1 Perplexica

1.1 相关地址

- 原 `github` 地址

  > https://github.com/ItzCrazyKns/Perplexica

- 更改后的（自己的）

  > https://github.com/qinchihongye/Perplexica

- 一直刷新处理(issue)

  > https://github.com/ItzCrazyKns/Perplexica/issues/180

- 新增 oneapi（功能不完整）

  > https://github.com/ItzCrazyKns/Perplexica/issues/367

1.2 其他修改

1.2.1 更改 dockerfile

- 更改后端 dockerfile

  > backend.dockerfile

  增加

```
RUN apt-get update
RUN apt-get install -y vim
```

1.2.2 更改yaml文件

- 更改IP 地址

```
        args:
          - NEXT_PUBLIC_API_URL=http://127.0.0.1:3001/api
          - NEXT_PUBLIC_WS_URL=ws://127.0.0.1:3001
```

将 `127.0.0.1` 替换为运行 `Perplexica` 服务器的 IP地

- 更改前端端口

```
      ports:
        - 3000:3000

      ports:
        - 8345:3000
```

将 `perplexica-frontend:` 下的 ports 改为 `8345:3000`

1.2.3 更改图标

- 更改展示网页来源时候的图标

  先找个图标，命名为 `favicon.ico`

  图标放置位置

> /ui/components/favicon.ico

更改文件

> /ui/components/MessageSources.tsx

替换下面第一行的为第二行（共三处）

```
src={`https://s2.googleusercontent.com/s2/favicons?domain_url=${source.metadata.url}`}

src='./favicon.ico'
```

1.2.4 更改后端配置文件

- 配置文件地址

> /cofig.toml

```
[API_KEYS]
OPENAI = "sk-PJrYIIj1QwodFT4n00Df8956B465411389C44dF11eDc07Da"
GROQ = ""
ANTHROPIC = ""
GEMINI = ""

[API_ENDPOINTS]
OLLAMA = "http://host.docker.internal:11434"
OPENAIBASEURL = "http://38.54.50.54:8025/v1"
SELFSEARCHURL = "http://host.docker.internal"
SEARXNG = "http://localhost:32768"

```

```
[GENERAL]
PORT = 3_001
SIMILARITY_MEASURE = "cosine"
KEEP_ALIVE = "5m"
PYPORT = 8_013
LLM_NAME = "qwen2.5-72b-instruct"
TIMEOUT = 5
RETRIEVALURL = "http://js1.blockelite.cn:19575/search"
```

