Artificial Intelligence and Robotics Research 人工智能与机器人研究, 2026, 15(1), 65-77
Published Online January 2026 in Hans. https://www.hanspub.org/journal/airr
https://doi.org/10.12677/airr.2026.151008

Hans汉斯

C/Python混合编程下高质量信息隐藏研究与实现

陈子超，丁海洋*，陈帅龙

北京印刷学院信息工程学院，北京

收稿日期：2025年10月29日；录用日期：2025年12月23日；发布日期：2026年1月5日

摘　要

C++在处理信息隐藏算法时具有高效的内存管理和快速执行速度，而Python则在数据处理和脚本编写方面表现出极高的易用性和灵活性。为了融合二者的优势，平衡计算性能与开发效率，本文提出了一种基于Ctypes技术的C/Python混合编程下高质量信息隐藏方案。该方案的核心贡献在于实现方式的创新：首先利用C++语言设计并实现了基于离散傅里叶变换(DFT)的信息隐藏算法，并引入基于人类视觉系统(HVS)模型的自适应嵌入策略增强其性能，然后将其编译为高效率、可移植的动态链接库(DLL)；随后，借助Ctypes技术将DLL文件与Python程序深度融合，成功实现了在彩色图像上信息隐藏的精准嵌入与高效提取。本方法不仅验证了将C++编写的高性能信息隐藏算法移植到Python平台的可行性，而且在嵌入的鲁棒性、提取的准确性和算法运行效率等方面均展现出较好性能。这为信息隐藏技术在不同编程环境中的灵活应用提供了一种高效、可靠的解决方案。

关键词

变换域，信息隐藏，DFT，Ctypes，人类视觉系统

Research and Implementation of High Quality Information Hiding in C/Python Mixed Programming

Zichao Chen, Haiyang Ding*, Shuailong Chen

School of Information Engineering, Beijing Institute of Graphic Communication, Beijing

Received: October 29, 2025; accepted: December 23, 2025; published: January 5, 2026

*通讯作者。

文章引用: 陈子超, 丁海洋, 陈帅龙. C/Python 混合编程下高质量信息隐藏研究与实现[J]. 人工智能与机器人研究, 2026, 15(1): 65-77. DOI: 10.12677/airr.2026.151008

Abstract

Processing information hiding algorithms, C++ has efficient memory management and fast execution speed, while Python shows high ease of use and flexibility in data processing and scripting. In order to integrate the advantages of the two and balance the computing performance and development efficiency, this paper proposes a high-quality information hiding scheme based on Ctypes technology in C/Python mixed programming. The core contribution of the scheme lies in the innovation of the implementation method: firstly, the information hiding algorithm based on discrete Fourier transform (DFT) is designed and implemented by using C++ language, and the adaptive embedding strategy based on human visual system (HVS) model is introduced to enhance its performance, and then it is compiled into an efficient and portable dynamic link library (DLL); Subsequently, with the help of ctypes technology, DLL files and python programs are deeply integrated, and the accurate embedding and efficient extraction of information hiding on color images are successfully realized. This method not only verifies the feasibility of transplanting the high-performance information hiding algorithm written in C++ to the Python platform, but also shows good performance in the robustness of embedding, the accuracy of extraction and the efficiency of the algorithm. This provides an efficient and reliable solution for the flexible application of information hiding technology in different programming environments.

Keywords

Transform Domain, Information Hiding, DFT, Ctypes, Human Visual System

1. 引言

随着数字时代的蓬勃发展，图像、视频等数字媒体的创作、分发和消费已变得前所未有的便捷。然而，这种便利性也给版权保护带来了严峻挑战，未经授权的复制和传播行为日益猖獗。因此，研究和应用稳健、安全的数字内容保护技术显得至关重要。信息隐藏技术作为其中的关键分支，通过嵌入秘密信息到数字化宿主信息中，以实现隐蔽传输的技术[1] [2]。其核心特征在于利用多媒体信号的冗余性和人类感知的不敏感性，使隐藏信息难以察觉。

图像信息隐藏算法根据其核心原理的差异，可以划分为空间域信息隐藏算法和变换域信息隐藏算法两大类。在空间域信息隐藏算法中，其实现方式主要是通过调整或修改载体图像的像素值来嵌入隐藏信息[3]。这种方法的复杂度相对较低，较为直观易懂。然而，其嵌入信息与图像内容耦合度高，导致鲁棒性较差，难以抵抗裁剪、压缩等常见攻击[4]-[6]。变换域信息隐藏算法的核心差异在于，它首先会将原始的载体图像从空间域转换到频域，通过这个过程对图像信息的不同频率成分进行分解。随后，隐藏信息被嵌入到选定的频域位置。尽管与空间域信息隐藏算法相比，变换域中嵌入的隐藏信息量可能较少，但它显著增强了对各种攻击的鲁棒性，即抵抗外界干扰或篡改的能力更强。这种方法在不可见性和鲁棒性之间取得了更好的平衡，因此成为当前主流的研究方向。在众多变换域技术中，离散余弦变换(DCT)和离散小波变换(DWT)应用广泛，例如文献[7]中提出了在视频信息隐藏中的 3 种离散余弦变换(Discrete Cosine Transform, DCT)算法和 1 种傅里叶变换(DFT)算法，这些算法也可被改编并应用于图像信息隐藏。文献[8]

