南開大學
Nankai University

计算机学院

软件工程实验报告

智慧编程助手性能分析

姓名：刘诺铭、刘修铭、朱子轩、陈佳卉、王祺鹏

学号：2110607、2112492、2110853、2110694、2110608

专业：计算机科学与技术、信息安全

2024 年 4 月 26 日

目录

1 整体设计思路

1.1 系统概述

本系统旨在通过一个 VSCode 扩展 API，提供智能代码维护功能，帮助开发者优化和修复代码，提高代码质量和开发效率。该 API 将利用机器学习技术，分析代码模式并提供实时的代码改进建议。

1.2 API 名称和框架

名称： CodeGuardian
框架： 基于 Node.js 的 VSCode 扩展，集成 TensorFlow 机器学习模型进行代码分析。

1.3 API 设计

1.3.1 功能概述

CodeGuardian 将提供以下核心功能：

- 代码缺陷检测：自动检测潜在的编程错误和代码异味（code smells）。
- 代码改进建议：基于最佳实践提出具体的代码改进建议。
- 代码趋势分析：分析代码库的历史变化，预测潜在的风险点。
- 实时代码反馈：在开发者编写代码时提供即时反馈和建议。
- 实现 kdump 分析：在 Linux 操作系统崩溃时，会进入 kdump 允许获取内核崩溃时的 dump stack 和故障现场，通常 dump 包括调用栈、寄存器和内存分析等信息，并且需要结合当前版本代码进行详细分析。

1.3.2 API 输入输出

输入： 用户在 VSCode 中编写的源代码。
输出： 代码分析报告，包括错误、警告、和改进建议。

1.3.3 技术细节

- 代码分析引擎：利用自然语言处理（NLP）和机器学习算法分析代码结构和语义，如 Transformer 模型。
- 模型训练：使用公开的代码库和历史修复数据训练模型，以识别常见的编程问题和修复模式。
- 实时反馈机制：通过 VSCode 的 API 钩子，实时分析用户输入的代码并提供反馈。

1.4 设计思想

1.4.1 用户中心设计

API 设计始终以用户体验为核心，确保功能直观易用，反馈及时有效，与 VSCode 的其他功能和界面无缝集成。

1.4.2 可扩展性

设计时考虑到未来可能加入的新功能和技术，例如支持更多编程语言和集成更先进的机器学习模型。

1.4.3 安全性与隐私

确保所有代码分析在本地执行，不上传用户数据至服务器，保护用户代码的安全与隐私。

CodeGuardian 旨在为开发者提供一个强大、智能、并且安全的代码维护工具，大幅提升代码质量和开发效率。

1.5 UML 类图

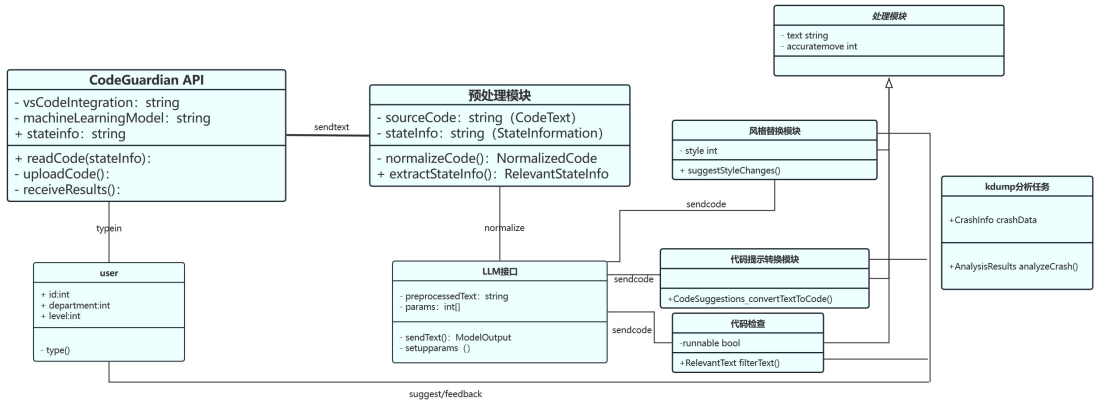

图 1.1: UML 类

UML 类图描述

CodeGuardian API

- 属性:
  - `- vsCodeIntegration: string` (私有)
  - `- machineLearningModel: string` (私有)
  - `+ stateinfo: string` (公有)
- 方法:
  - `+ readCode(stateInfo): void` (公有)
  - `- uploadCode(): void` (私有)
  - `- receiveResults(): void` (私有)

预处理模块

- 属性:
  - `- sourceCode: string` (私有)

  - - stateInfo: string (私有)

- 方法:
  - - normalizeCode(): NormalizedCode (私有)
  - + extractStateInfo(): RelevantStateInfo (公有)

LLAMA3 模型接口

- 属性:
  - - preprocessedText: string (私有)
  - - params: int[] (私有)
- 方法:
  - - sendText(): ModelOutput (私有)
  - - setupparams(): void (私有)

代码检查

- 属性:
  - - runnable: bool (私有)
- 方法:
  - + filterText(): RelevantText (公有)

代码提示转换模块

- 方法:
  - + convertTextToCode(): CodeSuggestions (公有)

风格替换模块

- 属性:
  - - style: int (私有)
- 方法:
  - + suggestStyleChanges(): void (公有)

kdump 分析任务

- 属性:
  - + crashData: CrashInfo (公有)
- 方法:
  - + analyzeCrash(): AnalysisResults (公有)

User

- 属性:
  - `+ id: int` (公有)
  - `+ department: int` (公有)
  - `+ level: int` (公有)
- 方法:
  - `- type(): void` (私有)

类之间的关系

关联关系

- `CodeGuardian API` 通过 `sendtext` 关联到 预处理模块。
- 预处理模块通过 `normalize` 关联到 `LLAMA3` 模型接口。
- `LLAMA3` 模型接口通过 `sendcode` 分别关联到 文本到代码提示转换模块、风格替换模块和代码检查

