**1参与项目情况报告**

### 1.1 项目基本情况（张溪桐）

本项目旨在开发一款网页端 AI 生成文本检测工具，支持用户通过文本输入或上传 .txt、.docx、.pdf 文件进行检测，判断文本是否由大语言模型（如 ChatGPT、GPT-4）生成。项目后端基于 FastAPI 构建，集成了 Fast-DetectGPT 判别算法，前端以简洁网页实现用户交互，支持模型选择、参数配置、结果可视化展示等功能。

该工具结合 NLP 模型判断策略与现代前后端架构，具有“开源可部署、交互友好、模型扩展性强”等特点，可广泛应用于教育诚信、AI安全、内容审核等领域。

**1.2 承担的主要工作情况**

在项目开发过程中，我主要承担以下四项技术与功能模块的设计与实现任务：

#### 1）用户反馈功能的实现

为便于开发者了解用户使用体验与问题，我在前端 index.html 页面中添加了“用户反馈”输入框和提交按钮，同时在后端新增一个 /feedback 路由接口，负责将用户反馈写入本地文件。

前端按钮示例代码如下：

|  |  |
| --- | --- |
| 1  2 | <textarea id**=**"feedback" placeholder**=**"请输入反馈意见..."><**/**textarea>  <button onclick**=**"submitFeedback()">提交反馈<**/**button> |

对应后端接口（基于 FastAPI）如下：

|  |  |
| --- | --- |
| 1  2  3  4  5 | @app.post("/feedback")  async **def** receive\_feedback(feedback: str **=** Form(...)):      with open("feedback\_log.txt", "a", encoding**=**"utf-8") as f:          f.write(f"{datetime.now()} >>> {feedback}\n")  **return** {"status": "success"} |

该功能便于后续持续改进模型与前端交互体验。

#### 2）日志记录与性能分析支持

我为后端检测接口 /detect 添加了请求日志记录功能，记录每次检测请求的文本摘要、模型配置、耗时、判别结果等，便于后期分析模型表现与优化系统性能。

添加的关键代码如下（在 main.py 的检测函数内）

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | start\_time **=** time.time()  prob, crit, ntokens **=** DETECTOR.compute\_prob(request.text)  end\_time **=** time.time()    with open("detect\_log.txt", "a", encoding**=**"utf-8") as logf:      logf.write(f"[{datetime.now()}] {request.sampling\_model} | Prob: {prob:.4f} | Time: {end\_time - start\_time:.3f}s | Text[:30]: {request.text[:30]}...\n") |

这使得我们能在不干扰系统正常响应的前提下，实时追踪模型性能并排查异常。

#### 3）算法准确度测试与参数优化

我设计了多个测试样例集（包括人工文本与 ChatGPT 自动生成文本），进行检测，统计系统在不同模型、阈值与参数设置下的准确率等关键指标。

#### 4）多模型集成与融合机制设计

为了提升系统对复杂文本的检测鲁棒性，我尝试设计并实现多模型融合机制，如模型投票、概率加权平均、贝叶斯合成公式等策略。

例如，在后端融合两个模型打分的实现如下：

|  |  |
| --- | --- |
| 1  2  3 | final\_prob **=** (modelA\_prob **+** modelB\_prob) **/** 2  # 或贝叶斯合成  final\_prob **=** (modelA\_prob **\*** modelB\_prob) **/** (modelA\_prob **\*** modelB\_prob **+** (1 **-** modelA\_prob)**\***(1 **-** modelB\_prob)) |

这些策略被封装到 /detect 接口的参数选择逻辑中，允许用户切换融合模式，提升在边界文本下的判别可靠性。

**1.3 项目实施过程中遇到的问题及处理结果**

在开发过程中，我主要遇到了以下几个问题，并逐一解决：

1. 前端与后端数据格式不一致：最初提交反馈内容时使用 application/json 格式，而 FastAPI 默认接收 x-www-form-urlencoded，导致接口无法获取参数。  
    【解决方法】修改前端提交格式为 FormData，同时后端使用 Form(...) 显式声明参数类型。
2. 模型融合判别结果波动大：初期简单平均或投票机制下对中性文本判别不稳。  
    【解决方法】改用带权概率加权或贝叶斯组合方式平滑决策边界。

