水印算法原理：

1.水印生成原理：

密钥哈希：将用户输入的 secret_key 通过 MD5 哈希转换为 32 位整数，作为伪随机数生成器（PRNG）的种子：

self.rng=np.random.RandomState(int(hashlib.md5(secret_key.encode()).hexdigest(), 16))

位置选择：避开图像低频区域（能量集中，修改易影响视觉效果），选择高频区域（i>5 且 j>5）作为嵌入位置；

序列生成：在选定位置生成取值为 1 或-1 的二进制伪随机序列（代表水印比特），形成二维水印矩阵。

数学表示如下：

$$W(i,j)=\begin{cases}1 & \text{若 } rng(i,j)>0.5\\ -1 & \text{否则}\end{cases} \quad (i,j)\in \text{有效位置}$$

其中 rng(i,j)是[0,1)区间的均匀随机数

2.DCT 域水印算法：

DCT 变换原理：

$$F(u,v)=\\ \alpha(u)\alpha(v)\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}f(i,j)\cos\left(\frac{(2i+1)u\pi}{2N}\right)\cos\left(\frac{(2j+1)v\pi}{2N}\right)$$

逆 DCT 变换用于从频率域恢复空间域图像：

$$f(i,j)=\\ \sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)F(u,v)\cos\left(\frac{(2i+1)u\pi}{2N}\right)\cos\left(\frac{(2j+1)v\pi}{2N}\right)$$

嵌入与提取流程：

a.嵌入流程：

图像分块：将图像划分为 8×8 像素块（符合 JPEG 压缩标准）；

DCT 变换：对每个块执行 DCT，得到频率域系数矩阵；

系数修改：选择中频系数，嵌入水印：

$$F'(u,v) = F(u,v) + \alpha \cdot W(i,j)$$

IDCT 变换：对修改后的系数执行逆 DCT，恢复空间域图像块；

拼接输出：合并所有块，得到含水印图像。

b.提取流程：

分块与 DCT：对含水印图像分块并执行 DCT；

系数差值计算：

非盲提取（已知原始图像）：计算含水印图像与原始图像的 DCT 系数差，除以强度来恢复水印：

$$\hat{W}(i,j) = \frac{F'_w(u,v) - F_o(u,v)}{\alpha}$$

盲提取（未知原始图像）：直接从含水印图像的 DCT 系数中提取修改量，与水印模板匹配；

阈值化：将提取的连续值转换为 1 或-1 的二进制序列。

3.LSB 水印算法：

图像像素通常用 8 位整数表示（0~255），其最低位对视觉影响可忽略。LSB 嵌入通过修改最低位承载水印比特：

水印比特为 1 时，设置像素最低位为 1：

$$p' = (p\&0xFE)|1$$

水印比特为-1（代表 0）时，设置像素最低位为 0：

$$p' = p\&0xFE$$

数学表示如下：

$$p'(i,j) = p(i,j) - (p(i,j) \mod 2) + b(i,j)$$

提取原理：提取时直接读取像素的最低位：

$$\hat{b}(i,j) = p'(i,j) \mod 2$$

4.归一化相关系数（NCC）：

为量化提取水印与原始水印的一致性，采用归一化相关系数（NCC）：

$$NCC(W,\hat{W}) = \frac{\sum_{(i,j)\in\Omega}(W(i,j)-\mu_W)(\hat{W}(i,j)-\mu_{\hat{W}})}{\sqrt{\sum_{(i,j)\in\Omega}(W(i,j)-\mu_W)^2}\cdot\sqrt{\sum_{(i,j)\in\Omega}(\hat{W}(i,j)-\mu_{\hat{W}})^2}}$$

NCC 取值范围为[-1,1]，越接近 1 表示相似度越高

重点部分代码实现：

1.核心类：

WatermarkDetector 类采用模块化设计，核心方法分为 5 类：初始化与水印生成、嵌入算法、提取算法、攻击模拟、鲁棒性测试，具体结构如下：

a. 初始化:__init__、_generate_watermark

初始化密钥、PRNG，生成伪随机水印

b.嵌入算法：embed_watermark、_embed_dct、_embed_lsb

实现 DCT 与 LSB 两种嵌入逻辑

c. 提取算法:extract_watermark、_extract_dct、_extract_lsb`

对应两种嵌入算法的提取逻辑

d. 攻击模拟:apply_attack

模拟翻转、旋转、噪声等 10+种攻击

e. 评估与可视化 :calculate_similarity、robustness_test、_visualize_results

计算 NCC，测试鲁棒性，可视化结果

2. 水印生成（_generate_watermark）

```
def _generate_watermark(self, shape: Tuple[int, int], bits: int = 256) -> np.ndarray:
    watermark = np.zeros(shape, dtype=np.float32)
    # 筛选有效位置（避开低频区域）
    valid_positions = []
    for i in range(shape[0]):
        for j in range(shape[1]):
            if i > 5 and j > 5:
            # 低频区域通常在(i≤5,j≤5)，能量集中
                valid_positions.append((i, j))
    # 随机选择 bits 个位置，生成±1 序列
    selected = self.rng.choice(len(valid_positions), bits, replace=False)
    for idx in selected:
```

```
        i, j = valid_positions[idx]
        watermark[i, j] = 1 if self.rng.rand() > 0.5 else -
1
    return watermark
```