继承关系

抽象类：处理模块

风格替换模块、代码检查、代码提示均继承自处理模块这个抽象类。不同的子类负责处理不同的运维功能。

1.6 整体功能顺序图

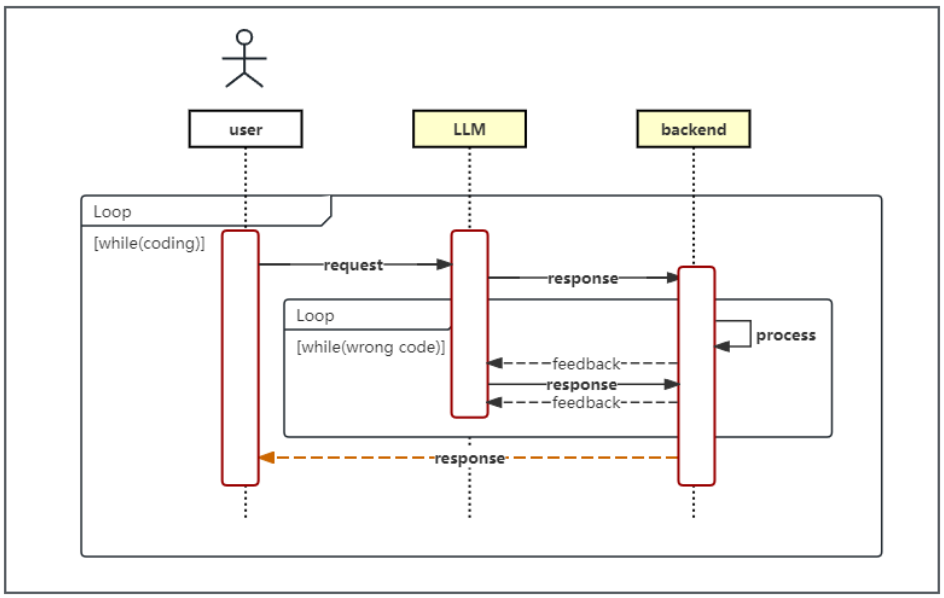

图 1.2: 整体顺序图

如图，给出了系统的整体的顺序图。用户在编码时，系统将自动生成与 LLM 交互的指令并予以发送，后台收到其响应后，将按照用户的使用申请情况对响应进行二次处理：如果符合要求，则作为建议反馈给用户；如果不符合要求，将重新反馈给 LLM 进行重复响应，直到符合要求为止。

1.7 整体功能状态图

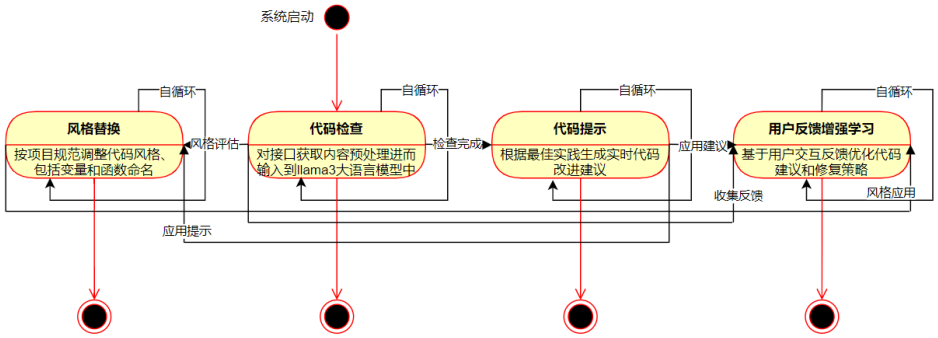

图 1.3: 整体状态图

这张整体功能状态图清晰地展示了编程助手软件的主要功能流程，在“系统启动”之后，首先进入“代码检查”状态，此处对接口获取内容预处理进而输入到 llama3 大语言模型中；完成检查后，软件提供“代码提示”以帮助改善代码质量；接着，用户可以选择进入“风格替换”状态，根据既定规则优化代码风格；最终，“用户反馈增强学习”状态收集用户的互动数据，用于持续改进软件性能。每个状态都有可能通过用户的选择或完成特定任务后直接结束，如图所示的独立终止状态。这种设计体现了软件的灵活性，允许在流程的任何阶段根据需要停止操作。

2 CodeGuardian API 接口设计

CodeGuardian API 的接口设计使 API 能够从 VSCode 中读取代码和其他相关状态信息，并上传给后端的机器学习模型处理，同时将模型的输出（如代码提示、代码检查、代码风格转换等）以适当的方式显示在 VSCode 中。

2.1 接口设计目标

该 API 接口的主要目标包括：

- 实现与 VSCode 环境的无缝集成，能够实时读取编辑器中的代码和状态信息。
- 将读取的信息格式化为字符串，并安全地上传至后端服务。
- 接收后端服务的处理结果，并根据不同功能模块将结果反馈给用户。
- 提高代码编写效率和质量，辅助开发者进行代码优化和风格统一。
- 通过理解对应版本的 kernel 代码，完成自动化 dump 信息的分析，给出初步故障诊断建议，简化故障定位流程。

2.2 接口功能模块

2.2.1 代码和状态信息读取模块

该模块负责从 VSCode 编辑器中读取当前活动文件的代码内容以及相关的编辑状态信息。主要功能包括：

- 捕获当前活动窗口中的源代码。
- 读取代码编辑的相关上下文，如光标位置、选中的文本等。
- 将捕获的数据格式化为文本文件，并准备用于上传。

2.2.2 数据上传模块

此模块负责将格式化后的代码文本安全上传到后端机器学习模型。主要功能包括：

- 加密代码文本以保护代码隐私。
- 通过安全的 API 调用将加密数据发送到服务器。
- 处理任何网络错误或数据传输问题。

2.2.3 功能输出展示模块

该模块接收后端模型处理的结果，并根据不同的输出类型在 VSCode 中以适当方式展示。主要功能包括：

- 接收并解析后端返回的数据（如代码检查结果、风格建议等）。
- 在 VSCode 的相应位置显示提示信息或高亮显示问题代码。
- 提供用户交互界面，允许用户选择是否应用某些代码修改建议。

2.3 安全与性能考虑

在设计 API 时，特别重视安全性和性能：

- 所有数据传输均采用加密技术，确保代码数据的安全性。
- 设计高效的数据处理流程，确保 API 响应迅速，不会显著影响 VSCode 的性能。

3 代码检查 LLM 接口设计

代码检查 LLM 接口的主要功能是将从 API 接口获取的代码文本及其他相关状态信息经过预处理，进而输入到 llama3 大语言模型中。该接口设计的目的是利用 llama3 模型的能力，根据历史信息和当前输入文本，输出适用于下游任务的文本序列。

