Vue 前端代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <link rel="icon" href="/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="vue-element-admin 的 vue3 版本" />
    <meta name="keywords" content="vue-element-admin,vue3-element-admin" />
    <title>图像去雾系统</title>
</head>
<body>
<div id="app" class="app"></div>
<script type="module" src="/src/main.ts"></script>
</body>
</html>

import { createApp } from "vue";
import App from "./App.vue";
import router from "@/router";
import { setupStore } from "@/store";
import { setupDirective } from "@/directive";
import "@/permission";
// 本地 SVG 图标
import "virtual:svg-icons-register";
// 国际化
import i18n from "@/lang/index";
// 样式
import "element-plus/theme-chalk/dark/css-vars.css";
import "@/styles/index.scss";
import "uno.css";
const app = createApp(App);
// 全局注册 自定义指令(directive)
setupDirective(app);
// 全局注册 状态管理(store)
setupStore(app);
app.use(router).use(i18n).mount("#app");
import { DehazeIndex, ImageInfo, ModelInfo } from "@/api/dehaze/types";
import requestPy from "@/utils/request-py";
import { AxiosProgressEvent, AxiosPromise } from "axios";
export function getModelApi(): AxiosPromise<ModelInfo[]> {
    return requestPy({
        url: "/model/",
        method: "get",
    });
}
export function uploadImageApi(file: File): AxiosPromise<ImageInfo> {
    return requestPy({
        url: "/upload/",
        method: "post",
        data: file,
        headers: {
            "Content-Type": "Image/png",
```

```
        },
    });
}
export function downloadApi(image_name: string): AxiosPromise<File> {
    return requestPy({
        url: `/download/${image_name}/`,
        method: "get",
    });
}
export function dehazeApi(
    haze_image: string,
    model_name: string
): AxiosPromise<ImageInfo> {
    return requestPy({
        url: "/dehazeImage/",
        method: "post",
        data: { haze_image, model_name },
    });
}
export function calculateIndexApi(
    haze_image: string,
    clear_image: string,
    onUpload: ((progressEvent: AxiosProgressEvent) => void) | undefined
): AxiosPromise<DehazeIndex> {
    return requestPy({
        url: "/calculateIndex/",
        method: "post",
        data: { haze_image, clear_image },
        onUploadProgress: onUpload,
    });
}
export interface ModelInfo {
    value: string;
    label: string;
    children?: ModelInfo;
}
export interface DehazeIndex {
    psnr: string;
    ssim: string;
}
export interface ImageInfo {
    image_name: string;
}
// 系统设置
interface DefaultSettings {
    title: string;
    showSettings: boolean;
    tagsView: boolean;
    fixedHeader: boolean;
    sidebarLogo: boolean;
    layout: string;
    theme: string;
    size: string;
    language: string;
}
```

```
const defaultSettings: DefaultSettings = {
    title: "图像去雾系统",
    showSettings: true,
    tagsView: false,
    fixedHeader: false,
    sidebarLogo: true,
    layout: "left",
    theme: "light",
    size: "default", // default |large |small
    language: "zh-cn", // zh-cn| en
};
export default {
    // 路由国际化
    route: {
        dashboard: "首页",
        document: "项目文档",
    },
    // 登录页面国际化
    login: {
        title: "图像去雾系统",
        username: "用户名",
        password: "密码",
        login: "登 录",
        verifyCode: "验证码",
    },
    // 导航栏国际化
    navbar: {
        dashboard: "首页",
        logout: "注销",
    },
};
// 创建 axios 实例
import axios, { AxiosResponse } from "axios";

const service = axios.create({
    baseURL: import.meta.env.VITE_APP_PYTHON_API,
    timeout: 50000,
});
service.interceptors.response.use(
    (response: AxiosResponse) => {
        const { code, msg } = response.data;
        if (code === "00000") {
            return response.data;
        }
        if (response.data instanceof ArrayBuffer) {
            return response;
        }
        ElMessage.error(msg || response.data);
        return Promise.reject(new Error(msg || "Error"));
    },
    (error: any) => {
        if (error.response.data) {
            const { msg } = error.response.data;
```

```
            ElMessage.error(msg || "系统出错");
        }
        return Promise.reject(error.message);
    }
);
export const imageBaseURL: string = import.meta.env.VITE_APP_IMG_URL;
// 导出 axios 实例
export default service;
export default defaultSettings;
import vue from "@vitejs/plugin-vue";
import { UserConfig, ConfigEnv, loadEnv, defineConfig } from "vite";
import AutoImport from "unplugin-auto-import/vite";
import Components from "unplugin-vue-components/vite";
import { ElementPlusResolver } from "unplugin-vue-components/resolvers";
import Icons from "unplugin-icons/vite";
import IconsResolver from "unplugin-icons/resolver";
import { createSvgIconsPlugin } from "vite-plugin-svg-icons";
import { viteMockServe } from "vite-plugin-mock";
import visualizer from "rollup-plugin-visualizer";
import UnoCSS from "unocss/vite";
import path from "path";
import viteCompression from "vite-plugin-compression";
const pathSrc = path.resolve(__dirname, "src");
export default defineConfig(({ mode }: ConfigEnv): UserConfig => {
    const env = loadEnv(mode, process.cwd());
    return {
        resolve: {
            alias: {
                "@": pathSrc,
            },
        },
        css: {
            // CSS 预处理器
            preprocessorOptions: {
                //define global scss variable
                scss: {
                    javascriptEnabled: true,
                    additionalData: `
            @use "@/styles/variables.scss" as *;
          `,
                },
            },
        },
        server: {
            host: "0.0.0.0",
            port: Number(env.VITE_APP_PORT),
            open: true, // 运行是否自动打开浏览器
            proxy: {
                // 反向代理解决跨域
                [env.VITE_APP_BASE_API]: {
                    target: env.VITE_APP_TARGET_URL,
                    changeOrigin: true,
                    rewrite: (path) =>
                        path.replace(
```

```
                        new RegExp("^" + env.VITE_APP_BASE_API),
                        env.VITE_APP_TARGET_BASE_API
                    ), // 替换 /dev-api 为 target 接口地址
                },
                [env.VITE_APP_PYTHON_API]: {
                    target: env.VITE_APP_PYTHON_URL,
                    changeOrigin: true,
                    rewrite: (path) =>
                        path.replace(
                            new RegExp("^" + env.VITE_APP_PYTHON_API),
                            env.VITE_APP_TARGET_BASE_API
                        ),
                },
            },
        },
        plugins: [
            vue(),
            UnoCSS({}),
            AutoImport({
                // 自动导入 Vue 相关函数，如：ref, reactive, toRef 等
                imports: ["vue", "@vueuse/core"],
                eslintrc: {
                    enabled: false,
                    filepath: "./.eslintrc-auto-import.json",
                    globalsPropValue: true,
                },
                resolvers: [
                    // 自动导入 Element Plus 相关函数，如：ElMessage, ElMessageB
ox... (带样式)
                    ElementPlusResolver(),
                    IconsResolver({}),
                ],
                vueTemplate: true,
                // 配置文件生成位置(false:关闭自动生成)
                // dts: false,
                dts: "src/types/auto-imports.d.ts",
            }),
            Components({
                resolvers: [
                    // 自动导入 Element Plus 组件
                    ElementPlusResolver(),
                    // 自动导入图标组件
                    IconsResolver({
                        // @iconify-json/ep 是 Element Plus 的图标库
                        enabledCollections: ["ep"],
                    }),
                ],
                // 指定自定义组件位置(默认:src/components)
                dirs: ["src/**/components"],
                // 配置文件位置(false:关闭自动生成)
                // dts: false,
                dts: "src/types/components.d.ts",
            }),
            Icons({
```

```
            // 自动安装图标库
            autoInstall: true,
        }),
        createSvgIconsPlugin({
            // 指定需要缓存的图标文件夹
            iconDirs: [path.resolve(pathSrc, "assets/icons")],
            // 指定 symbolId 格式
            symbolId: "icon-[dir]-[name]",
        }),
        // 代码压缩
        viteCompression({
            verbose: true, // 默认即可
            disable: true, // 是否禁用压缩，默认禁用，true 为禁用,false 为开
启，打开压缩需配置 nginx 支持
            deleteOriginFile: true, // 删除源文件
            threshold: 10240, // 压缩前最小文件大小
            algorithm: "gzip", // 压缩算法
            ext: ".gz", // 文件类型
        }),
        viteMockServe({
            ignore: /^_/,
            mockPath: "mock",
            enable: mode === "development",
            // https://github.com/anncwb/vite-plugin-mock/issues/9
        }),
        visualizer({
            filename: "./stats.html",
            open: false,
            gzipSize: true,
            brotliSize: true,
        }),
    ],
    // 预加载项目必需的组件
    optimizeDeps: {
        include: [
            "vue",
            "vue-router",
            "pinia",
            "axios",
            "element-plus/es/components/form/style/css",
            "element-plus/es/components/form-item/style/css",
            "element-plus/es/components/button/style/css",
            "element-plus/es/components/input/style/css",
            "element-plus/es/components/input-number/style/css",
            "element-plus/es/components/switch/style/css",
            "element-plus/es/components/upload/style/css",
            "element-plus/es/components/menu/style/css",
            "element-plus/es/components/col/style/css",
            "element-plus/es/components/icon/style/css",
            "element-plus/es/components/row/style/css",
            "element-plus/es/components/tag/style/css",
            "element-plus/es/components/dialog/style/css",
            "element-plus/es/components/loading/style/css",
            "element-plus/es/components/radio/style/css",
```

```
                "element-plus/es/components/radio-group/style/css",
                "element-plus/es/components/popover/style/css",
                "element-plus/es/components/scrollbar/style/css",
                "element-plus/es/components/tooltip/style/css",
                "element-plus/es/components/dropdown/style/css",
                "element-plus/es/components/dropdown-menu/style/css",
                "element-plus/es/components/dropdown-item/style/css",
                "element-plus/es/components/sub-menu/style/css",
                "element-plus/es/components/menu-item/style/css",
                "element-plus/es/components/divider/style/css",
                "element-plus/es/components/card/style/css",
                "element-plus/es/components/link/style/css",
                "element-plus/es/components/breadcrumb/style/css",
                "element-plus/es/components/breadcrumb-item/style/css",
                "element-plus/es/components/table/style/css",
                "element-plus/es/components/tree-select/style/css",
                "element-plus/es/components/table-column/style/css",
                "element-plus/es/components/select/style/css",
                "element-plus/es/components/option/style/css",
                "element-plus/es/components/pagination/style/css",
                "element-plus/es/components/tree/style/css",
                "element-plus/es/components/alert/style/css",
                "element-plus/es/components/radio-button/style/css",
                "element-plus/es/components/checkbox-group/style/css",
                "element-plus/es/components/checkbox/style/css",
                "element-plus/es/components/tabs/style/css",
                "element-plus/es/components/tab-pane/style/css",
                "element-plus/es/components/rate/style/css",
                "element-plus/es/components/date-picker/style/css",
                "element-plus/es/components/notification/style/css",
                "@vueuse/core",
                "sortablejs",
                "path-to-regexp",
                "echarts",
                "@wangeditor/editor",
                "@wangeditor/editor-for-vue",
                "vue-i18n",
                "codemirror",
            ],
        },
    };
});
```

```
<script setup lang="ts">
  import { uploadImageApi } from "@/api/dehaze";
  import { imageBaseURL } from "@/utils/request-py";
  import { UploadRawFile, UploadRequestOptions } from "element-plus";
  const props = defineProps({
    modelValue: {
      type: String,
      default: "",
    },
    title: {
      type: String,
      default: "点击上传文件",
```

```
    },
  });
  const emit = defineEmits(["update:modelValue", "getImageInfo"]);
  const imageName = useVModel(props, "modelValue", emit);
  const imgUrl = computed(() => {
    return imageName.value ? imageBaseURL + "/" + imageName.value + "/" : "";
  });
  const isLoading = ref(false);
  async function uploadFile(options: UploadRequestOptions): Promise<any> {
    isLoading.value = true;
    try {
      const { data } = await uploadImageApi(options.file);
      imageName.value = data.image_name;
    } finally {
      isLoading.value = false;
    }
  }
  async function handleBeforeUpload(file: UploadRawFile) {
    if (file.size > 20 * 1024 * 1024) {
      ElMessage.warning("上传图片不能大于 20M");
      return false;
    }
    const image = new Image();
    image.src = URL.createObjectURL(file);
    await image.decode();
    // 向父组件传递用户所上传的图片的分辨率
    emit("getImageInfo", image.naturalWidth, image.naturalHeight);
    return true;
  }
</script>
<template>
  <el-upload
      v-model="imgUrl"
      class="single-uploader"
      :show-file-list="false"
      list-type="picture"
      :before-upload="handleBeforeUpload"
      :http-request="uploadFile"
  >
    <img v-if="imgUrl" :src="imgUrl" class="single" alt="" />
    <div v-else-if="isLoading" class="single-uploader-placeholder">
      <span>上传中...</span>
    </div>
    <div v-else class="single-uploader-placeholder">
      <el-icon class="single-uploader-icon"><i-ep-plus /></el-icon>
      <span style="margin-top: 5px">{{ title }}</span>
    </div>
  </el-upload>
</template>
<style scoped>
  .single-uploader .single {
    display: block;
    width: 35vmax;
    height: 35vmax;
  }
```

```
</style>
<style>
  .single-uploader .el-upload {
    position: relative;
    overflow: hidden;
    cursor: pointer;
    border: 1px dashed var(--el-border-color);
    border-radius: 6px;
    transition: var(--el-transition-duration-fast);
  }
  .single-uploader .el-upload:hover {
    border-color: var(--el-color-primary);
  }
  .single-uploader-placeholder {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    width: 35vmax;
    height: 35vmax;
  }
  .single-uploader-icon {
    font-size: 28px;
    color: #8c939d;
    text-align: center;
  }
</style>
<script setup lang="ts">
import { useUserStore } from "@/store/modules/user";
import { useTransition, TransitionPresets } from "@vueuse/core";
defineOptions({
  // eslint-disable-next-line
  name: "Dashboard",
  inheritAttrs: false,
});
const userStore = useUserStore();
const date: Date = new Date();
const greetings = computed(() => {
  const hours = date.getHours();
  if (hours >= 6 && hours < 8) {
    return "晨起披衣出草堂，轩窗已自喜微凉 ！";
  } else if (hours >= 8 && hours < 12) {
    return "上午好 ！";
  } else if (hours >= 12 && hours < 18) {
    return "下午好🍎！";
  } else if (hours >= 18 && hours < 24) {
    return "晚上好 ！";
  } else if (hours >= 0 && hours < 6) {
    return "偷偷向银河要了一把碎星，只等你闭上眼睛撒入你的梦中，晚安 ！";
  }
});
const duration = 5000;
// 收入金额
const amount = ref(0);
```