[9] 中提出了 DWT 变换，DWT 变换虽在图像压缩和信号去噪领域表现出色，但其需要更多的存储空间去存储多层次分解的小波系数，环境受限，且基于小波特性的不同场景，需要更多的专业性知识。文献[10]提出了基于 DCT 和 DWT 信息隐藏的研究，将 DCT 和 DWT 的过程相结合。文献[11] [12]提出了基于 DFT 的图像信息隐藏算法的具体过程，然而，从理论算法到实际应用之间存在着一道显著的“工程鸿沟”。一方面，如 DFT 这类计算密集型算法，为了追求极致的运行效率，通常需要使用 C/C++等底层语言进行实现。文献[13]实现了一种离散傅里叶的信息隐藏方式，文献[14]在时间域上使用了信息隐藏算法中的最低有效位(Least Significant Bit, LSB)算法，该算法是在字节的最低有效位(LSB)进行信息隐藏，嵌入的信息隐藏图像肉眼甚至难以完全发现，但安全性有待提升。文献[15]调查了不同变换域信息隐藏的特点。现代的数据处理、应用开发和科学研究越来越多地依赖于 Python 生态系统，因为它拥有简洁的语法和海量的第三方库，能够极大地提升开发效率[16]。如何在保持核心算法高性能的同时，又能享受上层应用开发的便捷性，成为了一个亟待解决的技术难题。尽管已有研究尝试通过 Java 本地接口(JNI)等方式在特定平台解决类似问题[17] [18]，但在 Python 环境中，系统性地利用 Ctypes 库构建高性能、安全且自适应的信息隐藏应用方案[19] [20]，仍有待深入探索。

为了解决上述问题，本文提出了一种创新的工程实现方案：将一个结合人类视觉系统(HVS)模型的自适应 DFT 信息隐藏算法用 C++实现，并编译为动态链接库(DLL)，再利用 Python 的 Ctypes 库进行调用。HVS 模型的引入，使得算法能根据图像内容的复杂性自适应地调整嵌入强度，在纹理丰富的区域嵌入更强的鲁棒性的信息隐藏，在平滑区域则嵌入更弱的信息隐藏，从而在保证高不可见性的前提下，最大化信息隐藏的鲁棒性。本方案不仅解决了 C++/Python 混合编程在信息隐藏领域的应用问题，也通过引入 HVS 模型提升了信息隐藏算法的智能化水平。

2. 相关介绍

2.1. Ctypes 介绍

Ctypes 是一个 Python 库，它为 Python 与外部代码(尤其是 C 语言编写的代码)的交互提供了强大的支持。该库提供了与 C 语言兼容的数据类型，允许 Python 程序调用动态链接库(DLL)或共享库(.so 文件)中定义的函数。通过 Ctypes 模块，开发者可以以纯 Python 的方式封装这些外部库，从而在 Python 程序中方便地利用这些库的功能。利用 Ctypes 技术调用已编写好的 C++信息隐藏算法程序，可以充分发挥 C 语言贴近底层、运行效率高的优势，节省大量编程时间，并快速地在 Python 中进行调用，降低在 Python 上实现信息隐藏算法的时间成本。

2.1.1. Ctypes 流程

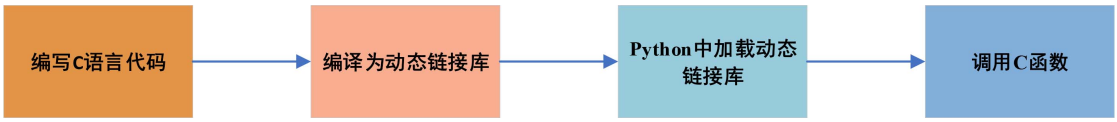

Figure 1. Ctypes flow chart
图 1. Ctypes 流程图

该流程如图 1 所示，主要包括：

1) 编写 C/C++源代码：此阶段专注于核心算法的逻辑实现，例如本文中的 DFT 信息隐藏嵌入与提取函数。开发者可以利用 C++的全部性能优势，包括指针操作、手动内存管理和高效的数学运算库，而不必担心上层应用的复杂性。

2) 编译为动态链接库：使用 C++编译器(如 Visual Studio 的 cl.exe 或 Linux 下的 g++)将源代码编译成平台兼容的动态库文件。这一步将高性能代码模块化，创建了一个独立、可重用的组件，为后续的调用做好了准备。

3) Python 中加载动态库：在 Python 脚本中，使用 ctypes.CDLL()或 ctypes.WinDLL()加载编译好的库文件。这个过程非常轻量级，Python 解释器会将动态库映射到其内存空间，使得库中的函数可以被访问。