3.1 接口设计目标

代码检查 LLM 接口设计的主要目标如下：

- 将源代码和相关状态信息有效地转化为模型可接受的文本格式。
- 利用 llama3 大语言模型的强大能力，进行代码质量分析和优化建议的生成。
- 提供高质量的模型输出，以便下游任务能够准确执行代码提示、代码检查和代码风格转换等功能。

3.2 接口功能顺序图

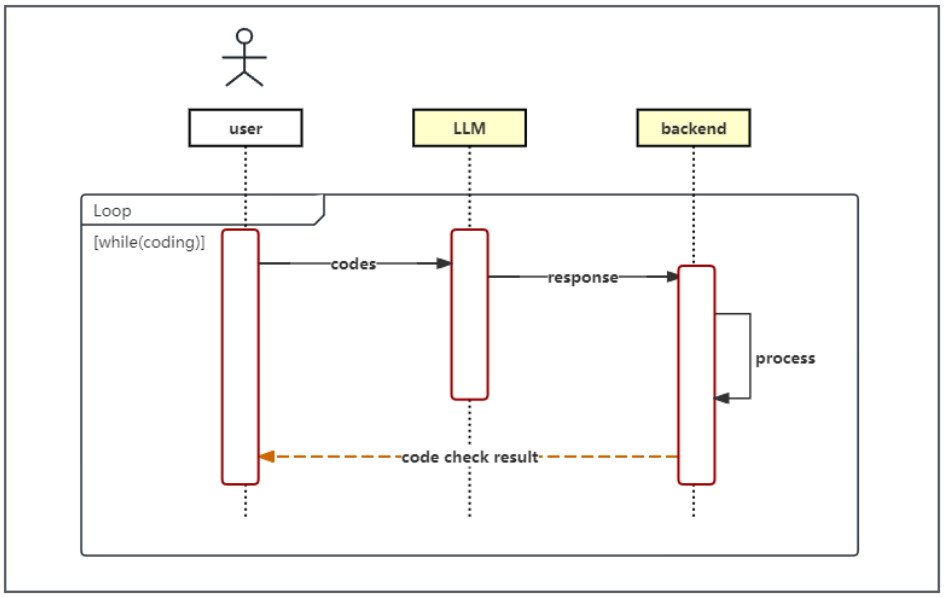

图 3.4: 代码检查

如图，给出了代码检查部分的顺序图。用户在编写代码时，软件读取用户的编码内容，创建 LLM 指令，并发送到 LLM 读取响应，得到响应后，后台进行一定处理，然后作为修改建议反馈给用户。重复进行，直到编码完成。

3.3 接口功能状态图

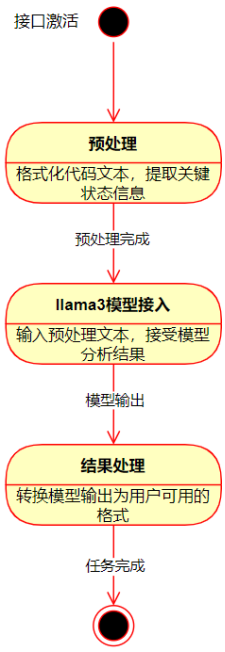

图 3.5: 代码检查状态图

状态图开始于“接口激活”，标志着代码检查流程的启动。首个状态为“预处理模块”，这里将源代码格式化并准备数据，以供后续步骤使用；接着，进入“llama3 模型接入”状态，该状态负责将预处理好的代码送入机器学习模型并等待其分析结果；分析完成后，状态转移到“结果处理模块”，在此处理并格式化模型的输出，使之适用于前端展示；最终，流程以“任务完成”结束，标识整个代码检查的完成。

3.4 接口功能模块

3.4.1 预处理模块

该模块负责处理从 API 接口接收到的源代码和状态信息，转换成适合模型处理的文本格式。具体功能包括：

- 标准化代码文本格式，确保一致性和兼容性。
- 提取关键状态信息，如光标位置、代码块关系等，并将其融入文本中。
- 清洗和过滤无关信息，保留对模型预测最有价值的数据。

3.4.2 llama3 模型接入模块

此模块负责将预处理后的文本输入到 llama3 大语言模型中，并处理模型的输出结果。主要功能包括：

- 管理与 llama3 模型的 API 调用，包括数据传输和结果接收。

- 调整模型输入参数（如 token 数限制、温度等），以优化输出结果。
- 解析模型输出，将其格式化为适合下游任务处理的数据结构。

3.5 性能与安全性

在设计接口时，特别关注性能和安全性的问题：

- 确保数据在传输过程中的安全，使用加密技术保护代码文本和状态信息的隐私。
- 设计高效的数据处理流程，以减少模型调用的延迟，并提升整体响应速度。

代码检查 LLM 接口旨在为下游任务提供准确、高效的文本序列输出，大幅提升代码质量分析和优化建议的准确性和实用性。

4 代码提示下游任务设计

代码提示下游任务的主要目标是处理从 LLM 模型（如 llama3）接收到的代码文本序列，并将其转换为实用的代码提示。设计采用基于 BERT 或类似的文本到文本框架来实现代码提示的生成和优化。

4.1 任务设计目标

代码提示下游任务设计的主要目标包括：

- 有效地从模型输出中提取与代码编写直接相关的文本序列。
- 转换这些文本序列为实用的、具体的代码提示，帮助开发者提升编码效率和质量。
- 确保代码提示的准确性和实用性，以提高开发者对工具的信赖和满意度。

