



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

CS3321-数据库技术-大作业报告

Connected-Advisor: Find Your Best Tutor

学院 电子信息与电气工程学院

专业 IEEE Honored Class

姓名 朱鹏翔, 刘易洲, 黄奔皓

2024 年 5 月 27 日

目录

1 需求分析	2
1.1 设计初衷与动机	2
1.2 数据字典	3
1.3 数据流图	3
2 概念设计	5
3 系统展示	7
3.1 介绍界面	7
3.2 主界面	7
3.3 功能页	7
3.3.1 图表交互	7
3.3.2 列表视图	9
3.3.3 搜索功能	10
3.3.4 数据上传页	11
3.3.5 统计数据页	12
3.3.6 Connected Advisor 小助手	12
4 系统实现	12
4.1 整体框架介绍	14
4.2 步骤一：数据库搭建	15
4.3 步骤二：API 构建	15
4.4 步骤三：搭建前端框架	15
5 局限性与未来的工作	19
6 组内分工	20
7 总结与致谢	20

1 需求分析

1.1 设计初衷与动机

找到合适的导师是每位科研人员迈向成功的重要一步。无论是即将毕业的本科生寻找研究导师，还是研究生和博士生选择科研方向和导师，深入了解潜在导师的信息都至关重要。虽然导师的学术能力和研究方向可以通过 Google Scholar、个人主页或学校

官网等公开渠道获取，但关于导师的个人品质、师生关系、资源和人脉等信息则难以直接获取。

在这种情况下，通过导师的现有学生、合作伙伴或其他联系人来获取这些信息显得尤为重要。此外，如果能通过共同认识的人引荐，与导师的沟通会更加自然，合作也会更顺畅。为此，我们开发了“Connected Advisor”系统。该系统的灵感来源于著名的“六度分离理论”，该理论指出，通过最多六个人的关系网，我们就能够认识任何一个陌生人。

“Connected Advisor”让用户可以轻松查询和筛选导师信息，并查看导师与其他导师的关系图谱。系统支持在图谱上进行便捷的跳转操作，帮助用户构建和拓展自己的学术和职业网络。这不仅提高了寻找导师的效率，还增加了选择的透明度和可能性。

我们特意选择使用“Advisor”一词而不是“Scholar”，是因为“Advisor”可以直译为“建议者”。**我们希望强调的是，每一个节点的学者对用户的信息价值——这些节点不仅是科研技术上的指导者，更是科研经验和教训的提供者，能够在选导师、科研方向等方面给予宝贵的建议。**

1.2 数据字典

Tab. 1 为 Connected Advisor 的数据字典。注意，其中我们做了最小程度的简化。此外，`connections.connection-strength` 在实际应用中并不起作用，我们最终采取的实现策略是动态计算。将这一项保留的原因是我们考虑后续工作中对计算得到的数值进行缓存，以加快数据读取与展示的过程，给用户以更好的交互体验。

1.3 数据流图

这个章节展示了 Connected Advisor 的数据流图。Fig. 1 展示了顶层数据流，即用户向我们的系统提交查询，或是上传/更新 Advisor 信息的请求后，我们的系统为用户提供相应的反馈信息。

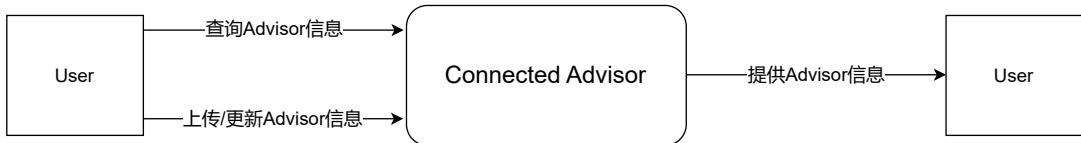


图 1：顶层数据流图

Fig. 2 展示了中间层数据流，它细化了顶层数据流图中的过程。我们在图中突出了对于用户上传的 Advisor 信息存在的审核机制。

Fig. 3 为顶层数据流，它进一步细化了中间层数据流图中的过程，展示了更详细的数据查询步骤和有管理员的审核回路。

表 1: Connected Advisor 数据字典 (已做简化)

数据项	含义/别名	类型	长度	取值范围	取值含义
advisor.id	导师 id	字符串	24	非空	唯一标识导师
advisor.name	导师姓名	字符串	255	非空	
advisor.position	导师职位	字符串	255	非空	
advisor.contacts.email	导师邮箱	字符串	255	符合邮箱格式	
advisor.contacts.twitter	导师推特	字符串	255	符合 url 格式	
advisor.contacts.linkedin	导师领英	字符串	255	符合 url 格式	
advisor.publication.google-scholar	导师谷歌学术	字符串	255	符合 url 格式	
advisor.publication.dblp	导师 dblp	字符串	255	符合 url 格式	
advisor.publication.research-gate	导师 research gate	字符串	255	符合 url 格式	
advisor.publication.semantic-scholar	导师 semantic scholar	字符串	255	符合邮箱格式	
advisors.department	部门名称	字符串	255	非空	
advisors.institute	机构名称	字符串	255	非空	
advisor.homepage	导师主页	字符串	255	符合 url 格式	
advisor.github	导师 github	字符串	255	符合 url 格式	
advisor.tags	标签	字符串列表	无限制		
advisor.descriptions	描述	文本	无限制		
advisor.picture	头像	图片 url	255		
advisor.connections	导师人脉	字符串	255		
paper.id	论文 id	字符串	24	非空	唯一标识论文
paper.year	发表时间 (年)	字符串	255	符合日期格式	
paper.name	论文名	字符串	255	非空	
paper.abstract	论文摘要	文本	无限制		
paper.url	论文链接	字符串	255	符合 url 格式	
paper.authors	作者	字符串列表	无限制	外码 (advisor.id)	
relation.id	关系 id	字符串	24	非空	唯一标识关系
relation.id-1	关系中的角色 1	字符串	24	非空	
relation.id-2	关系中的角色 2	字符串	24	非空	
relation.role-1	角色 1	字符串	255	非空	
relation.role-2	角色 2	字符串	255	非空	
relation.type	关系类型	字符串	255	非空	
relation.duration.start	关系开始时间	日期时间	255	符合日期时间格式	
relation.duration.end	关系结束时间	日期时间	255	符合日期时间格式	
connections.id	连接 id	字符串	24	非空	唯一标识连接
connections.id-1	连接中的角色 1	字符串	24	非空	
connections.id-2	连接中的角色 2	字符串	24	非空	
connections.relations	关系集合	字符串列表	非空	外码 (relation.id)	
connections.collaborated-papers	合作论文集合	字符串列表	无限制	外码 (paper.id)	
connections.last-connected	最后连接时间	日期时间	255	符合日期时间格式	
connections.connection-strength	连接强度	整数	11	≥ 0	

