个人信息脱敏

题目: 基于 Stable Diffusion 的“文生图”/“图生图”系统

一、 序言

包括项目的基本介绍，项目的有关背景（类似的项目的实现情况，包括现在市面上的项目），以及项目的特色之处。

本项目旨在使用 Stable Diffusion 技术搭建一个“文生图”/“图生图”Web 网站。

Stable Diffusion 的是由 Stablility AI 发布的基于深度学习的由文本到图像的生成模型，其核心思想是用 CLIP 模型将文本信息转为语义向量，通过采样器（如 DDPM/DDIM/PLMS）将语义向量输入到 Unet 网络控制纯噪声隐变量的去噪，在迭代去噪的过程中向隐变量注入语义信息，最终得到含有丰富语义的隐空间向量。这个含有丰富语义信息、去除噪声后的隐变量使用解码器映射到像素空间后，即可得到一张包含语义信息的有效照片，从而实现了文本到图片的跨模态转换。

LoRA（低秩适应模型）是微软提出的由于大语言模型的微调技术，由于大模型具有巨量的参数，对整个模型进行微调以适应特定任务或领域的成本非常高，因此 LoRA 在每个 Transformer 块中插入可训练层，只对可训练层的参数进行训练，就能在保证微调效果的前提下实现轻量化的模型调校。

LoRA 技术与 Stable Diffusion 技术结合，极大降低了 SD 模型的训练时间和训练成本，通过为 Stable Diffusion 模型注入特定风格的 LoRA 权重，使得模型能够生成各种风格的图像。

随着 Stable Diffusion 技术的开源，市面上基于 Stable Diffusion 技术搭建的“文生图”/“图生图”的产品越来越多。但是从技术角度和用户实际体验角度来说，大部分产品普遍存在以下问题：

- **种类受限：**生成图像的种类和多样性受到所使用的基模型的限制，若用户选取基模型不恰当，可能无法生成指定种类的图像；
- **质量与时长难以平衡：**生成图像的时长受基模型规模的影响，较大的基模型可能导致生成过程缓慢，增加用户等待的时间，对服务器的显卡负担也更大；而较差的基模型又无法满足用户对作图品质的需求；
- **提示词冗长：**在基模型固定的情况下，生图质量很大程度上取决于用户提示词的准确性和清晰度。若提示词不明确或模糊，则生图结果可能会偏离预期；若要获得较高质量的图像，则用户要需要精心编写冗长的提示词，间接提高了用户使用门槛；
- **未能利用 LoRA：**在调研的过程中发现大部分产品不支持更换作图风格或调用 LoRA，亦或者 LoRA 调用操作过程复杂，对用户存在使用门槛，使得 Stable Diffusion 的一大特性未能发挥；
- **网站功能单一：**大部分产品仅提供生图功能，而缺少生图后的功能衔接，如图片编辑等。

因此，我们的项目要在实现“文生图”/“图生图”功能的基础上，解决上述问题，为用户提供更灵活更实用的产品，提高用户体验，项目特色（功能）如下：

- **功能齐全：**网站除了提供文生图/图生图的核心参数供用户选择，还提供多种采样方法、高清化采样技术、多图生成等功能供用户使用；
- **丰富的作图种类：**网站选择 GhostMix 全能模型作为基模型，使用大量 2.5D 图片作为训练集，自身可以适应 2D、3D、动漫、真人等各种风格的作图种类需求；另外，GhostMix 对 LoRA 有很强的兼容性，可以使用 LoRA 进一步提高作图种类的分化能力；
- **质量与时长平衡：**GhostMix 模型仅 4G 大小，保持较高的作图质量同时极大压缩作图时长，减少用户等待时间；
- **提示词内嵌：**将用户输入和提示词模板在后端进行拼接，用户只需要输入少量关键词即可生成高质量图像，无需编写冗长的提示词以获得高质量图像；
- **风格选择：**在基模型基础上结合 LoRA 提高作图风格分化能力，将 LoRA 模型调用和提示词模板封装为水墨风、像素风、复古漫画风、胶片风等九个作图风格供用户使用，并提供样例图供用户参考，用户无需考虑 LoRA 调用过程和提示词工程即可享受 LoRA 的效果；
- **二次编辑：**网站除了作图功能，还为用户提供了二次编辑功能，用户可以在生成的图像上进行二次编辑；
- **历史作图与参数导入：**网站为每位用户记录生成的图像以及对应的作图参数，用户不仅可以查看历史作图记录，还可以一键导入图片对应的作图参数；
- **用户友好：**网站只保留了作图最核心、最简单的参数供用户填写，极大降低了使用门槛。