4.2 任务功能顺序图

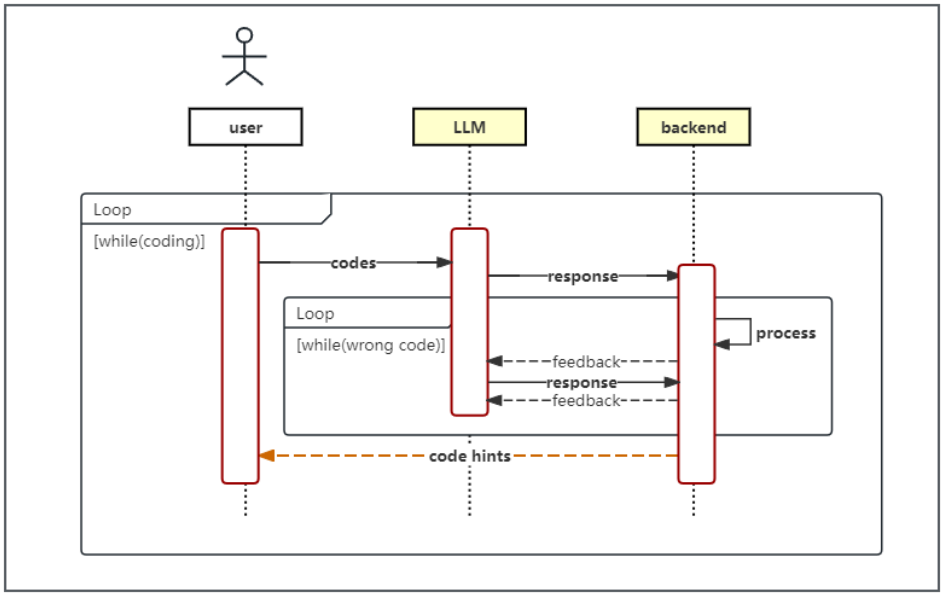

图 4.6: 代码提示

如图，给出了代码提示部分的顺序图。用户在编写代码时，软件读取用户的编码内容，创建 LLM 指令，并发送到 LLM 读取响应，得到响应后，后台进行一定处理，如果给出的代码提示有明显错误，则再发送给 LLM 进行处理，重复进行，直到后台无法检测出错误，此时再将结果作为建议反馈给用户，用户再得到建议后，可以选择接受或者拒绝该提示。重复进行，直至编码结束。

4.3 任务功能状态图

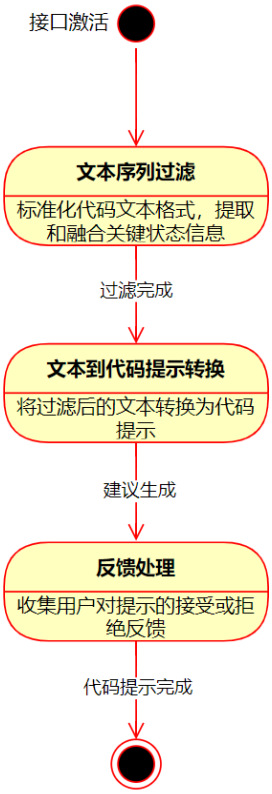

图 4.7: 代码提示状态图

状态图的起始点是“接口激活”，代表功能的启，紧接着是“文本序列过滤”状态，处理输入的代码文本，并提取必要信息；这一过程完成后，流程移至“文本到代码提示转换”状态，这里将处理过的文本转换成具体的代码建议；最终，达到“反馈处理”状态，可能涉及用户反馈接收和系统设置调整；经过一系列逻辑处理后，到达“任务完成”状态，标志着整个流程的结束。

4.4 功能模块设计

4.4.1 文本序列过滤模块

此模块负责分析 LLM 模型输出的文本序列，并过滤掉与代码提示不直接相关的内容。主要功能包括：

- 使用自然语言处理技术识别和删除无关的文本序列。
- 提取与当前编码任务相关的关键信息，如 API 调用、语法结构建议等。

4.4.2 文本到代码提示转换模块

采用基于 BERT 的架构，此模块将过滤后的文本序列转换为具体的代码提示。主要功能包括：

- 解析处理过的文本，生成具体的编程语言构造。
- 根据开发者的当前代码上下文，优化提示内容，以提供最相关的建议。

- 利用文本到文本转换技术，确保输出的自然度和适用性。

4.5 性能与用户体验

在设计任务时，特别关注性能和用户体验：

- 设计快速响应的处理流程，减少从接收模型输出到显示代码提示的延迟。
- 提供用户友好的交互界面，允许开发者根据个人偏好调整提示设置。

代码提示下游任务旨在为开发者提供精确、及时的编码支持，使得编码过程更加高效和准确。该任务不仅增强了开发者的代码质量，也提升了开发效率和工具的实用性。

5 风格替换下游任务设计

本文档详细描述了风格替换下游任务的设计，该任务的主要目标是处理从 LLM 模型（如 llama3）接收到的代码文本序列，并将其转换为具体的代码风格替换提示。设计采用基于 GPT 或类似的文本到文本框架来实现风格替换的生成和优化，不同于之前使用的 BERT 框架，以便更好地处理风格转换的创造性和复杂性。

5.1 任务设计目标

风格替换下游任务设计的主要目标包括：

- 有效地从模型输出中提取与代码风格替换直接相关的文本序列。
- 转换这些文本序列为具体的代码风格替换提示，通常包括对变量名、函数名的命名风格建议等。
- 提供可选的风格替换建议而非强制替换，以尊重开发者的编程习惯和项目特定规范。

5.2 任务功能顺序图

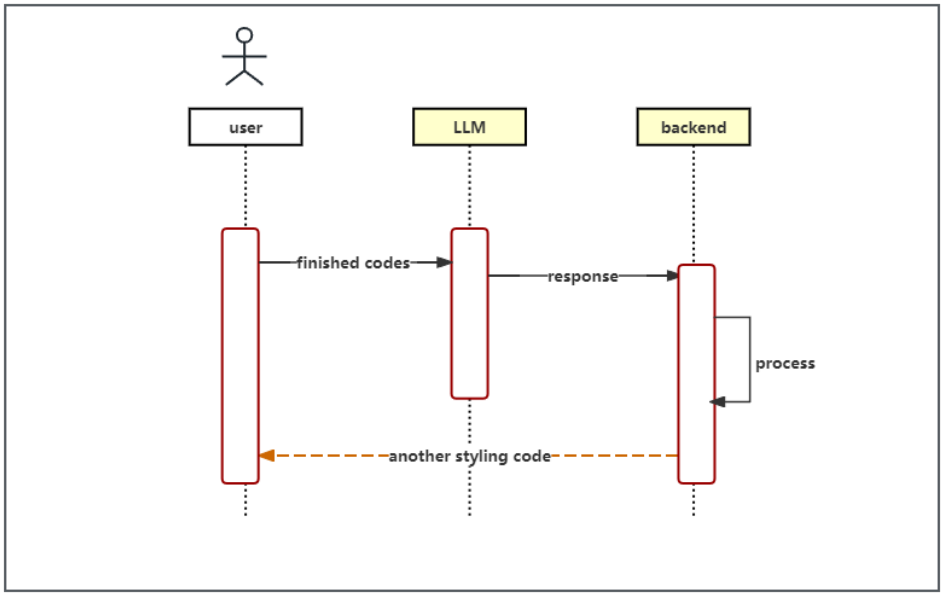

图 5.8: 代码风格替换

如图，给出了代码风格替换部分的顺序图。用户在编码完成后，可根据目标用户，选择要更换的目标代码的风格，然后系统将创建 LLM 指令发送给 LLM，LLM 根据已掌握的目标风格的代码对用户的编码进行重构，进行代码风格的替换。后台接收到响应后进行一定的处理，然后以建议反馈给用户。