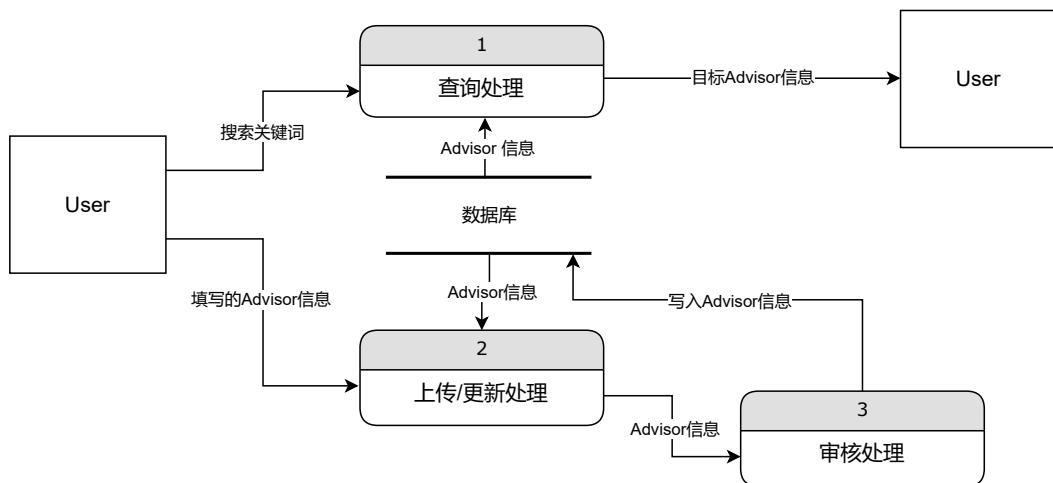


图 2: 中间层数据流图

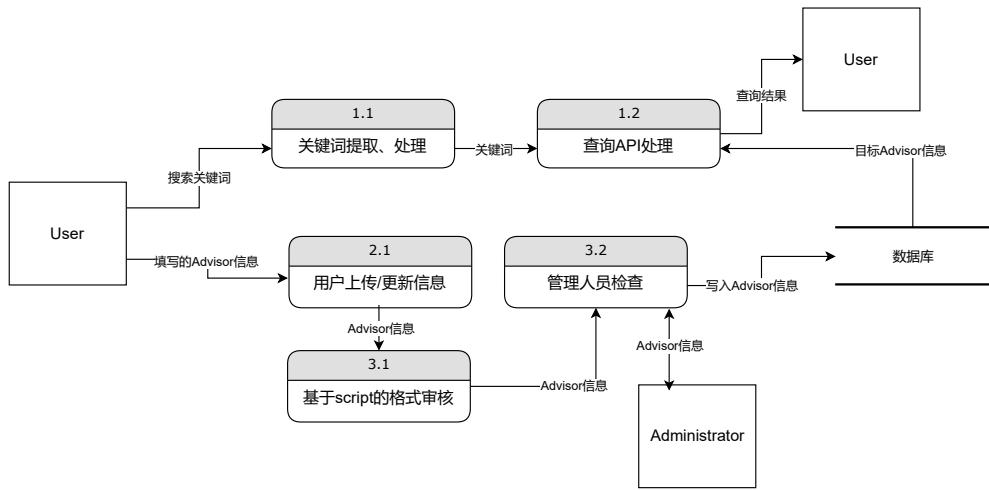


图 3: 底层数据流图

2 概念设计

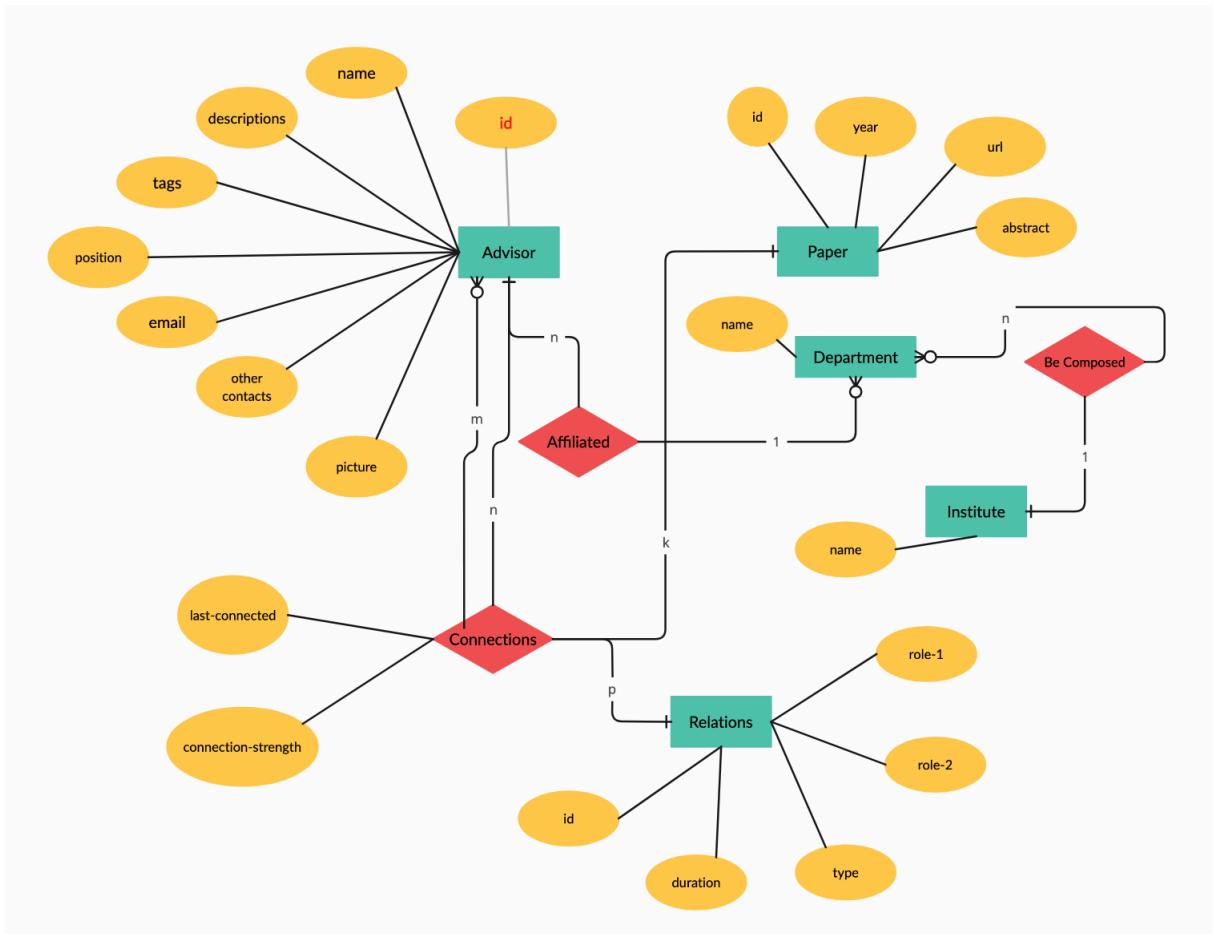


图 4: Connected Advisors: ER 图。其中 Department 和 Institute 在实际数据存储中被归并到了 Advisor 的属性里。

Fig. 4 中展示了 Connected Advisor 的 ER 图。其中有 Advisor, Paper, Department, Institute, Relations 五个实体，和 Affiliated, Connections 以及 Be Composed 三个关系。

由于 Connected Advisor 的数据性质更适合图数据库，我们在实践过程中使用了 MongoDB 数据库而不是传统的关系型数据库。MongoDB 是用了文档存储的方式，更为灵活直观。由于我们并非使用传统的关系型数据库，所以从应用的角度出发，我们并没有进一步从 ER 图推导关系模型。

特性	MongoDB	关系型数据库
数据模型	文档存储 (BSON/JSON)	表格存储
模式设计	动态模式，灵活的文档结构	固定模式，预定义表结构
扩展性	水平扩展，通过分片技术实现	垂直扩展，通过增加硬件资源实现
高可用性	通过复制集实现高可用性	通过主从复制和集群实现高可用性
查询语言	MongoDB 查询语言 (MQL)	SQL (结构化查询语言)
事务支持	支持多文档事务，但较关系型数据库有限	完整的事务支持，ACID 属性
数据类型	支持多种数据类型，包括嵌套文档和数组	关系型数据类型，通常为标量值
应用场景	大数据处理、实时分析、内容管理系统等	传统业务系统、财务系统、ERP 等

表 2: MongoDB 与关系型数据库对比

3 系统展示

在这一章节中，我们将对 Connected Advisor 的系统进行功能展示。

3.1 介绍界面

系统的介绍界面由标题、来自主创团队的简短介绍和对系统模块的图片展示组成。其中图片展示为瀑布流排版，美观直接地展示了 Connected Advisors 内部的一些交互图片。点击最下方的 Enter Main Page 按钮即可进入主界面。