基于这些目标，我们最终要以 Stable Diffusion 技术为核心，设计出界面简洁、特色鲜明、功能完善、用户友好、简单易用的“文生图”/“图生图”网站。

二、 项目设计

如图 1 所示，项目由前端和后端两个部分组成：

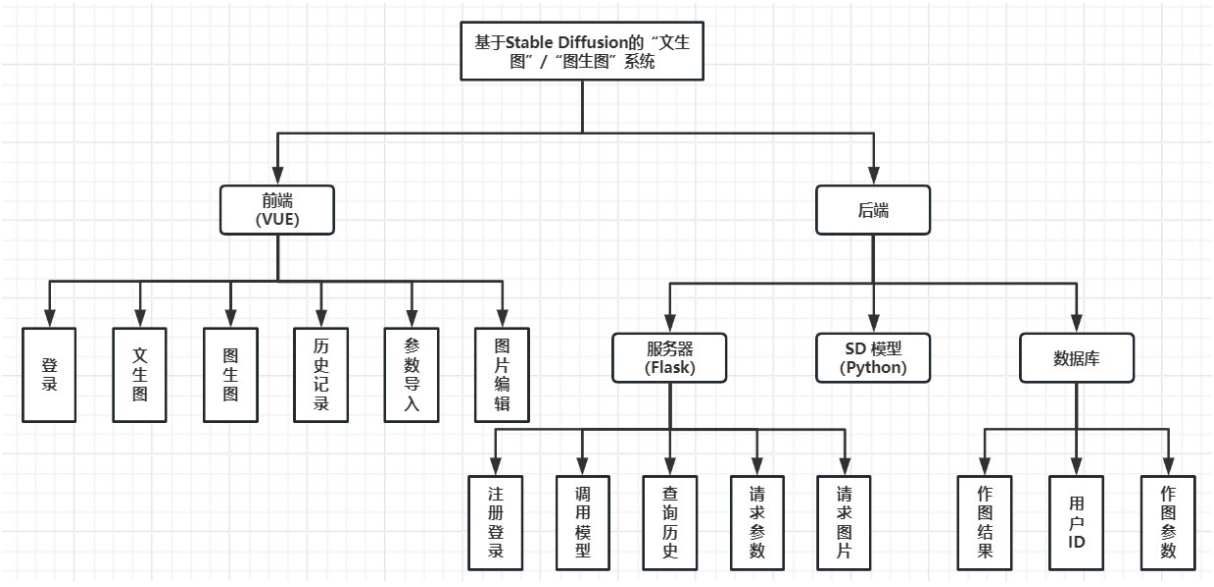

图 1 项目功能框架图

其中，前端 Web 界面分解为登录、文生图、图生图、历史记录、参数导入、图片编辑六个功能模块；

后端分解为与前端页面连接的服务器、部署在本地的 SD 模型、数据库三个部分：服务器有注册登录、调用模型作图、查询历史记录、返回前端参数、返回前端图片等功能模块；SD 模型则负责接收服务器传过来的参数生成图片，然后将结果返回给服务器；数据库则存储用户 ID，以及用户每次作图生成的图片和作图参数。

基于以上模块，我们可以绘制出基础的业务流程图：

① 用户注册/登录到系统；

② 用户选择“文生图”/“图生图”功能，输入关键词、作图风格等必要参数，然后由前端将作图参数打包请求至后端。

③ 后端解析参数，并根据作图风格等信息处理参数，添加提示词、调用 LoRA，然后对参数进行二次封装，并调用 SD 模型作图。作图结果和作图参数返回给前端展示，同时在数据库中保存。

④ 用户在前端查看作图结果，可以选择图片编辑，对生成图像进行二次加工。

⑤ 用户也可以选择历史记录功能，按日期新到旧查看作图记录，也可以将每张图片对应的作图参数导入到作图界面中。后端从数据库中查询到对应的图片和作图参数，即返回至前端展示。