4) 调用 C 函数：完成上述步骤后，C++函数就可以像一个普通的 Python 函数一样被调用。Ctypes 在后台处理了数据类型的转换和函数调用的细节，使得整个过程对于 Python 开发者来说是透明的。

5) 参数类型和返回值类型的注意事项：C 语言函数的参数类型和 Python 中的数据类型是不同的，这是确保两种语言间数据正确交互的关键步骤。需要在 Python 中明确指定 C 函数的参数类型(.argtypes)和返回值类型(.restype)。如果不这样做，Ctypes 会进行默认的类型猜测，这可能导致数据被错误解释，甚至引发内存访问错误或程序崩溃。

2.1.2. Ctypes 流程举例

以下案例展示了如何在 Python 中通过 Ctypes 调用 C++动态链接库：

1) 编写 C 语言代码：假设有一个简单的 C 文件 example.c，其中包含一个用于加法的函数：

```
    #include <stdio.h>
int add(int a，int b){
        return a + b;
}
```

2) 将 C 代码编译为 DLL 或.so 文件：假设有一个简单的 C 文件 example.c，需要将 C 语言代码编译成动态链接库(DLL 文件或.so 文件)。这一步骤的具体操作取决于你的操作系统。在 Windows 系统下，可以使用 Visual Studio 等工具将 C 代码编译成 DLL 文件。

3) 在 Python 中加载动态链接库：在 Python 中加载动态链接库需要导入 Ctypes 模块。

4) 在 Python 中加载动态链接库和参数类型和返回值类型的指定：然后编译这个 C 文件生成一个动态链接库(DLL 文件或.so 文件，取决于操作系统)。接下来，我们可以使用 Ctypes 模块在 Python 系统中调用这个函数，代码如下：

```
import ctypes
#加载动态链接库
lib = ctypes.CDLL(./example.so’)
#调用加法函数
result = lib.add(4.5)
print(result) #输出:9
```

通过上述代码，我们可以看到 Ctypes 技术使得 Python 程序能够轻松地调用 C 语言编写的动态链接库中的函数，实现了 Python 与 C 语言的无缝交互。

2.2. 信息隐藏方法

2.2.1. 信息嵌入

本文信息隐藏算法采用文献[2]中的 DFT 的信息隐藏算法。对于处理后的频域数据，我们将对位于正中心的部分以及与其等比例、占据频域面积 1/4 的区域进行分块处理。具体的分块策略是，将这些区域分割成多个 4*4 大小的块。对一个块中，在处理数据时，我们仅针对两个特定的部分进行操作，这两部分

的数据分布如图 2 所示。在图中，以块字符“X”标识的部分为第 1 部分，而以字符“O”标识的部分为第 2 部分。

将第 1 部分的数据能量总和定义为 e_1，第 2 部分的能量总和定义为 e_2。对于信息隐藏的强度，我们使用值 k 来表示。在嵌入数据时，需要确保这些数据满足特定的条件。

$$\begin{cases} e_1 - e_2 > k, \text{嵌入数据为0} \\ e_2 - e_1 > k, \text{嵌入数据为1} \end{cases} \tag{1}$$

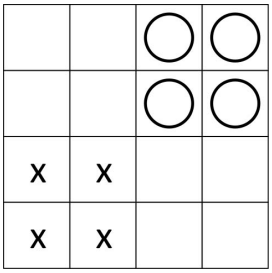

Figure 2. Data distribution
图 2. 数据分布图

当嵌入比特为 0 时，若原始频域系数不满足条件，则需按公式(2)进行调整：

$$\begin{cases} \text{delta} = \left(k - e_1 + e_2\right)/8 \\ P_{1i} = P_{1i} + \text{delta}; i = 1,2,3,4 \\ P_{2i} = P_{2i} - \text{delta}; i = 1,2,3,4 \end{cases} \tag{2}$$

若生成的数据为 1 时，有公式(3)：

$$\begin{cases} \text{delta} = \left(k - e_1 + e_2\right)/8 \\ P_{1i} = P_{1i} + \text{delta}; i = 1,2,3,4 \\ P_{2i} = P_{2i} - \text{delta}; i = 1,2,3,4 \end{cases} \tag{3}$$

公式中定义的 p_1 代表第 1 部分的数据，这些数据被安排在 4*4 矩阵的左下角四个单元格内，元素按照从左到右、从上到下的顺序进行排列。我们使用 p_{1i} 来表示 p_1 中的第 i 个元素。另一方面，p_2 代表第 2 部分的数据，它分布在 4*4 矩阵的右上角四个单元格中，并且元素的排列顺序与 p_1 相同。p_{2i} 则代表第二部分中从上到下的第 i 个元素。另外，delta 是一个增量值，用于调整两组元素中的能量大小的数据。信息隐藏的嵌入容量 capacity 的计算公式如公式(4)为：

$$\text{capacity} = \left|\text{width}/4\right| \times \left|\text{heighy}/8\right| \tag{4}$$