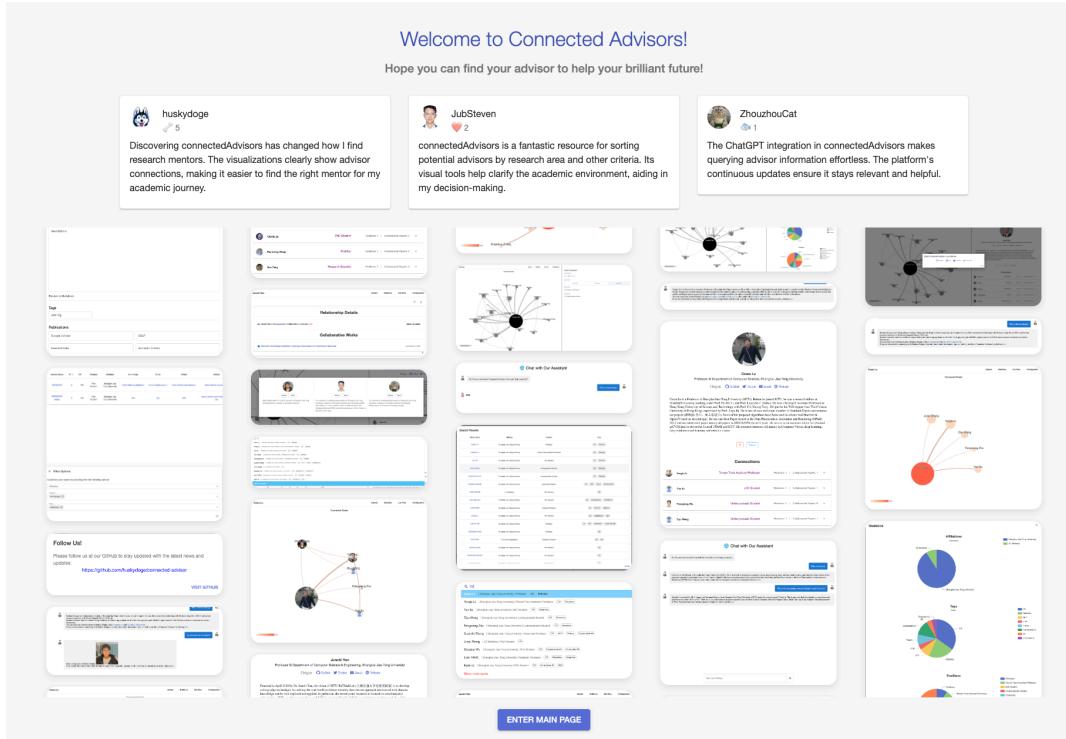


图 5: Connected Advisor 介绍界面展示

3.2 主界面

主界面 (Fig. 6) 由搜索栏、导航栏、Advisor 网络及 Advisor 卡片组成，是用户与系统交互的主要方式。

3.3 功能页

3.3.1 图表交互

Advisor 网络图表是 Connected Advisors 的主要界面 (Fig. 7)。对于主节点，我们采用了明显的颜色和加大的节点大小来表示。图支持拖动和放大。当鼠标悬浮在不同节

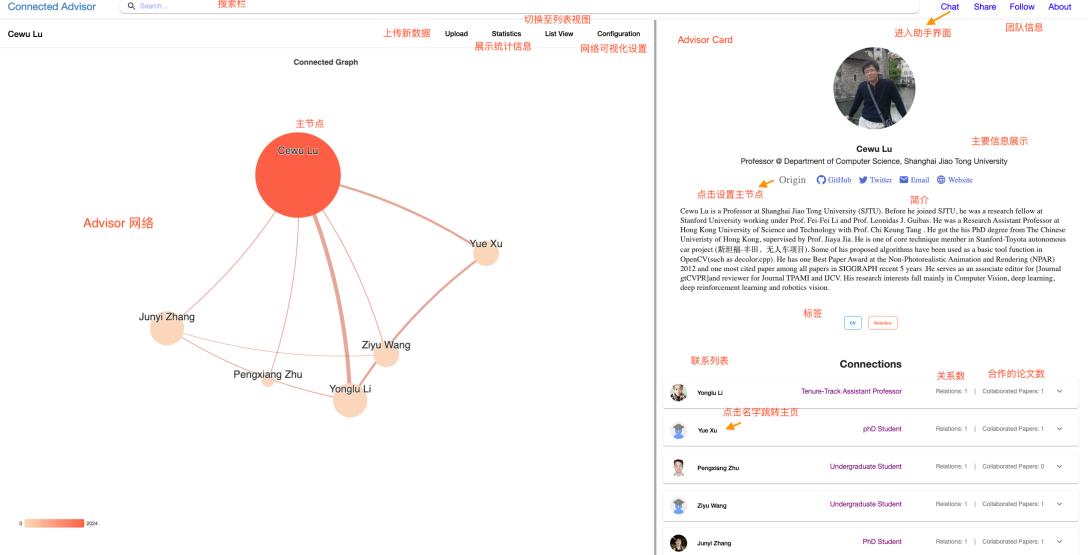


图 6: 主界面展示

点上时，右侧的 AdvisorCard 会跟随变化，展示 Advisor 的信息，且会显示该 Advisor 的 Influence Factor。当鼠标悬浮在边上时，会展示两个 Advisor 之间的 relations 信息。

图中节点的颜色与其和主节点之间最近一次合作时间有关。合作时间越近，其颜色越深；而边的粗细则和两个 Advisor 之间的 relationFactor 成正比。

假设 $|Advisor.connections| = k$ ，即该 Advisor 有 k 个人脉，那么：

$$\text{InfluenceFactor} = k + \sum_{i=1}^k |Advisor.connections[i].collaboratedPapers| \quad (1)$$

relationFactor 定义如下：

$$\text{relationFactor} = \sum_{i \in \mathcal{S}} W_i \cdot S_i, \quad \mathcal{S} = \{\text{tag}, \text{relation}, \text{paper}\} \quad (2)$$

其中：

$$S_{\text{tag}} = |Advisor1.tags \cap Advisor2.tags|$$

$$S_{\text{paper}} = |Connection.CollaboratedPapers|$$

$$S_{\text{relation}} = \sum_{r \in \mathcal{R}} RMap[r.type] \cdot (r.end - r.start)$$

其中 W_i 为固定值，我们设定了 $W_{\text{tag}} = 2, W_{\text{relation}} = 10, W_{\text{paper}} = 5$

主节点切换 当鼠标悬浮在某个主节点上之后，我们可以通过点击 Origin 按钮便捷地切换主节点。

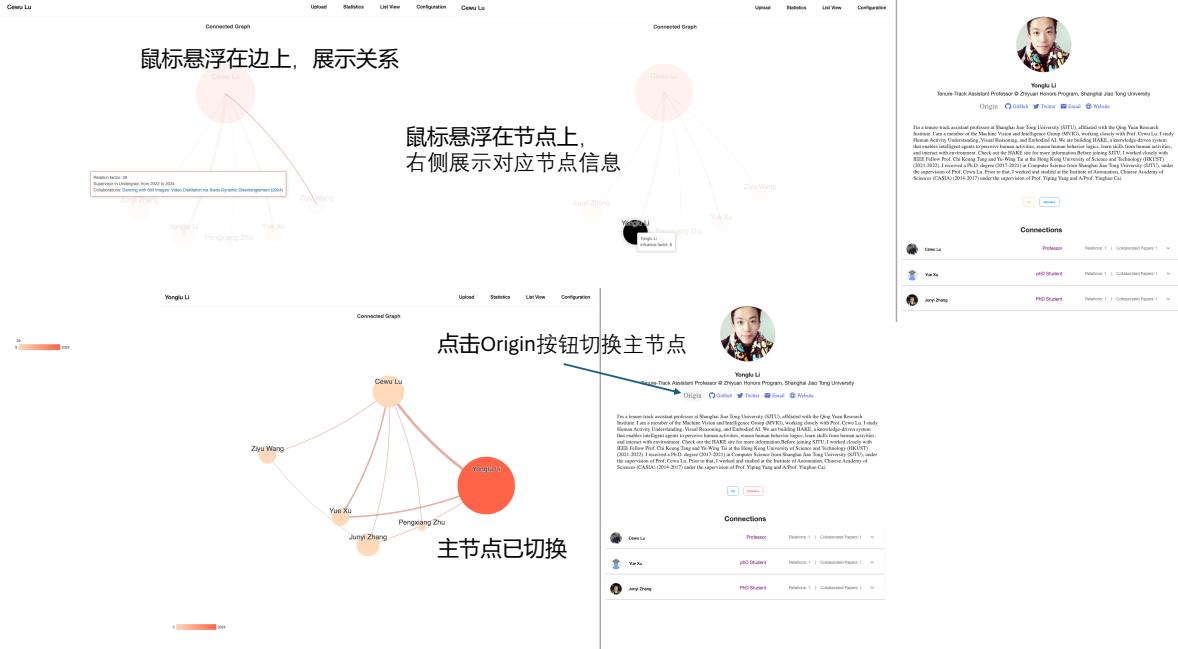


图 7: 图表交互功能展示

图表设置 为了为用户提供更好的体验，我们加入了图表主题设置功能 (Fig. 8):

- 可调节图表为有向图或无向图。
- 可设置为力导向图 (force) 或环形图 (circular)
- 支持调整图表的深度，从 1 度到 3 度不等。1 度表示从主节点出发一步可达的节点，2 度表示两步可达的节点，3 度表示三步可达的节点。由于计算量呈指数级增长，我们仅支持到三度人脉。
- 可设置图表的不同配色方案。
- 可选择是否在节点上显示 Advisor 的头像。

3.3.2 列表视图

图视图直观易懂，但当数据过多时，会有些混乱，为此我们额外引入了列表视图 (Fig. 9)，即和主节点有所联系的人脉列表。列表中我们展示了每个 Advisor 的基础信息，我们还加入了以下功能。