5.3 任务功能状态图

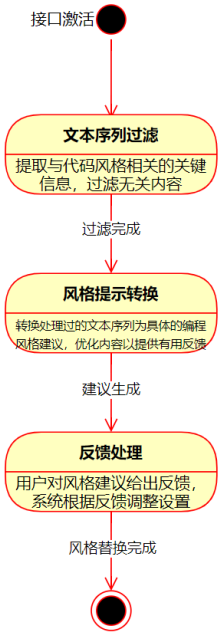

图 5.9: 代码风格替换状态图

图从“接口激活”开始，代表着任务的启动，接下来是“文本序列过滤”状态，负责处理和准备输入数据，提取与代码风格直接相关的文本序列；然后，进入“风格提示转换”状态，此处将过滤后的文本转换成具体的编程风格建议；最终，达到“反馈处理”状态，可能涉及用户反馈接收和系统设置调整；状态图以“任务完成”结束，整个风格替换过程圆满结束。

5.4 功能模块设计

5.4.1 文本序列过滤模块

此模块负责分析 LLM 模型输出的文本序列，并过滤掉与风格替换不直接相关的内容。主要功能包括：

- 使用自然语言处理技术识别和删除无关的文本序列。
- 提取与代码风格和编程规范相关的关键信息，如命名约束、代码布局建议等。

5.4.2 文本到风格提示转换模块

采用类 GPT 的架构，此模块将过滤后的文本序列转换为具体的风格替换提示。主要功能包括：

- 解析处理过的文本，生成具体的编程风格建议。
- 根据开发者的当前代码上下文，优化提示内容，以提供最相关和有用的反馈。
- 利用文本到文本转换技术，确保输出的自然度和适用性。

5.5 性能与用户体验

在设计任务时，特别关注性能和用户体验：

- 设计快速响应的处理流程，减少从接收模型输出到显示风格替换提示的延迟。
- 提供用户友好的交互界面，允许开发者根据个人偏好调整风格替换设置。

风格替换下游任务旨在为开发者提供精确、及时的代码风格支持，使得代码更加规范和一致。该任务不仅增强了代码的可读性，也提升了整体的编码质量和维护性。

6 kdump 分析任务设计

6.1 任务设计目标

在 Linux 操作系统崩溃时，会进入 kdump 允许获取内核崩溃时的 dump stack 和故障现场，通常 dump 包括调用栈、寄存器和内存分析等信息，并且需要结合当前版本代码进行详细分析。

该任务旨在利用语言模型的知识，通过理解对应版本的 kernel 代码，完成自动化 dump 信息的分析，给出初步故障诊断建议，简化故障定位流程。

6.2 任务功能顺序图

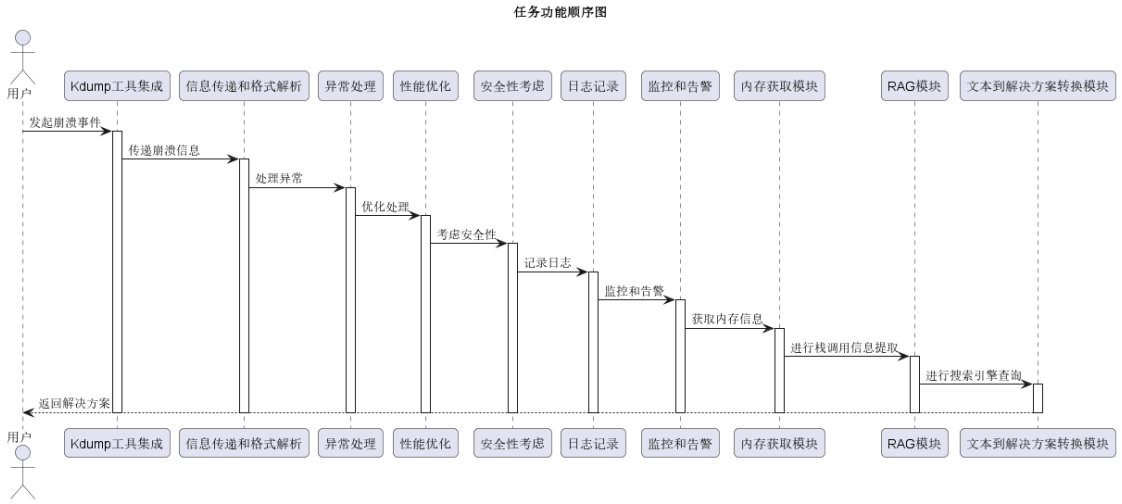

图 6.10: kdump 分析

本图描述了系统中不同模块之间的交互流程，展示了系统中各个模块之间的交互流程，以及它们如何共同处理崩溃信息并提供解决方案给用户。以下是对图中主要组件和操作的思路介绍：

- **用户（User）**：系统的最终用户或者操作者，触发系统的某些功能。
- **Kdump 工具集成（Kdump）**：在系统发生崩溃事件时，Kdump 工具会被触发，传递崩溃信息给后续处理模块。
- **信息传递和格式解析（Info）**：接收并解析来自 Kdump 工具的崩溃信息，将其转换成系统可理解的格式。
- **异常处理（Exception）**：处理解析后的异常情况，确保系统能够尽快恢复正常状态。
- **性能优化（Performance）**：对处理过程进行优化，提高系统的性能和响应速度。
- **安全性考虑（Security）**：在处理过程中确保系统安全，防止出现安全漏洞。
- **日志记录（Log）**：记录处理过程中的重要事件和信息，以备将来审查和分析。
- **监控和告警（Monitor）**：监视系统状态，发现异常情况时发出警报，通知相关人员进行处理。
- **内存获取模块（Memory）**：获取内存信息，供后续分析使用。
- **RAG 模块**：根据内存信息提取栈调用信息，进行搜索引擎查询获取相关内容。
- **文本到解决方案转换模块（Conversion）**：整理错误位置信息，返回给用户供进一步处理和故障诊断。