式中：capacity 的单位为 bit；而图片的宽度则被称为 width，高度则被称为 height。

2.2.2. 信息提取

信息隐藏提取部分和信息隐藏嵌入类似，不同于信息隐藏嵌入时使用的公式，信息隐藏提取的公式(5)为：

$$\begin{cases} e_1 > e_2, \text{嵌入数据为0} \\ e_2 > e_1, \text{嵌入数据为1} \end{cases} \tag{5}$$

信息隐藏提取的结果是原始数据的二进制形式即为 0 或 1 数据，需要通过 ASCII 转二进制的逆变换恢复出原始数据。

2.3. 结合人类视觉系统(HVS)的自适应嵌入策略

为了进一步优化信息隐藏的不可见性与鲁棒性之间的平衡，本算法在传统固定强度嵌入方法的基础上，引入了基于人类视觉系统(HVS)感知特性的自适应嵌入策略。HVS 理论指出，人眼对图像中平滑区域(如天空、墙壁)的微小扰动非常敏感，而对纹理复杂、边缘丰富的区域(如草地、毛发)内的相似扰动则不敏感，这种现象被称为视觉掩蔽效应(Visual Masking Effect)。利用该效应，可以在不牺牲不可见性的前提下，于不同区域嵌入不同强度的信息隐藏信息，从而最大化整体鲁棒性。

具体使用方法过程：

1) 区域复杂度分析：在嵌入信息隐藏前，对于每一个待嵌入信息隐藏的图像块，我们分析其在原始图像空间域中对应的区域。通过计算该区域像素的方差来量化其纹理复杂度。方差越大，代表该区域纹理越复杂、细节越丰富。

2) 自适应强度计算：我们不使用全局统一的嵌入强度，而是根据每个区域的方差动态计算一个自适应强度 k_i。其计算公式为：

$$k_i = k_{base} * \left(1 + \alpha * Var_i\right) \tag{6}$$

其中，k_{base} 是用户设定的基础强度，Var_i 是当前区域的方差，α 是一个调节因子，用于控制方差对强度的影响程度。

3) 智能嵌入：在嵌入信息隐藏比特时，使用这个动态计算出的 k_i 来调整频域系数。这意味着，在纹理复杂的区域，算法会自动使用更高的强度来嵌入信息隐藏，使其更难被移除；而在平滑区域，则会自动降低强度，以确保信息隐藏的不可见性。

3. 基于 Ctypes 的变换域图像信息隐藏算法

本章详细阐述了基于 C++/Python 混合编程的自适应信息隐藏算法的总体框架、C++核心库的设计与实现，以及 Python 调用层的具体实现方法。

3.1. 总开发流程图

本方案的创新之处在于其混合编程的工程架构，旨在融合 C++的高性能与 Python 的高效率。总体框架分为两个层次，如图 3 所示。

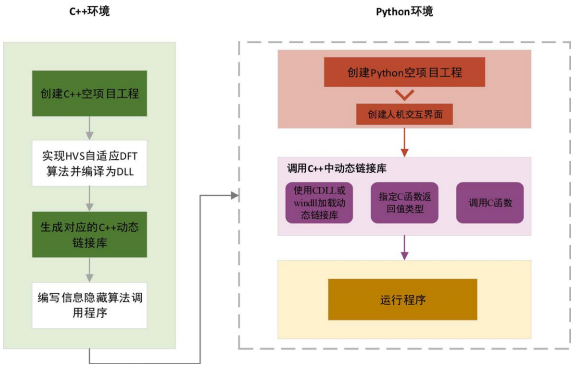

Figure 3. General development flow chart
图 3. 总开发流程图

第一阶段：C++核心库开发

1) 创建 C++项目：

2) 在 Visual Studio 中创建一个新的 C++项目，并选择“动态链接库(DLL)”模板。

3) 实现核心算法：在 C++项目中编写全部核心的信息隐藏算法，包括图像文件读写、颜色空间转换、DFT/IFFT 变换以及 HVS 自适应嵌入等功能。

4) 导出函数接口：使用 extern "C"__declspec(dllexport)关键字，将需要被 Python 调用的核心函数(如 EmbedWatermark, ExtractWatermark)声明为可供外部调用的接口。

5) 生成动态链接库：编译整个 C++项目，成功生成一个.dll 文件(例如 Watermark.dll)。

第二阶段：Python 应用层开发

6) 创建 Python 项目：创建一个 Python 项目，并将上一步生成的.dll 文件复制到项目目录下。

7) 编写 Python 调用程序：

8) 运行 Python 项目，验证整个流程是否能够成功嵌入和提取信息隐藏。

3.2. 核心信息隐藏算法流程

本节将作为一个整体，按照数据流的顺序，完整阐述信息隐藏从嵌入到提取的完整算法流程。

3.2.1. 图像预处理

算法的输入为一张 24 位真彩色 BMP 图像。无论是嵌入还是提取，算法首先对图像进行预处理。通过读取 BMP 文件的 BITMAPFILEHEADER 和 BITMAPINFOHEADER，精确获取图像的宽度、高度、位深等元数据。根据文件头中的偏移量信息，直接从文件中读取原始的像素数据。

