

第一章 A

第二章 B

第三章 AIGC: 点亮内容创作的未来

3.1 GPT 代码开发和编写代码

在这个数字化时代，编程已经成为连接人类智慧与机器世界的桥梁。从智能手机到人工智能，每一行代码都在构建着我们未来的数字文明。但编程并非易事，它需要深奥的知识、严谨的逻辑和丰富的经验。

而 GPT（生成式预训练变换模型）的出现，正在改变这一现状。这项革命性的技术不仅能够理解人类的语言，更能理解和生成高质量的代码。这不仅是技术创新的进步，更是人类智慧与机器智能融合的新起点。

本节将带读者进入这个激动人心的技术领域，探索 GPT 如何成为每一位开发者的强大助力。无论您是编程新手还是经验丰富的开发者，都将在这里发现新的可能性，开启代码创作的新纪元。

3.1.1 使用 GPT 进行代码开发

背景与挑战

在传统的软件开发过程中，开发者需要面对一系列复杂的任务：

- **需求理解**：将业务需求转化为技术实现
- **系统设计**：规划系统的架构和模块
- **编码实现**：编写高质量的代码
- **测试维护**：确保代码的稳定性和可维护性

这些环节不仅耗时耗力，还需要开发者具备跨领域的知识。GPT 的出现为解决这些问题提供了新的思路。

GPT 如何助力代码开发

GPT 通过以下方式改变传统的开发模式：

1. **自动化生成文档**：根据现有代码自动生成 API 说明、用户手册
2. **智能代码审查**：提供代码改进意见，发现潜在问题
3. **快速原型开发**：根据需求描述快速生成初始代码框架
4. **跨语言支持**：在不同编程语言之间进行转换和适配

为什么选择 GPT ?

1. 提高效率: 减少重复性工作, 加速开发流程
2. 增强质量: 通过智能检查提升代码稳定性
3. 跨越鸿沟: 连接业务需求与技术实现
4. 创新可能: 探索传统方法难以触及的解决方案

实例演示: GPT 在开发中的应用

1. 生成 API 文档

- 输入代码片段, 输出详细的接口说明:

```
// 用户输入一段 REST API 的代码 GET /users Response: "users": [ "id": 1, "name":  
"Alice", "id": 2, "name": "Bob" ]
```

```
// GPT 生成的 API 文档 用户管理接口
```

获取用户列表 “http GET /api/users

响应格式

```
"users": [ "id": number, "name": string ]
```

错误码

- 401: 未认证
- 500: 内部错误
-

2. 代码审查与优化

用户输入的原始代码 `def calculate_average(numbers): sum = 0
for number in numbers:
 sum += number
average = sum / len(numbers)
return average`

GPT 建议的优化版本 `def calculate_average(numbers):
 if not numbers:
 return 0.0
 return sum(numbers) / len(numbers)`

3. 快速原型开发

- 用户需求描述: “需要一个在线投票系统, 用户可以创建投票、查看结果统计。”
- GPT 生成的代码框架:

```
models.py class Poll(models.Model): title = models.CharField(max_length = 200) description =  
models.TextField() created_at = models.DateTimeField(auto_now_add = True) votes =  
models.ManyToManyField(User, through = 'Vote')
```

```
class Vote(models.Model): user = models.ForeignKey(User, on_delete = models.CASCADE) poll =  
models.ForeignKey(Poll, on_delete = models.CASCADE) choice = models.CharField(max_length =  
200)
```

```
views.py @login_required def create_poll(request): if request.method == 'POST':  
    form = PollForm(request.POST) if form.is_valid(): poll = form.save(commit =
```

```
False)poll.created_by = request.userpoll.save()return redirect('poll_list')else : form =
PollForm()return render(request,'create_poll.html', 'form' : form)
```

3.1.2 使用 GPT 进行代码编写

背景与挑战

在日常编程中，开发者常常面临以下难题：

- **语法错误**：拼写错误、括号不匹配等问题
- **逻辑问题**：算法实现、条件判断等复杂逻辑
- **风格统一**：保持代码的一致性和可读性
- **知识获取**：快速查找 API、框架的使用方法

GPT 通过强大的语义理解和生成能力，为这些问题提供了创新性的解决方案。

GPT 如何提升编码效率

1. **智能代码补全**：根据上下文预测下一步代码
2. **错误修正建议**：识别并提出修复方案
3. **代码风格调整**：统一格式、优化命名
4. **技术文档查找**：即时获取 API 说明和使用示例

为什么选择 GPT ?