```
const amountOutput = useTransition(amount, {
  duration: duration,
  transition: TransitionPresets.easeOutExpo,
});
amount.value = 150;
// 访问数
const visitCount = ref(0);
const visitCountOutput = useTransition(visitCount, {
  duration: duration,
  transition: TransitionPresets.easeOutExpo,
});
visitCount.value = 2180;
//消息数
const messageCount = ref(0);
const messageCountOutput = useTransition(messageCount, {
  duration: duration,
  transition: TransitionPresets.easeOutExpo,
});
messageCount.value = 15;
// 订单数
const orderCount = ref(0);
const orderCountOutput = useTransition(orderCount, {
  duration: duration,
  transition: TransitionPresets.easeOutExpo,
});
orderCount.value = 154;
</script>
<template>
  <div class="dashboard-container">
    <!-- 用户信息 -->
    <el-row class="mb-8">
      <el-card class="w-full">
        <div class="flex justify-between flex-wrap">
          <div class="flex items-center">
            <img
              class="user-avatar"
              :src="userStore.avatar + '?imageView2/1/w/80/h/80'"
            />
            <span class="ml-[10px] text-[16px]">
              {{ userStore.nickname }}
            </span>
          </div>
          <div class="leading-[40px]">
            {{ greetings }}
          </div>
          <div class="space-x-2 flex items-center justify-end">
            <el-link target="_blank" type="danger" href="http://10.16.90.26/"
              > 土味锌的阅读笔记</el-link
            >
            <el-divider direction="vertical" />
            <el-link
              target="_blank"
              type="success"
              href="https://gitee.com/earthy-zinc"
```

```
              >Gitee</el-link
            >
            <el-divider direction="vertical" />
            <el-link
              target="_blank"
              type="primary"
              href="https://github.com/earthy-zinc"
              >GitHub
            </el-link>
          </div>
        </div>
      </el-card>
    </el-row>
    <!-- 数据卡片 -->
    <el-row :gutter="40" class="mb-4">
      <el-col :xs="24" :sm="12" :lg="6" class="mb-4">
        <div class="data-box">
          <div
            class="text-[#40c9c6] hover:!text-white hover:bg-[#40c9c6] p-3 rou
nded"
          >
            <svg-icon icon-class="uv" size="3em" />
          </div>
          <div class="flex flex-col space-y-3">
            <div class="text-[var(--el-text-color-secondary)]">访问数</div>
            <div class="text-lg text-right">
              {{ Math.round(visitCountOutput) }}
            </div>
          </div>
        </div>
      </el-col>
      <!--消息数-->
      <el-col :xs="24" :sm="12" :lg="6" class="mb-4">
        <div class="data-box">
          <div
            class="text-[#36a3f7] hover:!text-white hover:bg-[#36a3f7] p-3 rou
nded"
          >
            <svg-icon icon-class="message" size="3em" />
          </div>
          <div class="flex flex-col space-y-3">
            <div class="text-[var(--el-text-color-secondary)]">模型数</div>
            <div class="text-lg text-right">
              {{ Math.round(messageCountOutput) }}
            </div>
          </div>
        </div>
      </el-col>
      <el-col :xs="24" :sm="12" :lg="6" class="mb-4">
        <div class="data-box">
          <div
            class="text-[#f4516c] hover:!text-white hover:bg-[#f4516c] p-3 rou
nded"
          >
```

```
            <svg-icon icon-class="money" size="3em" />
          </div>
          <div class="flex flex-col space-y-3">
            <div class="text-[var(--el-text-color-secondary)]">
              累计去雾数量
            </div>
            <div class="text-lg text-right">
              {{ Math.round(amountOutput) }}
            </div>
          </div>
        </div>
      </el-col>
      <el-col :xs="24" :sm="12" :lg="6" class="mb-2">
        <div class="data-box">
          <div
            class="text-[#34bfa3] hover:!text-white hover:bg-[#34bfa3] p-3 rou
nded"
          >
            <svg-icon icon-class="shopping" size="3em" />
          </div>
          <div class="flex flex-col space-y-3">
            <div class="text-[var(--el-text-color-secondary)]">
              累计评估数量
            </div>
            <div class="text-lg text-right">
              {{ Math.round(orderCountOutput) }}
            </div>
          </div>
        </div>
      </el-col>
    </el-row>
    <!-- Echarts 图表 -->
    <el-row :gutter="40">
      <el-col :sm="24" :lg="8" class="mb-4">
        <BarChart
          id="barChart"
          height="400px"
          width="100%"
          class="bg-[var(--el-bg-color-overlay)]"
        />
      </el-col>
      <el-col :xs="24" :sm="12" :lg="8" class="mb-4">
        <PieChart
          id="pieChart"
          height="400px"
          width="100%"
          class="bg-[var(--el-bg-color-overlay)]"
        />
      </el-col>
      <el-col :xs="24" :sm="12" :lg="8" class="mb-4">
        <RadarChart
          id="radarChart"
          height="400px"
          width="100%"
          class="bg-[var(--el-bg-color-overlay)]"
```

```
        />
      </el-col>
    </el-row>
  </div>
</template>
<style lang="scss" scoped>
.dashboard-container {
  position: relative;
  padding: 24px;
  .user-avatar {
    width: 40px;
    height: 40px;
    border-radius: 50%;
  }
  .data-box {
    display: flex;
    justify-content: space-between;
    padding: 20px;
    font-weight: bold;
    color: var(--el-text-color-regular);
    background: var(--el-bg-color-overlay);
    border-color: var(--el-border-color);
    box-shadow: var(--el-box-shadow-dark);
  }
  .svg-icon {
    fill: currentcolor !important;
  }
}
</style>
<!--  线 + 柱混合图 -->
<template>
  <el-card>
    <template #header>
      <div class="title">
        去雾效果柱状图
        <el-tooltip effect="dark" content="点击试试下载" placement="bottom">
          <i-ep-download class="download" @click="downloadEchart" />
        </el-tooltip>
      </div>
    </template>
    <div :id="id" :class="className" :style="{ height, width }"></div>
  </el-card>
</template>
<script setup lang="ts">
  import * as echarts from "echarts";
  const props = defineProps({
    id: {
      type: String,
      default: "barChart",
    },
    className: {
      type: String,
      default: "",
    },
    width: {
```

```
    type: String,
    default: "200px",
    required: true,
  },
  height: {
    type: String,
    default: "200px",
    required: true,
  },
});
const options = {
  grid: {
    left: "2%",
    right: "2%",
    bottom: "10%",
    containLabel: true,
  },
  tooltip: {
    trigger: "axis",
    axisPointer: {
      type: "cross",
      crossStyle: {
        color: "#999",
      },
    },
  },
  legend: {
    x: "center",
    y: "bottom",
    data: ["有雾图像", "无雾图像", "PSNR", "SSIM"],
    textStyle: {
      color: "#999",
    },
  },
  xAxis: [
    {
      type: "category",
      data: [
        "C2PNet",
        "DehazeFormer",
        "MB-TaylorFormer",
        "MixDehazeNet",
        "RIDCP",
      ],
      axisPointer: {
        type: "shadow",
      },
    },
  ],
  yAxis: [
    {
      type: "value",
      min: 0,
      max: 3000,
      interval: 500,
```

```
      axisLabel: {
        formatter: "{value} ",
      },
    },
    {
      type: "value",
      min: 0,
      max: 50,
      interval: 10,
      axisLabel: {
        formatter: "{value}%",
      },
    },
  ],
  series: [
    {
      name: "有雾图像",
      type: "bar",
      data: [1200, 500, 2500, 1800, 800],
      barWidth: 20,
      itemStyle: {
        color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
          { offset: 0, color: "#83bff6" },
          { offset: 0.5, color: "#188df0" },
          { offset: 1, color: "#188df0" },
        ]),
      },
    },
    {
      name: "无雾图像",
      type: "bar",
      data: [3000, 1000, 2400, 1600, 800],
      barWidth: 20,
      itemStyle: {
        color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
          { offset: 0, color: "#25d73c" },
          { offset: 0.5, color: "#1bc23d" },
          { offset: 1, color: "#179e61" },
        ]),
      },
    },
    {
      name: "PSNR",
      type: "line",
      yAxisIndex: 1,
      data: [25, 30, 26, 38, 41],
      itemStyle: {
        color: "#67C23A",
      },
    },
    {
      name: "SSIM",
      type: "line",
      yAxisIndex: 1,
      data: [14, 25, 30, 35, 40],
```

```
        itemStyle: {
          color: "#409EFF",
        },
      },
    ],
  };
  const chart = ref<any>("");
  onMounted(() => {
    // 图表初始化
    chart.value = markRaw(
        echarts.init(document.getElementById(props.id) as HTMLDivElement)
    );
    chart.value.setOption(options);
    // 大小自适应
    window.addEventListener("resize", () => {
      chart.value.resize();
    });
  });
  const downloadEchart = () => {
    // 获取画布图表地址信息
    const img = new Image();
    img.src = chart.value.getDataURL({
      type: "png",
      pixelRatio: 1,
      backgroundColor: "#fff",
    });
    // 当图片加载完成后，生成 URL 并下载
    img.onload = () => {
      const canvas = document.createElement("canvas");
      canvas.width = img.width;
      canvas.height = img.height;
      const ctx = canvas.getContext("2d");
      if (ctx) {
        ctx.drawImage(img, 0, 0, img.width, img.height);
        const link = document.createElement("a");
        link.download = `去雾效果图.png`;
        link.href = canvas.toDataURL("image/png", 0.9);
        document.body.appendChild(link);
        link.click();
        link.remove();
      }
    };
  };
</script>
<style lang="scss" scoped>
  .title {
    display: flex;
    justify-content: space-between;
    .download {
      cursor: pointer;
      &:hover {
        color: #409eff;
      }
    }
```

```
  }
</style>
<!-- 漏斗图 -->
<template>
  <div :id="id" :class="className" :style="{ height, width }"></div>
</template>
<script setup lang="ts">
  import * as echarts from "echarts";
  const props = defineProps({
    id: {
      type: String,
      default: "funnelChart",
    },
    className: {
      type: String,
      default: "",
    },
    width: {
      type: String,
      default: "200px",
      required: true,
    },
    height: {
      type: String,
      default: "200px",
      required: true,
    },
  });
  const options = {
    title: {
      show: true,
      text: "订单线索转化漏斗图",
      x: "center",
      padding: 15,
      textStyle: {
        fontSize: 18,
        fontStyle: "normal",
        fontWeight: "bold",
        color: "#337ecc",
      },
    },
    grid: {
      left: "2%",
      right: "2%",
      bottom: "10%",
      containLabel: true,
    },
    legend: {
      x: "center",
      y: "bottom",
      data: ["Show", "Click", "Visit", "Inquiry", "Order"],
    },
    series: [
      {
        name: "Funnel",
```

```
        type: "funnel",
        left: "20%",
        top: 60,
        bottom: 60,
        width: "60%",
        sort: "descending",
        gap: 2,
        label: {
          show: true,
          position: "inside",
        },
        labelLine: {
          length: 10,
          lineStyle: {
            width: 1,
            type: "solid",
          },
        },
        itemStyle: {
          borderColor: "#fff",
          borderWidth: 1,
        },
        emphasis: {
          label: {
            fontSize: 20,
          },
        },
        data: [
          { value: 60, name: "Visit" },
          { value: 40, name: "Inquiry" },
          { value: 20, name: "Order" },
          { value: 80, name: "Click" },
          { value: 100, name: "Show" },
        ],
      },
    ],
  };
  onMounted(() => {
    const chart = echarts.init(
        document.getElementById(props.id) as HTMLDivElement
    );
    chart.setOption(options);
    window.addEventListener("resize", () => {
      chart.resize();
    });
  });
</script>
<!-- 饼图 -->
<template>
  <el-card>
    <template #header> 去雾方法饼图 </template>
    <div :id="id" :class="className" :style="{ height, width }"></div>
  </el-card>
</template>
<script setup lang="ts">
```

```
import * as echarts from "echarts";
const props = defineProps({
  id: {
    type: String,
    default: "pieChart",
  },
  className: {
    type: String,
    default: "",
  },
  width: {
    type: String,
    default: "200px",
    required: true,
  },
  height: {
    type: String,
    default: "200px",
    required: true,
  },
});
const options = {
  grid: {
    left: "2%",
    right: "2%",
    bottom: "10%",
    containLabel: true,
  },
  legend: {
    top: "bottom",
    textStyle: {
      color: "#999",
    },
  },
  series: [
    {
      name: "Nightingale Chart",
      type: "pie",
      radius: [50, 130],
      center: ["50%", "50%"],
      roseType: "area",
      itemStyle: {
        borderRadius: 1,
        color: function (params: any) {
          //自定义颜色
          const colorList = ["#409EFF", "#67C23A", "#E6A23C", "#F56C6C"];
          return colorList[params.dataIndex];
        },
      },
      data: [
        { value: 58, name: "解码器-编码器" },
        { value: 27, name: "Transformer" },
        { value: 10, name: "无监督" },
        { value: 5, name: "物理模型" },
```