6.3 任务功能状态图

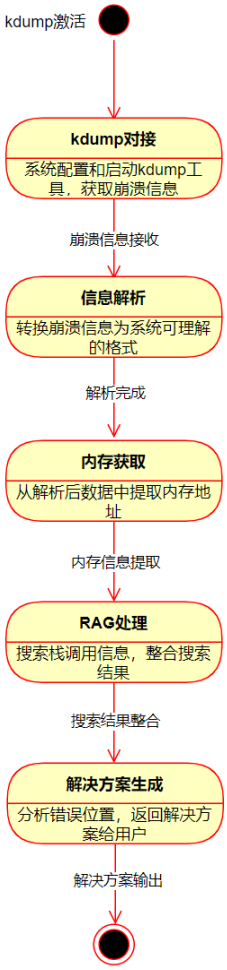

图 6.11: kdump 分析

状态图从“kdump 激活”开始，这一步表示系统已经进入 kdump 模式，准备分析崩溃信息；随后流程进入“kdump 对接”状态，其中系统与 kdump 工具集成，接收到崩溃信息；接下来，系统进入“信息解析”状态，负责将接收到的数据转换成可处理的格式；然后，是“内存获取”状态，此时系统从解析后的数据中提取内存地址；在“RAG 处理”状态，系统执行搜索查询以整合栈调用信息；最后，“解决方案生成”状态负责将分析结果整理并反馈给用户，流程以“分析完成”结束。

6.4 功能模块设计

6.4.1 kdump 对接模块

Kdump 对接模块是系统中的一个重要组成部分，负责在 Linux 内核崩溃时转入 kdump 获取崩溃的详细信息，并将其传递给后续处理模块进行分析。下面是该模块的设计方案：

- **Kdump 工具集成:** 确保系统已经正确配置和启用 kdump，并在崩溃事件发生时自动触发 kdump 机制。这需要与 kdump 工具进行集成，以便在需要时能够及时获取崩溃信息。
- **信息传递和格式解析:** 设计合适的数据结构来存储从 kdump 获取的信息，例如调用栈、寄存器

内容等。开发解析模块，将 kdump 输出的原始数据解析为系统可理解的格式，以便后续模块处理和分析。

- **异常处理:** 考虑如何处理 kdump 过程中可能出现的异常情况，例如数据损坏、解析错误等。实现错误处理机制，能够记录异常情况并向管理员报告，以确保系统的稳定性和可靠性。
- **性能优化:** 优化 kdump 对接模块的性能，以确保在系统崩溃时能够快速且可靠地获取信息。使用多线程或异步处理等技术，提高处理效率并减少对系统性能的影响。
- **安全性考虑:** 确保 kdump 对接模块的安全性，防止恶意攻击者利用 kdump 机制获取系统敏感信息。实施权限控制和身份验证机制，限制对 kdump 信息的访问和使用。
- **日志记录:** 记录 kdump 对接模块的操作日志，包括崩溃事件的时间、原因、处理过程等信息。这有助于后续审查和分析，并能够及时发现系统中存在的问题。
- **监控和告警:** 实现监控和告警机制，能够及时发现并响应系统崩溃事件。通过监控系统状态和检测异常行为，及时通知相关人员进行处理，确保系统的稳定性和可靠性。

6.4.2 内存获取模块

内存获取模块负责将上个模块得到的信息输入模型，并从模型中获取需要获取的内存的地址，自动给出需要的内存地址的内容。该模块的设计如下：

- **数据输入:** 从信息传递和格式解析模块接收解析后的数据，包括崩溃时的内存地址、寄存器内容等。
- **模型查询:** 将接收到的数据输入到预先训练好的模型中，模型将根据输入的信息给出需要获取的内存地址的推荐列表。
- **内存获取:** 根据模型给出的推荐列表，自动获取相关内存地址的内容，以备后续分析使用。

6.4.3 RAG 模块

RAG 模块负责将栈调用情况输入搜索引擎，搜索相关内容，并将相关结果输入模型中。该模块的设计如下：

- **栈调用信息提取:** 从信息传递和格式解析模块接收到的数据中提取栈调用情况，包括函数调用关系、调用参数等信息。
- **搜索引擎查询:** 将提取到的栈调用信息作为关键词，输入到搜索引擎中进行查询，获取与崩溃原因相关的文档和资料。
- **结果整合:** 将搜索引擎查询结果整合并筛选，提取出与问题相关的信息，并输入到模型中进行进一步分析和处理。

6.4.4 文本到解决方案转换模块

文本到解决方案转换模块负责将输出的模型分析得到的可能的错误位置整理后返回给用户。该模块的设计如下：

- **模型分析:** 接收并分析上述模块输出的结果，包括内存获取模块和 RAG 模块的输出内容。
- **错误位置整理:** 整理模型分析得到的可能的错误位置，包括故障发生的原因、可能影响的范围等信息，并将其格式化以便用户理解。
- **结果返回:** 将整理后的错误位置信息返回给用户，以供用户进行进一步的故障诊断和处理。

7 在线交互式用户反馈增强学习设计

本文档详细描述了在线交互式用户反馈增强学习系统的设计，该系统旨在通过收集用户反馈和历史调用记录，不断优化多文件协同代码优化处理的性能。系统将特别强调数据库的设计，以确保有效存储和利用历史数据，提升下游任务的输出质量和用户满意度。

7.1 设计目标

该系统设计的主要目标包括:

- 收集并存储用户反馈和系统调用历史，建立一个丰富的数据资源库。
- 使用历史数据来增强 LLM 模型的调用效果，使其更加符合用户的具体需求。
- 实现一个高度适应性的系统，可以根据用户的实时反馈进行自我优化。
- 提供一个用户友好的界面，使用户可以轻松地提供反馈并看到其反馈如何影响系统行为。