1. **即时帮助**：随时提供编程指导和建议
2. **提高质量**：通过智能纠错提升代码可靠性
3. **节省时间**：减少查找资料的时间成本
4. **创新思路**：启发新的解决方案和实现方式

实例演示：GPT 在编码中的应用

1. 根据自然语言生成代码

- 用户输入：“我需要一个函数，可以计算两个数的和，并返回结果。”
- GPT 生成代码：

```
def add_numbers(a,b): """Calculatethesumoftwonumbers.""" return a + b
```

2. 修复语法错误

- 用户输入错误代码：

```
print("Hello World!)
```

 GPT 检测到缺少引号，修正为：

```
print("Hello World!")
```

3. 优化代码逻辑

- 用户输入低效代码：

```
total = 0 for num in numbers: total += num average = total / len(numbers)
```

GPT 建议更简洁的方式：

```
average = sum(numbers) / len(numbers) if numbers else 0
```

4. 代码风格统一

- 用户输入不一致的命名规范：

```
function getUsername(id) return users[id].name;
```

```
function GetUserEmail(id) return users[id].email;
```

GPT 调整为一致的驼峰式命名：

```
function getUserInfo(id, field) return users[id][field];
```

3.1.3 动手试试

在前面的介绍中，已经有了实例演示的例子，那么我们现在再尝试动手使用 GPT 写几个小例子吧～

正所谓古人云：“学而不思则罔，思而不学则殆”。

摩斯密码

摩尔斯电码（Morse code）也被称作摩斯密码，是一种时通时断的信号代码，通过不同的排列顺序来表达不同的英文字母、数字和标点符号。它发明于 1837 年，是一种早期的数字化通信形式。不同于现代化的数字通讯，摩尔斯电码只使用零和一两种状态的二进制代码，它的代码包括五种：短促的点信号“·”，保持一定时间的长信号“—”，表示点和划之间的停顿、每个词之间中等的停顿，以及句子之间长的停顿。

各位读者肯定在众多影视作品中可以看到，那么现在我们就让 GPT 来完成一个这样的简单例子。

用户输入：“现在我想编写一个将自然语言转化为摩斯密码的代码，使用 Python 代码实现，就以”I LIKE U“这个句子为例实现一下。”

GPT 回复：

为了将自然语言句子转换为摩斯密码，我们可以按照以下步骤实现：

1. **创建摩斯密码字典**：包含所有英文字母及其对应的摩斯码。
2. **处理输入字符串**：将其转为大写，并去除非字母字符，分割成单词列表。
3. **转换每个单词**：将每个字母转换为摩斯码，并用单个空格分隔。
4. **合并单词间的摩斯码**：使用三个空格分隔不同单词。


以下是完整的 Python 代码：

```
创建摩斯密码字典 mos_codes = 'A' : ' .-' , 'B' : ' -...' , 'C' : ' - . -' , 'D' : ' - ..' , 'E' : ' .' , 'F' : ' .. - .' , 'G' : ' - - .' ,
获取输入并处理 input_str = input(" ").upper() cleaned = " ".join([cifcinmos_codesorc == '
'else"forcininput_str]]) words = cleaned.split()
转换为摩斯码 morse_words = [] for word in words : morse_word = [mos_codes[char] for char in word] morse_word
合并单词之间的空格 final_morse = ' '.join(morse_words) print(final_morse)
示例输入输出：
```

- 输入：“I LIKE U”
- 输出：“.. -.. .. -.- . .-”

这段代码能够将输入句子转换为对应的摩斯密码，每个字母用单个空格分隔，单词之间用三个空格分隔。

通过我们本地运行，得到了正确的结果：



```
请输入要转换的句子: I LIKE U
.. -.. .. -.- . .-
```

图 3.1: 摩斯密码输出

很有意思不是吗？现在你可以用摩斯密码对你想说的话进行加密了，你可以尝试继续让 GPT 改进，让这个简单代码不断完善并且可以处理中文相关的信息，动手试试看吧！

恺撒密码

恺撒密码（英语：Caesar cipher），或称**恺撒加密**、**恺撒变换**、**变换加密**，是一种最简单且最为人知的加密技术。它是一种替换加密的技术，明文中的所有字母都在字母表上向后（或向前）按照一个固定数目进行偏移后被替换成密文。例如，当偏移量是 3 的时候，所有的字母 A 将被替换成 D，B 变成 E，以此类推。这个加密方法是以罗马共和时期恺撒的名字命名的，当年恺撒曾用此方法与其将军们进行联系。