```
        ],
      },
    ],
  };
  onMounted(() => {
    const chart = echarts.init(
        document.getElementById(props.id) as HTMLDivElement
    );
    chart.setOption(options);
    window.addEventListener("resize", () => {
      chart.resize();
    });
  });
</script>
<!-- 雷达图 -->
<template>
  <el-card>
    <template #header> 数据集情况雷达图 </template>
    <div :id="id" :class="className" :style="{ height, width }"></div>
  </el-card>
</template>
<script setup lang="ts">
  import * as echarts from "echarts";
  const props = defineProps({
    id: {
      type: String,
      default: "radarChart",
    },
    className: {
      type: String,
      default: "",
    },
    width: {
      type: String,
      default: "200px",
      required: true,
    },
    height: {
      type: String,
      default: "200px",
      required: true,
    },
  });
  const options = {
    grid: {
      left: "2%",
      right: "2%",
      bottom: "10%",
      containLabel: true,
    },
    legend: {
      x: "center",
      y: "bottom",
      data: ["简单雾霾图", "困难雾霾图", "真实雾霾图"],
      textStyle: {
```

```
        color: "#999",
      },
    },
    radar: {
      // shape: 'circle',
      radius: "60%",
      indicator: [
        { name: "RESIDE" },
        { name: "Dense-Haze" },
        { name: "I-Haze" },
        { name: "O-Haze" },
        { name: "RS-Haze" },
        { name: "NH-Haze" },
      ],
    },
    series: [
      {
        name: "Budget vs spending",
        type: "radar",
        itemStyle: {
          borderRadius: 6,
          color: function (params: any) {
            //自定义颜色
            const colorList = ["#409EFF", "#67C23A", "#E6A23C", "#F56C6C"];
            return colorList[params.dataIndex];
          },
        },
        data: [
          {
            value: [400, 100, 200, 600, 300, 100],
            name: "简单雾霾图",
          },
          {
            value: [300, 100, 100, 200, 600, 100],
            name: "困难雾霾图",
          },
          {
            value: [800, 300, 200, 100, 600, 500],
            name: "真实雾霾图",
          },
        ],
      },
    ],
  };
  onMounted(() => {
    const chart = echarts.init(
        document.getElementById(props.id) as HTMLDivElement
    );
    chart.setOption(options);
    window.addEventListener("resize", () => {
      chart.resize();
    });
  });
</script>
```

```
<script setup lang="ts">
  import { calculateIndexApi, dehazeApi, getModelApi } from "@/api/dehaze";
  import { ModelInfo } from "@/api/dehaze/types";
  import SingleUploadPy from "@/components/Upload/SingleUploadPy.vue";
  import { useAppStore } from "@/store/modules/app";
  import { imageBaseURL } from "@/utils/request-py";
  import { ElMessage } from "element-plus";
  const dehazeModels = ref<ModelInfo[]>();
  const selectedDehazeModel = ref("");
  const appStore = useAppStore();
  onMounted(async () => {
    const { data } = await getModelApi();
    dehazeModels.value = data;
  });
  const hazeImage = reactive({
    name: "",
    height: 0,
    width: 0,
  });
  const clearImage = reactive({
    name: "",
    height: 0,
    width: 0,
  });
  const outputImage = reactive({
    name: "",
    height: 0,
    width: 0,
  });
  const dehazedImgUrl = computed(() => {
    return outputImage.name ? imageBaseURL + "/" + outputImage.name + "/" : "";
  });
  const isLoading = ref(false);
  const loadingText = ref("开始去雾");
  async function dehazeImage() {
    if (selectedDehazeModel.value.length === 0) {
      ElMessage.warning("请选择去雾模型！");
      return;
    }
    if (hazeImage.name.length === 0) {
      ElMessage.warning("请上传有雾图像！");
      return;
    }
    isLoading.value = true;
    loadingText.value = "正在去雾";
    try {
      const { data } = await dehazeApi(hazeImage.name, selectedDehazeModel.val
ue);
      outputImage.name = data.image_name;
    } finally {
      isLoading.value = false;
      loadingText.value = "开始去雾";
    }
  }
```

```
  const psnr = ref(0);
  const psnrEvaluate = computed(() => {
    if (psnr.value > 40) return "奇迹";
    else if (psnr.value > 30) return "优秀";
    else if (psnr.value > 25) return "良好";
    else if (psnr.value > 20) return "一般";
    else if (psnr.value > 1) return "不及格";
    else return "-";
  });
  const ssim = ref(0);
  const ssimEvaluate = computed(() => {
    if (ssim.value > 0.95) return "奇迹";
    else if (ssim.value > 0.8) return "优秀";
    else if (ssim.value > 0.6) return "良好";
    else if (ssim.value > 0.4) return "一般";
    else if (ssim.value > 0.01) return "不及格";
    else return "-";
  });
  const vi = ref(0);
  const viEvaluate = computed(() => "-");
  const ri = ref(0);
  const riEvaluate = computed(() => "-");
  const comprehensiveReview = computed(() => {
    if (ssim.value === 0 || psnr.value === 0) return "-";
    const score = (psnr.value / 40 + ssim.value / 1) / 2;
    let result;
    if (score > 0.95) result = "简直是奇迹";
    else if (score > 0.8) result = "非常优秀";
    else if (score > 0.6) result = "良好";
    else if (score > 0.4) result = "一般";
    else if (score > 0) result = "太差了！不及格";
    else result = "无法评价";
    return `该图像去雾效果${result}，在 PSNR 表现${psnrEvaluate.value}、SSIM 指标
上的表现${ssimEvaluate.value}！`;
  });
  async function calculateDehazeIndex() {
    if (
        hazeImage.height !== clearImage.height ||
        hazeImage.width !== clearImage.width
    ) {
      ElMessage.warning("基准无雾图像和传入的有雾图像分辨率不对应，无法计算指标");
      return;
    }
    if (outputImage.name.length === 0 || clearImage.name.length === 0) {
      ElMessage.warning("未点击去雾或者未上传基准无雾图像，无法去雾");
      return;
    }
    const { data } = await calculateIndexApi(
        outputImage.name,
        clearImage.name,
        (progressEvent) => {
          if (progressEvent.total) {
            evaluatePercentage.value = Math.round(
```

```
                (progressEvent.loaded / progressEvent.total) * 100
            );
          } else {
            evaluatePercentage.value = 100;
          }
        }
    );
    psnr.value = Math.floor(parseFloat(data.psnr) * 100) / 100;
    ssim.value = Math.floor(parseFloat(data.ssim) * 10000) / 10000;
  }
  function clearEvaluateResult() {
    clearImage.name = "";
    clearImage.height = 0;
    clearImage.width = 0;
    evaluatePercentage.value = 0;
    psnr.value = 0;
    ssim.value = 0;
  }
  function getHazeImageInfo(width: number, height: number) {
    hazeImage.width = width;
    hazeImage.height = height;
  }
  function getClearImageInfo(width: number, height: number) {
    clearImage.width = width;
    clearImage.height = height;
  }
  function resetForm() {
    clearEvaluateResult();
    outputImage.name = "";
    outputImage.height = 0;
    outputImage.width = 0;
    hazeImage.name = "";
    hazeImage.width = 0;
    hazeImage.height = 0;
    selectedDehazeModel.value = "";
  }
  const dialogVisible = ref(false);
  const drawerVisible = ref(false);
  const evaluatePercentage = ref(0);
</script>
<template>
  <div class="app-container">
    <h1 style="margin: 5px 0 10px; text-align: center">多模型图像去雾系统</h1>
    <div class="operate-panel">
      <el-cascader
          placeholder="请选择去雾模型"
          v-model="selectedDehazeModel"
          :options="dehazeModels"
          filterable
          :props="{ emitPath: false }"
          :clearable="true"
      />
      <div class="operate-panel-right">
        <el-button type="primary" @click="dehazeImage" :loading="isLoading"
        >{{ loadingText }}
```

```
        </el-button>
        <el-button @click="dialogVisible = !dialogVisible">评估效果</el-button>
        <el-button @click="drawerVisible = !drawerVisible">历史记录</el-button>
        <el-button type="warning" @click="resetForm">重置页面</el-button>
      </div>
    </div>
    <div class="image-show-container">
      <single-upload-py
          v-model="hazeImage.name"
          title="上传有雾图像"
          @get-image-info="getHazeImageInfo"
      />
      <span style="margin: 0 20px"></span>
      <el-image
          style="width: 35vmax; height: 35vmax"
          :src="dehazedImgUrl"
          fit="fill"
          alt="@/assets/photo.png"
      >
        <template #error>
          <div class="image-show-placeholder">
            <div class="image-show-text">
              请上传有雾图像<br />并点击"开始去雾"获取无雾图像
            </div>
          </div>
        </template>
      </el-image>
    </div>
    <el-dialog
        v-model="dialogVisible"
        :width="appStore.device === 'mobile' ? '100%' : '70%'"
        title="评估效果"
        top="2vh"
        class="dialog-class"
    >
      <el-alert
          :closable="false"
          show-icon
          description="评估去雾模型的去雾效果需要上传用于比较的基准无雾图像，系统才能够计算出模型去雾效果和真实的无雾图像之间的差距。真实的无雾图像需要和原始的有雾图像拍摄位置、图像宽高大小一致，否则会出现错误"
      />
      <div class="dialog-content">
        <single-upload-py
            style="margin-right: 20px"
            v-model="clearImage.name"
            title="上传基准无雾图像"
            @get-image-info="getClearImageInfo"
        />
        <div class="dialog-content-right">
          <div class="dialog-content-right-up">
            <el-button type="primary" @click="calculateDehazeIndex"
            >开始评估
            </el-button>
```

```
  <el-button type="info" @click="clearEvaluateResult"
  >清空结果
  </el-button>
</div>
<div class="dialog-content-right-center">
  <div style="display: flex; justify-content: center; width: 100%">
    <el-progress
        v-if="evaluatePercentage !== 100"
        type="dashboard"
        :percentage="evaluatePercentage"
        :color="0"
    />
  </div>
  <el-result v-if="evaluatePercentage === 100" icon="success" />
</div>
<div>
  <div class="text-class">
    <span style="margin-right: 16px">雾霾图像分辨率</span>
    {{ hazeImage.width + " * " + hazeImage.height }}
  </div>
  <div class="text-class">
    <span style="margin-right: 16px">基准图像分辨率</span>
    {{ clearImage.width + " * " + clearImage.height }}
  </div>
  <el-descriptions :column="2" size="large">
    <el-descriptions-item :span="1" :min-width="50" label="PSNR"
    >{{ psnr }}
    </el-descriptions-item>
    <el-descriptions-item :min-width="50" label="评价"
    >{{ psnrEvaluate }}
    </el-descriptions-item>
    <el-descriptions-item :min-width="50" label="SSIM"
    >{{ ssim }}
    </el-descriptions-item>
    <el-descriptions-item :min-width="50" label="评价"
    >{{ ssimEvaluate }}
    </el-descriptions-item>
    <el-descriptions-item :min-width="50" label="VI"
    >{{ vi }}
    </el-descriptions-item>
    <el-descriptions-item :min-width="50" label="评价"
    >{{ viEvaluate }}
    </el-descriptions-item>
    <el-descriptions-item :min-width="50" label="RI"
    >{{ ri }}
    </el-descriptions-item>
    <el-descriptions-item :min-width="50" label="评价"
    >{{ riEvaluate }}
    </el-descriptions-item>
    <el-descriptions-item :width="150" label="综合点评">
      {{ comprehensiveReview }}
    </el-descriptions-item>
  </el-descriptions>
</div>
```

```
        </div>
      </div>
    </el-dialog>
    <el-drawer
        :size="appStore.device === 'mobile' ? '100%' : '650'"
        v-model="drawerVisible"
        direction="rtl"
        title="历史记录"
    >
      <el-table>
        <el-table-column prop="id" label="序号" />
        <el-table-column prop="hazeModel" label="去雾模型" />
        <el-table-column prop="type" label="操作类别" />
        <el-table-column prop="operationTime" label="操作时间" />
        <el-table-column prop="detail" label="详情" />
      </el-table>
    </el-drawer>
  </div>
</template>
<style>
  .operate-panel {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin: 0 15px 20px;
  }
  .operate-panel-right {
    display: flex;
    flex-wrap: wrap;
    align-content: space-between;
  }
  .image-show-container {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    margin: 0 15px;
  }
  .image-show-placeholder {
    display: flex;
    align-content: center;
    align-items: center;
    justify-content: center;
    width: 35vmax;
    height: 35vmax;
    border: 1px dashed var(--el-border-color);
    border-radius: 6px;
  }
  .image-show-text {
    text-align: center;
  }
  .dialog-content {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
```

```
    margin-top: 15px;
  }
  .dialog-content-right {
    display: flex;
    flex-direction: column;
    flex-grow: 1;
    justify-content: space-between;
  }
  .dialog-content-right-up {
    display: flex;
    justify-content: space-evenly;
  }
  .dialog-content-right-center {
    height: 130px;
  }
  .text-class {
    padding: 1px 1px 16px;
    color: var(--el-text-color-primary);
  }
</style>
<template>
  <div class="login-container">
    <el-form
        ref="loginFormRef"
        :model="loginData"
        :rules="loginRules"
        class="login-form"
    >
      <div class="flex text-white items-center py-4 title-wrap">
        <span class="text-2xl flex-1 text-center title">
          {{ $t("login.title") }}
        </span>
        <lang-select class="text-white! cursor-pointer" />
      </div>
      <el-form-item prop="username">
        <div class="p-2 text-white">
          <svg-icon icon-class="user" />
        </div>
        <el-input
            ref="username"
            v-model="loginData.username"
            class="flex-1"
            size="large"
            :placeholder="$t('login.username')"
            name="username"
        />
      </el-form-item>
      <el-tooltip
          :disabled="isCapslock === false"
          content="Caps lock is On"
          placement="right"
      >
        <el-form-item prop="password">
          <span class="p-2 text-white">
            <svg-icon icon-class="password" />
```