为了在不显著影响图像色度的前提下嵌入信息隐藏，需将像素数据从 RGB 颜色空间转换到 YUV 颜色空间。信息隐藏信息将只嵌入在代表图像亮度的 Y 通道中，而代表色度的 U 和 V 通道则在整个过程中保持不变。其转换公式依据 ITU-R BT.601 标准：

$$\begin{cases} Y = 0.229R + 0.587G + 0.114B \\ U = -0.1684R - 0.3316G + 0.5B + 128 \\ V = 0.5R - 0.4187G - 0.0813B + 128 \end{cases} \tag{7}$$

3.2.2. 频域变换

为利用频域的特性隐藏信息，需对预处理后得到的亮度(Y)通道矩阵进行二维离散傅里叶变换(2D-DFT)。

1) 变换区域选取：为满足快速傅里叶变换(FFT)对数据尺寸的要求，算法会从亮度矩阵的中心截取一个边长为 2 的 N 次幂(如 256 × 256，512 × 512)的最大正方形区域作为计算对象。

2) 2D-FFT 计算：通过可分离性原理，将二维 FFT 分解为两次一维 FFT 来高效计算。首先对截取区域的每一行执行一维 FFT，然后对变换结果的每一列再次执行一维 FFT，得到该区域的二维频域复数矩阵。

3) 频谱中心化：为便于在视觉不敏感的中频区域进行操作，需对 FFT 输出的频谱进行象限交换，将代表图像低频信息的四个角落移动到频谱中心。

3.2.3. 基于 HVS 的自适应信息隐藏嵌入

本算法引入了基于人类视觉系统(HVS)感知特性的自适应嵌入策略，旨在将有限的失真预算智能地分配到最不易被察觉的区域，从而在保证高不可见性的前提下，最大化信息隐藏的整体鲁棒性。

低频区域集中了图像的主要能量，对其修改会严重影响图像质量；而高频区域则容易在 JPEG 压缩、滤波等常见攻击中被滤除。因此，中频区域是实现鲁棒性与不可见性的最佳平衡点。该区域在逻辑上被划分为若干不重叠的 4*4 小块，每个小块用于嵌入 1 比特信息。

与文献[2]中采用固定强度嵌入的方法不同，本算法对每一个待嵌入的比特动态计算其最佳嵌入强度。该过程的核心是建立一个能模拟 HVS 纹理掩蔽效应的计算模型。如图 4 所示。

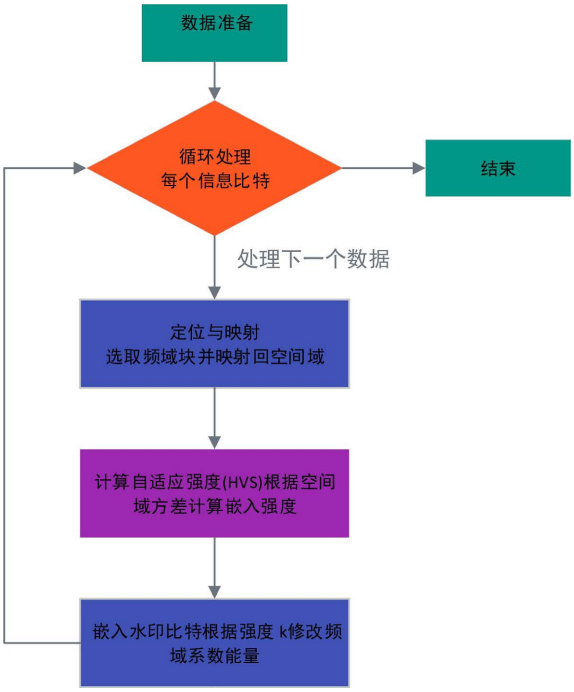

Figure 4. Flow chart of adaptive information hiding based on HVS
图 4. 基于 HVS 的自适应信息隐藏流程图

具体方法如下：

1) 定位与映射：对于频域中每一个待嵌入信息的 4*4 的数据块，我们将其位置反向映射回其在原始图像空间域(Y 通道)中所对应的像素区域，并计算该区域内所有像素值的方差(Variance)。方差能够有效地反映数据的离散程度，一个区域的方差越大，说明其纹理越复杂，HVS 的掩蔽能力也越强。因此，局部方差可以作为纹理掩蔽能力的一个可靠的数学近似。

2) 自适应强度计算：基于量化的局部纹理复杂度，我们建立了一个动态调整嵌入强度的数学模型。对于第 i 个图像块，其实际嵌入强度 k_i 由以下公式(6)确定。该模型确保了在纹理平滑的区域，嵌入强度 k_i 趋近于基础强度 kbase，以保护视觉质量；而在纹理复杂的区域，嵌入强度 k_i 将被显著放大，以提升局部信息隐藏的鲁棒性。