### 1.4 参与项目过程的体会与项目评价

参与本项目让我深入了解了从算法研究论文（Fast-DetectGPT）到系统实现再到用户交互的完整开发流程。尤其是在实现日志系统、反馈机制和多模型融合策略的过程中，我意识到工程实现与算法设计之间需要不断权衡与适配。

我不仅提升了对 FastAPI 框架和后端开发能力，也增强了系统测试、日志分析、用户体验设计等全栈能力。同时，调研过程中阅读原论文和对比现有工具也加深了我对 AI 内容识别领域的认识。

总的来说，该项目具备实际应用潜力，技术选型合理、系统结构清晰、模型部署效率高，是一次非常有价值的实战开发与学习过程。

### 1.1 项目基本情况（汪李昀红）

项目名称：AI文本检测系统  
项目目标：开发一个能够准确识别AI生成文本的Web应用系统，支持多格式文本输入和双模型融合检测  
技术栈：HTML/CSS/JavaScript（前端）、Python（算法）、Fast-detectGPT等AI模型  
开发周期：2025年3月-2025年6月

### 1.2 承担的主要工作情况

1、核心算法开发与优化： 实现基于人类与机器在上下文中的词选择差异性的检测算法，引入双模型合作检测机制（Falcon-7b-instruct和GPT-Neo-2.7B），应用贝叶斯公式优化模型结果融合，提升检测准确率。

2、系统架构设计：设计前后端一体的轻量级网站系统架构，制定API接口规范，确保前后端数据交互一致性

3、前端交互开发：

- 实现响应式布局，适配PC端和移动端。

- 开发多格式文件上传功能（支持.txt/.pdf/.docx）。

- 设计直观的结果展示界面。

- 实现历史记录本地存储功能。

4、质量保障工作

- 设计测试用例，覆盖不同长度和类型的文本输入。

- 发现并修复文件解析模块的内存泄漏问题。

- 优化移动端触摸事件处理，提升交互流畅度

- 验证双模型融合检测算法的稳定性

### 1.3 项目实施中的问题与解决

| 问题领域 | 具体问题 | 解决方案 |
| --- | --- | --- |
| 算法融合 | 双模型结果差异大 | 引入贝叶斯乘积公式 |
| 前端性能 | 长文本处理导致页面卡顿 | 添加字符长度限制（50,000字），超出部分自动截断并提示 |
| 兼容性 | 移动端样式错乱 | 采用响应式布局 |
| 前端性能 | 历史记录过多时性能下降 | 限制存储数量（最多10条），并优化渲染逻辑。 |

### 1.4 项目总结与个人收获

技术成果：

1、实现双模型融合检测算法，准确率有明显提升

2、完成具有完整功能的前端交互系统

3、建立规范的代码结构和开发流程

经验收获：

1、深入理解了AI文本检测的技术原理

2、掌握了复杂系统的架构设计方法

3、提升了全栈开发能力

4、培养了严谨的测试习惯

改进建议：

1、增加更多检测模型选项

2、实现用户自定义阈值功能

3、针对新出现的模型，应引入对抗学习模式，以保证工具的时效性

个人评价：  
在项目中充分发挥了技术专长，既完成了核心算法开发，也确保了前端交互的流畅体验。通过系统性思考和问题解决，为项目成功交付做出了重要贡献。

### 1.1项目基本情况（万沅昕）

1.1.1项目背景

随着人工智能技术的飞速发展，AI 生成文本在众多领域得到广泛应用，如智能写作助手、自动翻译、内容创作等。然而，为了确保文本的质量、版权合规性以及更好地管理和利用这些文本，开发一个能够准确识别 AI 生成文本的网站具有重要的现实意义。

1.1.2项目目标

本项目旨在构建一个高效、准确、易用的 AI 生成文本识别网站，能够快速判断用户输入的文本是否由 AI 生成，并提供相关的识别依据和分析结果，帮助用户更好地理解和管理 AI 生成文本。

