n'
    matches = re.findall(pattern, log_text)
    for match in matches:
        logs[match[0]].append(match[1])
    result = ""
    for log_type, log_msgs in logs.items():
        result += f"{log_type}:{len(log_msgs)} in total\n"
        for i, log_msg in enumerate(log_msgs):
            result += f"{log_msg}"
            if i != len(log_msgs) - 1:
                result += "\n"
    return result
```

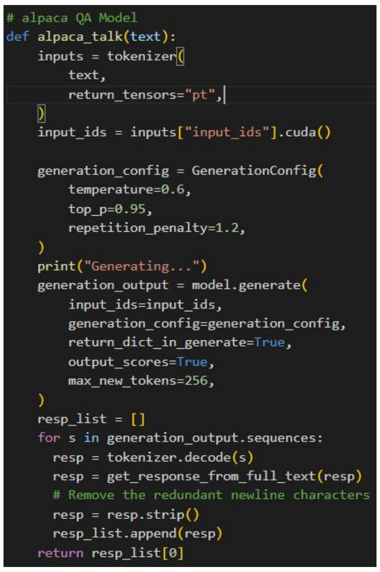
Fig.4.9 Extraction method

Answer: qa_service is used to implement this function, which is provided by the Alpaca-LoRA-LogQA model and encapsulated as the alpaca_talk method. Among them, before the logs are input into alpaca_talk, we use the question-based matching method for log filtering.



```python
# 1. Question Answering
def qa_service(instruction, input):
  # log filtering before questioning
  filtered_input = log_filtering(input)
  prompt = format_prompt(instruction, filtered_input)
  return alpaca_talk(prompt)
```

Fig.4.10 qa_service method

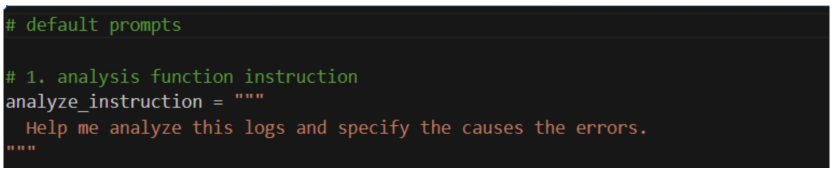


```python
# alpaca QA Model
def alpaca_talk(text):
    inputs = tokenizer(
        text,
        return_tensors="pt",
    )
    input_ids = inputs["input_ids"].cuda()

    generation_config = GenerationConfig(
        temperature=0.6,
        top_p=0.95,
        repetition_penalty=1.2,
    )
    print("Generating...")
    generation_output = model.generate(
        input_ids=input_ids,
        generation_config=generation_config,
        return_dict_in_generate=True,
        output_scores=True,
        max_new_tokens=256,
    )
    resp_list = []
    for s in generation_output.sequences:
      resp = tokenizer.decode(s)
      resp = get_response_from_full_text(resp)
      # Remove the redundant newline characters
      resp = resp.strip()
      resp_list.append(resp)
    return resp_list[0]
```

Fig.4.11 Core qa function

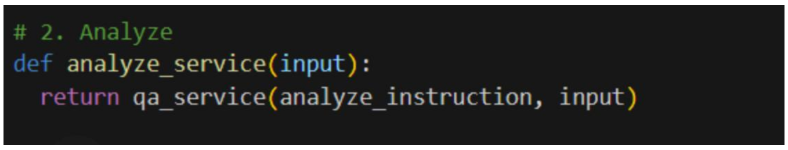
Except that the Extraction function is based on rule-based matching, and the Answer function is based on natural language dialogue, other functions of ILS, such as Analyze and Report, interact with the pre-trained model through specific Prompt Words. These Prompt Words serve as guides to help the model understand user needs and generate corresponding outputs. Specifically, both Analyze and Report are implemented based on the core question-and-answer function alpaca_talk (Fig.4.11), and the results are generated by calling the core function. The difference is that the Answer function accepts the text entered by the user as a prompt word, while Analyze and Report have pre-specified prompt words.

Analyze: The main goal of this function is to analyze text uploaded by users. If ILS detects that the text uploaded by the user contains information related to bug lines, that is to say, the result of extraction is not empty, then ILS will perform a special analysis on bug lines. Otherwise, ILS performs a comprehensive analysis of all preprocessed uploads. This process leverages the powerful ability of pre-trained models to discover hidden information from various patterns.