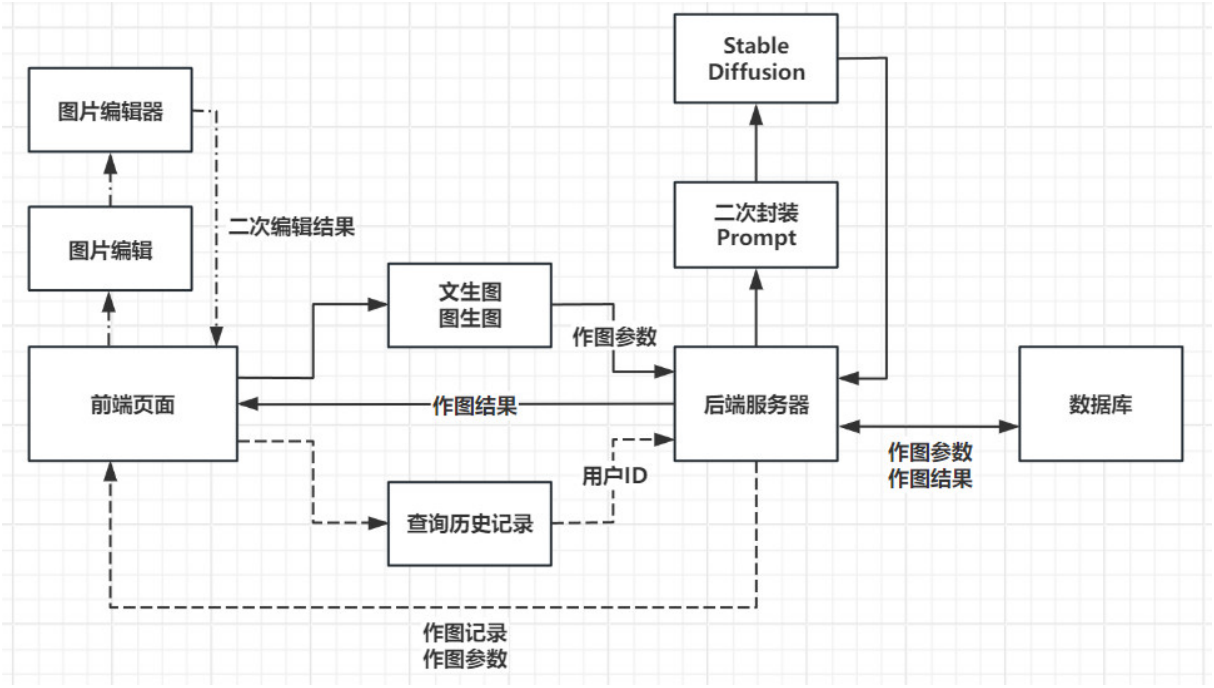

图 2 文生图流程图

因此，本项目使用前后端分离的模式进行开发，前端负责设计用户交互界面，获取到用户的操作信息；后端负责解析前端发送的请求，并串联 Stable Diffusion 模型和数据库，进行生图操作以及用户数据持久化操作。

三、 方法与实现

我在项目中负责后端服务器功能设计、数据库设计和 Stable Diffusion 模型调用的实现，以及前后端连接中的历史记录、图片二次编辑、文生图/图生图等功能的实现。

下面按照顺序讲解各功能具体实现方法：

3.1 Stable Diffusion 模型调用（Python+Torch）

本项目调用 Stable Diffusion 模型作图的方案是使用 AUTOMATIC1111 开发的 Stable Diffusion WebUI，该项目整合了 Stable Diffusion 模型的主要功能，并提供了一个简单的图形化界面和丰富的 API 模型以供其他开发者调用。

使用前需要将 CheckPoint 模型、LoRA 模型、VAE、Embeddding 文件放至指定目录，然后搭建好 Stable Diffusion 运行的必要环境，使用 Python 启动后，会自动在 7860 端口建立监听服务器。该服务器的特定路由收到携带作图参数的 POST 请求后，会自动解析请求内容，调用用户放至的模型作图，并将作图结果作为响应返回。

因此，后端服务器可以将封装好的参数以 POST 请求格式发送到 7860 端口，通过 WebUI 提供的 API 功能，实现对 Stable Diffusion 模型的调用。

3.2 数据库设计

本项目中需要持久化的信息有每位用户的作图记录，作图记录包括用户 ID、作图类型、作图时间、生成的图片和作图参数。

由于生成的图片和作图参数通过文件保存，而且总体信息量不多，因此我们采用的方案是不引入额外的数据库，直接使用用户 ID+作图类型+图片/参数作为多级目录的索引，作图时间+图片编号作为文件索引，映射到用户的每条记录。

以用户 lzm 的历史记录为例，database 目录下包含 lzm 目录，lzm 目录下包含 i2i 和 t2i 目录（对应图生图和文生图），然后 i2i/t2i 目录下是 args 和 images 目录，分别存放参数文件和图片文件，使用作图时间的图片编号进行文件命名，如图 3 所示：

图 3 用户 lzm 的历史记录

这种数据库设计方案，在保证历史记录唯一、检索便捷的前提下，实现了作图图片和作图参数的一一映射，同时还减少了引入数据库带来的性能和内存开销。

3.3 服务器功能设计（Python+Flask）

Flask 是一个轻量级的 Python Web 框架，可以轻松部署 Web 服务器，具有很强的灵活性和可扩展性。其丰富的插件和强大的社区支持可以很好地满足本项目需求。

因此，本项目的服务器使用 Flask 作为服务器框架、使用 Python 语言进行编写。

服务器的主要功能有：文生图/图生图、保存/查询/删除历史记录、请求图片/作图参数，下面对各功能具体实现进行说明。

3.3.1 文生图/图生图

文生图/图生图功能是本项目的核心功能，现将该功能的实现进行分步：① 接收前端请求、② 二次封装参数、③ 调用模型作图、④ 保存作图记录、⑤ 返回作图结果。下面是对各流程的具体实现：

1. 接收前端请求

服务器收到前端向/txt2img 或/img2img 接口发送的 Post 请求后，使用 get_json 方法解析其中的作图参数，其中主要的作图参数有：正向提示词、反向提示词、长、宽、图片数量、作图种子、采样方式、采样步数、提示词引导系数、高清采样方式与放大倍数等；若是图生图任务，则参数中会增加 initial_image 字段。