配置项中 `OPENAIBASEURL`、`SELFSEARCHURL`、`PYPORT` 是 自己后加的。

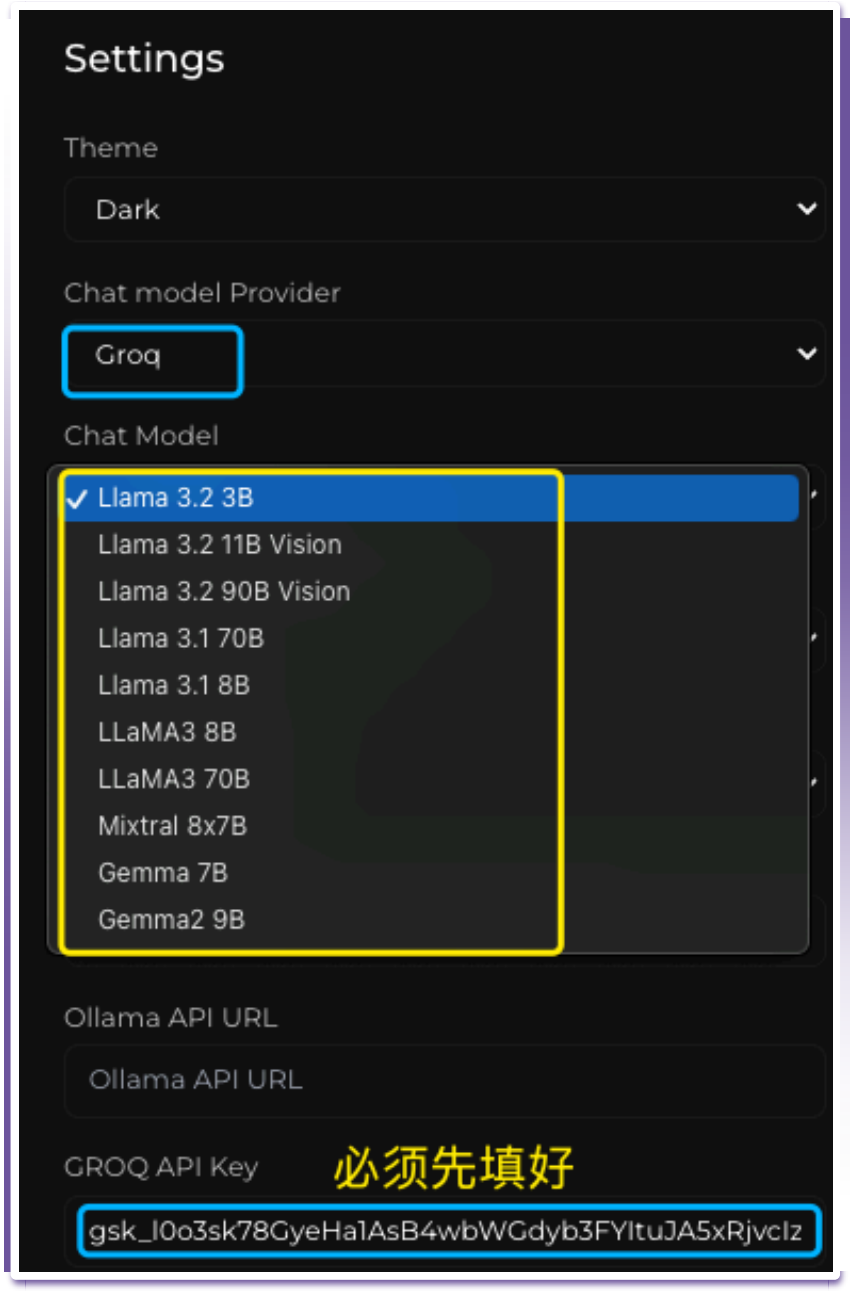
说明如下：

1. `OPENAI`： 可以使用 OpenAI 的 API-Key，也可以使用支持 OpenAI 接口的其他 API-Key。使用时需与 OPENAIBASEURL 配合,增删模型在如下地址设置。

   > src/lib/providers/openai.ts

2. `GROQ`： GROQ 的 api-key,超低延迟推理，只有设置后在前端页面才会在"Chat model Provider"选项中出现，并自动在"Chat Model"选项框中呈现支持的模型, 新模型可在 以下地址配置。

   > /src/lib/providers/groq.ts



3. `ANTHROPIC`: 同上，新模型可在以下地址设置

   > src/lib/providers/anthropic.ts

4. `GEMINI`:同上，新模型可在以下地址设置

   > src/lib/providers/gemini.ts

5. `OLLAMA`: ollama 服务的启动地址，若是 docker启动的填

   > **http://host.docker.internal:11434**

6. `SEARXNG`: SEARXNG服务地址。
7. `OPENAIBASEURL`： 用于指定基础 URL，可以是 OneAPI 的 baseurl，或者是符合 OpenAI 接口格式的 URL。

8. `SELFSEARCHURL`： 搜索接口地址（/src/lbi/searxng.ts 中使用的）。
9. `PORT`： 后端服务的端口
10. `SIMILARITY_MEASURE`： 要使用的相似性度
11. `KEEP_ALIVE`： 等待ollma 模型加载的时间
12. `PYPORT`： python 服务启动的端口。
13. `LLM_NAME` : python 服务中 query改写的模型
14. `TIMEOUT` : python 服务中检索服务的超时限制、
15. `RETRIEVALURL` : 由美芝提供的检索服务地址

1.3 增加配置项(config.toml)

- 以增加 `OPENAIBASEURL` 、 `SELFSEARCHURL` 、 `PYPORT` 为例

  将 `OPENAIBASEURL` 和 `SELFSEARCHURL` 增加在 config.toml 的[API_ENDPOINTS]下

  ```
  OPENAIBASEURL = ""
  SELFSEARCHURL = ""
  ```

  将 `PYPORT` 增加在[GENERAL]下面

  ```
  PYPORT =
  ```

- 备注
  1. 如果不需要在前端页面中更改，则只需要更改第一步 `更改 src/config.ts` 即可,后续直接在代码中 import,比如PYPORT。
  2. 记得如果在容器中更改，改完要编译

     ```
     npm run build
     ```

1.3.1 更改 src/config.ts

- 地址

  ```
  /src/config.ts
  ```

- `Config` 类型中的 `API_ENDPOINTS` 下 增加

```
OPENAIBASEURL: string;
SELFSEARCHURL: string;
```

GENERAL下增加

```
PYPORT: number;
```

- 增加导出函数

```
export const getOpenaiBaseUrl = () =>
loadConfig().API_ENDPOINTS.OPENAIBASEURL;

export const getSelfSearchUrl = () =>
loadConfig().API_ENDPOINTS.SELFSEARCHURL;

export const getPyPort = () => loadConfig().GENERAL.PYPORT;
```

1.3.2 更改 routes/config.ts

- 地址

> /src/routes/config.ts

- 新增

```
import {getOpenaiBaseUrl} from '../config';
import {getSelfSearchUrl} from '../config';

//在 60 行加入
config['openaiBaseUrl'] = getOpenaiBaseUrl();

//在 61 行加入
config['selfSearchUrl'] = getSelfSearchUrl();

//在 updatedConfig 中API_ENDPOINTS下加入
OPENAIBASEURL: config.openaiBaseUrl,
SELFSEARCHURL: config.selfSearchUrl,
```