```
# default prompts

# 1. analysis function instruction
analyze_instruction = """
  Help me analyze this logs and specify the causes the errors.
"""
```

Fig.4.12 Predefine the prompt words for Analyze

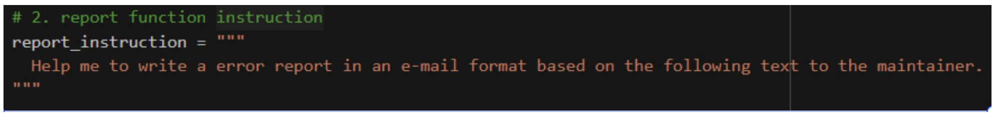


```
# 2. Analyze
def analyze_service(input):
  return qa_service(analyze_instruction, input)
```

Fig.4.13 The implementation method of Analyze

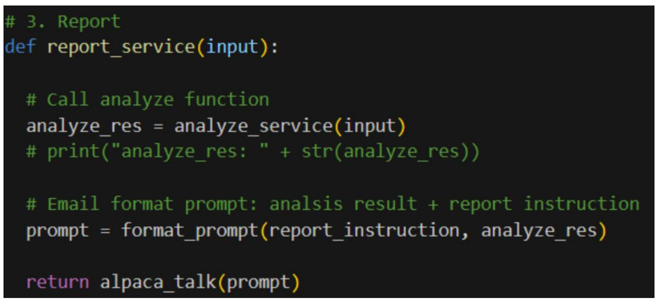
Report: The Report function is implemented on the basis of Analyze and is used to solve the transforming task. It can generate more general and instructive suggestions and present them in the form of emails. The specific implementation is divided into two steps: the first step: format_promp is used to integrate the prompts from two sources: 1) The service function analyze_service of Analyze is called in the report_service function as an input of the format_promp function. That is, the result of analyze_service in the clubhouse is used as a prompt word. 2) The pre-defined prompt word report_instruction is used as another input of the format_promp function, where the prompt word defines the output text in email format. Step 2: Call the core question-and-answer function alpaca_talk, use the integrated prompt as the input of the function, and get the final report result.

The corresponding relationship of the Prompt Words part is as follows:



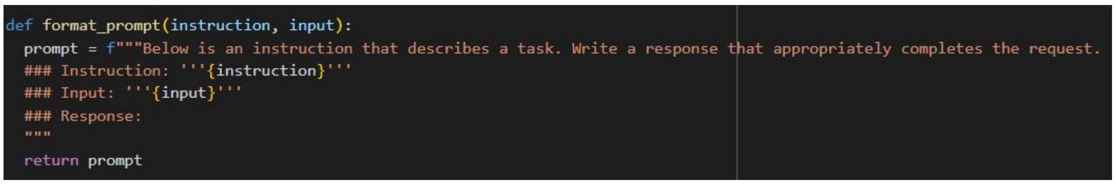
```
# 2. report function instruction
report_instruction = """
  Help me to write a error report in an e-mail format based on the following text to the maintainer.
"""
```

Fig.4.14 Predefine the prompt words of Report



```python
# 3. Report
def report_service(input):