2. 二次封装参数

前端发送的作图参数中可能有缺失、错误等异常情况，服务器要进行判断和处理，对于缺失值要使用默认值进行补全；

然后服务器要根据用户的“风格”参数，调用相应的 LoRA 模型，并为 prompt 和 negative_prompt 拼接合适的模板，如图 4、图 5 所示：

```python
def parse_request(dict):
    general_prompt = style_select(dict.get('style'))
    general_negprompt = "(worst quality, low quality:2), monochrome, zombie,overexposure, watermark,text,bad anatomy,bad hand,

    negative_prompt = general_negprompt + dict.get('negative_prompt') if dict.get('negative_prompt') else general_negprompt
    prompt = general_prompt + dict.get('prompt') + (":1.3") if dict.get('prompt') else general_prompt
    width = dict.get('width') if dict.get('width') else 512
    height = dict.get('height') if dict.get('height') else 512
    seed = dict.get('seed') if dict.get('seed') else -1
    batch_size = dict.get('batch_size') if dict.get('batch_size') else 1
    steps = dict.get('steps') if dict.get('steps') else 25
    cfg_scale = dict.get('cfg_scale') if dict.get('cfg_scale') else 7
    hr_upscaler = dict.get('hr_upscaler') if dict.get('hr_upscaler') else "R-ESRGAN 4x+ Anime6B"
    sampler_name = dict.get('sampler_index') if dict.get('sampler_index') else "DPM++ 2M Karras"
```

图 4 异常值处理

```python
def style_select(style):
    if style == "none": # 不限风格
        return "(masterpiece, best quality:1.3),"
    elif style == "detailed": # 细丽风格
        return "(masterpiece, best quality, official art, beautiful and aesthet
    elif style == "water_color": # 水彩风格
        return "(masterpiece, best quality:1.3),(flat color:1.3),(colorful:1.3),
    elif style == "Hand_drawn": # 手绘风格
        return "(masterpiece, best quality:1.3), ToonShading, bold outlines,(ma
    elif style == "pixel": # 像素风格
        return "(masterpiece, best quality:1.3), <lora:pk_trainer768_V1:0.6>"
    elif style == "Retro_comic": # 复古漫画
        return "1980s anime, retro fashion, muted pastel colors, by Tsukasa Hoj
    elif style == "mechanical_girl": # 机械少女
        return "1mechanical girl,((ultra realistic details)), portrait, global
    elif style == "Film": # 电影胶片
        return "<lora:FilmVelvia3:0.6>, masterpiece, best quality, solo, minimal makeup, tender smile, dainty neckline, nostalgic
    elif style == "CyberPunk": # 赛博朋克
        return "(masterpiece, best quality:1.3),face to screen,extremely high detailed, intricate, 8k, HDR, wallpaper, cinematic
```

LoRA

图 5 风格选择

3. 调用模型作图

在步骤 2 处理完作图参数后，将作图参数用 Json 格式二次封装，以 POST 请求发送至 localhost:7860 的/sdapi/v1/txt2img 和/sdapi/v1/img2img 路径，Stable Diffusion WebUI 会监听请求，解析请求中的参数，并调用设置好的模型文件进行作图。

4. 保存作图记录

收到 WebUI 的响应后，服务器要解析出响应中的'images'字段，以 Base64 方式进行解码出图片，然后获取用户的 ID 和当前时间，并将用户的作图参数和结果图片以日期和编号命名（多图情况）。

最后以文件形式保存至对应目录（若是图生图，则将参考图片以 Base64 字符串格式写入参数文件），如图 6 所示：

```python
for i, image_data in enumerate(r['images']):
    # 将图片解码
    image_base64 = image_data.split(",", 1)[0]
    image = Image.open(io.BytesIO(base64.b64decode(image_base64)))

    image_path = os.path.join(folder_path,'t2i' ,'images', f'{current_time}-{i}.png')
    args_path = os.path.join(folder_path,'t2i' ,'args', f'{current_time}-{i}.json')
    # 保存图片到文件夹
    image.save(image_path)
    # 保存参数到文件夹
    with open(args_path, 'w') as file:
        file.write(json.dumps(dict))
```

解析响应

保存至对应目录

图 6 保存作图记录

5. 返回作图结果

最后，服务器要将生成的图片发送至前端进行展示，对于异常情况，服务器也要返回相应的 error_message 以供前端分析和处理。

3.3.2 查询历史记录

查询历史记录需要前端传递用户 ID，然后服务器查询数据库该用户目录中，t2i 和 i2i 目录下的所有参数和图片文件，并将文件前缀（作图时间+编号）发送至前端。

前端接收到响应后，在页面中渲染历史记录列表，用户点击某一个具体记录后，再另外发送请求获取作图参数和图片进行展示。

除此之外，该接口还承担用户注册的功能，若服务器查询不到用户目录，则说明用户是第一次登录，系统会为该 ID 的用户创建用户目录，以及 i2i、t2i、args、images 等子目录，该过程视为用户注册。

3.3.3 删除作图记录

删除历史记录需要前端将用户 id、作图类型、文件前缀以 Delete 请求发送至 /delete/<id>/<type>/<filename>，服务器会根据路径检索到对应的图片文件和参数文件，然后进行删除，删除成功后向前端响应成功提示。