```
          </span>
          <el-input
              v-model="loginData.password"
              class="flex-1"
              placeholder="密码"
              :type="passwordVisible === false ? 'password' : 'input'"
              size="large"
              name="password"
              @keyup="checkCapslock"
              @keyup.enter="handleLogin"
          />
          <span class="mr-2" @click="passwordVisible = !passwordVisible">
            <svg-icon
                :icon-class="passwordVisible === false ? 'eye' : 'eye-open'"
                class="text-white cursor-pointer"
            />
          </span>
        </el-form-item>
      </el-tooltip>
      <!-- 验证码 -->
      <el-form-item prop="verifyCode">
        <span class="p-2 text-white">
          <svg-icon icon-class="verify_code" />
        </span>
        <el-input
            v-model="loginData.verifyCode"
            auto-complete="off"
            :placeholder="$t('login.verifyCode')"
            class="w-[60%]"
            @keyup.enter="handleLogin"
        />
        <div class="captcha">
          <img :src="captchaBase64" @click="getCaptcha" />
        </div>
      </el-form-item>
      <el-button
          size="default"
          :loading="loading"
          type="primary"
          class="w-full"
          @click.prevent="handleLogin"
      >{{ $t("login.login") }}
      </el-button>
      <!-- 账号密码提示 -->
      <div class="mt-4 text-white text-sm">
        <span>去雾体验账号: dehaze</span>
        <span class="ml-4"> 密码: 123456</span>
      </div>
    </el-form>
  </div>
</template>
<script setup lang="ts">
  import router from "@/router";
  import LangSelect from "@/components/LangSelect/index.vue";
```

```
import SvgIcon from "@/components/SvgIcon/index.vue";
// 状态管理依赖
import { useUserStore } from "@/store/modules/user";
// API 依赖
import { LocationQuery, LocationQueryValue, useRoute } from "vue-router";
import { getCaptchaApi } from "@/api/auth";
import { LoginData } from "@/api/auth/types";
const userStore = useUserStore();
const route = useRoute();
/**
 * 按钮 loading
 */
const loading = ref(false);
/**
 * 是否大写锁定
 */
const isCapslock = ref(false);
/**
 * 密码是否可见
 */
const passwordVisible = ref(false);
/**
 * 验证码图片 Base64 字符串
 */
const captchaBase64 = ref();
/**
 * 登录表单引用
 */
const loginFormRef = ref(ElForm);
const loginData = ref<LoginData>({
  username: "",
  password: "",
});
const loginRules = {
  username: [{ required: true, trigger: "blur" }],
  password: [{ required: true, trigger: "blur", validator: passwordValidator
}],
  verifyCode: [{ required: true, trigger: "blur" }],
};
/**
 * 密码校验器
 */
function passwordValidator(rule: any, value: any, callback: any) {
  if (value.length < 6) {
    callback(new Error("The password can not be less than 6 digits"));
  } else {
    callback();
  }
}
/**
 * 检查输入大小写状态
 */
function checkCapslock(e: any) {
  const { key } = e;
```

```
    isCapslock.value = key && key.length === 1 && key >= "A" && key <= "Z";
  }
  /**
   * 获取验证码
   */
  function getCaptcha() {
    getCaptchaApi().then(({ data }) => {
      const { verifyCodeBase64, verifyCodeKey } = data;
      loginData.value.verifyCodeKey = verifyCodeKey;
      captchaBase64.value = verifyCodeBase64;
    });
  }
  /**
   * 登录
   */
  function handleLogin() {
    loginFormRef.value.validate((valid: boolean) => {
      if (valid) {
        loading.value = true;
        userStore
            .login(loginData.value)
            .then(() => {
              const query: LocationQuery = route.query;
              const redirect = (query.redirect as LocationQueryValue) ?? "/";
              const otherQueryParams = Object.keys(query).reduce(
                  (acc: any, cur: string) => {
                    if (cur !== "redirect") {
                      acc[cur] = query[cur];
                    }
                    return acc;
                  },
                  {}
              );
              router.push({ path: redirect, query: otherQueryParams });
            })
            .catch(() => {
              // 验证失败，重新生成验证码
              getCaptcha();
            })
            .finally(() => {
              loading.value = false;
            });
      }
    });
  }
  onMounted(() => {
    getCaptcha();
  });
</script>
<style lang="scss" scoped>
  .login-container {
    width: 100%;
    min-height: 100%;
    overflow: hidden;
    background-color: #2d3a4b;
```

```
    .title-wrap {
      filter: contrast(30);
      .title {
        letter-spacing: 4px;
        animation: showup 3s forwards;
      }
      @keyframes showup {
        0% {
          letter-spacing: -20px;
        }
        100% {
          letter-spacing: 4px;
        }
      }
    }
    .login-form {
      width: 520px;
      max-width: 100%;
      padding: 160px 35px 0;
      margin: 0 auto;
      overflow: hidden;
      .captcha {
        position: absolute;
        top: 0;
        right: 0;
        img {
          width: 120px;
          height: 48px;
          cursor: pointer;
        }
      }
    }
  }
  .el-form-item {
    background: rgb(0 0 0 / 10%);
    border: 1px solid rgb(255 255 255 / 10%);
    border-radius: 5px;
  }
  .el-input {
    background: transparent;
    // 子组件 scoped 无效，使用 :deep
    :deep(.el-input__wrapper) {
      padding: 0;
      background: transparent;
      box-shadow: none;
      .el-input__inner {
        color: #fff;
        background: transparent;
        border: 0;
        border-radius: 0;
        caret-color: #fff;
        &:-webkit-autofill {
          box-shadow: 0 0 0 1000px transparent inset !important;
          -webkit-text-fill-color: #fff !important;
        }
```

```
        CaptchaResult captcha = easyCaptchaService.getCaptcha();
        return Result.success(captcha);
    }
}
```

python 后端代码

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys
def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'dehazing_system.settings')

    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()

from pathlib import Path
BASE_DIR = Path(__file__).resolve().parent.parent

DEBUG = True

ALLOWED_HOSTS = ['*']

# APPEND_SLASH = False
# 最大文件上传大小 20MB（单位：字节）
DATA_UPLOAD_MAX_MEMORY_SIZE = 20971520

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

```
ROOT_URLCONF = 'dehazing_system.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates']
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'dehazing_system.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilari
tyValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidato
r',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidat
or',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValida
tor',
    },
]
LANGUAGE_CODE = 'en-US'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True

STATIC_URL = 'static/'