1.3.3 更改 SettingsDialog.tsx

- 地址

> /ui/components/SettingsDialog.tsx

- `SettingsType` 类型中增加

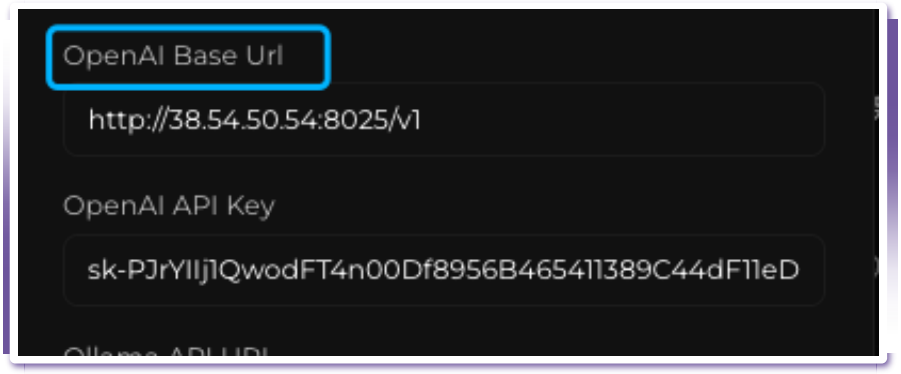
```
  openaiBaseUrl: string;

  selfSearchUrl: string;
```

- `Return` 中增加 (对应网页的设置页面，找个合适的位置)

1. 放在 `OpenAI API Key` 上面

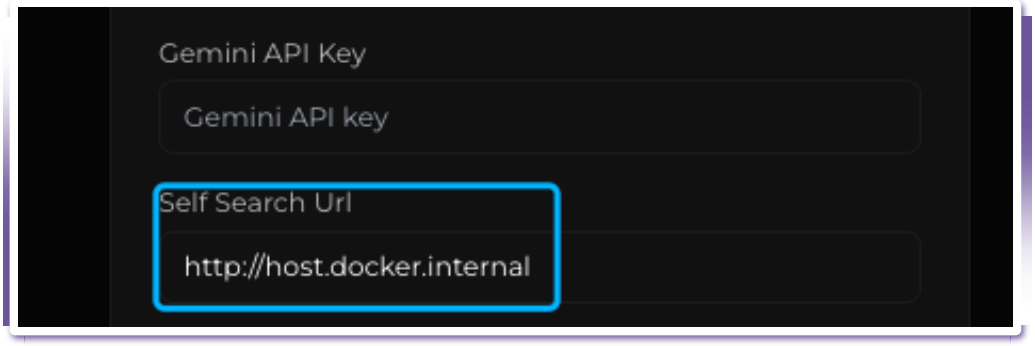
```
                    <div className="flex flex-col space-y-1">
                      <p className="text-black/70 dark:text-white/70 text-sm">
                        OpenAI Base Url
                      </p>
                      <Input
                        type="text"
                        placeholder="OpenAI Base Url"
                        defaultValue={config.openaiBaseUrl}
                        onChange={(e) =>
                          setConfig({
                            ...config,
                            openaiBaseUrl: e.target.value,
                          })
                        }
                      />
                    </div>
```



2. 放在 `Gemini API Key` 下面

```
                    <div className="flex flex-col space-y-1">
                      <p className="text-black/70 dark:text-white/70 text-sm">
                        Self Search Url
                      </p>
                      <Input
                        type="text"
                        placeholder="Self Search Url"
                        defaultValue={config.selfSearchUrl}
                        onChange={(e) =>
                          setConfig({
                            ...config,
                            selfSearchUrl: e.target.value,
                          })
```

```
14                                }
15                                />
16                            </div>
```



1.4 增加 models

1.4.1 修改 oneapi 的 baseurl

- 文件地址

> /src/lib/providers/openai.ts

- 增加代码

```
1   import { getOpenaiBaseUrl } from '../../config';
```

- 在 chatModels 下更改模型的默认 base_url

  原

```
1         'gpt-3.5-turbo': {
2           displayName: 'GPT-3.5 Turbo',
3           model: new ChatOpenAI({
4             openAIApiKey,
5             modelName: 'gpt-3.5-turbo',
6             temperature: 0.7,
7           }
8         },
```

  更改后

```
        'gpt-3.5-turbo': {
          displayName: 'GPT-3.5 Turbo',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'gpt-3.5-turbo',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        }
```

因使用 `customer openai` 指定 `baseurl` 和 `apikey` 会无法使用完整功能，所以直接在 openai.ts 中更改，增加 `baseURL: getOpenaiBaseUrl()`

备注： 新版中 `customer openai` 应该也是可以使用完整功能，但是因为 `需要自己填` model_name, 所以还是直接在 `openai.ts` 中配置 `baseURL`，这样可以直接在前端下拉框界面选择已经配置好的模型。

1.4.2 增加模型(以增加 qwen-long 为例子)

- 使用 `oneapi` 中转，需提前配置好模型(其他途径类似)
- 文件地址

> /src/lib/providers/openai.ts

- 代码

  在`chatModels` 下增加

```
        'qwen-long': {
          displayName: 'Qwen-Long',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'qwen-long',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
```