若出现记录不存在的异常情况，则会将错误信息进行返回，以供前端分析和处理。

3.3.4 获取图片/作图参数

前端将用户 id、作图类型、文件前缀发送至路径/get_images/<id>/<type>/<filename> 和 /get_args/<id>/<type>/<filename>，服务器根据路径在数据库检索到对应的图片文件和参数文件后，将文件内容以 JSON 字符串的格式响应到前端。

其中，图片文件转为 Base64 编码的字符串，并将宽和高信息一并返回至前端；而参数文件在存储时以及是 JSON 字符串格式，因此只需读出内容直接返回即可。

若出现记录不存在的异常情况，则会将错误信息进行返回，以供前端分析和处理。

3.3.5 接口文档和风格说明文档

除了编写后端代码，我还使用 Apifox 等工具为服务器的所有接口编写了 API 接口说明文档供前端使用，还利用 Apifox 的测试功能编写测试用例，根据用例测试结果进行代码代码重构和漏洞修复。（见附件—接口文档）

另外，我还制作了风格说明文档，并使用 GhostMix 模型为每种作图风格生成了样例图片在前端进行展示，用户选择作图风格时可以根据样例图片进行参考，提高体验。（见附件—风格说明文档）

3.4 前后端连接（VUE+Node.js）

除了后端的服务器、SD 模型调用、数据库设计以外，我还负责完成前后端连接中的图像二次编辑、查看历史记录和参数导入、文生图/图生图功能的实现。

3.4.1 图像二次编辑

美图是 PC 端与移动端上强大的图像编辑工具，其功能包括裁剪、旋转、调整亮度、对比度、AI 优化等。美图开放平台提供了免费的 SDK 以及开发文档以供开发人员使用，通过调用 SDK，网站可以在前端直接调用美图的编辑器，使用美图的编辑功能。

因此本项目通过继承美图 SDK 的方式，在前端实现二次编辑功能，极大减少开发工作量。

具体实现方法为，在前端导入美图 SDK 库，初始化美图编辑器。当用户点击“二次编辑”按键后，调用 openImageEditor 函数将图片 URL 传入美图编辑器，即可使用美图的图片编辑功能。

实现效果如图 7 所示，将生成的图片传入美图编辑器进行编辑操作：

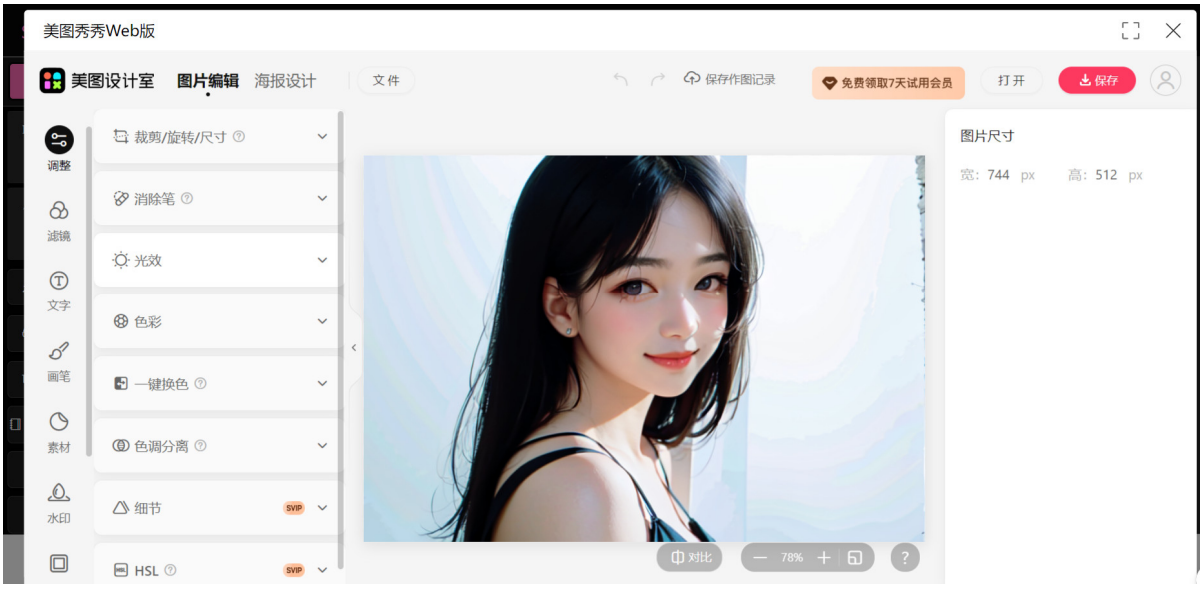

图 7 图片二次编辑

3.4.2 历史记录和参数导入

用户点击“历史记录”后，前端会将 id 发送至服务器历史记录 API，请求文生图和图生图记录。

由于服务器的响应数据是作图类型、作图时间和图片编号，因此前端需要对响应数据进行解析和处理，将每一条历史记录解析为 年月-日-记录 的格式，按多级目录的方式进行展示，并按作图时间由新到旧进行排序。