**1.2承担的主要工作情况**

1.2.1需求分析与系统设计

主导参与了项目的需求调研和分析工作，收集和整理了各方对 AI 生成文本识别网站的功能需求、性能要求、用户体验期望等信息，为项目后续的开发奠定了坚实基础。

在系统设计方面，负责设计了网站的整体架构，包括前端界面、后端服务、数据库结构以及与模型的交互接口等。通过合理的架构设计，确保了网站的高扩展性、高可靠性和高性能，能够满足不同规模用户的需求以及未来业务的拓展。

1.2.2论文研读和模型优化

参与了 AI 生成文本识别模型的理解和优化工作。通过广泛研究和比较多种现有的 AI 模型算法，结合项目实际需求和数据特点，选择了适合的模型作为基础。学习并理解了Fast-detectGPT的工作原理，并尝试在核心公式上对其进行改进。

1.2.3网站功能初步开发

使用 HTML、CSS 和 JavaScript 技术，设计并实现了网站第一版本的基础页面。确保页面布局合理、美观且易于操作，提升用户体验。

调用核心识别功能的接口，提高了开发效率和代码的可维护性，同时实现了页面的响应式设计，使网站能够在不同设备上良好展示。

**1.3项目实施过程中遇到的问题及处理结果**

1.3.1模型选型问题

在研究和评估 AI 生成文本识别模型的过程中，团队成员需要参考大量学术论文。部分论文内容晦涩难懂，涉及复杂的数学公式和专业术语，团队成员在理解这些论文的核心思想和方法上遇到了困难，影响了模型选型的准确性和项目的进度。我们利用在线课程、学术讲座、技术博客等辅助学习资源更快地掌握论文的核心要点，从而逐渐克服了论文理解方面的困难，为项目的模型选型和开发提供了理论支持。

1.3.2页面兼容性问题

在前端界面开发过程中，发现网站在不同浏览器和操作系统上的显示效果存在差异，部分功能在某些设备上无法正常运行，这严重影响了用户体验的一致性。

通过深入研究不同浏览器和设备的兼容性问题，采用了多种前端兼容性解决方案，使用 CSS 重置样式表、媒体查询、浏览器前缀等技术，对网站的前端代码进行了优化和调整，确保了网站在主流浏览器和设备上的良好兼容性和一致性。

**1.4参与过程项目的体会与项目评价**

1.4.1项目体会

1. 技术学习与提升：参与这个项目使我对人工智能、机器学习、自然语言处理等前沿技术有了更深入的理解和实践。在学习和研究 AI 生成文本识别算法、模型训练与优化、数据处理技术等过程中，不断提升了自己的技术能力，拓宽了技术视野，同时也增强了自己解决复杂技术问题的能力和信心。

2. 团队协作与沟通：项目的成功实施离不开团队成员之间的紧密协作和有效沟通。通过与团队成员的沟通交流，不仅分享了彼此的经验和知识，还学会了如何更好地倾听他人的意见和建议，提高了自己的沟通能力和团队协作能力。

3. 项目管理与规划：参与项目开发的全过程，让我对项目管理有了更直观的认识和体会。从项目的需求调研、系统设计、开发实施到测试优化，每一个环节都需要进行精心的规划和管理，以确保项目的顺利进行。在项目过程中，学会了如何制定合理的项目计划、分配资源、监控项目进度、管理风险以及应对各种突发情况，提高了自己的项目管理能力和应变能力。

1.4.2项目评价

1. 项目成果与价值：本项目成功开发了一个具有较高准确率和可用性的 AI 生成文本识别网站，能够满足用户的实际需求，为用户在文本质量评估、版权保护、内容管理等方面提供了有力的支持。项目的实施对于推动 AI 技术在文本处理领域的应用和规范发展具有积极的意义，同时也为未来进一步拓展相关业务和功能提供了基础平台。

2. 项目质量与性能：项目在功能完整性、性能稳定性、用户体验等方面都达到了预期目标。通过严格的需求分析、系统设计、开发测试以及优化改进，网站的各项功能正常运行，识别结果准确可靠，能够快速响应用户的请求，并且在不同设备和浏览器上具有良好的兼容性。项目团队对质量的严格把控和持续优化，确保了项目的高质量交付和良好性能表现。