快速跳转 通过点击列表中的人名可以快速跳转到以该 Advisor 作为主节点的页面。

排序 列表支持按照 influenceFactor, relationFactor 对 Advisor 进行升、降排序。

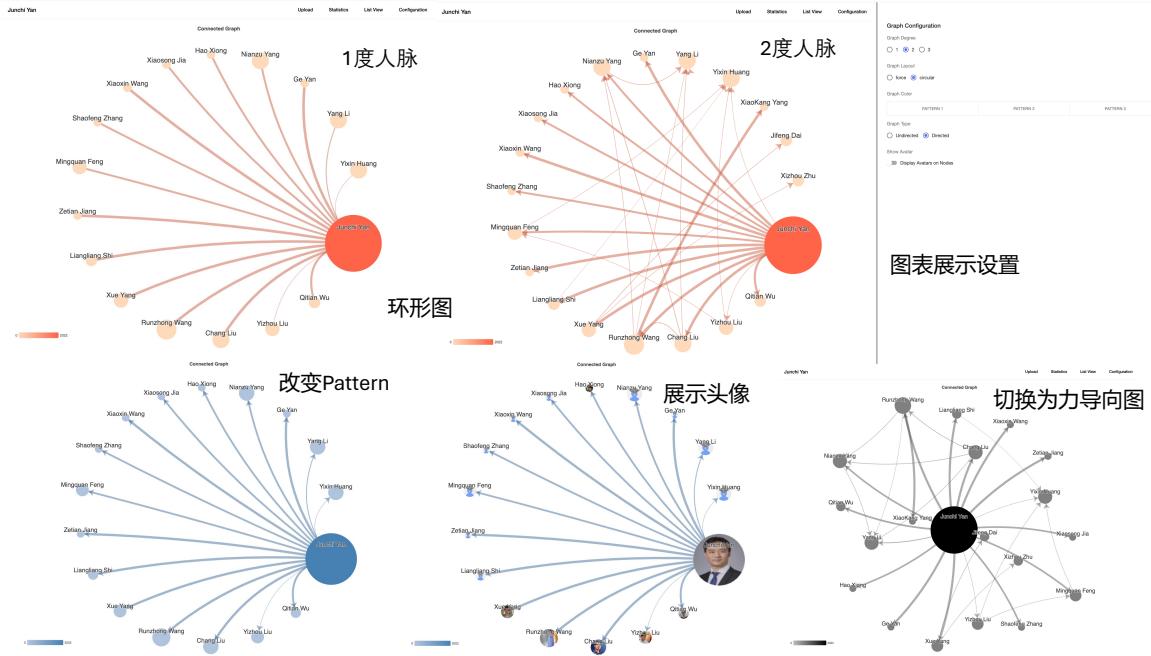


图 8: 图表设置功能展示

过滤 列表支持对 List 按照所属学校、机构，Advisor 的职位和标签进行筛选。

关系展开 通过点击 Show Relation 按钮可以得到关系的细节信息。

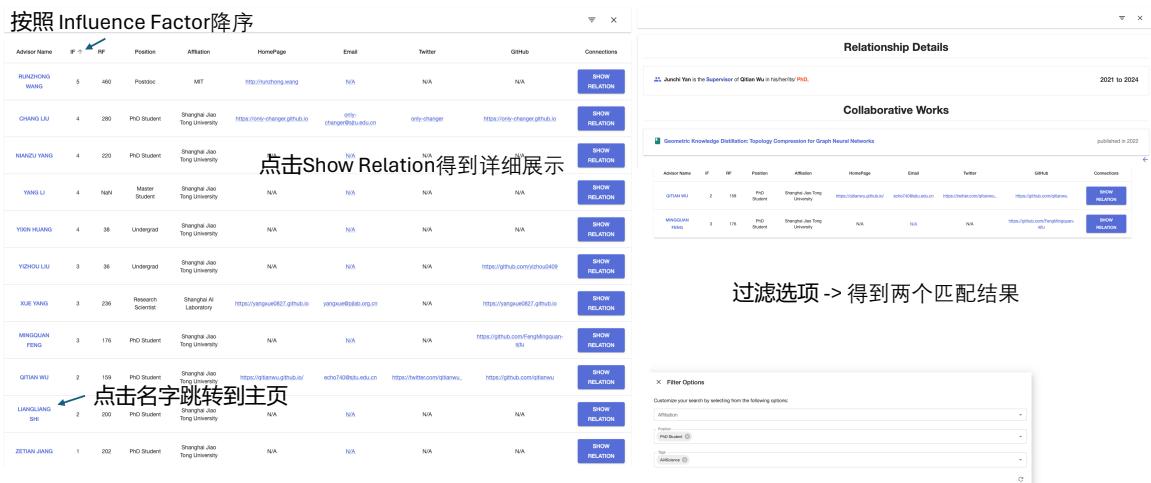


图 9: 列表视图展示

3.3.3 搜索功能

在顶部的搜索栏中 (Fig. 10)，我们支持了对 Advisor 的模糊搜索。用户可以输入 Advisor 名字、学校或机构名，甚至标签等。对于匹配数过多的结果，我们从美观角度出发，加入了展示更多按钮。

Connected Advisor

按照学校搜索

支持多种搜索方式

按照标签搜索

Search Results

Advisor Name	Affiliation	Position	Tags
Cewu Lu	Shanghai Jiao Tong University	Professor	CV Robotics
Yonglu Li	Shanghai Jiao Tong University	Tenure-Track Assistant Professor	CV Robotics
Yue Xu	Shanghai Jiao Tong University	PhD Student	CV Robotics
Ziyu Wang	Shanghai Jiao Tong University	Undergraduate Student	CV Robotics
Pengxiang Zhu	Shanghai Jiao Tong University	Undergraduate Student	CV Robotics
Benhuo Huang	Shanghai Jiao Tong University	Undergrad	NLP LLM
Quanshi Zhang	Shanghai Jiao Tong University	Associate Professor	CV NLP Theory Explainable AI
Lu Chen	Shanghai Jiao Tong University	PhD Student	Explainable AI Theory ML
Junyi Zhang	UC Berkeley	PhD Student	CV
Xiaoqian Wu	Shanghai Jiao Tong University	PhD Student	CV Explainable AI Embodied AI

Show more results

点击展开成列表

按照姓名搜索

Q, Ce

Cewu Lu	Shanghai Jiao Tong University Professor	CV Robotics
Junchi Yan	Shanghai Jiao Tong University Professor	CV NLP AI4Science Graph Learning
Qitian Wu	Shanghai Jiao Tong University PhD Student	ML Graph Learning AI4Science
Yizhou Liu	Shanghai Jiao Tong University Undergrad	AI4Science
Mingquan Feng	Shanghai Jiao Tong University PhD Student	AI4Science
Yixin Huang	Shanghai Jiao Tong University Undergrad	AI4Science

Show more results

CLOSE

图 10: 搜索功能展示

3.3.4 数据上传页

由于数据结构的复杂性，我们无法很好的通过爬虫技术进行数据获取。为此，我们在设计了数据上传功能。我们的系统支持用户上传新的 Advisor 信息和添加或更新 Advisors 之间的联系（Connection）信息。

Fig. 11展示了数据上传界面的内容。上传界面分为两部分：用于上传 Advisor 信息和用于上传 Connection 信息。

在上传 Advisor 的界面中可以看到，我们以表格的形式让用户可以上传新的 Advisor 信息。其中，我们对 Advisor 的信息部分支持了 Markdown 语法，以求包含更多信息（如链接等）。同时，我们还对输入信息的情况进行了检查设置，如当要求填写的 field 为空时，系统会提示内容为空；对 email 的格式我们也做了检查处理，以尽可能减少数据库管理人员检查数据的负担。

在上传 Connection 的界面中，我们让用户通过名字搜索两个已有 Advisor 的形式，更新或是添加两者之间新的 Connection。考虑到重名的情况，我们在搜索的返回结果中特意包含了职务和隶属机构的信息，以方便区分。当用户选择了要进行操作的两个 Advisors 后，界面就会展示出已经存在的联系信息，包括合作的论文和它们之间的关系（relation）。在点击 Add Paper 按钮后，会出现表单，此时用户可选择搜索已有的论文，也可以提交新的论文；点击 Add Relation 按钮后，则会出现添加关系的表单，用户按照 Advisor 的顺序填写相应的角色和信息即可。

图 11: 数据上传界面

3.3.5 统计数据页

我们还加入了对数据总体情况的图标展示 (Fig. 12)。点击 Statistics 按钮，即可看到对机构/学校，标签和职位的饼图统计，直观地展示了系统中的数据组成。

3.3.6 Connected Advisor 小助手

为了进一步方便用户与我们系统的交互，我们将 ChatGPT API 整合进了我们的系统。点击 Fig. 6 导航栏中的 Chat 按钮，即可进入聊天界面。

正如 Fig. 13 所展示的，用户可以询问关于某个 Advisor 的信息，可以询问两个 Advisor 之间的关系，可以要求助手向自己推荐一些某个领域的 Advisor，也可以通过综合的指令查找自己需要的 Advisor；另一方面，用户可以直接点击助手提供的信息中的链接，跳转到对应的主页面或者站外页面。我们希望通过这样一个 AI 助手让用户获得更便捷直接的交互体验。