用户选择某一条记录后，前端由会向后端服务器请求该记录对应的图片和作图参数进行展示，效果如图 8 所示：

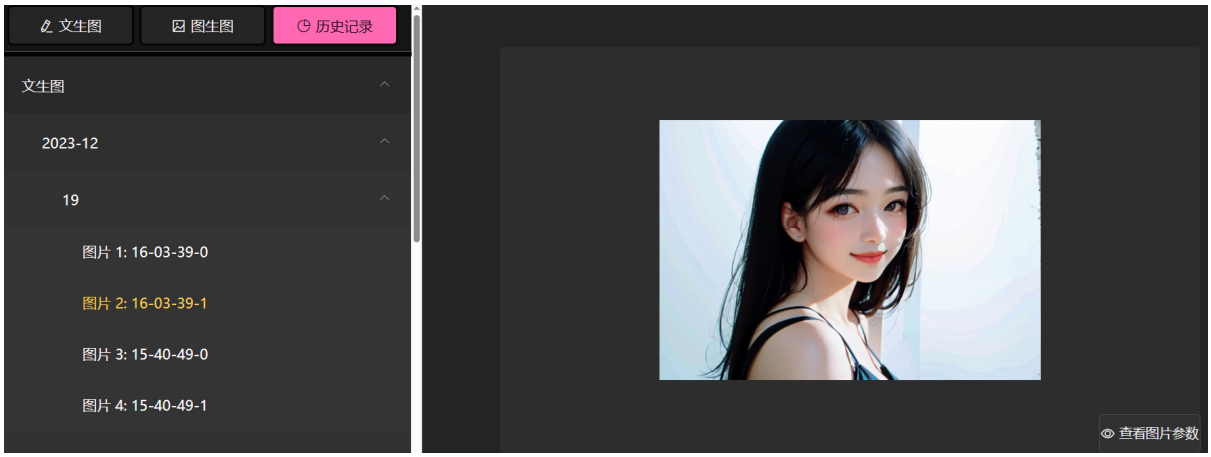

图 8 查询历史记录

若用户点击右下角“查看图片参数”，则会根据参数文件渲染出作图参数框。

用户可以点击一键复制，将参数导入并跳转到作图界面。（图生图适用，会将参考图片进行导入）

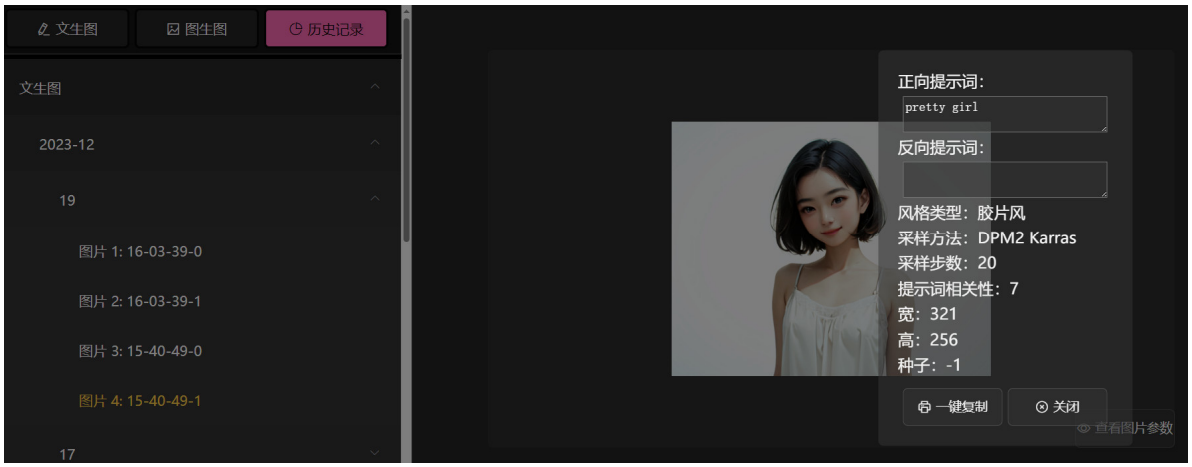

图 9 查询作图参数

3.4.3 文生图/图生图

该功能的逻辑简单，用户点击“生成图像”按键后，将用户输入的作图参数打包为 JSON 格式，使用 Ajax 以 POST 请求发送至后端 txt2img 或 img2img 接口，然后解析后端响应，按照响应中的宽和高参数，将图片 Base64 字符串进行解码，并渲染展示在前端。

如果批量生成多张图，则使用数组进行循环渲染，使得所有图像均可显示。

与“文生图”相比，“图生图”增加了 init_image 字段，用户上传图片后，前端需要将参考图转为 Base64 字符串，存入该 payload 的 init_image 字段中进行打包发送。