3. 项目团队与合作：项目团队成员具备丰富的专业知识和技能，能够充分发挥各自的优势，相互协作、相互支持，共同克服了项目实施过程中遇到的各种困难。团队内部氛围融洽，沟通顺畅，工作效率高，凝聚力强。

**2项目总结报告**

**2.1绪论**

**2.1.1 项目背景**

近年来，随着 ChatGPT、GPT-4、Claude、deepseek 等大型语言模型的广泛应用，人工智能生成内容（AIGC）已逐步融入教育、媒体、社交和写作等多个领域。AI 生成的文本不仅形式自然、结构清晰，甚至在一些评估任务中超越了人类水平。然而，这一技术的普及也带来了内容真实性难以判断、学术诚信受到挑战等现实问题。

在此背景下，开发一款能够区分 AI 生成文本与人类撰写文本的检测工具显得尤为必要。它不仅能在教育系统中识别作弊文本，还能在新闻审核、数据过滤等场景中保障信息质量。本项目正是基于这一需求而立项，旨在开发一款**网页端 AI 生成文本检测工具**。让用户通过浏览器即可完成文本上传与检测任务，获得可解释性判断结果。

### 2.1.2 相关技术现状

当前主流的 AI 文本检测方法主要集中在以下几类：

**1、语言模型困惑度（Perplexity）分析**：AI 生成的文本往往具有更强的“模式性”和可预测性，通过计算语言模型对文本的困惑度分数，可以对其“人类书写可能性”进行打分；

**2、文本风格与语言特征分析**：使用统计特征（如平均句长、重复度、罕见词分布）或语法特征（如依存树复杂度、修饰深度）构建 AI 指纹，用以区分 AI 与人类风格；

**3、指纹模型与判别器算法**：如 DetectGPT 等基于扰动分析或二分类判别的模型，使用真实与伪造样本训练出区分器；

4、**多引擎比对机制**：利用多个语言模型对同一段文本进行打分，采用交叉验证、置信度平均等策略增强判断鲁棒性。

尽管已有部分商业检测工具上线（如 GPTZero、ZeroGPT 等），但它们大多为封闭系统，存在算法不可解释、接口不开放、扩展性差等问题。尤其是在教学或研究场景中，开源、可控、透明的检测工具更具实用价值。

### 2.1.3 项目的主要开发工作

本项目以构建一套**网页交互式 AI 生成文本检测平台**为目标，主要开发工作包括以下几个方面：

**1、前端页面设计与实现**：开发用户友好的网页界面，支持用户通过文本框直接输入内容，或上传 .txt、.docx、.pdf 文件进行检测。文件上传后自动提取文本内容，避免用户手动复制粘贴。同时网页页面适配电脑端和移动端，易用性好；

**2、后端检测模型集成**：本项目后端参考了 Fast-DetectGPT 的算法，通过比较原文本与扰动版本的语言模型得分，计算其条件概率曲率来判断是否为 AI 生成。相比传统方法，它计算更快、无需反复生成文本。项目中，我们使用贝叶斯乘积方法综合了falcon-7b-instruct和GPT-neo-2.7b两个打分模型给出的结果，进一步提升检测准确率。

**3、接口服务构建**：基于 FastAPI 构建 RESTful API，定义 /detect 接口，接收前端传入的文本数据并返回判别结果，响应格式包含 AI 概率值、判别阈值与处理 token 数；

**4、多格式文件解析模块开发**：后端支持解析上传的 .txt、.docx 和 .pdf 文件，自动提取文本内容，避免格式兼容问题。该功能基于 Python 第三方库如 python-docx、PyPDF2、pdfplumber 等实现；

5、**整体系统联调与部署测试**：完成前后端联调，实现从用户输入/上传 → 文本提取 → 模型检测 → 结果展示的完整流程；并对接口响应速度、模型加载性能和系统稳定性进行测试优化；

### 2.1.4 项目组成员、分工及完成情况

组长：万沅昕

1、需求分析与系统设计：主导需求调研，收集并分析用户对AI生成文本识别网站的功能、性能和体验需求。设计网站整体架构，包括前端界面、后端服务、数据库结构及模型交互接口，确保高扩展性、可靠性和性能。