  其中

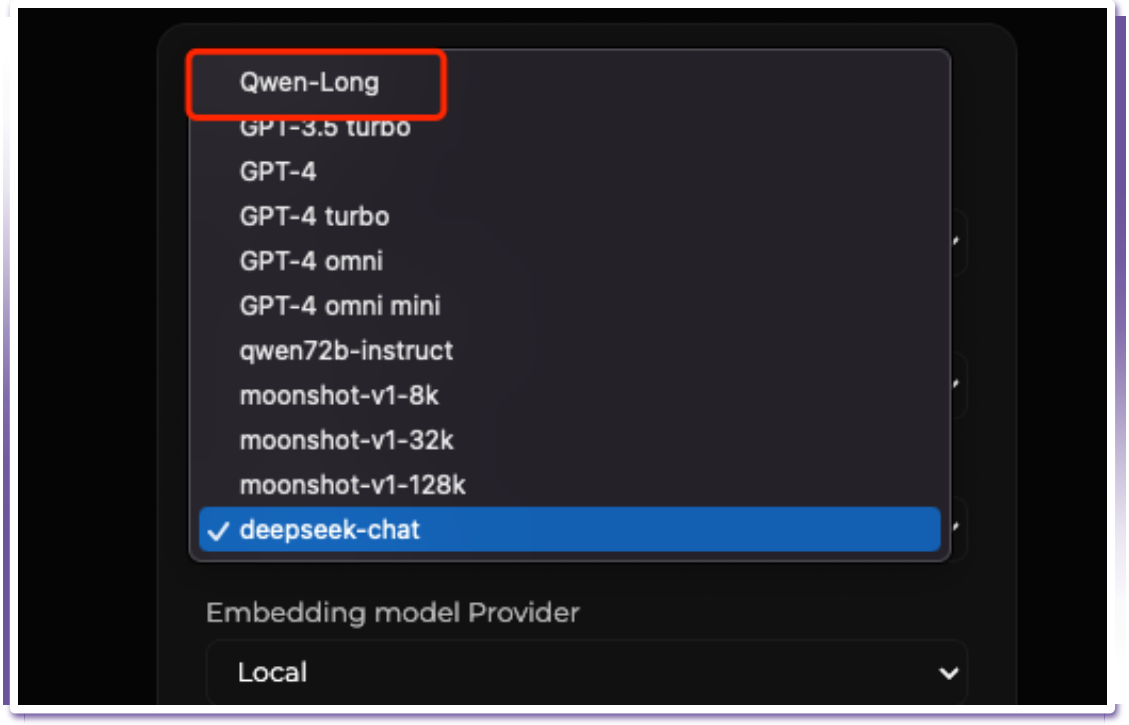
  1. `modelName`: 为 oneapi 中模型的名称；
  2. `displayName`: 为显示在前端中的模型名称；
  3. `baseURL`: 如果没有指定则默认为 openai 自己的 url，且这时 api-key 必须是原生的 openai api key。

  如果是在容器中更改，需要编译

> npm run build

编译后需要重启

> docker-compose restart



1.4.3 增加oneapi中的模型使用

- **东京机 oneapi**

> **http://38.54.50.54:8025/**

`令牌名称`： Perplexica专用

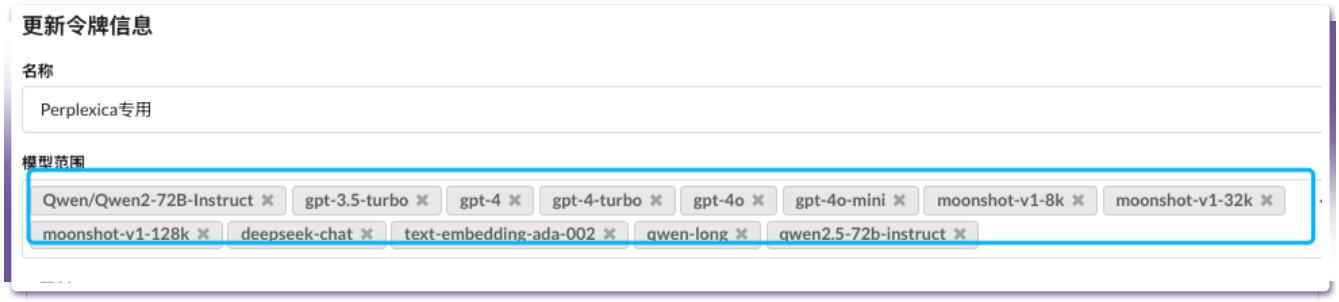
`base_url`

> **http://38.54.50.54:8025/v1**

`api-key`

> sk-PJrYIlj1QwodFT4n00Df8956B465411389C44dF11eDc07Da

自行增减 模型



- `openai.ts` 中配置 oneapi 的模型

> /src/lib/providers/openai.ts

在 `chatModels` 下增加

注意如果是在容器中修改完成后记得编译完成后重启

> npm run build

```
        'qwen-long': {
          displayName: 'Qwen-Long',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'qwen-long',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'gpt-3.5-turbo': {
          displayName: 'GPT-3.5 Turbo',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'gpt-3.5-turbo',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'gpt-4': {
          displayName: 'GPT-4',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'gpt-4',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'gpt-4-turbo': {
          displayName: 'GPT-4 turbo',
          model: new ChatOpenAI({
```

```
            openAIApiKey,
            modelName: 'gpt-4-turbo',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'gpt-4o': {
          displayName: 'GPT-4 omni',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'gpt-4o',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'gpt-4o-mini': {
          displayName: 'GPT-4 omni mini',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'gpt-4o-mini',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'Qwen/Qwen2-72B-Instruct': {
          displayName: 'qwen72b-instruct',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'Qwen/Qwen2-72B-Instruct',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'moonshot-v1-8k': {
          displayName: 'moonshot-v1-8k',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'moonshot-v1-8k',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'moonshot-v1-32k': {
          displayName: 'moonshot-v1-32k',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'moonshot-v1-32k',
```

```
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'moonshot-v1-128k': {
          displayName: 'moonshot-v1-128k',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'moonshot-v1-128k',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'deepseek-chat': {
          displayName: 'deepseek-chat',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'deepseek-chat',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
```