以“图生图”功能为例，最终效果图如图 10 所示：

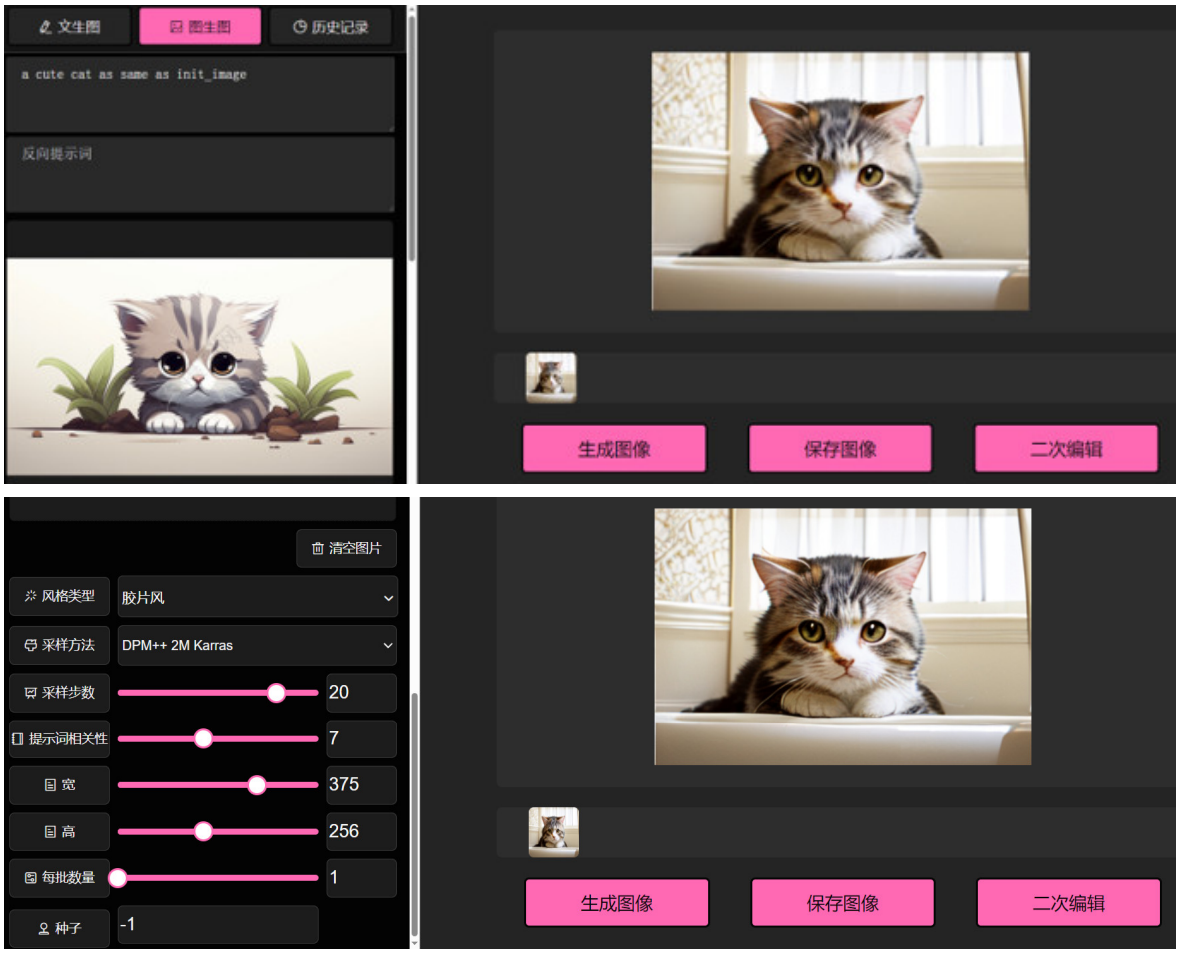

图 10“图生图”功能示例

至此，我在项目中的承担的任务以及使用的实现方法已经介绍完毕。本项目使用前后端分离的模式进行开发，我主要负责后端服务器功能设计、数据库设计、Stable Diffusion 模型调用的任务，以及前后端连接实现中的历史记录、图片二次编辑、文生图/图生图等功能的连接和实现。

在后端设计中，我借助基于 Stable Diffusion 技术的开源 WebUI 项目，简化了 Stable Diffusion 的调用过程；

然后我采样多级目录索引的方式设计了适用于本项目的文件数据库，将用户的历史请求数据持久化，避免引入额外的数据库，降低系统开销，还提高了检索和存储效率；

最后我使用 Python 作为编程语言，使用 Flask 作为框架搭建了一个轻量级服务器，该服务器能完成文生图/图生图、保存/查询/删除历史记录、请求图片/作图参数等操作，很好地连接了前端、Stable Diffusion 模型、数据库，能够快速处理用户的各类请求，对于异常情况也做出相应处理。

此外，我还为服务器接口编写了详细的接口文档和测试用例，为整个项目提供了清晰的指导和参考，提高团队协作效率和项目可维护性。另外，我还编写了风格说明文档以及样例图片，为用户作图风格选择提供更良好的使用体验。

在前后端连接实现中，我将前端的网页框架中的静态数据部分替换为与后端服务器交互获取的动态数据，然后我将文生图/图生图、查询历史记录、参数导入等功能的参数传递、接口调用、结果响应结果解析和渲染、页面跳转等功能逻辑进行实现。