  # Call analyze function
  analyze_res = analyze_service(input)
  # print("analyze_res: " + str(analyze_res))

  # Email format prompt: analsis result + report instruction
  prompt = format_prompt(report_instruction, analyze_res)

  return alpaca_talk(prompt)
```

Fig.4.15 Report implementation method



```python
def format_prompt(instruction, input):
  prompt = f"""Below is an instruction that describes a task. Write a response that appropriately completes the request.
  ### Instruction: '''{instruction}'''
  ### Input: '''{input}'''
  ### Response:
  """
  return prompt
```

Fig 4.16 The format_promp function in the Report method

5. Experiment and Evaluation

5.1 Training

In our experiment, we adopted the Alpaca-LoRA model, and optimized and improved it based on this method. We use LLaMA-7b as the base large language model, and its excellent performance and wide application provide a good foundation for our experiments.
We fine-tuned the model to fit our task, the Go language logging task. First, we use 150 Go log QA data collected, which covers a variety of common problems and scenarios in Go logs, providing rich fine-tuning samples for our model. We then took the Alpaca model's higher-quality generic-task dataset, alpaca-cleaned, and concatenated it with our dataset to get our fine-tuned dataset, called Alpaca-Cleaned-LogQA.

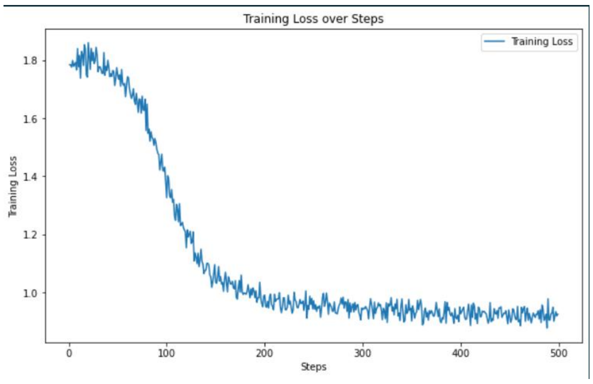


Fig.5.1 Training Loss

In the process of training, we borrow the method of Alpaca-LoRA for fine-tuning. In the process of fine-tuning the model, although the official claim is that the model can be fine-tuned at a low cost, in the experiment fine-tuning, we found that due to the large amount of data, a sufficiently powerful GPU is required. A100 is a powerful computing device capable of handling large amounts of data and complex computing tasks. In the fine-tuning process of this model, we used Google Colab Pro A100 GPU device for fine-tuning, and carried out 500 steps of training. The training loss curve is shown in Figure 5.1.

5.2 Evaluation

In the evaluation part, we used ROUGE as the evaluation metrics. ROUGE is a set of metrics commonly used to evaluate the quality of text generated by language models in NLP. The idea is to measure the overlap between the generated text and the reference answer.

1. ROUGE-N

This metric measures the n-gram overlap between the generated answer and the reference answer. It calculates the F1-score for different values of 'n'.

In this project, the 2 mainly used metrics are ROUGE-1 and ROUGE-2.

ROUGE-1 refers to the overlap of unigrams between the generated answer and reference answer.

ROUGE-2 refers to the overlap of bigrams between the generated answer and reference answer.

The calculation function is as follows:

$$Rouge{-}N = \frac{\sum\limits_{S \in ReferenceSummaries} \sum\limits_{gram_n \in S} Count_{match}(gram_n)}{\sum\limits_{S \in ReferenceSummaries} \sum\limits_{gram_n \in S} Count(gram_n)} \quad \text{.................. (1.1)}$$

2. ROUGE-L

This metric applies the longest common subsequence (LCS) between the generated answer and the reference answer in the calculation. It measures the recall of the LCS and computes F1-score based on that. The calculation function is as follows:

$$R_{lcs} = \frac{LCS(X,Y)}{m}$$
$$p_{lcs} = \frac{LCS(X,Y)}{n}$$
$$F_{lcs} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs}+\beta^2 P_{lcs}} \quad \text{................................. (1.2)}$$

In the formula, X refers to the generated answer, Y refers to the reference answer, LCS refers to the Longest Common Subsequence between X and Y, m and n refer to the length of X and Y.

3. Result

In the project, we used the test set of the LogQA dataset we collected to test performance. For each Q&A pair in the dataset, the question is fed into the language model, and the generated answer is compared with the reference answer to calculate the ROUGE F1 score. Finally, we calculate the average score for all Q&A pairs in the test set to be the final result for the model. The final evaluation result is shown in the following chart:

Model	Rouge-1	Rouge-2	Rouge-L
Alpaca Lora 7B	21.2	4.3	18.0
Alpaca Lora LogQA(Ours)	**24.3**	**6.2**	**21.5**