3. DCT 域嵌入（_embed_dct）

```
   def _embed_dct(self, img: np.ndarray, bits: int) ->
np.ndarray:
    h, w = img.shape
    # 调整尺寸为 8 的倍数（DCT 块大小）
    h = h - (h % 8)
    w = w - (w % 8)
    img = img[:h, :w].astype(np.float32)
    watermark = self._generate_watermark((h, w), bits)
    watermarked = np.copy(img)
    # 分块处理
    for i in range(0, h, 8):
        for j in range(0, w, 8):
            block = img[i:i+8, j:j+8]
            dct_block = dct(dct(block, axis=0, norm='ortho'),
axis=1, norm='ortho')  # 2D DCT
            # 嵌入水印（(3,3)位置）
            dct_block[3, 3] += self.alpha * watermark[i, j]
```

```
            # 逆DCT
            idct_block = idct(idct(dct_block, axis=1,
norm='ortho'), axis=0, norm='ortho')
            watermarked[i:i+8, j:j+8] = idct_block

    return np.clip(watermarked, 0, 255).astype(np.uint8)
    # 确保像素值在有效范围
```

4. LSB嵌入（_embed_lsb）

```
    def _embed_lsb(self, img: np.ndarray, bits: int) ->
np.ndarray:
    h, w = img.shape
    watermark = self._generate_watermark((h, w), bits)
    # 筛选水印位置并打乱（增强安全性）
    positions = [(i,j) for i in range(h) for j in range(w)
if watermark[i,j] != 0]
    self.rng.shuffle(positions)
    positions = positions[:bits]
    # 取前bits个位置
    watermarked = np.copy(img)
    for i, j in positions:
        bit = 1 if watermark[i, j] > 0 else 0
        watermarked[i, j] = (watermarked[i, j] & 0xFE) | bit
```

修改最低位

return watermarked.astype(np.uint8)

随机打乱嵌入位置，避免水印分布规律被破解；

通过位运算（`& 0xFE`保留高 7 位，bit 设置最低位）高效修改像素，计算复杂度低。

5.攻击模拟（apply_attack）

a.翻转（flip）:水平/垂直翻转图像（Image.FLIP_LEFT_RIGHT）

水印位置镜像，需提取时同步翻转模板

b.旋转（rotate）:旋转指定角度（Image.rotate）

水印几何变形，严重时可能超出图像范围

c.裁剪（crop）:裁剪边缘后缩放回原尺寸

丢失边缘水印信息，影响提取完整性

d.噪声（noise）:添加高斯噪声（np.random.normal）

干扰像素值，对 LSB 水印破坏性强

e.JPEG 压缩（jpeg）:降低保存质量（quality 参数）

丢失高频信息，对 DCT 高频系数影响大

6.鲁棒性测试（robustness_test）

```
def robustness_test(self, original_img: str, watermarked_img: str, output_dir: str = "attack_results",
                    method: str = 'dct', bits: int = 256) -> dict:
    # 生成原始水印与无攻击时的提取结果
```

```
    test_img = cv2.imread(original_img, cv2.IMREAD_GRAYSCALE)
    original_watermark = self._generate_watermark(test_img.shape, bits)
    extracted_original = self.extract_watermark(watermarked_img, original_img, bits, method)
    base_similarity = self.calculate_similarity(original_watermark, extracted_original)

    # 遍历攻击列表，计算每种攻击后的相似度
    attacks = [('flip', {'direction': 'horizontal'}), ('rotate', {'angle': 30}), ...]
    results = {"original": base_similarity}
    for attack_type, params in attacks:
        # 应用攻击并提取水印
        attack_path = os.path.join(output_dir, f"{attack_type}_result.png")
        self.apply_attack(watermarked_img, attack_type, attack_path, **params)
        extracted = self.extract_watermark(attack_path, original_img, bits, method)
```

```
            results[f"{attack_type}"]                                    =
self.calculate_similarity(original_watermark, extracted)

        # 可视化结果
        self._visualize_results(...)
        return results
```

通过可视化对比原始图像、含水印图像、水印模板及提取结果，直观展示性能。

算法优缺点总结：

1.DCT 域水印

鲁棒性强，抗压缩/噪声能力好，实现复杂，计算量

2.LSB 水印

实现简单，计算高效，不可见性极佳，鲁棒性弱，易被移除

参考文献：

1. Cox I J, Kilian J, Leighton F T, et al. "Secure spread spectrum watermarking for multimedia." *IEEE Transactions on Image Processing*, 1997.

2. Wolfgang R, Podilchuk C I, Delp E J. "Perceptual watermarks for digital images and video." *Proceedings of the IEEE*, 1999.

3. 潘正祥, 孙星明. 《数字水印技术及应用》. 国防工业出版社, 2009.

4. NIST. "Guidelines for Media Forensics". NIST SP 1800-16, 2018.