1.5 替换搜索部分

1.5.1 搜索

- 由美芝老师提供的知乎数据的搜索

  `url`

  > **http://js1.blockelite.cn:19575/search**

  `请求体`

```
{
    "query":query,
    "datasetIds":["661f9c02e50601c600e26eb9"],
    "topN": 9,
    "simThreshold": 0.0
}
```

- python连接 示例代码

```python
import requests

"""知乎数据的相关搜索接口"""
url = 'http://js1.blockelite.cn:19575/search'
body = {
    "query":query,
    "datasetIds":["661f9c02e50601c600e26eb9"],
    "topN": 2,
    "simThreshold": 0.0
}
res = requests.post(url,json=body).json()
conv_links,content_list = res['conv_links'],res['content_list']
```

- typescript 连接示例代码

```typescript
import axios from 'axios';

const url = "http://js1.blockelite.cn:19575/search";
const payload = {
    query: "什么是保险",
    datasetIds: ["661f9c02e50601c600e26eb9"],
    topN: 2,
    simThreshold: 0
};

axios.post(url, payload)
  .then(response => {
    // 处理响应数据
    console.log("conv_links:",response.data.conv_links);
    console.log("content_list:",response.data.content_list);
  })
  .catch(error => {
    // 处理错误情况
    console.error(error);
  });
```

1.5.2 直接使用接口

- 直接使用美芝老师提供的接口，query 是 string 类型

1.5.2.1 测试用代码

- 文件名

> search_use_zhihu.ts

```
npm install axios

tsc search_use_zhihu.ts

node search_use_zhihu.js
```

- 代码(topN =2 测试)

```
import axios from 'axios';

interface SearxngSearchOptions {
  categories?: string[];
  engines?: string[];
  language?: string;
  pageno?: number;
}

interface SearxngSearchResult {
  title: string;
  url: string;
  img_src?: string;
  thumbnail_src?: string;
  thumbnail?: string;
  content?: string;
  author?: string;
  iframe_src?: string;
}

export const searchSearxng = async (
  query: string,
  opts?: SearxngSearchOptions,
) => {
  const url = "http://js1.blockelite.cn:19575/search";
  console.log('url', url);
  const payload = {
    query: query,
    datasetIds: ["661f9c02e50601c600e26eb9"],
    topN: 2,
    simThreshold: 0
  };

  try {
    const response = await axios.post(url, payload);
    const contentList = response.data.content_list;
    const convLinks = response.data.conv_links;

    const results: SearxngSearchResult[] = [];
    const suggestions: any[] = [];

    contentList.forEach((val, idx) => {
      const dataDict = {
        url: val.url,
```

```
          title: convLinks[idx].title,
          content: val.content,
          engine: 'bing',
          template: 'default.html',
          engines:['bing'],
          positions: [idx + 1],
          score: val.score,
          category:'general'
        };
        results.push(dataDict);
      });
      console.log('这里是结果:convLinks,', convLinks)
      return { results, suggestions };
    } catch (error) {
      console.error('这里出现错误:', error);
      throw error;
    }
  };


  // 调用 searchSearxng 函数
  async function performSearch() {
    const query = '中国平安2023年业绩'; // 你的搜索查询
    const options: SearxngSearchOptions = {
        categories: ['general', 'news'], // 可选的搜索类别
        engines: ['bing', 'google'],        // 可选的搜索引擎
        language: 'zh-CN',        // 可选的搜索语言
        pageno: 1                 // 可选的页码
    };

    try {
        const response = await searchSearxng(query, options);
        console.log('搜索结果:', response.results);
        console.log('response 的 type',typeof response);
        console.log('results 的 type',typeof response.results);
        console.log('results 的 type333',typeof response.suggestions);
    } catch (error) {
        console.error('搜索出错:', error);
    }
  }


  // 执行搜索
  performSearch();
```

1.5.2.2 更改searxng.ts

- 如果使用了以下代码修改，则 config.toml 文件中的 SELFSEARCHURL填写

> SELFSEARCHURL = "**http://js1.blockelite.cn:19575/search**"

- 地址

> /src/lib/searxng.ts

原代码

```ts
import axios from 'axios';
import { getSearxngApiEndpoint } from '../config';

interface SearxngSearchOptions {
  categories?: string[];
  engines?: string[];
  language?: string;
  pageno?: number;
}

interface SearxngSearchResult {
  title: string;
  url: string;
  img_src?: string;
  thumbnail_src?: string;
  thumbnail?: string;
  content?: string;
  author?: string;
  iframe_src?: string;
}

export const searchSearxng = async (
  query: string,
  opts?: SearxngSearchOptions,
) => {
  const searxngURL = getSearxngApiEndpoint();

  const url = new URL(`${searxngURL}/search?format=json`);
  url.searchParams.append('q', query);

  if (opts) {
    Object.keys(opts).forEach((key) => {
      if (Array.isArray(opts[key])) {
        url.searchParams.append(key, opts[key].join(','));
        return;
      }
      url.searchParams.append(key, opts[key]);
    });
  }

  const res = await axios.get(url.toString());

  const results: SearxngSearchResult[] = res.data.results;
  const suggestions: string[] = res.data.suggestions;