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

```
import os
import uuid
import torch
DEVICE = 'cuda:0' if torch.cuda.is_available() else 'cpu'
PROJECT_PATH = os.path.dirname(os.path.abspath(__file__))
DATA_PATH = os.path.join(PROJECT_PATH, "data")
MODEL_PATH = os.path.join(PROJECT_PATH, "trained_model")
if __name__ == '__main__':
    image_name = str(uuid.uuid4()) + ".png"
    image_path = os.path.join(DATA_PATH, image_name)
    print(image_path)

import json
import os.path
import traceback
import uuid
from django.http import HttpResponse, HttpRequest
import benchmark.C2PNet.run
import benchmark.DehazeFormer.run
import benchmark.MixDehazeNet.run
import benchmark.CMFNet.run
import benchmark.DEANet.run
import benchmark.FogRemoval.run
import benchmark.ITBdehaze.run
import benchmark.RIDCP.run
from benchmark.metrics import calculate
from global_variable import DATA_PATH

dehaze_model = {
    'C2PNet/OTS.pkl': benchmark.C2PNet.run.dehaze,
    'C2PNet/ITS.pkl': benchmark.C2PNet.run.dehaze,
    'DehazeFormer/indoor/dehazeformer-b.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/indoor/dehazeformer-d.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/indoor/dehazeformer-l.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/indoor/dehazeformer-m.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/indoor/dehazeformer-s.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/indoor/dehazeformer-t.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/indoor/dehazeformer-w.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/outdoor/dehazeformer-b.pth': benchmark.DehazeFormer.run.deha
ze,
    'DehazeFormer/outdoor/dehazeformer-m.pth': benchmark.DehazeFormer.run.deha
ze,
    'DehazeFormer/outdoor/dehazeformer-s.pth': benchmark.DehazeFormer.run.deha
ze,
    'DehazeFormer/outdoor/dehazeformer-t.pth': benchmark.DehazeFormer.run.deha
ze,
    'DehazeFormer/reside6k/dehazeformer-b.pth': benchmark.DehazeFormer.run.deh
```

```
aze,
    'DehazeFormer/reside6k/dehazeformer-m.pth': benchmark.DehazeFormer.run.deh
aze,
    'DehazeFormer/reside6k/dehazeformer-s.pth': benchmark.DehazeFormer.run.deh
aze,
    'DehazeFormer/reside6k/dehazeformer-t.pth': benchmark.DehazeFormer.run.deh
aze,
    'DehazeFormer/rshaze/dehazeformer-b.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/rshaze/dehazeformer-m.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/rshaze/dehazeformer-s.pth': benchmark.DehazeFormer.run.dehaz
e,
    'DehazeFormer/rshaze/dehazeformer-t.pth': benchmark.DehazeFormer.run.dehaz
e,
    'MixDehazeNet/haze4k/MixDehazeNet-l.pth': benchmark.MixDehazeNet.run.dehaz
e,
    'MixDehazeNet/indoor/MixDehazeNet-l.pth': benchmark.MixDehazeNet.run.dehaz
e,
    'MixDehazeNet/indoor/MixDehazeNet-b.pth': benchmark.MixDehazeNet.run.dehaz
e,
    'MixDehazeNet/outdoor/MixDehazeNet-b.pth': benchmark.MixDehazeNet.run.deha
ze,
    'MixDehazeNet/outdoor/MixDehazeNet-l.pth': benchmark.MixDehazeNet.run.deha
ze,
    'MixDehazeNet/outdoor/MixDehazeNet-s.pth': benchmark.MixDehazeNet.run.deha
ze,
    'CMFNet/dehaze_I_OHaze_CMFNet.pth': benchmark.CMFNet.run.dehaze,
    'DEA-Net/HAZE4K/PSNR3426_SSIM9885.pth': benchmark.DEANet.run.dehaze,
    'DEA-Net/ITS/PSNR4131_SSIM9945.pth': benchmark.DEANet.run.dehaze,
    'DEA-Net/OTS/PSNR3659_SSIM9897.pth': benchmark.DEANet.run.dehaze,
    'FogRemoval/NH-HAZE_params_0100000.pt': benchmark.FogRemoval.run.dehaze,
    'ITBdehaze/best.pkl': benchmark.ITBdehaze.run.dehaze,
    'RIDCP/pretrained_RIDCP.pth': benchmark.RIDCP.run.dehaze,
}


def ok_response(data):
    message = {
        'code': '00000',
        'msg': '一切 ok',
        'data': data
    }
    return HttpResponse(json.dumps(message), content_type='application/json')


def error_response(code, msg):
    message = {
        'code': code,
        'msg': msg,
        'data': None
    }
    return HttpResponse(json.dumps(message), content_type='application/json')
```

```
def get_model(request: HttpRequest):
    result = []
    for index, key in enumerate(dehaze_model):
        # 首先将字符串按照 / 分割成数组
        parts = key.split('/')
        # 然后获取当前已经组装好的结果，准备继续向内部添加当前结点
        current = result
        # 遍历该数组，创建嵌套的数组
        for i, part in enumerate(parts):
            # 如果当前元素是数组的最后一个元素，也就是'DehazeFormer/indoor/dehazeformer-b.pth' 中的 'dehazeformer-b.pth'
            # 那么就将当前元素放入结果数组中
            if i == len(parts) - 1:
                current.append({'value': key, 'label': part.split(".")[0]})
            else:
                # 如果不是最后一个元素，则遍历结果数组，直到找到一个 key 和当前的元素一样的
                # 就更改当前结果数组
                found = False
                for child in current:
                    if child['value'] == part:
                        current = child['children']
                        found = True
                        break
                # 如果没有找到则创建一个新元素，插入到结果数组中，并且更新当前结果数组
                if not found:
                    new_node = {'value': part, 'label': part, 'children': []}
                    current.append(new_node)
                    current = new_node['children']
    return ok_response(result)


def upload_image(request: HttpRequest):
    image_name = str(uuid.uuid4()) + ".png"
    image_path = os.path.join(DATA_PATH, image_name)
    image = request.body
    # 保存前端传来的图片
    with open(image_path, "wb") as destination:
        destination.write(image)
    return ok_response({'image_name': image_name})


def download_image(request: HttpRequest, image_name: str):
    image_path = os.path.join(DATA_PATH, image_name)
    with open(image_path, "rb") as destination:
        return HttpResponse(destination.read(), content_type="image/png")


def dehaze_image(request: HttpRequest):
    data = json.loads(request.body)
    haze_image_name = data["haze_image"]
    model_name = data["model_name"]
```

```
    output_image_name = str(uuid.uuid4()) + ".png"
    haze_image_path = os.path.join(DATA_PATH, haze_image_name)
    output_image_path = os.path.join(DATA_PATH, output_image_name)

    try:
        dehaze = dehaze_model.get(model_name, None)
        if dehaze is not None:
            dehaze(haze_image_path, output_image_path, model_name)
        else:
            return error_response('1', "无法找到模型")
    except RuntimeError as e:
        traceback.print_exc()
        return error_response('1', e.__str__())

    return ok_response({'image_name': output_image_name})


def calculate_dehaze_index(request: HttpRequest):
    data = json.loads(request.body)
    haze_image_name = data["haze_image"]
    clear_image_name = data["clear_image"]
    haze_image_path = os.path.join(DATA_PATH, haze_image_name)
    clear_image_path = os.path.join(DATA_PATH, clear_image_name)

    psnr, ssim = calculate(haze_image_path, clear_image_path)
    return ok_response({'psnr': psnr, 'ssim': ssim})

from django.contrib import admin
from django.urls import path
import dehazing_system.photo
urlpatterns = [
    path('admin/', admin.site.urls),
    path('model/', dehazing_system.photo.get_model),
    path("upload/", dehazing_system.photo.upload_image),
    path('download/<str:image_name>/', dehazing_system.photo.download_image),
    path('dehazeImage/', dehazing_system.photo.dehaze_image),
    path('calculateIndex/', dehazing_system.photo.calculate_dehaze_index),
]
import numpy as np
from PIL import Image
from skimage.metrics import peak_signal_noise_ratio, structural_similarity


def calculate(haze_image_path: str, clear_image_path: str):
    haze = Image.open(haze_image_path).convert('RGB')
    clear = Image.open(clear_image_path).convert('RGB')
    haze = np.array(haze)
    clear = np.array(clear)
    current_psnr = peak_signal_noise_ratio(haze, clear)
    current_ssim = structural_similarity(haze, clear, channel_axis=2)
    return current_psnr, current_ssim

import torch
from PIL import Image
import torchvision.transforms as tfs
```

```
import torchvision.utils as torch_utils
from skimage.metrics import peak_signal_noise_ratio, structural_similarity
from benchmark.C2PNet.model import C2PNet
import os
# from benchmark.C2PNet.metrics import psnr, ssim
from global_variable import MODEL_PATH, DEVICE

def get_model(model_name: str):
    # 构造模型文件的绝对路径
    model_dir = os.path.join(MODEL_PATH, model_name)
    net = C2PNet(gps=3, blocks=19)
    ckp = torch.load(model_dir)
    net = net.to(DEVICE)
    net.load_state_dict(ckp['model'])
    net.eval()
    return net

def dehaze(haze_image_path: str, output_image_path: str, model_name: str = 'C2
PNet/OTS.pkl'):
    net = get_model(model_name)
    haze = Image.open(haze_image_path).convert('RGB')
    haze = tfs.ToTensor()(haze)[None, ::]
    haze = haze.to(DEVICE)
    with torch.no_grad():
        pred = net(haze)
    ts = torch.squeeze(pred.clamp(0, 1).cpu())
    torch_utils.save_image(ts, output_image_path)

import torch
import torch.nn as nn


def default_conv(in_channels, out_channels, kernel_size, bias=True):
    return nn.Conv2d(in_channels, out_channels, kernel_size, padding=(kernel_s
ize // 2), bias=bias)


class CALayer(nn.Module):
    def __init__(self, channel):
        super(CALayer, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.ca = nn.Sequential(
            nn.Conv2d(channel, channel // 8, 1, padding=0, bias=True),
            nn.ReLU(inplace=True),
            nn.Conv2d(channel // 8, channel, 1, padding=0, bias=True),
            nn.Sigmoid()
        )

    def forward(self, x):
        y = self.avg_pool(x)
        y = self.ca(y)
        return x * y


class PDU(nn.Module):  # physical block
```

```
    def __init__(self, channel):
        super(PDU, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.ka = nn.Sequential(
            nn.Conv2d(channel, channel // 8, 1, padding=0, bias=True),
            nn.ReLU(inplace=True),
            nn.Conv2d(channel // 8, channel, 1, padding=0, bias=True),
            nn.Sigmoid()
        )
        self.td = nn.Sequential(
            default_conv(channel, channel, 3),
            default_conv(channel, channel // 8, 3),
            nn.ReLU(inplace=True),
            default_conv(channel // 8, channel, 3),
            nn.Sigmoid()
        )

    def forward(self, x):
        a = self.avg_pool(x)
        a = self.ka(a)
        t = self.td(x)
        j = torch.mul((1 - t), a) + torch.mul(t, x)
        return j


class Block(nn.Module):  # origin
    def __init__(self, conv, dim, kernel_size, ):
        super(Block, self).__init__()
        self.conv1 = conv(dim, dim, kernel_size, bias=True)
        self.act1 = nn.ReLU(inplace=True)
        self.conv2 = conv(dim, dim, kernel_size, bias=True)
        self.calayer = CALayer(dim)
        self.pdu = PDU(dim)

    def forward(self, x):
        res = self.act1(self.conv1(x))
        res = res + x
        res = self.conv2(res)
        res = self.calayer(res)
        res = self.pdu(res)
        res += x
        return res


class Group(nn.Module):
    def __init__(self, conv, dim, kernel_size, blocks):
        super(Group, self).__init__()
        modules = [Block(conv, dim, kernel_size) for _ in range(blocks)]
        modules.append(conv(dim, dim, kernel_size))
        self.gp = nn.Sequential(*modules)

    def forward(self, x):
        res = self.gp(x)
        res += x
        return res
```

```
class C2PNet(nn.Module):
    def __init__(self, gps, blocks, conv=default_conv):
        super(C2PNet, self).__init__()
        self.gps = gps
        self.dim = 64
        kernel_size = 3
        pre_process = [conv(3, self.dim, kernel_size)]
        assert self.gps == 3
        self.g1 = Group(conv, self.dim, kernel_size, blocks=blocks)
        self.g2 = Group(conv, self.dim, kernel_size, blocks=blocks)
        self.g3 = Group(conv, self.dim, kernel_size, blocks=blocks)
        self.ca = nn.Sequential(*[
            nn.AdaptiveAvgPool2d(1),
            nn.Conv2d(self.dim * self.gps, self.dim // 16, 1, padding=0),
            nn.ReLU(inplace=True),
            nn.Conv2d(self.dim // 16, self.dim * self.gps, 1, padding=0, bias=
True),
            nn.Sigmoid()
        ])
        self.pdu = PDU(self.dim)

        post_precess = [
            conv(self.dim, self.dim, kernel_size),
            conv(self.dim, 3, kernel_size)]

        self.pre = nn.Sequential(*pre_process)
        self.post = nn.Sequential(*post_precess)

    def forward(self, x1):
        x = self.pre(x1)
        res1 = self.g1(x)
        res2 = self.g2(res1)
        res3 = self.g3(res2)
        w = self.ca(torch.cat([res1, res2, res3], dim=1))
        w = w.view(-1, self.gps, self.dim)[:, :, :, None, None]
        out = w[:, 0, ::] * res1 + w[:, 1, ::] * res2 + w[:, 2, ::] * res3
        out = self.pdu(out)
        x = self.post(out)
        return x + x1


if __name__ == "__main__":
    net = C2PNet(gps=3, blocks=19)
    print(net)

import math
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.checkpoint as checkpoint
from timm.models.layers import DropPath, to_2tuple, trunc_normal_
```

```
class Mlp(nn.Module):
    def __init__(self, in_features, hidden_features=None, out_features=None, a
ct_layer=nn.GELU, drop=0.):
        super().__init__()
        out_features = out_features or in_features
        hidden_features = hidden_features or in_features
        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = act_layer()
        self.fc2 = nn.Linear(hidden_features, out_features)
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.drop(x)
        x = self.fc2(x)
        x = self.drop(x)
        return x


def window_partition(x, window_size):
    """
    Args:
        x: (B, H, W, C)
        window_size (int): window size

    Returns:
        windows: (num_windows*B, window_size, window_size, C)
    """
    B, H, W, C = x.shape
    x = x.view(B, H // window_size, window_size, W // window_size, window_size,
 C)
    windows = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(-1, window_size, w
indow_size, C)
    return windows


def window_reverse(windows, window_size, H, W):
    """
    Args:
        windows: (num_windows*B, window_size, window_size, C)
        window_size (int): Window size
        H (int): Height of image
        W (int): Width of image

    Returns:
        x: (B, H, W, C)
    """
    B = int(windows.shape[0] / (H * W / window_size / window_size))
    x = windows.view(B, H // window_size, W // window_size, window_size, windo
w_size, -1)
    x = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(B, H, W, -1)
    return x
```

```
class WindowAttention(nn.Module):
    r""" Window based multi-head self attention (W-MSA) module with relative p
osition bias.
    It supports both of shifted and non-shifted window.

    Args:
        dim (int): Number of input channels.
        window_size (tuple[int]): The height and width of the window.
        num_heads (int): Number of attention heads.
        qkv_bias (bool, optional):  If True, add a learnable bias to query, ke
y, value. Default: True
        qk_scale (float | None, optional): Override default qk scale of head_d
im ** -0.5 if set
        attn_drop (float, optional): Dropout ratio of attention weight. Defaul
t: 0.0
        proj_drop (float, optional): Dropout ratio of output. Default: 0.0
    """

    def __init__(self, dim, window_size, num_heads, qkv_bias=True, qk_scale=No
ne, attn_drop=0., proj_drop=0.):

        super().__init__()
        self.dim = dim
        self.window_size = window_size  # Wh, Ww
        self.num_heads = num_heads
        head_dim = dim // num_heads
        self.scale = qk_scale or head_dim ** -0.5

        # define a parameter table of relative position bias
        self.relative_position_bias_table = nn.Parameter(
            torch.zeros((2 * window_size[0] - 1) * (2 * window_size[1] - 1), n
um_heads))  # 2*Wh-1 * 2*Ww-1, nH

        # get pair-wise relative position index for each token inside the wind
ow
        coords_h = torch.arange(self.window_size[0])
        coords_w = torch.arange(self.window_size[1])
        coords = torch.stack(torch.meshgrid([coords_h, coords_w]))  # 2, Wh, W
w
        coords_flatten = torch.flatten(coords, 1)  # 2, Wh*Ww