4 系统实现

在上述的展示过程中，我们并没有过多的阐述 Connected Advisor 各个功能的实现过程。在这一章节中，我们将扼要地介绍 Connected Advisor 系统的实现流程，以尽可能帮助读者能够重建我们的系统。我们将从整体框架介绍引入，分几步介绍系统的搭建过程。由于我们提供了网页源代码，所以我们会忽略掉过多的细节，仅对几个重要的步骤进行重点讲解。

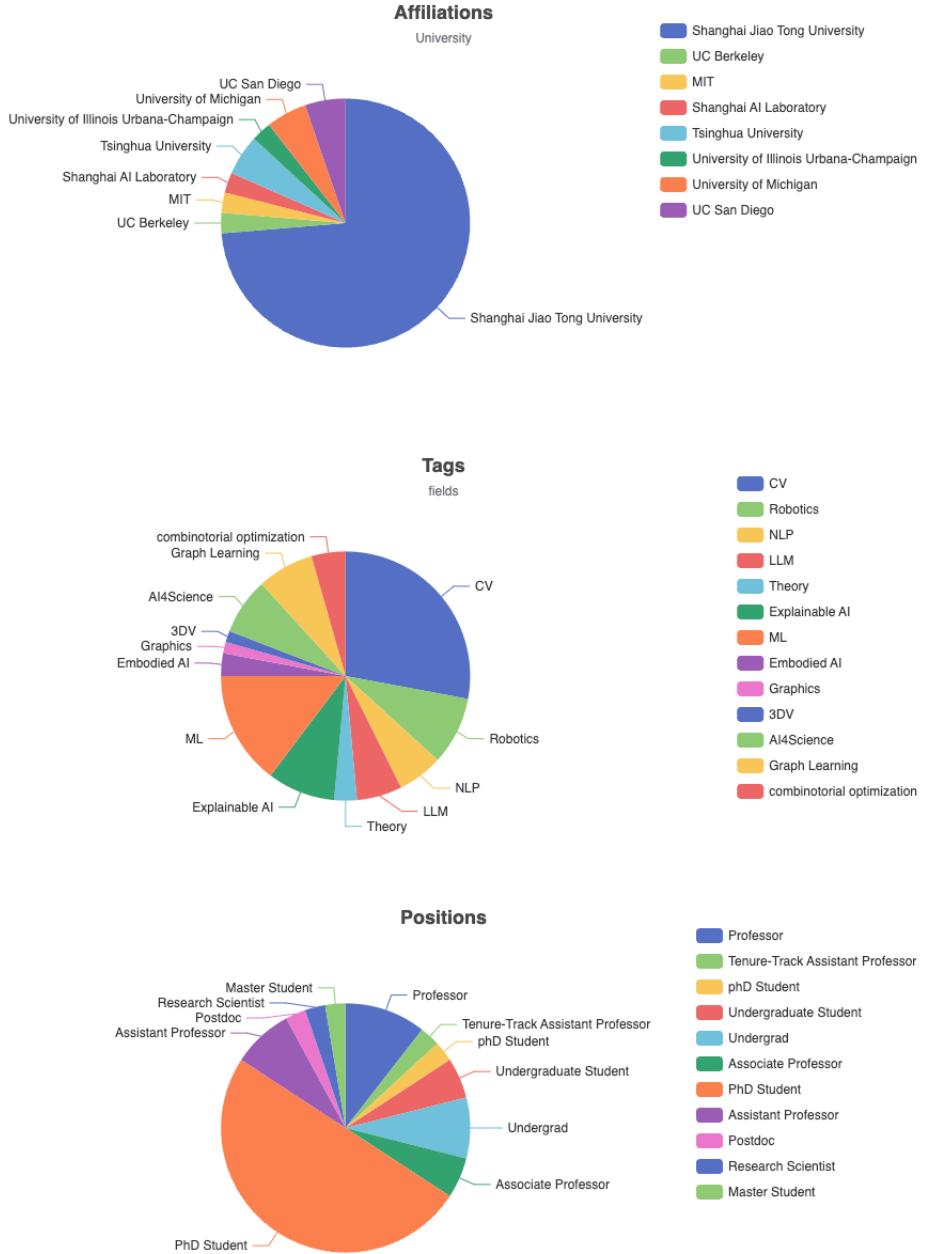


图 12: Connected Adviosrs 数据统计图

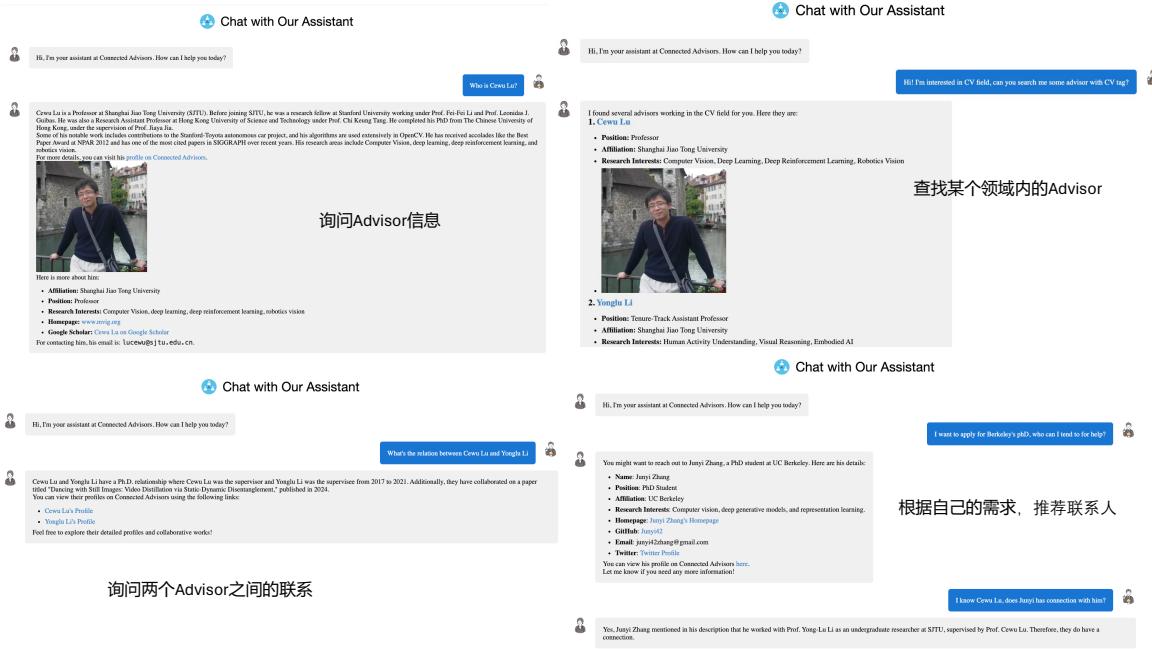


图 13: Connected Advisor 助手功能展示

4.1 整体框架介绍

Connected Advisor 的搭建采用了如下的框架与技术：

- 前端: Next.js + React
- 后端: MongoDB + Next API
- UI 框架: Material UI
- 可视化: Echarts.js

Connected Advisor 的前端框架采用了 Next.js¹。Next.js 是一个由 Vercel 开发的 React 框架，旨在简化构建现代 Web 应用的过程。它通过提供服务器端渲染 (SSR) 和 静态站点生成 (SSG) 等功能，使得开发者能够轻松构建高性能和 SEO 友好的 Web 应用。

在后端方面，我们使用了 MongoDB²数据库来存储数据。MongoDB 是一个面向文档的 NoSQL 数据库，以其灵活的模式设计和高扩展性著称。结合 Next API，我们能够高效地管理数据并提供稳定的后端服务。

对于项目整体 UI 设计，我们使用了 Material UI³。Material UI 是一个受 Google Material Design 启发的 React 组件库，通过其现代化的 UI 组件，能够快速便捷的帮助我们构建出色的用户界面。

¹NextJs 官方教程，通过一个小项目快速入门

²MongoDB 官方网址

³Material UI 官方网址

为了实现数据的可视化，我们采用了 Echarts.js⁴。Echarts.js 是一个开源的 JavaScript 可视化库，提供了丰富的图表类型和高度的可定制性，使我们能够以直观的方式展示数据。在我们的项目中，关系图、数据图均采用 echarts 进行可视化。

4.2 步骤一：数据库搭建

搭建 Connected Advisors 系统的第一步是建立数据库。我们根据此处的[教程](#)，将 MongoDB 数据库建立在了远程服务器上，

在 MongoDB 部署完毕后，我们根据数据字典 (Tab. 1 及 ER 图 (Fig. 4) 以及的设计创建如下表格：

- advisors: 存储每个 Advisor 的信息，每个 Advisor 由独立的 ID 标识。
- papers: 存储每个 Paper 的信息，每个 Paper 由独立的 ID 标识。
- relations: 存储两个 Advisor 之间的关系，例如是博士生与导师的关系、公司上级和下级的关系等。
- connections: 存储两个 Advisors 之间的具体联系。其本身由一个独立 ID 标识，两个 Advisor ID，若干 PaperID 以及若干 Relation ID 作为外码。其包含比 relation 更丰富的信息。