另外，我还通过在前端集成了美图 SDK 的方式实现图像二次编辑功能，编写了美图编辑器的调用逻辑，使得用户可以对生成的图像进行各种编辑，增加了用户对图像的可操作性，还为用户提供了更多个性化的选择，满足了不同用户对图像风格的独特需求。

最后，我使用隧道内网穿透技术，将网站映射到公网，使得其他用户可以访问和使用。

四、 总结与展望

4.1 课题总结

本次课程我选择了“基于 Stable Diffusion 的文生图/图生图系统”作为课题，通过整合前后端与 Stable Diffusion 深度学习模型，搭建了一个界面简洁、功能完善、特色鲜明、用户友好的文生图/图生图 Web 系统。我们设计的系统具有以下特性和亮点：

- 采用 GhostMix 模型作为基础模型，模型本身能够适应多种多样的作图风格需求，在保持作图质量的同时提高作图效率，实现高质量与时长的平衡；
- 利用 LoRA 技术对 Stable Diffusion 模型进行轻量化微调，提高模型对不同作图风格的分化能力，使用户能够更灵活地选择和创造图像；
- 引入提示词内嵌，用户只需输入少量关键词即可获得高质量 Prompt，从而生成高质量图像，降低了用户使用门槛；
- 提供二次编辑功能作为文生图/图生图功能的衔接，为用户在生成图像后进行进一步的个性化创作提供了便利；
- 对文生图和图生图历史记录进行存储，用户可以随时查看并对作图参数进行一键导入。

通过调研后进行功能设计，我们的系统弥补了其他竞品的缺陷，为用户带来更优的体验，出色地完成了课题要求。

4.2 技术难题与解决方案

在本项目中，我们面临了以下技术挑战：

- 缺乏深度学习领域的相关知识，对环境配置和模型调用不了解；
- 开发方案种类繁多，具体开发方法和开发框架难以选择；
- 缺乏实际开发经验，对团队协作、项目整合方法不熟悉；

基于以上问题，我们采取了以下解决方案

- 阅读相关文献资料，了解 Stable Diffusion 的技术原理，参考网络教程搭建深度学习环境，学习 Pytorch、Numpy 等库的用法，最终成功在本地部署并运行 Stable Diffusion 项目。针对 Stable Diffusion 技术，我还参考了其他开源方案，最终借助 WebUI 项目完成了 Stable Diffusion 的部署与服务器调用，降低了调用过程的复杂性。
- 通过技术调研，比较了不同的开发框架和方法，最终出于项目背景、团队成员的编程语言掌握程度、轻量化等因素的考量，选择 Python+Flask 框架作为后端服务器技术方案，选择 Vue+Node.js 作为前端界面技术方案，为项目的顺利开发奠定基础。
- 通过团队沟通，决定好开发模式（前后端分离）和任务分工；通过提前调研，列出了项目要完成的目标，并对开发进度进行规划。另外，我们还约定了开发规范，通过编写文档使得团队协作更加流畅，提高了项目的可维护性。

4.3 个人心得体会

本次项目分为五个阶段进行，分别是系统开发环境搭建、系统研究分析与设计、应用分析与开发、服务设计与实现、综合组织与部署。从系统设计到实现部署层层递进，最终完成项目开发的全流程。

在这个过程中，我首先收获的是技术上的提升，经过项目全流程的开发，我掌握了深度学习模型的部署和开发方法、Flask 框架的使用方法、数据库设计的技巧以及团队协作工具的使用。这些技能的掌握，为我未来的项目和职业发展奠定了坚实的基础。

另外，我深刻体会到团队协作对于项目成功的重要性。通过有效的沟通，我们在项目初期就规划好项目需求、工作目标和任务分工，并规划了合理的开发进度，充分发挥出团队两人的优势。通过编写文档和遵循开发规范，我们的团队协作过程非常顺畅。由于是第一次参与多人项目协作，这种协作的经验让我受益匪浅。

除了技术上的提升，我的另一大收获是开发过程中思维方式的转变。开发一个系统，不仅要从技术的视角考虑，更要从用户的视角考虑。比如本项目中的 Prompt 工程和 LoRA 模型的调用，从技术角度看可行，但是对于普通用户存在门槛。我们可以封装 Propmt 工程和 LoRA 模型调用的过程，帮助用户降低门槛，提高用户体验。从技术专业角度转向用户视角在开发中是非常重要的，要求我们不仅要关注系统的技术实现，还要关心用户在使用过程中的感受和需求。通过将技术复杂性从用户层面隐藏，使用户能够更轻松地使用系统，是提高产品吸引力和可用性的关键之一。

总的来说，通过这个项目，我不仅在技术上取得了进步，更重要的是培养了团队协作意识和以用户为中心的设计思维，这次项目开发经历是我项目开发道路的起点，也将作为我职业生涯发展道路的宝贵财富。

4.4 课题发展的建议

- 可以让不同课题内容的队伍互相体验课题成果，收集真实的用户反馈进行迭代开发。
- 可以讲解一些优秀案例（技术方案、团队协作方案）以供参考。