```

```
    return { results, suggestions };
  };
```

修改后代码

```
import axios from 'axios';
import { getSelfSearchUrl } from '../config';

interface SearxngSearchOptions {
  categories?: string[];
  engines?: string[];
  language?: string;
  pageno?: number;
}

interface SearxngSearchResult {
  title: string;
  url: string;
  img_src?: string;
  thumbnail_src?: string;
  thumbnail?: string;
  content?: string;
  author?: string;
  iframe_src?: string;
}

export const searchSearxng = async (
  query: string,
  opts?: SearxngSearchOptions,
) => {
  // const url = "http://js1.blockelite.cn:19575/search";
  const url = getSelfSearchUrl();
  console.log('url', url);
  const payload = {
    query: query,
    datasetIds: ["661f9c02e50601c600e26eb9"],
    topN: 10,
    simThreshold: 0
  };

  try {
    const response = await axios.post(url, payload);
    const contentList = response.data.content_list;
    const convLinks = response.data.conv_links;

    const results: SearxngSearchResult[] = [];
    const suggestions: any[] = [];

    contentList.forEach((val, idx) => {
      const dataDict = {
        url: val.url,
        title: convLinks[idx].title,
```

```
            content: val.content,
            engine: 'bing',
            template: 'default.html',
            engines:['bing'],
            positions: [idx + 1],
            score: val.score,
            category:'general'
          };
          results.push(dataDict);
        });
        console.log('这里是结果:convLinks,', convLinks)
        return { results, suggestions };
      } catch (error) {
        console.error('这里出现错误:', error);
        throw error;
      }
    };


    // // 调用 searchSearxng 函数
    // async function performSearch() {
    //   const query = '中国平安2023年业绩'; // 你的搜索查询
    //   const options: SearxngSearchOptions = {
    //       categories: ['general', 'news'], // 可选的搜索类别
    //       engines: ['bing', 'google'],       // 可选的搜索引擎
    //       language: 'zh-CN',        // 可选的搜索语言
    //       pageno: 1                 // 可选的页码
    //   };

    //   try {
    //       const response = await searchSearxng(query, options);
    //       console.log('搜索结果:', response.results);
    //       console.log('response 的 type',typeof response);
    //       console.log('results 的 type',typeof response.results);
    //       console.log('results 的 type333',typeof response.suggestions);
    //   } catch (error) {
    //       console.error('搜索出错:', error);
    //   }
    // }


    // // 执行搜索
    // performSearch();
```

1.5.3 使用 python 启动的接口

- 使用 python 启动的接口，也是调用美芝老师的接口，不同的是 query 是 string或者list 类型，其中 query 进入后会做判断，如果是 string 类型，则转换为 list 类型

1.5.3.1 测试用代码

- 文件名

> search_use_python.ts

```
npm install axios

tsc search_use_python.ts

node search_use_python.js
```

- 代码

```
import axios from 'axios';

interface SearxngSearchOptions {
  categories?: string[];
  engines?: string[];
  language?: string;
  pageno?: number;
}

interface SearxngSearchResult {
  title: string;
  url: string;
  img_src?: string;
  thumbnail_src?: string;
  thumbnail?: string;
  content?: string;
  author?: string;
  iframe_src?: string;
}


export const searchSearxng = async (
  query: string | string[],
  opts?: SearxngSearchOptions,
) => {
  if (typeof query === "string") { // 如果 query 是字符串类型，将其转换为单元素数组
    query = [query];
  } else if (!Array.isArray(query)) { // 如果 query 既不是字符串也不是字符串数组，抛出一个错误。
    throw new Error("query必须是 【字符串 】或者 【字符串数组】！！！！！");
  }
  const urlroot = "http://localhost:8013"
  const url = urlroot + "/retrieval";

  console.log('url', url);
  const payload = {
    query_list:query
  };
```

```
38
39    try {
40      const response = await axios.post(url, payload);
41      const results = response.data.results;
42      const suggestions = response.data.suggestions;
43      return { results, suggestions };
44    } catch (error) {
45      console.error('这里出现错误:', error);
46      throw error;
47    }
48  };
49
50  // 调用 searchSearxng 函数
51  async function performSearch() {
52    const query = '中国平安2023年业绩'; // 你的搜索查询
53    const options: SearxngSearchOptions = {
54        categories: ['general', 'news'], // 可选的搜索类别
55        engines: ['bing', 'google'],        // 可选的搜索引擎
56        language: 'zh-CN',        // 可选的搜索语言
57        pageno: 1                 // 可选的页码
58    };
59
60    try {
61        const response = await searchSearxng(query, options);
62        console.log('搜索结果:', response.results);
63        console.log('response 的 type',typeof response);
64        console.log('results 的 type',typeof response.results);
65        console.log('results 的 type333',typeof response.suggestions);
66    } catch (error) {
67        console.error('搜索出错:', error);
68    }
69  }
70
71
72  // 执行搜索
73  performSearch();
```