详细步骤请参考代码文件中的 `README.md`。

4.3 步骤二：API 构建

有了数据，我们需要通过构建 API，来搭建前后端之间的桥梁。下面我们将展示 API 设计以及如何在 NextJS 中创建与 MongoDB 链接的 API。

对于数据库中的每条数据，我们都要根据实际的需求去设置获取、修改、添加这三种类型的 API。Fig. 14 展示了一个添加 Advisor 的简单示例，Fig. 15 展示了对 Advisor 的模糊搜索 API，Fig. 16 则展示了更新 Advisor 的 API。

我们的 API 是基于功能设计的。根据网页中不同部分数据展示的需要，我们设计独立的互不干涉的 API，以保证功能实现的正确性。**在实现过程中，API 的设计也是要根据实际情况不断作出调整的。**

4.4 步骤三：搭建前端框架

搭建好前端的桥梁后，我们开始进行前端框架的搭建。我们首先对网页的路由路径进行设计。用户会首先进入系统的介绍界面 (Fig. 5)。点击 `enter main page` 按钮后就会进入我们的主界面。

⁴[Echarts 官方网址](#)

```
import type { NextApiRequest, NextApiResponse } from "next";
import { MongoClient } from "mongodb";

// MongoDB URL and database name
const MONGO_URL = process.env.MONGO_URL || ""; // Use the environment variable
const DB_NAME = "ConnectedAdvisor";

// Function to connect to the database
async function connectToDatabase() {
  const client = new MongoClient(MONGO_URL);
  await client.connect();
  return client.db(DB_NAME);
}

// API handler function
export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  console.log("Received request:", req.method, req.body);
  if (req.method === "POST") {
    try {
      const db = await connectToDatabase();
      console.log("Database connected");
      const collection = db.collection("connections");
      const result = await collection.insertOne(req.body);
      console.log("Document inserted", result);
      res.status(200).json({ ...result, _id: result.insertedId });
    } catch (error) {
      console.error("Error occurred:", error);
      res
        .status(500)
        .json({ error: "Unable to connect to the database or insert data" });
    }
  } else {
    res.setHeader("Allow", ["POST"]);
    res.status(405).end(`Method ${req.method} Not Allowed`);
  }
}
```

图 14: 该图中展示了添加 Advisor 的 API 代码。从图中可以看到，我们首先要通过创建 MongoClient 对象连接到 MongoDB 数据库；然后在 `handler` 函数中，我们用 `req` 中的内容去数据库中查询信息，并返回到 `res` 中。此外，我们需要加入对异常情况的处理，以方便我们更好地调试代码。

```

import { MongoClient, ObjectId } from "mongodb";

// MongoDB URL and database query_text
const MONGO_URL = process.env.MONGO_URL || "";
const DB_NAME = "ConnectedAdvisor";
const COLLECTION_NAME = "AdvisorTable";

async function connectToDatabase() {
  const client = new MongoClient(MONGO_URL);
  await client.connect();
  const db = client.db(DB_NAME);
  return { db, client };
}

export default async function handler(req, res) {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS");
  res.setHeader("Access-Control-Allow-Headers", "Content-Type");

  if (req.method !== "POST") {
    res.setHeader("Allow", ["POST"]);
    res.status(405).end(`Method ${req.method} Not Allowed`);
    return;
  }

  const { oid, name } = req.body;
  let query_text = name;
  if (!oid && !query_text) {
    res.status(400).json({ message: "Missing query parameters" });
    return;
  }
  try {
    const { db, client } = await connectToDatabase();
    let result;
    if (oid) {
      result = await db
        .collection(COLLECTION_NAME)
        .findOne({ _id: new ObjectId(oid) });
    } else if (query_text) {
      let words = query_text.split(/\s+/); // 拆分输入的query_text为单词数组
      let regexPatterns = words.map((word) => `(?=.*${word})`);
      let regexQuery = new RegExp(regexPatterns.join("|"), "i");

      result = await db
        .collection(COLLECTION_NAME)
        .find({
          $or: [
            { name: { $regex: regexQuery } },
            { tags: { $regex: regexQuery } },
            { position: { $regex: regexQuery } },
            { affiliation: { $regex: regexQuery } },
          ],
        })
        .toArray();
    }
    if (result) {
      res.status(200).json(result);
    } else {
      res.status(404).json({ message: "No matching document found" });
    }
    client.close();
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: "Internal Server Error" });
  }
}

```

图 15: 该图展示了输入部分信息模糊搜索 Advisor 的代码。与 Fig. 14 不同的地方在于，该 API 需要加入对搜索功能的支持。一方面，我们需要构造正则表达式进行匹配；另一方面，我们要借助 MongoDB 的搜索语法做到对若干属性同时进行搜索，找到任意满足要求的 Advisor 项，这由 `$or` 来完成。

```

// API handler function to update advisor information
export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  console.log("Received request:", req.method, req.body);
  if (req.method === "PUT") {
    // Ensure the request method is PUT for updating
    try {
      const db = await connectToDatabase();
      console.log("Database connected");
      const collection = db.collection(COLLECTION_NAME);

      // Extract _id from request body and convert it to ObjectId
      const { _id, ...updateData } = req.body;
      if (!_id) {
        res.status(400).json({ error: "Missing _id in request body" });
        return;
      }

      // Perform the update operation
      const result = await collection.updateOne(
        { _id: new ObjectId(_id) },
        { $set: updateData }
      );

      if (result.modifiedCount === 0) {
        res
          .status(404)
          .json({ message: "No matching document found to update" });
      } else {
        console.log("Document updated", result);
        res.status(200).json(result);
      }
    } catch (error) {
      console.error("Error occurred:", error);
      res.status(500).json({ error: "Internal Server Error" });
    }
  } else {
    res.setHeader("Allow", ["PUT"]);
    res.status(405).end(`Method ${req.method} Not Allowed`);
  }
}

```

图 16: 该图展示了更新 Advisor 数据的 API。其与添加 Advisor 的 Fig. 14 的不同之处在于，在 `handler` 函数中使用了 `collection.updateOne` 函数而非 `collection.insertOne`。前者需要输入要更新的数据的 ID，以及要更新的内容，并使用 `$ set` 进行更新。

我们采用了[动态路由](#)⁵的方式来控制数据的展示，即通过控制路由中代表 Advisor ID 和展示方式的字符串来控制相应内容展示。例如，

`http://localhost:3000/main/6607bc09eb00fa31e8d30829?view=graph`
中, `6607bc09eb00fa31e8d30829` 代表 Advisor 的 ID, `view=graph` 则代表图视图, 切换成 `view=list` 即展示列表视图。

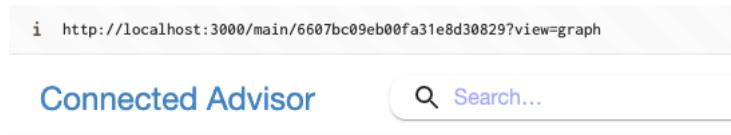


图 17: 动态路由实例展示

在动态路由设计的基础上，我们就可以很好的做到让对应的页面展示正确的数据。有了对数据流的把控，我们进而设计和实现各类组件，如 AdvisorCard, ListView, GraphView 等就会更加灵活和便捷。

关于组件的具体设计与实现，我们便不在报告中一一阐述；请参考代码中 `src/components` 部分下的各个代码。请参阅代码文件顶层的 `README.md`，在其中我们说明了各部分代码所实现的功能。

5 局限性与未来的工作

尽管我们的系统实现了诸多功能，但同时也有诸多不足，我们在此进行简要讨论。

数据 在数据规模上，由于我们数据结构的复杂性，我们无法使用传统的爬虫技术对数据进行爬取；尽管我们尝试了基于大语言模型的非结构化数据爬取，但其效果仍然不理想。这迫使我们开发了数据上传模块。实际应用场景中，我们的数据是需要定期更新的，这种需求的技术实现是颇有难度的。在我们的项目中，我们并没有实现这一功能。此外，对于上传的数据，我们仅仅设计了格式合法性、选项合法性等初步的检查，对于一些注入攻击我们的系统是没有特殊的防范措施的。

可视化 由于我们使用了 `echarts.js` 进行可视化，其本身并没有对图数据的展示进行优化，所以在渲染速度上仍然会有所欠缺。尽管我们使用了浏览器缓存技术进行加速，但是初次的数据可视化载入仍然需要相对长的时间。我们希望在未来的工作中能通过选择更合适的框架、对算法进行优化等方法来加速图表的渲染，为用户提供更好的交互体验。

⁵[NextJS-动态路由](#)

小助手 目前，Chat 助手的功能仍然有所局限，包括不能保存历史记录、缺少更灵活的业内搜索功能等。将大语言模型融合到网页 APP 是技术发展的趋势，我们希望在未来的工作中对此做进一步的探索。

6 组内分工

组内分工如下：

- 朱鹏翔：项目构思与需求设计、后端数据库搭建、数据爬虫设计与尝试
- 刘易洲：概念设计、数据收集、数据维护
- 黄奔皓：项目构思与需求设计、前后端代码接口设计与实现、数据收集

7 总结与致谢

在这次大作业的实践过程中，我们深刻领会了数据库技术在网页应用中的基础地位，探索与学习了当下的前沿前端、后端技术。无论是关于需求设计、数据库选择还是接口搭建，每一次课下和郭老师的交流都让我们有所收获。我们十分感谢郭老师在我们完成本次大作业的过程中给予的悉心指导！

Connected Advisor | SJTU-CS3321-Group-Project

Welcome to Connected Advisors!