Table 1.1 Different ROUGE F1-scores for each model on LogQA Dataset

From the table, we can see after fine-tuning on the LogQA dataset, the performance of Alpaca Lora LogQA had a significant improvement on the task of answering questions related to the log.

6. Conclusion and Improvement

6.1 Summary

In the course of our research, we have achieved key milestones and gained significant insights, outlined as follows:

- Dataset Development. In our research, we utilized web crawlers to aggregate log-related content from the StackOverflow platform. We further employed the GPT model to expand this initial data set, thereby establishing our own unique Question-Answer (QA) dataset.
- Log Text Filtering. Given that an excessive volume of log text could potentially impact the performance of model inference, we implemented a question text matching method. This method served to filter out irrelevant log text, thereby optimizing the Question Answering functionality of the user interface.
- Function Realization. The core functions of Extraction, Analysis, Reporting, and Answering have been effectively realized. These capabilities enable four distinct operations: error line extraction, cause analysis, email text summary, and question and answer reply pertaining to the log content.
- Model Utilization. For our log task, we chose to deploy the Alpaca-LoRA model, finely tuning it to balance performance improvement against controllable training costs.
- Deployment Strategy. Our design embraces the separation of front-end and back-end elements. The front-end utilizes the Flask framework, while the back-end is deployed in Google Colab. We use Flask and Ngrok to expose the model API to the Wide Area Network (WAN), facilitating front-end calls and connections.
- Deployment Strategy. Our design embraces the separation of front-end and back-end elements. The front-end utilizes the Flask framework, while the back-end is deployed in

Google Colab. We use Flask and Ngrok to expose the model API to the Wide Area Network (WAN), facilitating front-end calls and connections.

6.2 Future work and improvement

Looking forward, there still exist a number of areas in our work which we believe could benefit from further development and improvement, including:

- Format Flexibility. At present, the log content fed into the system must conform to a specific format. In the future, we aim to incorporate the capacity to process logs in a variety of formats.
- Report Function Enhancement. Within the current system, the Report function generates email text but does not yet possess the ability to automatically forward emails. We envisage one-click email forwarding as a key future development.
- Dataset and Resources. At this stage, our dataset remains insufficient and the available resource format is limited. Future work will address these limitations.
- Model Optimization. Our model's reasoning speed is currently slow, indicating inadequate training levels. Hence, performance optimization is a necessary area for future improvements.

7. Reference

[1] yahma. "Alpaca cleaned dataset". *https://huggingface.co/datasets/yahma/alpaca-cleaned (2023)*

[2] Wang, Yizhong, et al. "Self-Instruct: Aligning Language Model with Self Generated Instructions." *arXiv preprint arXiv:2212.10560* (2022).

[3] Elastic-stack. *https://www.elastic.co/cn/elastic-stack/*

[4] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[5] Ouyang, Long, et al. "Training language models to follow instructions with human feedback." Advances in Neural Information Processing Systems 35 (2022): 27730-27744.

[6] Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." arXiv preprint arXiv:2106.09685 (2021).

[7] Taori, Rohan, et al. "Alpaca: A Strong, Replicable Instruction-Following Model." Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html (2023).

[8] tloen. "Alpaca-LoRA". *https://github.com/tloen/alpaca-lora (2023).*

[9] Meta, A. I. "Introducing LLaMA: A foundational, 65-billion-parameter large language model." *Meta AI. https://ai. facebook. com/blog/large-language-model-llama-meta-ai* (2023).

[10] Lin, Chin-Yew. "Rouge: A package for automatic evaluation of summaries." *Text summarization branches out*. 2004.