3) 信息隐藏嵌入：在每个频域块中，算法选取两个特定的、不重叠的子区域(例如，左下角和右上角的四个系数)，并分别计算它们的幅值之和，记为 e_1 与 e_2。根据当前待嵌入的比特值(0 或 1)以及利用公式(6)计算出的自适应强度 k_i，算法通过修改这两个子区域的系数幅值，确保它们满足以下关系之一：若嵌入比特为‘0’，则需满足：$e_1 - e_2 > k_i$ 若嵌入比特为‘1’，则需满足：$e_2 - e_1 > k_i$。

若原始系数的能量关系不满足上述强度要求，算法将计算出需要弥补的最小能量差，并将其按比例分配给两个子区域的系数，通过增大一个区域的能量、同时减小另一个区域的能量，来建立起一个足够

“深”且与局部纹理复杂度相匹配的能量差“鸿沟”。这种基于动态强度的修改，确保了信息隐藏在平滑区域足够隐蔽，在复杂区域则足够顽健。

4) 重复上述过程，直至所有信息比特嵌入完毕。

3.2.4. 信息隐藏提取

信息隐藏提取是一个盲过程，无需原始图像。对含信息隐藏图像执行相同的预处理(2.2.1)和频域变换(2.2.2)后，遍历频域中所有嵌入信息的 4 × 4 块。在每个块中，通过比较能量 e_1 和 e_2 的大小关系(若 $e_1 > e_2$ 则为‘0’，反之为‘1’)即可恢复出原始的二进制比特流，并最终重构为信息隐藏字符串。

3.3. 混合编程实现策略

将 2.2 节所述的理论算法落地，关键在于如何实现 C++核心与 Python 上层的桥接。本方案采用 Ctypes 技术作为实现这一策略的基石。

3.3.1. C++函数接口导出

在 C++侧，算法的全部功能被封装在一个 Watermarker 类中。为使其能被 Python 等外部语言调用，需要设计并导出两个简洁的、符合 C 语言调用规范的接口函数。这通过 extern\"C"__declspec(dllexport)声明来实现，extern\"C"：指示编译器使用 C 语言的命名约定，避免 C++的函数名修饰(Name Mangling)，从而保证函数名在 DLL 中是可预测的、稳定的。__declspec(dllexport)：明确告诉链接器，将这个函数放入动态链接库的导出表中，使其对外部可见。

```
define DLLEXPORT externdeclspec(dllexport)
#导出嵌入函数
LLEXPORT int EmbedWatermark(...);
#导出提取函数
DLLEXPORT int ExtractWatermark(...).
```

3.3.2. Python 与 Ctypes 桥接

在 Python 侧，ctypes 库扮演了“翻译官”和“调用者”的双重角色。

1) 动态库加载：使用 ctypes.CDLL()函数加载 C++编译生成的 DLL 文件。此操作在运行时将 DLL 加载到当前进程的内存空间。

2) 函数原型定义：这是实现稳定跨语言调用的核心步骤。必须为每个 C++函数精确定义其 Python 端的函数原型，即严格指定其参数类型(.argtypes)和返回值类型(.restype)。例如，C++中的 const char*对应 Python 中的 ctypes.c_char_p，int 对应 ctypes.c_int。这一过程确保了当 Python 调用 C 函数时，ctypes 能够正确地将 Python 对象(如字符串、数字)打包成 C 语言兼容的内存格式进行传递。

```
#定义 EmbedWatermaxrk 函数原型
embed func argtypes =[ctypes.c_char_p, ctypes.c_char_p, ctypes.c_int, ctypes.c_char_p]
embed_func.restype = ctypes.c_int
```

即可使用生成的应用库。

4. 实验结果

通过运行编写的 ctypes 的变换域信息隐藏算法应用，验证变换域信息隐藏算法是否能够通过 Ctypes 技术在 Python 平台上调用嵌入信息隐藏和提取信息隐藏，以达到隐藏信息、研究和学习的目的。

4.1. 实验环境与评价指标

在本次实验中，所使用的电脑操作系统是 Windows 10，编程环境方面，我们选用了 Visual Studio 2022 作为 C++语言的开发工具，而 Python 的开发则依赖于 Python 3.9 版本，并通过 Pycharm Community Edition 2024.1 这款免费的集成开发环境(IDE)来进行。这些软件版本的选择确保了实验环境的稳定性和兼容性。

评价指标：主要为不可见性指标：峰值信噪比(PSNR)和鲁棒性指标：比特误码率(BER)。PSNR (Peak Signal-to-Noise Ratio)是衡量含信息隐藏图像与原始图像失真程度的常用标准。其值越高，代表图像质量越好，信息隐藏的不可见性也越强。单位为分贝(dB)。BER (Bit Error Rate)用于衡量在遭受攻击后，提取出的信息隐藏信息与原始信息相比，错误比特所占的比例。BER 值越低(理想值为 0)，代表提取越准确，算法的鲁棒性越强。计算公式为：BER = 错误比特数/总比特数。

4.2. 实验展示

测试的数据如图 5 所示。图 5(a)是大小为 512 × 512 的 Tiffany 图。图 5(b)是大小为 256 × 256 的 24 位 Peppers 图。强度值以 50,000 为例。