1.5.3.2 更改searxng.ts

- 如果使用了以下代码修改，则 config.toml 文件中的 SELFSEARCHURL和 PYPORT填写

> SELFSEARCHURL = 服务器 ip
>
> PYPORT = python 后端服务的地址

既使用 python 启动的后端服务 的接口，后续在代码中会转换成

"http://${SELFSEARCHURL}:${PYPORT} /retrieval"的检索接口

- 地址

> /src/lib/searxng.ts

原代码:同上

修改后

```ts
import axios from 'axios';
import { getSelfSearchUrl } from '../config';

interface SearxngSearchOptions {
  categories?: string[];
  engines?: string[];
  language?: string;
  pageno?: number;
}

interface SearxngSearchResult {
  title: string;
  url: string;
  img_src?: string;
  thumbnail_src?: string;
  thumbnail?: string;
  content?: string;
  author?: string;
  iframe_src?: string;
}


export const searchSearxng = async (
  query: string | string[],
  opts?: SearxngSearchOptions,
) => {
  if (typeof query === "string") { // 如果 query 是字符串类型，将其转换为单元素数组
    query = [query];
  } else if (!Array.isArray(query)) { // 如果 query 既不是字符串也不是字符串数组，抛出一个错误。
    throw new Error("query必须是 【字符串 】或者 【字符串数组】！！！！！");
  }

  // const url = "http://localhost:8013/retrieval";
  const urlroot = getSelfSearchUrl();
  const url = urlroot + "/retrieval";

  console.log('python 服务检索的url: ', url);
  const payload = {
    query_list:query
  };

  try {
    const response = await axios.post(url, payload);
```

```
    const results = response.data.results;
    const suggestions = response.data.suggestions;
    return { results, suggestions };
  } catch (error) {
    console.error('这里出现错误:', error);
    throw error;
  }
};

// // 调用 searchSearxng 函数
// async function performSearch() {
//   const query = '中国平安2023年业绩'; // 你的搜索查询
//   const options: SearxngSearchOptions = {
//       categories: ['general', 'news'], // 可选的搜索类别
//       engines: ['bing', 'google'],       // 可选的搜索引擎
//       language: 'zh-CN',       // 可选的搜索语言
//       pageno: 1                // 可选的页码
//   };

//   try {
//       const response = await searchSearxng(query, options);
//       console.log('搜索结果:', response.results);
//       console.log('response 的 type',typeof response);
//       console.log('results 的 type',typeof response.results);
//       console.log('results 的 type333',typeof response.suggestions);
//   } catch (error) {
//       console.error('搜索出错:', error);
//   }
// }


// // 执行搜索
// performSearch();
```

1.5.4 更改 docker-compose.yaml 文件

1.5.4.1 删除 Searxng

- 原 Perplexica 搜索部分是 SearXNG，这个需要启动一个独立的容器，现在将搜索部分替换后，直接将 SearXNG 部分给删除（注释掉）

```
  # searxng:
  #   image: docker.io/searxng/searxng:latest
  #   volumes:
  #     - ./searxng:/etc/searxng:rw
  #   ports:
  #     - 4000:8080
```

```
#   networks:
#     - perplexica-network
#   restart: unless-stopped



    # depends_on:
#     - searxng
```

1.6 接口相关

1.6.1 图片视频接口返回

1.6.1.1 输入值

- 用户 query

1.6.1.2 返回值

- **results**：由多个字典组成的列表

  每个字典需要包含的字段原代码

```
title: string;
url: string;
img_src: string;
thumbnail_src: string;
thumbnail: string;
content: string;
author: string;
iframe_src: string;
```

- **suggestions**:默认返回空列表即可
- 备注：

  图片接口的请求参数中的 `engines: ['bing images', 'google images']`

  视频接口的请求参数中的 `engines: ['youtube']`

1.6.1.3 图片接口

- 原代码

```
    RunnableLambda.from(async (input: string) => {
      const res = await searchSearxng(input, {
        engines: ['bing images', 'google images'],
```

```
        });//使用 searchSearxng 函数执行搜索，指定搜索引擎为 'bing images' 和 'google images'。

        const images = [];

        res.results.forEach((result) => {//遍历搜索结果，筛选出包含图片源 (img_src)、链接 (url) 和标题 (title) 的结果。
          if (result.img_src && result.url && result.title) {
            images.push({
              img_src: result.img_src,
              url: result.url,
              title: result.title,
            });
          }
        });

        return images.slice(0, 10);// 返回前 10个结果
      }),
```

results 中需要包含的字段示例(两条)：


```
[
  {
    "template":"images.html",
    "url": "https://www.baogaoting.com/info/494035",
    "thumbnail_src": "https://tse1.mm.bing.net/th?id=OIP.xWvO2Z4yyAQg2rNYt3QAkAHaEK&pid=15.1",
    "img_src": "https://imgcdn.baogaoting.com/PDFImage/2023/08/31/cbba8ccc6b91467f9add74be3ca5fbfd.jpg",
    "title": "中国平安2023年中期业绩报告-48页_报告-报告厅",
    "engine": "bing images",
    "engines": [
      "bing images"
    ],
    "positions": [
      1
    ],
    "score": 1.0,
    "category": "images"
  },
  {
    "template": "images.html",
```

```
    "url": "https://www.deviantart.com/blinchikart/art/xiaomi-mira-968479829",
    "thumbnail_src": "https://222.png",
    "img_src": "https://222.png",
    "title": "xiaomi mira by BlinchikART, visual art",
    "engines": [
      "deviantart"
    ],
    "positions": [
      1
    ],
    "score": 1.0,
    "category": "images"
  },
]
```