Hope you can find your advisor to help your brilliant future!

 huskydoge
5

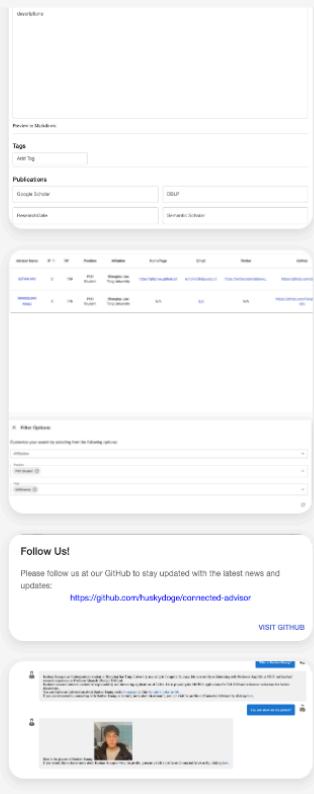
Discovering connectedAdvisors has changed how I find research mentors. The visualizations clearly show advisor connections, making it easier to find the right mentor for my academic journey.

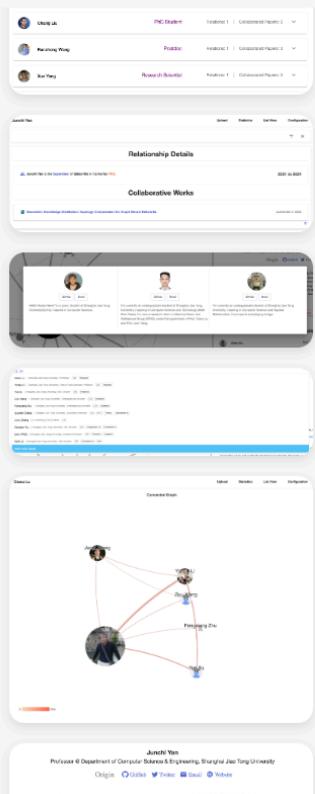
 JubSteven
2

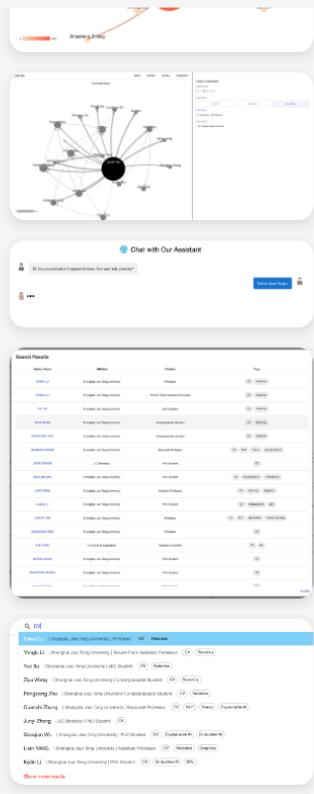
connectedAdvisors is a fantastic resource for sorting potential advisors by research area and other criteria. Its visual tools help clarify the academic environment, aiding in my decision-making.

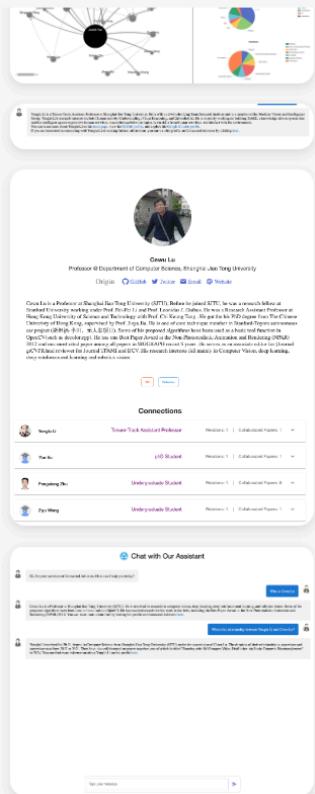
 ZhouzhouCat
1

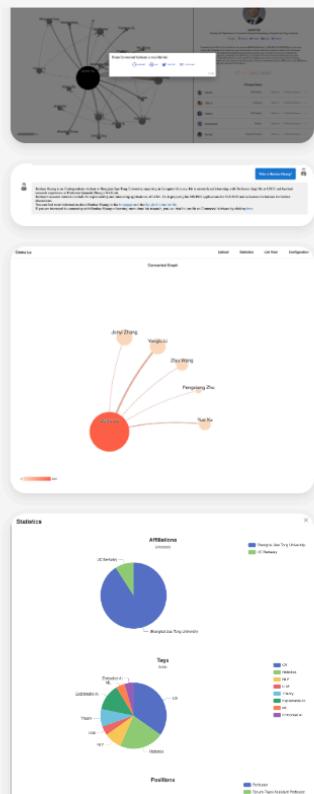
The ChatGPT integration in connectedAdvisors makes querying advisor information effortless. The platform's continuous updates ensure it stays relevant and helpful.





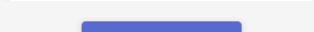


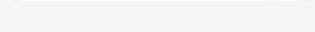














[ENTER MAIN PAGE](#)

设计灵感

找到合适的导师是每位科研人员迈向成功的重要一步。无论是即将毕业的本科生寻找研究导师，还是研究生和博士生选择科研方向和导师，深入了解潜在导师的信息都至关重要。虽然导师的学术能力和研究方向可以通过 Google Scholar、个人主页或学校官网等公开渠道获取，但关于导师的个人品质、师生关系、资源和人脉等信息则难以直接获取。

在这种情况下，通过导师的现有学生、合作伙伴或其他联系人来获取这些信息显得尤为重要。此外，如果能通过共同认识的人引荐，与导师的沟通会更加自然，合作也会更顺畅。为此，我们开发了“Connected Advisor”系统。该系统的灵感来源于著名的“六度分离理论”，该理论指出，通过最多六个人的关系网，我们就能够认识任何一个陌生人。

“Connected Advisor”让用户可以轻松查询和筛选导师信息，并查看导师与其他导师的关系图谱。系统支持在图谱上进行便捷的跳转操作，帮助用户构建和拓展自己的学术和职业网络。这不仅提高了寻找导师的效率，还增加了选择的透明度和可能性。

我们特意选择使用“Advisor”一词而不是“Scholar”，是因为“Advisor”可以直译为“建议者”。我们希望强调的是，每一个节点的学者对用户的信息价值——这些节点不仅是科研技术上的指导者，更是科研经验和教训的提供者，能够在选导师、科研方向等方面给予宝贵的建议。

代码结构

这里我们展示出主要的代码结构：

```
- code
  - README.md: 文档
  - public/: 网站中用到的静态资源，如各种图像等
  - src/:
    - components/: 各种组件
      - mainPage/: 主界面中使用的组件
        - advisorCardComponents/ : AdvisorCard中使用的组件
        - dataRender/ : 实现 graph render 和 list render
        - uploadInfo/ : 处理上传 Advisor 功能
        - mainContent.tsx : 主界面顶层文件
        - ...
    - wrapped_api/ : 按照实体类型对各种API进行包装
    - topMenu.tsx : 顶部菜单栏
    - const.tsx : 存储常量
    - interface.tsx : 定义数据接口类型
    - ImageGallery.tsx : 介绍界面的瀑布流图
    - testimony.tsx : 介绍界面的“用户”反馈
    - MessageForm.tsx : chat 界面提交信息组件
```

```
- MessageList.tsx : chat界面展示信息组件
- searchTable.tsx : 展开搜索结果
- ...
- pages/
  - main/ : 动态路由
  - api/ : 存储了所有API
    - openai/ : 调用OpenAI-API
    - ...
  - _app.tsx : 应用主入口, 对网页整体做了一些外观主题上的定义
  - index.tsx : 介绍界面
  - chat.tsx : 聊天助手界面
- styles/ : 定义css styles
```