(a) Tiffany　　(b) papers

Figure 5. Carrier image
图 5. 载体图像

图 6 是在变换域中经过嵌入之后转换为时域的含隐藏信息的图像。在图像提取过程中，隐藏的字符串是“1234Abc”，提取的字符串的最终结果是“1234Abc”

(a) Tiffany　　(b) papers

Figure 6. Embedded information hiding image
图 6. 嵌入信息隐藏的图像

4.3. 测试与对比

在测试与对比过程中，选用尺寸为 256 × 256 的 24 位 papers 图两者的平均值作为基准图像。

4.3.1. 嵌入强度对算法性能的影响

使用 Peppers 图像，固定信息隐藏信息，分别在强度为[10,000, 30,000, 50,000, 70,000, 90,000]的条件下嵌入信息隐藏，并在未受攻击的情况下直接提取，记录每组的 PSNR 和 BER 值。

Table 1. Average value of image indexes before attack
表 1. 未攻击前图像指标平均值

信息隐藏强度	10,000	20,000	50,000	70,000	90,000
BER	0.125	0.010	0.000	0.0000	0.000
PSNR/dB	48.12	42.58	38.75	36.21	34.19

从表 1 中数据可以看出：当强度过低(10,000)时，虽然 PSNR 极高，但嵌入的能量差不足以抵抗计算和保存过程中的精度损失，导致提取失败(BER > 0)。当强度达到 30,000 及以上时，信息隐藏均能被正确提取。随着强度的增加，PSNR 值稳定下降，意味着图像失真度逐渐增大。

综合考虑，选择强度 = 50,000 作为后续实验的基础强度。在此设置下，算法既能保证信息隐藏的完整性(BER = 0)，又能在可接受的范围内保持较高的图像质量(PSNR > 38 dB)，达到了较好的平衡。

4.3.2. 常见攻击下的鲁棒性测试

对使用 peppers.bmp 生成的含信息隐藏图像，分别施加 JPEG 压缩、噪声、滤波和几何攻击，然后提取信息隐藏并计算 BER。

Table 2. BER value after relevant attacks
表 2. 相关攻击后 BER 值

攻击类型	攻击参数	BER
JPEG 压缩	质量 90%	0.000
	质量 70%	0.025
	质量 50%	0.088
高斯噪声	方差 0.001	0.013
	方差 0.003	0.045
	方差 0.005	0.019
椒盐噪声	方差 0.01	0.038
	方差 0.03	0.094
	比例 0.05	0.128
滤波攻击	中值滤波(5*5 窗口)	0.050
	中值滤波(5*5 窗口)	0.138
	高斯滤波(Sigma1.0)	0.075
几何攻击	旋转 + 反旋(0.5 度)	0.063
	旋转 + 反旋(1.0 度)	0.113
	旋转 + 反旋(2.0 度)	0.213
	剪切(裁掉右侧 10%)	0.118
	剪切(裁掉右侧 10%)	0.25
	剪切(裁掉右侧 10%)	0.47

从表 2 中数据可以看出：该算法对 JPEG 压缩和高斯噪声表现出较强的鲁棒性，即使在质量为 70%的压缩和一定程度的噪声下，误码率也保持在较低水平。对中值滤波和椒盐噪声的抵抗力尚可，出现少量误码。对几何攻击(特别是旋转和剪切)的抵抗力由于自适应算法影响，几何攻击造成的影响具有随机性。结论：本算法的鲁棒性强弱顺序为：JPEG 压缩 > 噪声 > 滤波 > 几何攻击。

4.3.3. HVS 自适应模型性能对比

使用较高的固定强度 70,000，分别对平滑图像 lena.bmp 和复杂图像 peppers.bmp 嵌入信息隐藏再记录四张生成图像的 PSNR 值，并对它们施加统一的 JPEG 压缩攻击(质量 50%)，记录攻击后的 BER 值。使用较低的基础强度 40,000，但启用 HVS 模型，对同样两张图像嵌入信息隐藏。

Table 3. Performance comparison between foundation strength and HVS adaptive model
表 3. 基础强度与 HVS 自适应模型性能对比

图像	嵌入方式	基础强度	PSNR (dB)	BER (JPEG 50%)
Tiffany	固定强度	70,000	35.88	0.063
	HVS 自适应	40,000	41.23	0.075
Peppers	固定强度	70,000	36.15	0.088
	HVS 自适应	400,00	39.54	0.088

从表 3 中数据可以得出结论：

对于平滑图像 Tiffany：HVS 组的 PSNR 值(41.23 dB)显著高于固定强度组(35.88 dB)，说明 HVS 模型智能地降低了在平滑区域的嵌入强度，极大地保护了图像质量。尽管其 BER 略高，但仍在同一数量级，证明用较低的基础强度达到了相近的鲁棒性。

对于复杂图像 Peppers：HVS 组的 PSNR 值同样更高。同时，两组的 BER 值完全相同，这表明 HVS 模型在纹理区域“智能地”提升了局部强度，用 40,000 的基础强度达到了 70,000 固定强度的抗攻击效果。