        relative_coords = coords_flatten[:, :, None] - coords_flatten[:, None,
 :]  # 2, Wh*Ww, Wh*Ww
        relative_coords = relative_coords.permute(1, 2, 0).contiguous()  # Wh*
Ww, Wh*Ww, 2
        relative_coords[:, :, 0] += self.window_size[0] - 1  # shift to start
from 0
        relative_coords[:, :, 1] += self.window_size[1] - 1
        relative_coords[:, :, 0] *= 2 * self.window_size[1] - 1
        relative_position_index = relative_coords.sum(-1)  # Wh*Ww, Wh*Ww
        self.register_buffer("relative_position_index", relative_position_inde
x)

        self.qkv = nn.Linear(dim, dim * 3, bias=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop)
        self.proj = nn.Linear(dim, dim)
```

```
        self.proj_drop = nn.Dropout(proj_drop)

        trunc_normal_(self.relative_position_bias_table, std=.02)
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, x, mask=None):
        """
        Args:
            x: input features with shape of (num_windows*B, N, C)
            mask: (0/-inf) mask with shape of (num_windows, Wh*Ww, Wh*Ww) or N
one
        """
        B_, N, C = x.shape
        qkv = self.qkv(x).reshape(B_, N, 3, self.num_heads, C // self.num_head
s).permute(2, 0, 3, 1, 4)
        q, k, v = qkv[0], qkv[1], qkv[2]  # make torchscript happy (cannot use
 tensor as tuple)

        q = q * self.scale
        attn = (q @ k.transpose(-2, -1))

        relative_position_bias = self.relative_position_bias_table[self.relati
ve_position_index.view(-1)].view(
            self.window_size[0] * self.window_size[1], self.window_size[0] * s
elf.window_size[1], -1)  # Wh*Ww,Wh*Ww,nH
        relative_position_bias = relative_position_bias.permute(2, 0, 1).conti
guous()  # nH, Wh*Ww, Wh*Ww
        attn = attn + relative_position_bias.unsqueeze(0)

        if mask is not None:
            nW = mask.shape[0]
            attn = attn.view(B_ // nW, nW, self.num_heads, N, N) + mask.unsque
eze(1).unsqueeze(0)
            attn = attn.view(-1, self.num_heads, N, N)
            attn = self.softmax(attn)
        else:
            attn = self.softmax(attn)

        attn = self.attn_drop(attn)

        x = (attn @ v).transpose(1, 2).reshape(B_, N, C)
        x = self.proj(x)
        x = self.proj_drop(x)
        return x

    def extra_repr(self) -> str:
        return f'dim={self.dim}, window_size={self.window_size}, num_heads={se
lf.num_heads}'

    def flops(self, N):
        # calculate flops for 1 window with token length of N
        flops = 0
        # qkv = self.qkv(x)
        flops += N * self.dim * 3 * self.dim
```

```
        # attn = (q @ k.transpose(-2, -1))
        flops += self.num_heads * N * (self.dim // self.num_heads) * N
        #  x = (attn @ v)
        flops += self.num_heads * N * N * (self.dim // self.num_heads)
        # x = self.proj(x)
        flops += N * self.dim * self.dim
        return flops


class SwinTransformerBlock(nn.Module):
    def __init__(self, dim, input_resolution, num_heads, window_size=7, shift_
size=0,
                 mlp_ratio=4., qkv_bias=True, qk_scale=None, drop=0., attn_dro
p=0., drop_path=0.,
                 act_layer=nn.GELU, norm_layer=nn.LayerNorm):
        super().__init__()
        self.dim = dim
        self.input_resolution = input_resolution
        self.num_heads = num_heads
        self.window_size = window_size
        self.shift_size = shift_size
        self.mlp_ratio = mlp_ratio
        if min(self.input_resolution) <= self.window_size:
            # if window size is larger than input resolution, we don't partiti
on windows
            self.shift_size = 0
            self.window_size = min(self.input_resolution)
        assert 0 <= self.shift_size < self.window_size, "shift_size must in 0-
window_size"

        self.norm1 = norm_layer(dim)
        self.attn = WindowAttention(
            dim, window_size=to_2tuple(self.window_size), num_heads=num_heads,

            qkv_bias=qkv_bias, qk_scale=qk_scale, attn_drop=attn_drop, proj_dr
op=drop)

        self.drop_path = DropPath(drop_path) if drop_path > 0. else nn.Identit
y()
        self.norm2 = norm_layer(dim)
        mlp_hidden_dim = int(dim * mlp_ratio)
        self.mlp = Mlp(in_features=dim, hidden_features=mlp_hidden_dim, act_la
yer=act_layer, drop=drop)

        if self.shift_size > 0:
            attn_mask = self.calculate_mask(self.input_resolution)
        else:
            attn_mask = None

        self.register_buffer("attn_mask", attn_mask)

    def calculate_mask(self, x_size):
        # calculate attention mask for SW-MSA
        H, W = x_size
        img_mask = torch.zeros((1, H, W, 1))  # 1 H W 1
```

```
        h_slices = (slice(0, -self.window_size),
                    slice(-self.window_size, -self.shift_size),
                    slice(-self.shift_size, None))
        w_slices = (slice(0, -self.window_size),
                    slice(-self.window_size, -self.shift_size),
                    slice(-self.shift_size, None))
        cnt = 0
        for h in h_slices:
            for w in w_slices:
                img_mask[:, h, w, :] = cnt
                cnt += 1

        mask_windows = window_partition(img_mask, self.window_size)  # nW, win
dow_size, window_size, 1
        mask_windows = mask_windows.view(-1, self.window_size * self.window_si
ze)
        attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)
        attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).maske
d_fill(attn_mask == 0, float(0.0))

        return attn_mask

    def forward(self, x, x_size):
        H, W = x_size
        B, L, C = x.shape
        # assert L == H * W, "input feature has wrong size"

        shortcut = x
        x = self.norm1(x)
        x = x.view(B, H, W, C)

        # cyclic shift
        if self.shift_size > 0:
            shifted_x = torch.roll(x, shifts=(-self.shift_size, -self.shift_si
ze), dims=(1, 2))
        else:
            shifted_x = x

        # partition windows
        x_windows = window_partition(shifted_x, self.window_size)  # nW*B, win
dow_size, window_size, C
        x_windows = x_windows.view(-1, self.window_size * self.window_size, C)
  # nW*B, window_size*window_size, C

        # W-MSA/SW-MSA (to be compatible for testing on images whose shapes ar
e the multiple of window size
        if self.input_resolution == x_size:
            attn_windows = self.attn(x_windows, mask=self.attn_mask)  # nW*B,
window_size*window_size, C
        else:
            attn_windows = self.attn(x_windows, mask=self.calculate_mask(x_siz
e).to(x.device))

        # merge windows
        attn_windows = attn_windows.view(-1, self.window_size, self.window_siz
```

```
e, C)
        shifted_x = window_reverse(attn_windows, self.window_size, H, W)  # B
H' W' C

        # reverse cyclic shift
        if self.shift_size > 0:
            x = torch.roll(shifted_x, shifts=(self.shift_size, self.shift_siz
e), dims=(1, 2))
        else:
            x = shifted_x
        x = x.view(B, H * W, C)

        # FFN
        x = shortcut + self.drop_path(x)
        x = x + self.drop_path(self.mlp(self.norm2(x)))

        return x

    def extra_repr(self) -> str:
        return f"dim={self.dim}, input_resolution={self.input_resolution}, num
_heads={self.num_heads}, "
               f"window_size={self.window_size}, shift_size={self.shift_size},
 mlp_ratio={self.mlp_ratio}"

    def flops(self):
        flops = 0
        H, W = self.input_resolution
        # norm1
        flops += self.dim * H * W
        # W-MSA/SW-MSA
        nW = H * W / self.window_size / self.window_size
        flops += nW * self.attn.flops(self.window_size * self.window_size)
        # mlp
        flops += 2 * H * W * self.dim * self.dim * self.mlp_ratio
        # norm2
        flops += self.dim * H * W
        return flops


class PatchMerging(nn.Module):
    def __init__(self, input_resolution, dim, norm_layer=nn.LayerNorm):
        super().__init__()
        self.input_resolution = input_resolution
        self.dim = dim
        self.reduction = nn.Linear(4 * dim, 2 * dim, bias=False)
        self.norm = norm_layer(4 * dim)

    def forward(self, x):
        """
        x: B, H*W, C
        """
        H, W = self.input_resolution
        B, L, C = x.shape
        assert L == H * W, "input feature has wrong size"
        assert H % 2 == 0 and W % 2 == 0, f"x size ({H}*{W}) are not even."
```

```
        x = x.view(B, H, W, C)

        x0 = x[:, 0::2, 0::2, :]  # B H/2 W/2 C
        x1 = x[:, 1::2, 0::2, :]  # B H/2 W/2 C
        x2 = x[:, 0::2, 1::2, :]  # B H/2 W/2 C
        x3 = x[:, 1::2, 1::2, :]  # B H/2 W/2 C
        x = torch.cat([x0, x1, x2, x3], -1)  # B H/2 W/2 4*C
        x = x.view(B, -1, 4 * C)  # B H/2*W/2 4*C

        x = self.norm(x)
        x = self.reduction(x)

        return x

    def extra_repr(self) -> str:
        return f"input_resolution={self.input_resolution}, dim={self.dim}"

    def flops(self):
        H, W = self.input_resolution
        flops = H * W * self.dim
        flops += (H // 2) * (W // 2) * 4 * self.dim * 2 * self.dim
        return flops


class BasicLayer(nn.Module):
    def __init__(self, dim, input_resolution, depth, num_heads, window_size,
                 mlp_ratio=4., qkv_bias=True, qk_scale=None, drop=0., attn_dro
p=0.,
                 drop_path=0., norm_layer=nn.LayerNorm, downsample=None, use_c
heckpoint=False):

        super().__init__()
        self.dim = dim
        self.input_resolution = input_resolution
        self.depth = depth
        self.use_checkpoint = use_checkpoint

        # build blocks
        self.blocks = nn.ModuleList([
            SwinTransformerBlock(dim=dim, input_resolution=input_resolution,
                                 num_heads=num_heads, window_size=window_size,

                                 shift_size=0 if (i % 2 == 0) else window_size
 // 2,
                                 mlp_ratio=mlp_ratio,
                                 qkv_bias=qkv_bias, qk_scale=qk_scale,
                                 drop=drop, attn_drop=attn_drop,
                                 drop_path=drop_path[i] if isinstance(drop_pat
h, list) else drop_path,
                                 norm_layer=norm_layer)
            for i in range(depth)])

        # patch merging layer
        if downsample is not None:
```

```
            self.downsample = downsample(input_resolution, dim=dim, norm_layer
=norm_layer)
        else:
            self.downsample = None

    def forward(self, x, x_size):
        for blk in self.blocks:
            if self.use_checkpoint:
                x = checkpoint.checkpoint(blk, x, x_size)
            else:
                x = blk(x, x_size)
        if self.downsample is not None:
            x = self.downsample(x)
        return x

    def extra_repr(self) -> str:
        return f"dim={self.dim}, input_resolution={self.input_resolution}, dep
th={self.depth}"

    def flops(self):
        flops = 0
        for blk in self.blocks:
            flops += blk.flops()
        if self.downsample is not None:
            flops += self.downsample.flops()
        return flops

class RSTB(nn.Module):
    def __init__(self, dim, input_resolution, depth, num_heads, window_size,
                 mlp_ratio=4., qkv_bias=True, qk_scale=None, drop=0., attn_dro
p=0.,
                 drop_path=0., norm_layer=nn.LayerNorm, downsample=None, use_c
heckpoint=False,
                 img_size=224, patch_size=4, resi_connection='1conv'):
        super(RSTB, self).__init__()
        self.dim = dim
        self.input_resolution = input_resolution
        self.residual_group = BasicLayer(dim=dim,
                                         input_resolution=input_resolution,
                                         depth=depth,
                                         num_heads=num_heads,
                                         window_size=window_size,
                                         mlp_ratio=mlp_ratio,
                                         qkv_bias=qkv_bias, qk_scale=qk_scale,

                                         drop=drop, attn_drop=attn_drop,
                                         drop_path=drop_path,
                                         norm_layer=norm_layer,
                                         downsample=downsample,
                                         use_checkpoint=use_checkpoint)

        if resi_connection == '1conv':
            self.conv = nn.Conv2d(dim, dim, 3, 1, 1)
        elif resi_connection == '3conv':
            # to save parameters and memory
```

```
            self.conv = nn.Sequential(nn.Conv2d(dim, dim // 4, 3, 1, 1), nn.Le
akyReLU(negative_slope=0.2, inplace=True),
                                      nn.Conv2d(dim // 4, dim // 4, 1, 1, 0),
                                      nn.LeakyReLU(negative_slope=0.2, inplace
=True),
                                      nn.Conv2d(dim // 4, dim, 3, 1, 1))

        self.patch_embed = PatchEmbed(
            img_size=img_size, patch_size=patch_size, in_chans=0, embed_dim=di
m,
            norm_layer=None)

        self.patch_unembed = PatchUnEmbed(
            img_size=img_size, patch_size=patch_size, in_chans=0, embed_dim=di
m,
            norm_layer=None)

    def forward(self, x, x_size):
        # with torch.backends.cudnn.flags(enabled=False):
        return self.patch_embed(self.conv(self.patch_unembed(self.residual_gro
up(x, x_size), x_size))) + x

    def flops(self):
        flops = 0
        flops += self.residual_group.flops()
        H, W = self.input_resolution
        flops += H * W * self.dim * self.dim * 9
        flops += self.patch_embed.flops()
        flops += self.patch_unembed.flops()

        return flops


class PatchEmbed(nn.Module):
    def __init__(self, img_size=224, patch_size=4, in_chans=3, embed_dim=96, n
orm_layer=None):
        super().__init__()
        img_size = to_2tuple(img_size)
        patch_size = to_2tuple(patch_size)
        patches_resolution = [img_size[0] // patch_size[0], img_size[1] // pat
ch_size[1]]
        self.img_size = img_size
        self.patch_size = patch_size
        self.patches_resolution = patches_resolution
        self.num_patches = patches_resolution[0] * patches_resolution[1]

        self.in_chans = in_chans
        self.embed_dim = embed_dim

        if norm_layer is not None:
            self.norm = norm_layer(embed_dim)
        else:
            self.norm = None

    def forward(self, x):
```

```
        x = x.flatten(2).transpose(1, 2)  # B Ph*Pw C
        if self.norm is not None:
            x = self.norm(x)
        return x

    def flops(self):
        flops = 0
        H, W = self.img_size
        if self.norm is not None:
            flops += H * W * self.embed_dim
        return flops

class PatchUnEmbed(nn.Module):
    def __init__(self, img_size=224, patch_size=4, in_chans=3, embed_dim=96, n
orm_layer=None):
        super().__init__()
        img_size = to_2tuple(img_size)
        patch_size = to_2tuple(patch_size)
        patches_resolution = [img_size[0] // patch_size[0], img_size[1] // pat
ch_size[1]]
        self.img_size = img_size
        self.patch_size = patch_size
        self.patches_resolution = patches_resolution
        self.num_patches = patches_resolution[0] * patches_resolution[1]

        self.in_chans = in_chans
        self.embed_dim = embed_dim

    def forward(self, x, x_size):
        B, HW, C = x.shape
        x = x.transpose(1, 2).view(B, self.embed_dim, x_size[0], x_size[1])  #
 B Ph*Pw C
        return x

    def flops(self):
        flops = 0
        return flops

class Upsample(nn.Sequential):
    def __init__(self, scale, num_feat):
        m = []
        if (scale & (scale - 1)) == 0:  # scale = 2^n
            for _ in range(int(math.log(scale, 2))):
                m.append(nn.Conv2d(num_feat, 4 * num_feat, 3, 1, 1))
                m.append(nn.PixelShuffle(2))
        elif scale == 3:
            m.append(nn.Conv2d(num_feat, 9 * num_feat, 3, 1, 1))
            m.append(nn.PixelShuffle(3))
        else:
            raise ValueError(f'scale {scale} is not supported. ' 'Supported sc
ales: 2^n and 3.')