数据库结构

```
- ConnectedAdvisor
  - advisors
  - connections
  - relations
  - papers
```

根据这个结构, 将 `mongodb-data` 下的 `.json` 文件 [import](#) 进 MongoDB 数据库中。

项目启动流程

在开始之前, 请确保你的系统已经安装了以下软件:

- **Node.js**: 确保安装最新的 LTS 版本。你可以从[Node.js 官网](#)下载并安装。
- **npm** (Node.js 包管理器) : 通常随 Node.js 一起安装。

下载项目

首先, 通过 `git clone` 将项目下载到本地。

安装依赖

在项目根目录运行以下命令来安装项目所需的依赖项:

```
npm install
```

这会根据 `package.json` 文件中的依赖项列表安装所有必要的包。

构建项目

运行以下命令来构建项目:

```
npm run build
```

这个命令将会生成一个 `.next` 文件夹, 其中包含了打包好的项目文件。

额外信息

如果你在启动项目时遇到问题, 可以参考以下命令来进行排查:

- 检查 Node.js 版本:

```
node -v
```

- 检查 npm 版本:

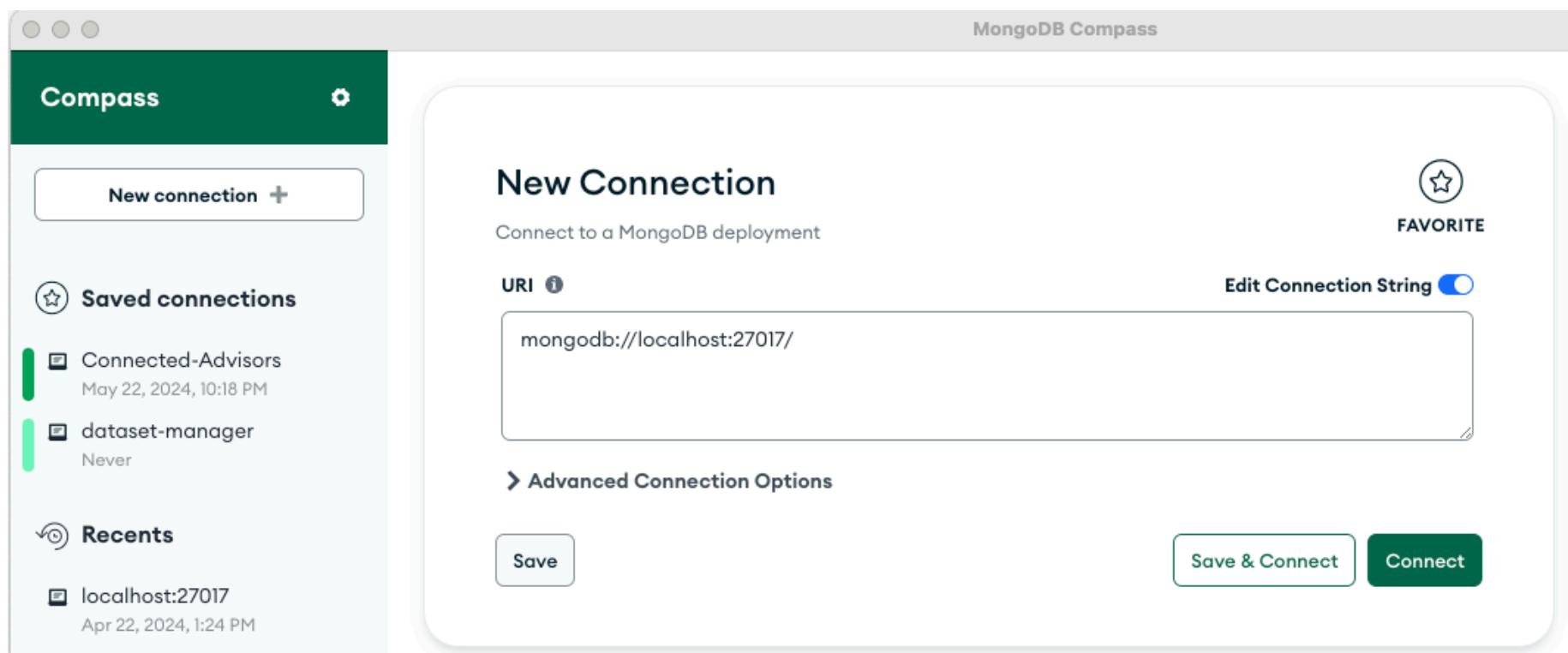
```
npm -v
```

- 清除 npm 缓存 (如果遇到依赖安装问题) :

```
npm cache clean --force
```

连接数据库

要确保正确连接到 MongoDB 客户端。请参考以下步骤：



远程服务器

如果将 MongoDB 部署到远程服务器上，请按以下操作进行数据库连接。在图中可以看到，我们将连接到 `mongodb://localhost:27017`。MongoDB 会在远程服务器的 27017 端口启动。

为了从本地访问远程 MongoDB 实例，需要进行端口转发。请按照以下步骤操作：

1. 打开终端。
2. 输入以下命令以进行端口转发：

```
ssh -L 27017:localhost:27017 -N -f -l username server_ip
```

其中，`username` 是远程服务器的用户名，`server_ip` 是远程服务器的 IP 地址。此命令将远程服务器上的 27017 端口转发到本地的 27017 端口。

本地

如果你在本地搭建 MongoDB，步骤会有所不同。请参考以下操作：

在 Windows 上安装和启动 MongoDB

1. 下载 MongoDB 安装程序：
 - 访问 [MongoDB 下载中心](#)。
 - 选择 Windows 版本，下载 `.msi` 安装文件。
2. 安装 MongoDB：
 - 双击下载的 `.msi` 文件，启动安装向导。
 - 在安装过程中，选择“Complete”安装类型，这会安装所有 MongoDB 工具和功能。
 - 选择安装 MongoDB Compass（可选）。
3. 配置 MongoDB 作为服务：
 - 在安装向导中，选择“Install MongoDB as a Service”，这会将 MongoDB 配置为 Windows 服务并在安装完成后自动启动。
4. 启动 MongoDB 服务：
 - 安装完成后，MongoDB 服务应已自动启动。你可以通过 Windows 服务管理器检查和管理 MongoDB 服务。
5. 验证安装：
 - 打开命令提示符，输入以下命令检查 MongoDB 版本：

```
mongo --version
```

详细步骤可以参考 MongoDB 官方文档和 [TutorialsTeacher](#) 的指南。

在 Linux 上安装和启动 MongoDB

1. 导入公钥：

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
```

2. 创建列表文件:

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
```

3. 更新包列表并安装 MongoDB:

```
sudo apt-get update  
sudo apt-get install -y mongodb-org
```

4. 启动 MongoDB:

```
sudo systemctl start mongod
```

5. 启用开机启动:

```
sudo systemctl enable mongod
```

6. 验证安装:

- 输入以下命令检查 MongoDB 服务状态:

```
sudo systemctl status mongod
```

详细步骤可以参考 [MongoDB 官方文档](#)。

在 macOS 上安装和启动 MongoDB

1. 通过 Homebrew 安装 MongoDB:

- 首先安装 Homebrew（如果尚未安装），然后运行以下命令：

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. 添加 MongoDB 存储库:

```
brew tap mongodb/brew
```

3. 安装 MongoDB:

```
brew install mongodb-community@4.4
```

4. 启动 MongoDB:

```
brew services start mongodb/brew/mongodb-community
```

5. 验证安装:

- 打开终端，输入以下命令检查 MongoDB 版本：

```
mongo --version
```

详细步骤可以参考 [MongoDB 官方文档](#)。

MongoDB 默认使用的端口是 27017。当你启动 MongoDB 服务器时，如果没有指定其他端口，它会默认在 27017 端口上监听。

配置环境变量

在项目根目录下创建一个 `.env.local` 文件，并输入以下内容：

```
MONGO_URL=mongodb://localhost:27017/  
OPENAI_API_KEY=<YOUR OPENAI KEY>
```

- `MONGO_URL`：配置为 `mongodb://localhost:27017/` 以连接本地的 MongoDB 实例。
- `OPENAI_API_KEY`：替换 `<YOUR OPENAI KEY>` 为自己的 OpenAI API 密钥。

启动项目

使用以下命令启动项目：

```
npm run dev
```

项目将在浏览器中运行， 默认地址是 `http://localhost:3000`。

请使用稳定的 VPN ， 否则会存在图片加载不出来的情况。