1.6.1.4 视频接口

- 原代码

```
      RunnableLambda.from(async (input: string) => {
        const res = await searchSearxng(input, {
          engines: ['youtube'],
        });

        const videos = [];

        res.results.forEach((result) => {
          if (
            result.thumbnail &&
            result.url &&
            result.title &&
            result.iframe_src
          ) {
            videos.push({
              img_src: result.thumbnail,
              url: result.url,
              title: result.title,
              iframe_src: result.iframe_src,
            });
          }
        });

        return videos.slice(0, 10);
      })
```

results 中需要包含的字段示例(两条)：(必须包含 url,title,img_src,iframe_src)


```
[
    {
        "template": "videos.html",
        "url": "https://www.youtube.com/watch?v=np2SOHeGpCA",
        "title": "中國歷史24個朝代，每個朝代滅亡原因是什麼？",
        "content": "",
        "author": "歷史面面觀",
        "length": "30:42",
        "iframe_src": "https://www.youtube-nocookie.com/embed/np2SOHeGpCA",
        "engine": "youtube",
        "engines": [
            "youtube"
        ],
        "positions": [
            1
        ],
        "score": 1.0,
        "category": "videos"
    },
    {
        "url": "https://www.youtube.com/watch?v=mF3I8nKlPFc",
        "title": "快速有趣地看完中国历史，不同朝代和著名人物 | 中国历史时间轴 | 简懂笔记",
        "content": "",
        "author": "简懂笔记",
        "length": "9:15",
        "template": "videos.html",
        "iframe_src": "https://www.youtube-nocookie.com/embed/mF3I8nKlPFc",
        "thumbnail": "https://i.ytimg.com/vi/mF3I8nKlPFc/hqdefault.jpg",
        "engine": "youtube",
        "engines": [
            "youtube"
        ],
        "positions": [
```

```
        2
      ],
      "score": 0.5,
      "category": "videos"
    }]
```

1.6.1.5 东京机上的 searxng 测试(服务不一定开了)

- curl

```
curl --location --request POST 'http://38.54.50.54:4000/search?format=json&q=中国的历史朝代&categories=images&engines=bing%20images&language=zh-CN&pageno=1'
```

1.6.2 python 服务

- buld 镜像

```
docker build -t  perplexica-rag:1.0   . -f py.dockerfile
```

- 启动

```
docker run -d -p 8012:8012 --name perplexica-rag perplexica-rag:1.0
```

- helath

```
curl --location 'http://localhost:8013/health'
```

- 改写接口

```
curl --location 'http://localhost:8013/rewrite_query' \
--header 'Content-Type: application/json' \
--data '{
    "query":"24年的大学生就业情况怎么样"
}'
```

- 检索接口

```
curl --location 'http://localhost:8013/retrieval' \
--header 'Content-Type: application/json' \
--data '{
    "query_list": ["24年的大学生就业市场是什么样的", "24年大学生就业的主要行业和岗位有哪些", "如何提高24年大学生的就业竞争力"]
}
'
```

1.6.3 Perplexica 自带的后端接口

- health

```
curl --location 'http://127.0.0.1:3001/api'
```

- 获取 config（包含 config.toml 中的内容以及 mode列表）

```
curl --location 'http://127.0.0.1:3001/api/config'
```

- 更改 config (可更改 config.tom 中的内容)

```
curl --location 'http://127.0.0.1:3001/api/config' \
--header 'Content-Type: application/json' \
--data '{
  "groqApiKey": "11111"
}'
```

- 获取所有聊天

```
curl --location 'http://127.0.0.1:3001/api/chats'
```

- 获取已经配置的 model 和 embedding model

```
curl --location 'http://127.0.0.1:3001/api/models'
```

- 根据 chatid 获取 chat (每个页面的所有聊天),chatid 可从获取所有聊天中找到

```
curl --location 'http://127.0.0.1:3001/api/chats/${chatid}'
```

- 根据聊天历史生成建议，聊天历史可从上面找到

```
curl --location 'http://127.0.0.1:3001/api/suggestions' \
--header 'Content-Type: application/json' \
--data '{
    "chatHistory": [
        {
            "id": 16,
            "content": "什么是保险",
            "chatId": "c539389ea79786145734b6289b9f0095e14b8b32",
            "messageId": "033eee2ea15ddc",
            "role": "user",
            "metadata": "{\"createdAt\":\"2024-11-28T01:26:35.194Z\"}"
        },
        {
            "id": 17,
            "content": "保险是一种金融工具，旨在为个人或企业提供财务保护，以应对意外损失。以下是关于保险的详细解释：",
            "chatId": "c539389ea79786145734b6289b9f0095e14b8b32",
            "messageId": "9266de787dab70",
            "role": "assistant",
            "metadata": ""
```

```
            }
        ],
        "chatModel": { "provider": "openai", "model": "qwen-long" }
        }'
```

- discover 页面

```
curl --location 'http://127.0.0.1:3001/api/discover'
```