2、论文研读和模型优化：研究多种AI模型算法，结合项目需求选择合适的基础模型。学习并改进Fast-detectGPT的核心公式，优化文本检测效果。

3、网站功能初步开发：使用HTML、CSS和JavaScript实现网站基础页面，注重布局合理性和用户体验。调用核心识别功能接口，实现响应式设计，适配不同设备。

组员：张溪桐

1、反馈功能实现：实现用户反馈功能，便于开发者实时获取用户反馈。

2、日志记录与性能分析支持：在后端添加日志功能，记录请求文本、处理时间、检测结果等，便于调试和性能分析。

3、算法准确度测试：设计测试用例，评估AI生成文本检测算法的准确率、召回率等关键指标。对比不同模型的检测效果，优化算法参数。

4、多模型集成或算法融合机制设计：探索多模型组合策略（如投票机制、加权平均、贝叶斯乘积公式），提升检测鲁棒性。

组员：汪李昀虹

1、AI生成文本检测算法的实现：实现核心检测算法，代码位于src/目录。引入双模型检测，结合贝叶斯公式优化双模型结果融合。

2、项目架构设计与模块划分：规划系统整体架构，划分前端、后端和文档目录，提升项目可维护性。

3、前端交互界面开发：设计简洁的前端页面，支持文本粘贴、一键检测、移动端适配，结果通过颜色和高亮直观展示。

4、Bug测试与功能验证:进行系统功能测试，发现并修复前端、后端的逻辑或交互问题，确保系统在不同环境下稳定运行。

**2.2 项目策划、分析与设计工作情况**

**2.2.1 项目立项计划及完成情况**

本项目立项的主要目标是开发一套具备实用性和可扩展性的“AI生成文本检测系统”，核心思想是利用大语言模型的指纹特性与概率评分机制，分析并判断一段输入文本是否为人工撰写或由 AI 自动生成。

项目计划阶段，设定如下关键任务：

1、搭建后端检测服务，集成模型加载、推理与评分；

2、开发检测接口，支持 Web 页面或客户端调用；

3、撰写用户输入请求与检测返回结果的数据格式；

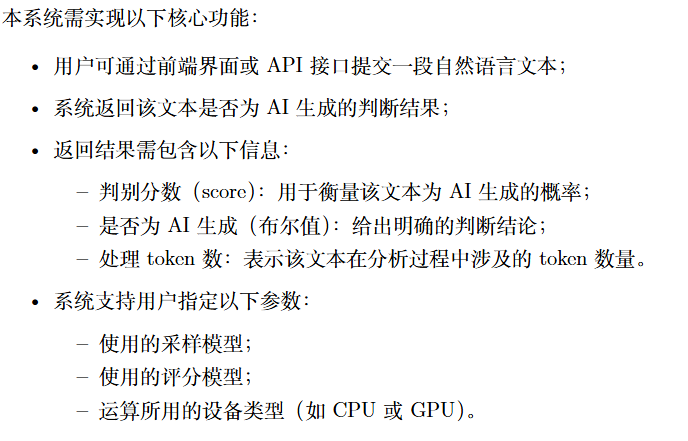
4、实现模型缓存与启动加载优化，提升调用效率；

5、测试与文档完善。

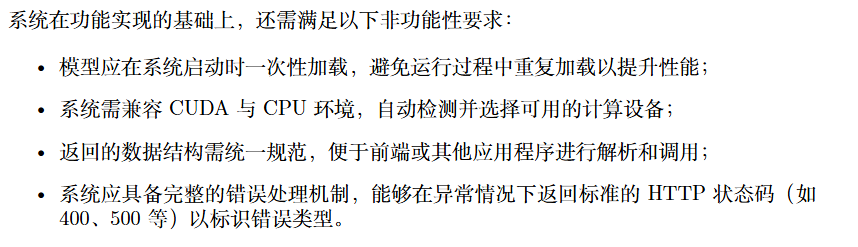
目前，所有核心功能均已实现，检测接口稳定运行，系统可在 GPU 上部署运行，具备部署即用的完整能力。