HVS 自适应模型是有效的。它成功地将更多的“失真预算”分配到不敏感的复杂区域，同时减少了在敏感平滑区域的修改，从而能够在达到同等级鲁棒性的前提下，获得显著更高的图像质量(更高的 PSNR)。

5. 结论

本文围绕信息隐藏技术在高性能计算与高效率开发环境之间的应用鸿沟问题展开研究，成功设计并实现了一种基于 Ctypes 的 C++/Python 混合编程框架。该方案将一个基于离散傅里叶变换(DFT)并结合了人类视觉系统(HVS)自适应模型的信息隐藏算法在 C++中实现为高性能动态链接库，并由 Python 上层应用进行灵活调用，有效解决了计算性能与开发效率之间的矛盾。

但目前的算法在提取信息隐藏时，需要预先知道原始信息隐藏信息的长度，这在某些应用场景下可能受限。此外，实验数据表明，算法对于旋转、剪切等几何攻击的抵抗力仍有待加强。

基金项目

北京市高等教育学会项目(MS2023204)；北京市数字教育研究课题(BDEC2023619095, BDEC2024ZX042)；北京教育科学十四五规划课题(CDDB24253)；北京教育督导学会重点课题(BESA202420190034)；北京印刷学院学科建设和研究生教育专项(21090325004)；中国成人教育协会“数字赋能教育”2024 年度一般课题(2024-SJYB-007S)；北京印刷学院青年卓越项目(Ea202411)。

参考文献

[1] Podilchuk, C.I. and Delp, E.J. (2001) Digital Watermarking: Algorithms and Applications. *IEEE Signal Processing Magazine*, **18**, 33-46. https://doi.org/10.1109/79.939835

[2] Samtani, R. (2009) Ongoing Innovation in Digital Watermarking. *Computer*, **42**, 92-94. https://doi.org/10.1109/mc.2009.93

[3] Ma, Y., Wang, S., Song, J., Yu, Y., Sun, W. and Bian, J. (2020) Comparative and Analysis of Spatial Domain and Frequency Domain Digital Image Watermarking Algorithm. *Journal of Physics*: *Conference Series*, **1486**, Article 032026. https://doi.org/10.1088/1742-6596/1486/3/032026

[4] 眭新光. 文本信息隐藏及分析技术研究[D]: [博士学位论文]. 郑州: 解放军信息工程大学, 2007.

[5] 轩璐, 鲁晓辉. 一种基于空域均值的图像信息隐藏算法[J]. 长江信息通信, 2022, 35(9): 24-26.

[6] 轩璐. 一种基于关系的空域信息隐藏算法[J]. 电脑编程技巧与维护, 2022(7): 141-143+176.

[7] 丁海洋. 基于 DCT 和 DFT 视频盲信息隐藏算法的研究与实现[J]. 北京印刷学院学报, 2012, 20(4): 32-35.

[8] 刘旭. 基于小波变换的数字水印隐藏与检测算法设计[J]. 西安文理学院学报自然科学版), 2018, 21(4): 42-44+72.

[9] 张华宇, 汪国强. 基于 DWT 的数字视频信息隐藏算法[J]. 科技创新导报, 2018, 15(33): 100+102.

[10] 王先春, 郭杰荣, 胡惟文, 等. 基于 DCT 和 DWT 域的音频信息隐藏算法[J]. 计算机工程与设计, 2008(16): 4389-4391.

[11] 王海文, 李杰, 王鸿林, 等. 基于二维离散余弦变换的图像信息隐藏算法研究[J]. 印刷与数字媒体技术研究, 2023(3): 119-124.

[12] 许静. 一种基于 YUV 空间的 DCT 彩色图像信息隐藏算法[J]. 科技信息(学术研究), 2007(30): 415-416.

[13] 王树梅, 张文斌. 一种基于傅里叶变换的鲁棒信息隐藏算法[J]. 湖南理工学院学报(自然科学版), 2019, 32(3): 17-22.

[14] 杨烨, 孙容海, 施林甫, 等. 面向大文件的多载体图像信息隐藏方法[J]. 信息技术, 2018(5): 52-54.

[15] 陈欣欣, 玄宇, 牟歌, 等. 基于变换域的信息隐藏技术[J]. 科技创新与应用, 2014(35): 55.

[16] Guy, K.K. (2008) Automatic C Library Wrapping—Ctypes from the Trenches. *Python Papers*, **3**, 1-7.

[17] 秦定武, 丁海洋, 张凡, 等. 基于 Android 的变换域数字图像信息隐藏算法[J]. 通信技术, 2023, 56(1): 119-125.

[18] 许勇, 李佩佩. 基于 Android 系统视频信息隐藏算法的研究[J]. 信息技术, 2016(10): 113-116.

[19] Yegulalp, S. (2016) PyPy Update Makes Nice with Python C Extensions. *InfoWorld*, **2**, 18-23.

[20] Alex, H. (2008) Ctypes. Ctypes Run! *Python Papers*, **2**, 38-43.