这次我们再尝试一个古典密码学中的经典加密，根据苏维托尼乌斯于公元二世纪写的《恺撒传》中的记载，恺撒曾用此方法对重要的军事信息进行加密，它的原理非常简单，就是将所有字母偏移固定的长度，就像下图：

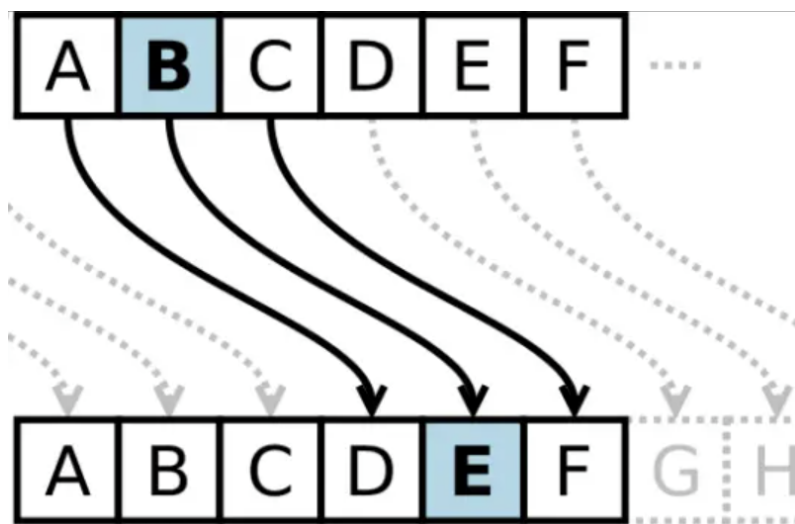


图 3.2: 恺撒加密示意图

那么现在我们同样用 GPT 来实现一个这样简单的例子：

用户输入：“现在我想编写一个将自然语言转化为恺撒密码的代码，偏移量为 3，使用 Python 代码实现，就以 My Fault forgive me for saying I like you here 这个句子为例实现一下。”

GPT 回复：好的！凯撒密码是一种替换加密技术，每个字母在字母表上向后移动固定的位数（这里是 3 位）。例如，‘A’ 会被转换为‘D’，‘B’ 会变成‘E’，依此类推。对于字母表的末尾字母（如‘X’、‘Y’、‘Z’），它们会循环回到字母表的开头。

以下是将自然语言转化为凯撒密码的 Python 代码实现：

```
def caesar_cipher(text, shift = 3): result = []
    for c in text:
        if c.isupper():
            new_char = chr((ord(c) - ord('A') + shift) % 26 + ord('A'))
        elif c.islower():
            new_char = chr((ord(c) - ord('a') + shift) % 26 + ord('a'))
        else:
            new_char = c
        result.append(new_char)
    return ''.join(result)
```

示例句子 sentence = "My Fault forgive me for saying I like you here"

```
encrypted_sentence = caesar_cipher(sentence, shift = 3)
print("Original sentence: ", sentence)
print("Encrypted sentence: ", encrypted_sentence)
```

代码解释：

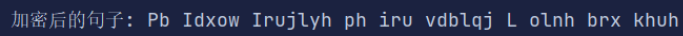
1. `caesar_cipher` 函数接受两个参数：`text`（要加密的文本）和 `shift`（偏移量，默认为 3）。
2. 遍历输入文本中的每个字符。
3. 对于大写字母，计算其在字母表中的新位置，并将结果转换回字符。
4. 对于小写字母，同样进行处理。
5. 非字母字符（如空格、标点符号等）保持不变。
6. 将所有处理后的字符拼接成最终的加密字符串。

示例运行：

输入句子：“My Fault forgive me for saying I like you here” 加密后的输出会是：“Pb Idxow irujlyh ph iru vdbqlj L olnh brx khuh”

这个代码可以将任何自然语言文本转换为凯撒密码，适用于需要简单加密的场景。

通过我们本地运行，得到了正确的结果：



```
加密后的句子: Pb Idxow Irujlyh ph iru vdblj j L olnh brx khuh
```

图 3.3: 恺撒加密输出图

Eulooldqw! (Brilliant!) 现在你可以用恺撒密码对你想说的话进行加密了，你可以尝试继续让 GPT 改进，让它可以解密恺撒密码，动手试试吧！