7.2 功能顺序图

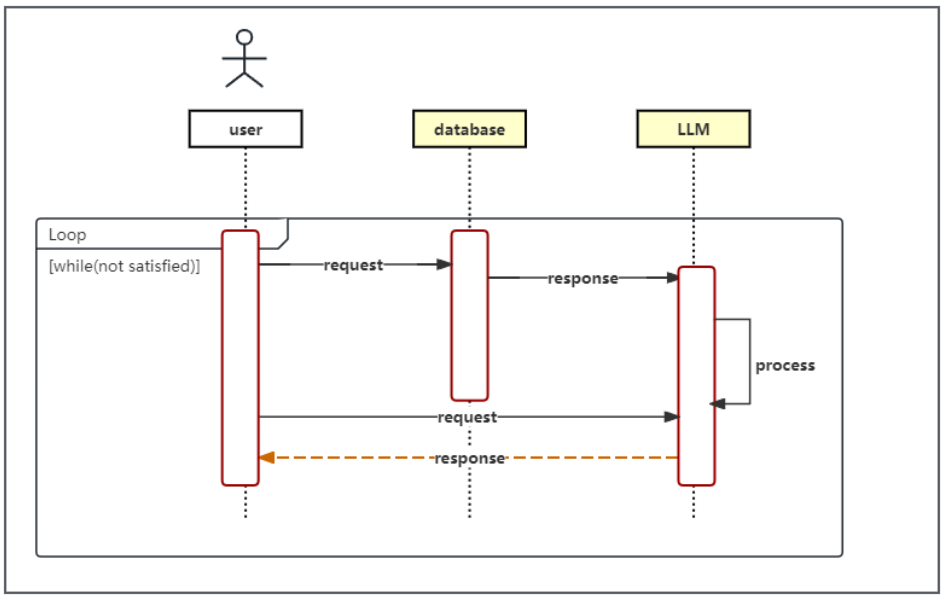

图 7.12: 在线交互式用户反馈增强学习

如图，给出了在线交互式用户反馈增强学习部分的顺序图。系统收集用户反馈和历史调用记录，存入数据库，之后用户再跟 LLM 发送请求时，将读取数据库中内容，即用户与 LLM 的交互记录，并结合用户的特定请求进行响应。

7.3 任务功能状态图

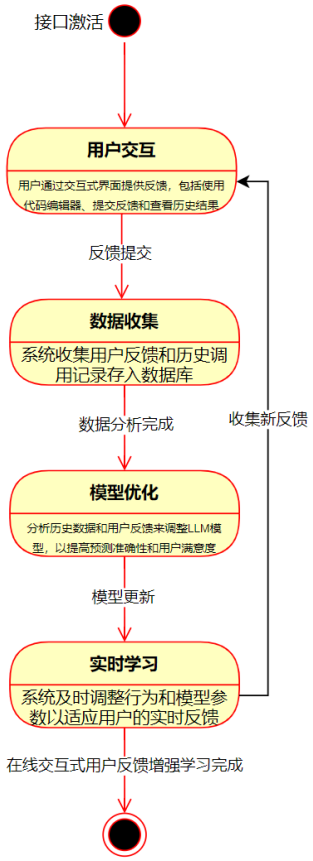

图 7.13: 在线交互式用户反馈增强学习状态图

它从“用户交互”开始，用户在此输入反馈，并触发系统的下一步动作；接下来是“数据收集”阶段，系统在这里收集用户的输入并保存到数据库；随后进入“模型优化”状态，系统分析数据库中的数据并更新 LLM 模型以改善预测效果；最后，系统进入“实时学习”状态，在这里，系统根据用户的即时反馈进行自我调整，实时优化模型响应；整个流程结束时，系统等待下一轮用户交互，以此继续学习和自我完善的循环。

7.4 系统概述

该设计涵盖以下主要组成部分：

- 用户界面设计：用于收集用户反馈和展示优化结果。
- 数据库设计：用于存储用户反馈和调用历史。
- 模型增强逻辑：利用收集的数据改善模型预测的逻辑。

7.5 用户界面和反馈机制

7.5.1 用户界面设计

用户界面（UI）设计旨在为用户提供一个直观、易用的平台，用于输入反馈和查看系统对其输入的响应。UI 应包括以下特点：

- 交互式代码编辑器，支持高亮显示和代码提示。
- 反馈输入界面，允许用户简单地点击或填写表单来提供对系统建议的反馈。
- 历史反馈和结果查看区域，使用户能够查看过去的输入和系统的响应，以及这些交互如何改变了系统行为。

7.5.2 反馈机制设计

为了有效收集和利用用户反馈，设计一个结构化的反馈系统非常关键。这应包括：

- 反馈分类：用户可以对不同类型的系统输出（如错误检查、风格建议）提供反馈。
- 反馈强度：用户不仅可以标记反馈的内容，还可以评价其重要性或紧急程度，帮助系统优先处理。
- 自动反馈学习：系统应能从用户的隐式行为（如接受/忽略建议的频率）学习，无需显式输入。