        super(Upsample, self).__init__(*m)


class UpsampleOneStep(nn.Sequential):
```

```
    def __init__(self, scale, num_feat, num_out_ch, input_resolution=None):
        self.num_feat = num_feat
        self.input_resolution = input_resolution
        m = []
        m.append(nn.Conv2d(num_feat, (scale ** 2) * num_out_ch, 3, 1, 1))
        m.append(nn.PixelShuffle(scale))
        super(UpsampleOneStep, self).__init__(*m)

    def flops(self):
        H, W = self.input_resolution
        flops = H * W * self.num_feat * 3 * 9
        return flops


class SwinIR(nn.Module):
    def __init__(self, img_size=64, patch_size=1, in_chans=3,
                 embed_dim=96, depths=[6, 6, 6, 6], num_heads=[6, 6, 6, 6],
                 window_size=7, mlp_ratio=4., qkv_bias=True, qk_scale=None,
                 drop_rate=0., attn_drop_rate=0., drop_path_rate=0.1,
                 norm_layer=nn.LayerNorm, ape=False, patch_norm=True,
                 use_checkpoint=False, upscale=2, img_range=1., upsampler='',
resi_connection='1conv',
                 **kwargs):
        super(SwinIR, self).__init__()
        num_in_ch = in_chans
        num_out_ch = in_chans
        num_feat = 64
        self.img_range = img_range
        if in_chans == 3:
            rgb_mean = (0.4488, 0.4371, 0.4040)
            self.mean = torch.Tensor(rgb_mean).view(1, 3, 1, 1)
        else:
            self.mean = torch.zeros(1, 1, 1, 1)
        self.upscale = upscale
        self.upsampler = upsampler
        self.window_size = window_size

        self.conv_first = nn.Conv2d(num_in_ch, embed_dim, 3, 1, 1)

        self.num_layers = len(depths)
        self.embed_dim = embed_dim
        self.ape = ape
        self.patch_norm = patch_norm
        self.num_features = embed_dim
        self.mlp_ratio = mlp_ratio

        # split image into non-overlapping patches
        self.patch_embed = PatchEmbed(
            img_size=img_size, patch_size=patch_size, in_chans=embed_dim, embe
d_dim=embed_dim,
            norm_layer=norm_layer if self.patch_norm else None)
        num_patches = self.patch_embed.num_patches
        patches_resolution = self.patch_embed.patches_resolution
        self.patches_resolution = patches_resolution
```

```
        # merge non-overlapping patches into image
        self.patch_unembed = PatchUnEmbed(
            img_size=img_size, patch_size=patch_size, in_chans=embed_dim, embe
d_dim=embed_dim,
            norm_layer=norm_layer if self.patch_norm else None)

        # absolute position embedding
        if self.ape:
            self.absolute_pos_embed = nn.Parameter(torch.zeros(1, num_patches,
 embed_dim))
            trunc_normal_(self.absolute_pos_embed, std=.02)

        self.pos_drop = nn.Dropout(p=drop_rate)

        # stochastic depth
        dpr = [x.item() for x in torch.linspace(0, drop_path_rate, sum(depth
s))]  # stochastic depth decay rule

        # build Residual Swin Transformer blocks (RSTB)
        self.layers = nn.ModuleList()
        for i_layer in range(self.num_layers):
            layer = RSTB(dim=embed_dim,
                         input_resolution=(patches_resolution[0],
                                           patches_resolution[1]),
                         depth=depths[i_layer],
                         num_heads=num_heads[i_layer],
                         window_size=window_size,
                         mlp_ratio=self.mlp_ratio,
                         qkv_bias=qkv_bias, qk_scale=qk_scale,
                         drop=drop_rate, attn_drop=attn_drop_rate,
                         drop_path=dpr[sum(depths[:i_layer]):sum(depths[:i_lay
er + 1])],  # no impact on SR results
                         norm_layer=norm_layer,
                         downsample=None,
                         use_checkpoint=use_checkpoint,
                         img_size=img_size,
                         patch_size=patch_size,
                         resi_connection=resi_connection

                         )
            self.layers.append(layer)
        self.norm = norm_layer(self.num_features)

        # build the last conv layer in deep feature extraction
        if resi_connection == '1conv':
            self.conv_after_body = nn.Conv2d(embed_dim, embed_dim, 3, 1, 1)
        elif resi_connection == '3conv':
            # to save parameters and memory
            self.conv_after_body = nn.Sequential(nn.Conv2d(embed_dim, embed_di
m // 4, 3, 1, 1),
                                                 nn.LeakyReLU(negative_slope=0.
2, inplace=True),
                                                 nn.Conv2d(embed_dim // 4, emb
ed_dim // 4, 1, 1, 0),
                                                 nn.LeakyReLU(negative_slope=0.
```

```
2, inplace=True),
                                                 nn.Conv2d(embed_dim // 4, emb
ed_dim, 3, 1, 1))

        if self.upsampler == 'pixelshuffle':
            # for classical SR
            self.conv_before_upsample = nn.Sequential(nn.Conv2d(embed_dim, num
_feat, 3, 1, 1),
                                                      nn.LeakyReLU(inplace=Tru
e))
            self.upsample = Upsample(upscale, num_feat)
            self.conv_last = nn.Conv2d(num_feat, num_out_ch, 3, 1, 1)
        elif self.upsampler == 'pixelshuffledirect':
            # for lightweight SR (to save parameters)
            self.upsample = UpsampleOneStep(upscale, embed_dim, num_out_ch,
                                            (patches_resolution[0], patches_re
solution[1]))
        elif self.upsampler == 'nearest+conv':
            # for real-world SR (less artifacts)
            assert self.upscale == 4, 'only support x4 now.'
            self.conv_before_upsample = nn.Sequential(nn.Conv2d(embed_dim, num
_feat, 3, 1, 1),
                                                      nn.LeakyReLU(inplace=Tru
e))
            self.conv_up1 = nn.Conv2d(num_feat, num_feat, 3, 1, 1)
            self.conv_up2 = nn.Conv2d(num_feat, num_feat, 3, 1, 1)
            self.conv_hr = nn.Conv2d(num_feat, num_feat, 3, 1, 1)
            self.conv_last = nn.Conv2d(num_feat, num_out_ch, 3, 1, 1)
            self.lrelu = nn.LeakyReLU(negative_slope=0.2, inplace=True)
        else:
            # for image denoising and JPEG compression artifact reduction
            self.conv_last = nn.Conv2d(embed_dim, num_out_ch, 3, 1, 1)

        self.apply(self._init_weights)

    def _init_weights(self, m):
        if isinstance(m, nn.Linear):
            trunc_normal_(m.weight, std=.02)
            if isinstance(m, nn.Linear) and m.bias is not None:
                nn.init.constant_(m.bias, 0)
        elif isinstance(m, nn.LayerNorm):
            nn.init.constant_(m.bias, 0)
            nn.init.constant_(m.weight, 1.0)

    @torch.jit.ignore
    def no_weight_decay(self):
        return {'absolute_pos_embed'}

    @torch.jit.ignore
    def no_weight_decay_keywords(self):
        return {'relative_position_bias_table'}

    def check_image_size(self, x):
        _, _, h, w = x.size()
        mod_pad_h = (self.window_size - h % self.window_size) % self.window_si
```

```
ze
        mod_pad_w = (self.window_size - w % self.window_size) % self.window_si
ze
        x = F.pad(x, (0, mod_pad_w, 0, mod_pad_h), 'reflect')
        return x

    def forward_features(self, x):
        x_size = (x.shape[2], x.shape[3])
        x = self.patch_embed(x)
        if self.ape:
            x = x + self.absolute_pos_embed
        x = self.pos_drop(x)

        for layer in self.layers:
            x = layer(x, x_size)

        x = self.norm(x)  # B L C
        x = self.patch_unembed(x, x_size)

        return x

    def forward(self, x):
        H, W = x.shape[2:]
        x = self.check_image_size(x)

        self.mean = self.mean.type_as(x)
        x = (x - self.mean) * self.img_range

        if self.upsampler == 'pixelshuffle':
            # for classical SR
            x = self.conv_first(x)
            x = self.conv_after_body(self.forward_features(x)) + x
            x = self.conv_before_upsample(x)
            x = self.conv_last(self.upsample(x))
        elif self.upsampler == 'pixelshuffledirect':
            # for lightweight SR
            x = self.conv_first(x)
            x = self.conv_after_body(self.forward_features(x)) + x
            x = self.upsample(x)
        elif self.upsampler == 'nearest+conv':
            # for real-world SR
            x = self.conv_first(x)
            x = self.conv_after_body(self.forward_features(x)) + x
            x = self.conv_before_upsample(x)
            x = self.lrelu(self.conv_up1(torch.nn.functional.interpolate(x, sc
ale_factor=2, mode='nearest')))
            x = self.lrelu(self.conv_up2(torch.nn.functional.interpolate(x, sc
ale_factor=2, mode='nearest')))
            x = self.conv_last(self.lrelu(self.conv_hr(x)))
        else:
            # for image denoising and JPEG compression artifact reduction
            x_first = self.conv_first(x)
            res = self.conv_after_body(self.forward_features(x_first)) + x_fir
st
            x = x + self.conv_last(res)
```

```
        x = x / self.img_range + self.mean

        return x[:, :, :H*self.upscale, :W*self.upscale]

    def flops(self):
        flops = 0
        H, W = self.patches_resolution
        flops += H * W * 3 * self.embed_dim * 9
        flops += self.patch_embed.flops()
        for i, layer in enumerate(self.layers):
            flops += layer.flops()
        flops += H * W * 3 * self.embed_dim * self.embed_dim
        flops += self.upsample.flops()
        return flops

if __name__ == '__main__':
    upscale = 4
    window_size = 8
    height = (1024 // upscale // window_size + 1) * window_size
    width = (720 // upscale // window_size + 1) * window_size
    model = SwinIR(upscale=2, img_size=(height, width),
                   window_size=window_size, img_range=1., depths=[6, 6, 6, 6],

                   embed_dim=60, num_heads=[6, 6, 6, 6], mlp_ratio=2, upsample
r='pixelshuffledirect')
    print(model)
    print(height, width, model.flops() / 1e9)

    x = torch.randn((1, 3, height, width))
    x = model(x)
    print(x.shape)

import os.path
import torch
import torch.nn.functional as F
from torch import nn as nn
import numpy as np
import math
from global_variable import MODEL_PATH
from .dcn import ModulatedDeformConvPack, modulated_deform_conv
from .network_swinir import RSTB
from .ridcp_utils import ResBlock, CombineQuantBlock
from .vgg_arch import VGGFeatureExtractor

WEIGHT_PATH = os.path.join(MODEL_PATH, 'RIDCP/weight_for_matching_dehazing_Fli
ckr.pth')


class DCNv2Pack(ModulatedDeformConvPack):
    def forward(self, x, feat):
        out = self.conv_offset(feat)
        o1, o2, mask = torch.chunk(out, 3, dim=1)
        offset = torch.cat((o1, o2), dim=1)
        mask = torch.sigmoid(mask)
```

```
        offset_absmean = torch.mean(torch.abs(offset))
        if offset_absmean > 50:
            print(f'Offset abs mean is {offset_absmean}, larger than 50.')

        return modulated_deform_conv(x, offset, mask, self.weight, self.bias,
self.stride, self.padding,
                                     self.dilation, self.groups, self.deformab
le_groups)


class VectorQuantizer(nn.Module):
    def __init__(self, n_e, e_dim, weight_path=WEIGHT_PATH, beta=0.25,
                 LQ_stage=False, use_weight=True, weight_alpha=1.0):
        super().__init__()
        self.n_e = int(n_e)
        self.e_dim = int(e_dim)
        self.LQ_stage = LQ_stage
        self.beta = beta
        self.use_weight = use_weight
        self.weight_alpha = weight_alpha
        if self.use_weight:
            self.weight = nn.Parameter(torch.load(weight_path))
            self.weight.requires_grad = False
        self.embedding = nn.Embedding(self.n_e, self.e_dim)

    def dist(self, x, y):
        if x.shape == y.shape:
            return (x - y) ** 2
        else:
            return torch.sum(x ** 2, dim=1, keepdim=True) +
                torch.sum(y ** 2, dim=1) - 2 *
                torch.matmul(x, y.t())

    def gram_loss(self, x, y):
        b, h, w, c = x.shape
        x = x.reshape(b, h * w, c)
        y = y.reshape(b, h * w, c)

        gmx = x.transpose(1, 2) @ x / (h * w)
        gmy = y.transpose(1, 2) @ y / (h * w)

        return (gmx - gmy).square().mean()

    def forward(self, z, gt_indices=None, current_iter=None, weight_alpha=Non
e):
        """
        Args:
            z: input features to be quantized, z (continuous) -> z_q (discrete)

               z.shape = (batch, channel, height, width)
            gt_indices: feature map of given indices, used for visualization.
        """
        # reshape z -> (batch, height, width, channel) and flatten
        z = z.permute(0, 2, 3, 1).contiguous()
```

```
        z_flattened = z.view(-1, self.e_dim)

        codebook = self.embedding.weight

        d = self.dist(z_flattened, codebook)
        if self.use_weight and self.LQ_stage:
            if weight_alpha is not None:
                self.weight_alpha = weight_alpha
            d = d * torch.exp(self.weight_alpha * self.weight)

        # find closest encodings
        min_encoding_indices = torch.argmin(d, dim=1).unsqueeze(1)
        min_encodings = torch.zeros(min_encoding_indices.shape[0], codebook.sh
ape[0]).to(z)
        min_encodings.scatter_(1, min_encoding_indices, 1)

        if gt_indices is not None:
            gt_indices = gt_indices.reshape(-1)

            gt_min_indices = gt_indices.reshape_as(min_encoding_indices)
            gt_min_onehot = torch.zeros(gt_min_indices.shape[0], codebook.shap
e[0]).to(z)
            gt_min_onehot.scatter_(1, gt_min_indices, 1)

            z_q_gt = torch.matmul(gt_min_onehot, codebook)
            z_q_gt = z_q_gt.view(z.shape)

        # get quantized latent vectors
        z_q = torch.matmul(min_encodings, codebook)
        z_q = z_q.view(z.shape)

        e_latent_loss = torch.mean((z_q.detach() - z) ** 2)
        q_latent_loss = torch.mean((z_q - z.detach()) ** 2)

        if self.LQ_stage and gt_indices is not None:
            # codebook_loss = self.dist(z_q, z_q_gt.detach()).mean() \
            # + self.beta * self.dist(z_q_gt.detach(), z)
            codebook_loss = self.beta * self.dist(z_q_gt.detach(), z)
            texture_loss = self.gram_loss(z, z_q_gt.detach())
            # print("codebook loss:", codebook_loss.mean(), "\ntexture_loss: ",
 texture_loss.mean())
            codebook_loss = codebook_loss + texture_loss
        else:
            codebook_loss = q_latent_loss + e_latent_loss * self.beta

        # preserve gradients
        z_q = z + (z_q - z).detach()

        # reshape back to match original input shape
        z_q = z_q.permute(0, 3, 1, 2).contiguous()

        return z_q, codebook_loss, min_encoding_indices.reshape(z_q.shape[0],
1, z_q.shape[2], z_q.shape[3])

    def get_codebook_entry(self, indices):
```

```
        b, _, h, w = indices.shape

        indices = indices.flatten().to(self.embedding.weight.device)
        min_encodings = torch.zeros(indices.shape[0], self.n_e).to(indices)
        min_encodings.scatter_(1, indices[:, None], 1)

        # get quantized latent vectors
        z_q = torch.matmul(min_encodings.float(), self.embedding.weight)
        z_q = z_q.view(b, h, w, -1).permute(0, 3, 1, 2).contiguous()
        return z_q


class SwinLayers(nn.Module):
    def __init__(self, input_resolution=(32, 32), embed_dim=256,
                 blk_depth=6,
                 num_heads=8,
                 window_size=8,
                 **kwargs):
        super().__init__()
        self.swin_blks = nn.ModuleList()
        for i in range(4):
            layer = RSTB(embed_dim, input_resolution, blk_depth, num_heads, wi
ndow_size, patch_size=1, **kwargs)
            self.swin_blks.append(layer)

    def forward(self, x):
        b, c, h, w = x.shape
        x = x.reshape(b, c, h * w).transpose(1, 2)
        for m in self.swin_blks:
            x = m(x, (h, w))
        x = x.transpose(1, 2).reshape(b, c, h, w)
        return x


class MultiScaleEncoder(nn.Module):
    def __init__(self,