以上部分设计确保了用户可以有效地与系统互动，同时系统可以从这些互动中学习并自我优化。

7.6 数据库设计

7.6.1 数据库架构

为了支持多文件协同代码优化处理及其下游任务，设计一个高效且可扩展的数据库架构至关重要。我们参考 OpenAI 和 Ollama 的内容，设计如下数据库：

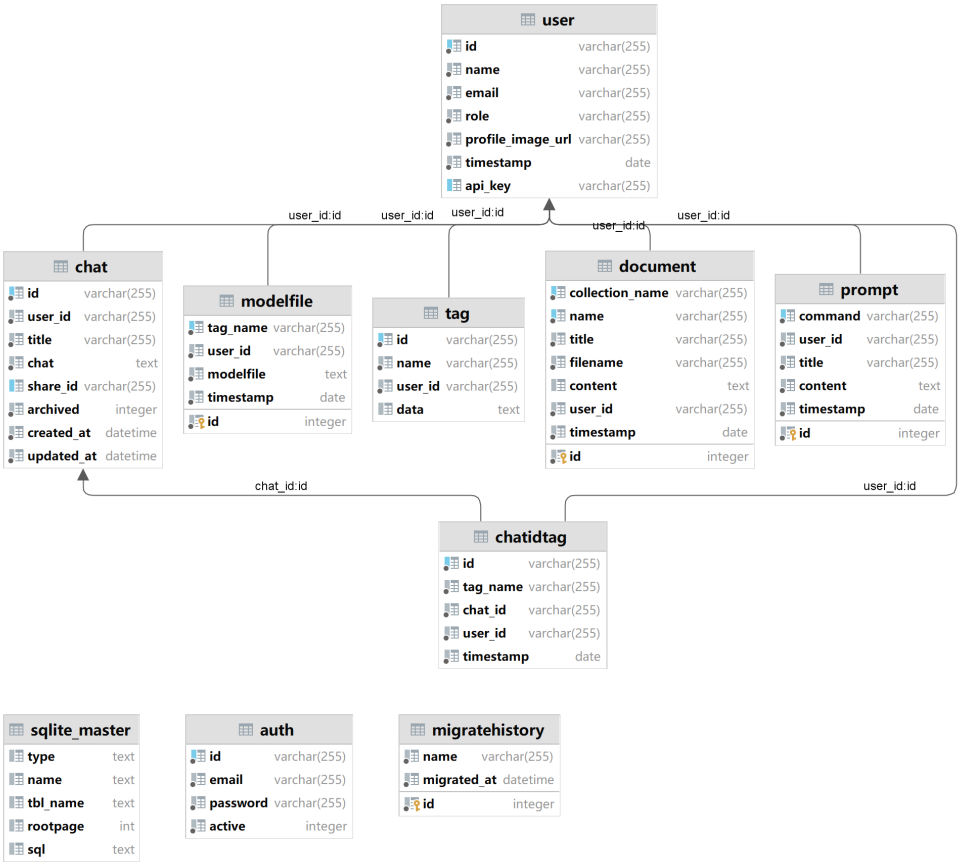

图 7.14: 数据库设计图

对图中各个表的解释如下：

表 7.1: **user（用户表）**

序号	属性名	说明	字段类型
1	id	用户 id	varchar(255)
2	name	用户名	varchar(255)
3	email	用户邮箱	varchar(255)
4	role	用户角色（管理员或用户）	varchar(255)
5	profile_image_url	用户头像	varchar(255)
6	timestamp	用户创建时间	date
7	api_key	用户 API 密钥	varchar(255)

表 7.2: **chat（对话记录表）**

序号	属性名	说明	字段类型
1	id	对话 id	varchar(255)
2	user_id	用户 id	varchar(255)
3	title	对话标题	varchar(255)
4	chat	对话内容，以 json 形式存储	text
5	share_id	对话分享 id	varchar(255)
6	archived	是否归档	integer
7	created_at	对话创建时间	datetime
8	updated_at	对话最近更新时间	datetime

表 7.3: **modelfile（模型文件表）**

序号	属性名	说明	字段类型
1	tag_name	模型标签名称	varchar(255)
2	user_id	用户 id	varchar(255)
3	modelfile	模型文件路径	text
4	timestamp	模型文件创建时间戳	date
5	id	模型 id	integer

表 7.4: **tag（标签表）**

序号	属性名	说明	字段类型
1	id	标签 id	varchar(255)
2	name	标签名	varchar(255)
3	user_id	用户 id	varchar(255)
4	data	内容	text

表 7.5: **document（文件映射表）**

序号	属性名	说明	字段类型
1	collection_name	文件编号	varchar(255)
2	name	文件命名	varchar(255)
3	title	文件标题	varchar(255)
4	fielname	源文件名	varchar(255)
5	timestamp	创建时间	date

表 7.6: **prompt（提示词表）**

序号	属性名	说明	字段类型
1	command	提示词命令	varchar(255)
2	user_id	用户 id	varchar(255)
3	title	提示词标题	varchar(255)
4	content	提示词内容	text
5	timestamp	创建时间	date
6	id	提示 id	integer

表 7.7: **chatidtag（对话标签）**

序号	属性名	说明	字段类型
1	id	关系 id	varchar(255)
2	tag_name	标签名	varchar(255)
3	chat_id	对话 id	varchar(255)
4	user_id	用户 id	varchar(255)
5	timestamp	创建时间	date

以下为自动创建的表：

表 7.8: **sqlite_master（master 表）**

序号	属性名	说明	字段类型
1	type	项目类型	text
2	name	项目名称	text
3	tbl_name	从属表名	text
4	rootpage	项目在数据库中的页编号	int
5	sql	创建该项目的 SQL 语句	text

表 7.9: **auth（用户表）**

序号	属性名	说明	字段类型
1	id	用户 id	varchar(255)
2	email	用户邮箱	varchar(255)
3	password	用户密码	varchar(255)
4	active	是否活跃（在线）	integer

表 7.10: **migratehistory（迁移历史表）**

序号	属性名	说明	字段类型
1	name	迁移名	varchar(255)
2	migrated_at	迁移时间	datetime
3	id	迁移 id	integer

7.6.2 数据管理策略

为确保数据的有效管理和利用，采取以下策略：

- **数据完整性**：实施一系列约束和触发器以维护数据的准确性和一致性。
- **安全性**：采用加密存储敏感信息，如用户个人信息，并实施访问控制策略以限制数据访问。
- **备份与恢复**：定期备份数据库，确保在数据丢失或损坏时可以迅速恢复。

这一部分的设计旨在为系统提供一个坚实的数据基础，支持复杂的数据查询、更新和存储操作，同时确保数据的安全和可靠。

7.7 模型增强逻辑

7.7.1 历史数据的利用

利用历史数据来增强模型预测包括以下关键步骤：

- **数据挖掘**：对历史记录表和反馈表中的数据进行挖掘，识别常见的用户需求和问题点。
- **模型训练**：使用用户的反馈和历史互动数据来微调 LLM 模型，使其输出更加符合用户的期望和需求。
- **动态更新**：根据新收集的数据定期更新模型，确保模型的输出始终保持最新和最有效。

7.7.2 实时反馈学习

实施实时反馈学习机制，使系统能够即时调整其行为以适应用户的反馈，包括：

- **快速迭代**：系统设计为可快速迭代和更新，能够即时应用新的用户反馈到模型调整中。
- **用户反馈循环**：建立一个闭环反馈系统，即用户的每次反馈都会被用来改善后续的系统表现。

这一部分设计的目的是确保系统不仅能够处理和响应用户的反馈，还能从中学习和自我完善，从而不断提升用户满意度和系统的整体效率。