                 in_channel,
                 max_depth,
                 input_res=256,
                 channel_query_dict=None,
                 norm_type='gn',
                 act_type='leakyrelu',
                 LQ_stage=True,
                 **swin_opts,
                 ):
        super().__init__()
        self.LQ_stage = LQ_stage
        ksz = 3

        self.in_conv = nn.Conv2d(in_channel, channel_query_dict[input_res], 4,
 padding=1)

        self.blocks = nn.ModuleList()
        self.up_blocks = nn.ModuleList()
        self.max_depth = max_depth
```

```
        res = input_res
        for i in range(max_depth):
            in_ch, out_ch = channel_query_dict[res], channel_query_dict[res //
 2]
            tmp_down_block = [
                nn.Conv2d(in_ch, out_ch, ksz, stride=2, padding=1),
                ResBlock(out_ch, out_ch, norm_type, act_type),
                ResBlock(out_ch, out_ch, norm_type, act_type),
            ]
            self.blocks.append(nn.Sequential(*tmp_down_block))
            res = res // 2

        if LQ_stage:
            self.blocks.append(SwinLayers(**swin_opts))

    def forward(self, input):
        # input.requires_grad = True
        x = self.in_conv(input)
        for idx, m in enumerate(self.blocks):
            with torch.backends.cudnn.flags(enabled=False):
                x = m(x)
        return x

class DecoderBlock(nn.Module):

    def __init__(self, in_channel, out_channel, norm_type='gn', act_type='leak
yrelu'):
        super().__init__()

        self.block = []
        self.block += [
            nn.Upsample(scale_factor=2),
            nn.Conv2d(in_channel, out_channel, 3, stride=1, padding=1),
            ResBlock(out_channel, out_channel, norm_type, act_type),
            ResBlock(out_channel, out_channel, norm_type, act_type),
        ]

        self.block = nn.Sequential(*self.block)

    def forward(self, input):
        return self.block(input)


class WarpBlock(nn.Module):
    def __init__(self, in_channel):
        super().__init__()
        self.offset = nn.Conv2d(in_channel * 2, in_channel, 3, stride=1, paddi
ng=1)
        self.dcn = DCNv2Pack(in_channel, in_channel, 3, padding=1, deformable_
groups=4)

    def forward(self, x_vq, x_residual):
        x_residual = self.offset(torch.cat([x_vq, x_residual], dim=1))
        feat_after_warp = self.dcn(x_vq, x_residual)
```

```
        return feat_after_warp


class MultiScaleDecoder(nn.Module):
    def __init__(self,
                 in_channel,
                 max_depth,
                 input_res=256,
                 channel_query_dict=None,
                 norm_type='gn',
                 act_type='leakyrelu',
                 only_residual=False,
                 use_warp=True
                 ):
        super().__init__()
        self.only_residual = only_residual
        self.use_warp = use_warp
        self.upsampler = nn.ModuleList()
        self.warp = nn.ModuleList()
        res = input_res // (2 ** max_depth)
        for i in range(max_depth):
            in_channel, out_channel = channel_query_dict[res], channel_query_d
ict[res * 2]
            self.upsampler.append(nn.Sequential(
                nn.Upsample(scale_factor=2),
                nn.Conv2d(in_channel, out_channel, 3, stride=1, padding=1),
                ResBlock(out_channel, out_channel, norm_type, act_type),
                ResBlock(out_channel, out_channel, norm_type, act_type),
            )
            )
            self.warp.append(WarpBlock(out_channel))
            res = res * 2

    def forward(self, input, code_decoder_output):
        x = input
        for idx, m in enumerate(self.upsampler):
            with torch.backends.cudnn.flags(enabled=False):
                if not self.only_residual:
                    x = m(x)
                    if self.use_warp:
                        x_vq = self.warp[idx](code_decoder_output[idx], x)
                        # print(idx, x.mean(), x_vq.mean())
                        x = x + x_vq * (x.mean() / x_vq.mean())
                    else:
                        x = x + code_decoder_output[idx]
                else:
                    x = m(x)
        # print()
        return x


class VQWeightDehazeNet(nn.Module):
    def __init__(self,
                 *,
                 in_channel=3,
```

```
            codebook_params=None,
            gt_resolution=256,
            LQ_stage=False,
            norm_type='gn',
            act_type='silu',
            use_quantize=True,
            use_semantic_loss=False,
            use_residual=True,
            only_residual=False,
            use_weight=False,
            use_warp=True,
            weight_alpha=1.0,
            **ignore_kwargs):
        super().__init__()

        codebook_params = np.array(codebook_params)

        self.codebook_scale = codebook_params[:, 0]
        codebook_emb_num = codebook_params[:, 1].astype(int)
        codebook_emb_dim = codebook_params[:, 2].astype(int)

        self.use_quantize = use_quantize
        self.in_channel = in_channel
        self.gt_res = gt_resolution
        self.LQ_stage = LQ_stage
        self.use_residual = use_residual
        self.only_residual = only_residual
        self.use_weight = use_weight
        self.use_warp = use_warp
        self.weight_alpha = weight_alpha

        channel_query_dict = {
            8: 256,
            16: 256,
            32: 256,
            64: 256,
            128: 128,
            256: 64,
            512: 32,
        }

        # build encoder
        self.max_depth = int(np.log2(gt_resolution // self.codebook_scale[0]))

        self.multiscale_encoder = MultiScaleEncoder(
            in_channel,
            self.max_depth,
            self.gt_res,
            channel_query_dict,
            norm_type, act_type, LQ_stage
        )
        if self.LQ_stage and self.use_residual:
            self.multiscale_decoder = MultiScaleDecoder(
                in_channel,
                self.max_depth,
```

```
                self.gt_res,
                channel_query_dict,
                norm_type, act_type, only_residual, use_warp=self.use_warp
            )

        # build decoder
        self.decoder_group = nn.ModuleList()
        for i in range(self.max_depth):
            res = gt_resolution // 2 ** self.max_depth * 2 ** i
            in_ch, out_ch = channel_query_dict[res], channel_query_dict[res *
2]
            self.decoder_group.append(DecoderBlock(in_ch, out_ch, norm_type, a
ct_type))

        self.out_conv = nn.Conv2d(out_ch, 3, 3, 1, 1)
        self.residual_conv = nn.Conv2d(out_ch, 3, 3, 1, 1)

        # build multi-scale vector quantizers
        self.quantize_group = nn.ModuleList()
        self.before_quant_group = nn.ModuleList()
        self.after_quant_group = nn.ModuleList()

        for scale in range(0, codebook_params.shape[0]):
            quantize = VectorQuantizer(
                codebook_emb_num[scale],
                codebook_emb_dim[scale],
                LQ_stage=self.LQ_stage,
                use_weight=self.use_weight,
                weight_alpha=self.weight_alpha
            )
            self.quantize_group.append(quantize)

            scale_in_ch = channel_query_dict[self.codebook_scale[scale]]
            if scale == 0:
                quant_conv_in_ch = scale_in_ch
                comb_quant_in_ch1 = codebook_emb_dim[scale]
                comb_quant_in_ch2 = 0
            else:
                quant_conv_in_ch = scale_in_ch * 2
                comb_quant_in_ch1 = codebook_emb_dim[scale - 1]
                comb_quant_in_ch2 = codebook_emb_dim[scale]

            self.before_quant_group.append(nn.Conv2d(quant_conv_in_ch, codeboo
k_emb_dim[scale], 1))
            self.after_quant_group.append(CombineQuantBlock(comb_quant_in_ch1,
 comb_quant_in_ch2, scale_in_ch))

        # semantic loss for HQ pretrain stage
        self.use_semantic_loss = use_semantic_loss
        if use_semantic_loss:
            self.conv_semantic = nn.Sequential(
                nn.Conv2d(512, 512, 1, 1, 0),
                nn.ReLU(),
            )
            self.vgg_feat_layer = 'relu4_4'
```

```
            self.vgg_feat_extractor = VGGFeatureExtractor([self.vgg_feat_laye
r])

    def encode_and_decode(self, input, gt_indices=None, current_iter=None, wei
ght_alpha=None):
        # if self.training:
        #     for p in self.multiscale_encoder.parameters():
        #         p.requires_grad = True
        enc_feats = self.multiscale_encoder(input)

        if self.use_semantic_loss:
            with torch.no_grad():
                vgg_feat = self.vgg_feat_extractor(input)[self.vgg_feat_layer]


        codebook_loss_list = []
        indices_list = []
        semantic_loss_list = []
        code_decoder_output = []

        quant_idx = 0
        prev_dec_feat = None
        prev_quant_feat = None
        out_img = None
        out_img_residual = None
        x = enc_feats
        for i in range(self.max_depth):
            cur_res = self.gt_res // 2 ** self.max_depth * 2 ** i
            if cur_res in self.codebook_scale:  # needs to perform quantize
                if prev_dec_feat is not None:
                    before_quant_feat = torch.cat((x, prev_dec_feat), dim=1)
                else:
                    before_quant_feat = x
                feat_to_quant = self.before_quant_group[quant_idx](before_quan
t_feat)

                if weight_alpha is not None:
                    self.weight_alpha = weight_alpha
                if gt_indices is not None:
                    z_quant, codebook_loss, indices = self.quantize_group[quan
t_idx](feat_to_quant,

       gt_indices[quant_idx],

       weight_alpha=self.weight_alpha)
                else:
                    z_quant, codebook_loss, indices = self.quantize_group[quan
t_idx](feat_to_quant,

       weight_alpha=self.weight_alpha)
                if self.use_semantic_loss:
                    semantic_z_quant = self.conv_semantic(z_quant)
                    semantic_loss = F.mse_loss(semantic_z_quant, vgg_feat)
                    semantic_loss_list.append(semantic_loss)
                if not self.use_quantize:
```

```
                    z_quant = feat_to_quant
                after_quant_feat = self.after_quant_group[quant_idx](z_quant,
prev_quant_feat)
                codebook_loss_list.append(codebook_loss)
                indices_list.append(indices)
                quant_idx += 1
                prev_quant_feat = z_quant
                x = after_quant_feat
            x = self.decoder_group[i](x)
            code_decoder_output.append(x)
            prev_dec_feat = x
        out_img = self.out_conv(x)
        if self.LQ_stage and self.use_residual:
            if self.only_residual:
                residual_feature = self.multiscale_decoder(enc_feats, code_dec
oder_output)
            else:
                residual_feature = self.multiscale_decoder(enc_feats.detach(),
 code_decoder_output)
            out_img_residual = self.residual_conv(residual_feature)
        if len(codebook_loss_list) > 0:
            codebook_loss = sum(codebook_loss_list)
        else:
            codebook_loss = 0
        semantic_loss = sum(semantic_loss_list) if len(semantic_loss_list) els
e codebook_loss * 0
        return out_img, out_img_residual, codebook_loss, semantic_loss, feat_t
o_quant, z_quant, indices_list

    def decode_indices(self, indices):
        assert len(indices.shape) == 4, f'shape of indices must be (b, 1, h,
w), but got {indices.shape}'
        z_quant = self.quantize_group[0].get_codebook_entry(indices)
        x = self.after_quant_group[0](z_quant)
        for m in self.decoder_group:
            x = m(x)
        out_img = self.out_conv(x)
        return out_img

    @torch.no_grad()
    def test_tile(self, input, tile_size=240, tile_pad=16):
        batch, channel, height, width = input.shape
        output_height = height
        output_width = width
        output_shape = (batch, channel, output_height, output_width)
        # start with black image
        output = input.new_zeros(output_shape)
        tiles_x = math.ceil(width / tile_size)
        tiles_y = math.ceil(height / tile_size)
        # loop over all tiles
        for y in range(tiles_y):
            for x in range(tiles_x):
                # extract tile from input image
                ofs_x = x * tile_size
                ofs_y = y * tile_size
```

```
                # input tile area on total image
                input_start_x = ofs_x
                input_end_x = min(ofs_x + tile_size, width)
                input_start_y = ofs_y
                input_end_y = min(ofs_y + tile_size, height)
                # input tile area on total image with padding
                input_start_x_pad = max(input_start_x - tile_pad, 0)
                input_end_x_pad = min(input_end_x + tile_pad, width)
                input_start_y_pad = max(input_start_y - tile_pad, 0)
                input_end_y_pad = min(input_end_y + tile_pad, height)
                # input tile dimensions
                input_tile_width = input_end_x - input_start_x
                input_tile_height = input_end_y - input_start_y
                tile_idx = y * tiles_x + x + 1
                input_tile = input[:, :, input_start_y_pad:input_end_y_pad, in
put_start_x_pad:input_end_x_pad]
                # upscale tile
                output_tile = self.test(input_tile)
                # output tile area on total image
                output_start_x = input_start_x
                output_end_x = input_end_x
                output_start_y = input_start_y
                output_end_y = input_end_y
                # output tile area without padding
                output_start_x_tile = (input_start_x - input_start_x_pad)
                output_end_x_tile = output_start_x_tile + input_tile_width
                output_start_y_tile = (input_start_y - input_start_y_pad)
                output_end_y_tile = output_start_y_tile + input_tile_height
                # put tile into output image
                output[:, :, output_start_y:output_end_y,
                output_start_x:output_end_x] = output_tile[:, :, output_start_
y_tile:output_end_y_tile,
                                                           output_start_x_tile:output_end_
x_tile]
        return output

    @torch.no_grad()
    def test(self, input, weight_alpha=None):
        org_use_semantic_loss = self.use_semantic_loss
        self.use_semantic_loss = False

        # padding to multiple of window_size * 8
        wsz = 32
        _, _, h_old, w_old = input.shape
        h_pad = (h_old // wsz + 1) * wsz - h_old
        w_pad = (w_old // wsz + 1) * wsz - w_old
        input = torch.cat([input, torch.flip(input, [2])], 2)[:, :, :h_old + h
_pad, :]
        input = torch.cat([input, torch.flip(input, [3])], 3)[:, :, :, :w_old
+ w_pad]
        output_vq, output, _, _, _, after_quant, index = self.encode_and_decod
e(input, None, None,

  weight_alpha=weight_alpha)
        if output is not None:
```

```
            output = output[..., :h_old, :w_old]
        if output_vq is not None:
            output_vq = output_vq[..., :h_old, :w_old]
        self.use_semantic_loss = org_use_semantic_loss
        return output, index

    def forward(self, input, gt_indices=None, weight_alpha=None):
        if gt_indices is not None:
            # in LQ training stage, need to pass GT indices for supervise.
            dec, dec_residual, codebook_loss, semantic_loss, quant_before_feat
ure, quant_after_feature, indices = self.encode_and_decode(
                input, gt_indices, weight_alpha=weight_alpha)
        else:
            # in HQ stage, or LQ test stage, no GT indices needed.
            dec, dec_residual, codebook_loss, semantic_loss, quant_before_feat
ure, quant_after_feature, indices = self.encode_and_decode(
                input, weight_alpha=weight_alpha)
        return dec, dec_residual, codebook_loss, semantic_loss, quant_before_f
eature, quant_after_feature, indices
```