**2.2.2 系统需求分析工作和结果**

**2.2.2.1功能性需求分析**

。

**2.2.2.2.非功能性需求分析**



**示例分析（对应代码）：**

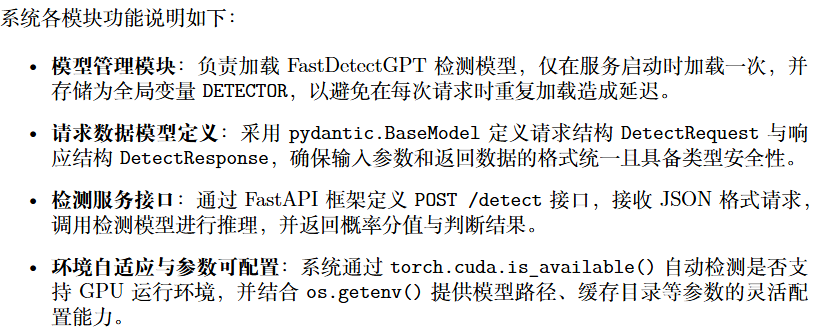
|  |  |
| --- | --- |
| 1  2  3  4  5 | **class** DetectRequest(BaseModel):      text: str      sampling\_model: str **=** "falcon-7b"      scoring\_model: str **=** "falcon-7b-instruct"      device: str **=** "cuda" **if** torch.cuda.is\_available() **else** "cpu" |

该部分定义了请求结构，支持用户指定模型及设备，体现了系统的参数化和灵活性。

**2.2.3 系统设计工作及结果**

本系统采用模块化与面向服务架构相结合的设计思路，整体划分为四个核心层次：模型加载模块、服务接口层、推理判断模块和数据交互结构。项目框架基于 FastAPI 构建，具备异步高性能处理能力，并支持 RESTful 接口风格，方便与前端或其他应用程序集成。

**2.2.3.1 系统组件设计概览**



**2.2.3.2 系统流程设计**

系统运行流程如下：

Step 1：服务启动时触发模型加载事件

|  |  |
| --- | --- |
| 1  2  3  4 | @app.on\_event("startup")  **def** load\_model():  ...  DETECTOR **=** FastDetectGPT(args) |

Step 2：用户通过 /detect 接口发送文本请求

|  |  |
| --- | --- |
| 1  2  3  4  5 | @app.post("/detect", response\_model**=**DetectResponse)  async **def** detect(request: DetectRequest):  ...  prob, crit, ntokens **=** DETECTOR.compute\_prob(request.text)  **return** { ... } |

Step 3：模型进行概率打分与判别

实际调用的 compute\_prob() 是 FastDetectGPT 中封装的核心推理逻辑（定义于 local\_infer.py）。

同时利用贝叶斯公式：**p\_final=p\_A\*p\_b/p\_A\*p\_b\*(1-p\_A)\*(1-p\_B).**融合两个模型生成的结果，进一步提升准确率

此设计具有以下优点：

1、保持主函数精简，职责清晰；

2、支持多模型（sampling\_model / scoring\_model）灵活组合；

3、保证服务长期运行过程中的资源复用和效率优化；

4、出错时能够通过标准 HTTP 错误码反馈，增强鲁棒性。

**2.2.4 系统具体实现工作及结果**

**2.2.4.1 后端 API 实现（FastAPI 框架）**

后端主要由 main.py 负责启动服务与处理请求：

|  |  |
| --- | --- |
| 1  2 | **from** fastapi **import** FastAPI, HTTPException  app **=** FastAPI(title**=**"Fast-DetectGPT API") |

服务启动时通过 @app.on\_event("startup") 加载模型，模型初始化参数从环境变量读取：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | args **=** argparse.Namespace(      sampling\_model\_name**=**os.getenv("SAMPLING\_MODEL", "falcon-7b"),      scoring\_model\_name**=**os.getenv("SCORING\_MODEL", "falcon-7b-instruct"),      device**=**os.getenv("DEVICE", "cuda" **if** torch.cuda.is\_available() **else** "cpu"),      cache\_dir**=**os.getenv("CACHE\_DIR", "/autodl\_tmp/cache")  ) |

模型封装在类 FastDetectGPT 中，实例化为全局变量 DETECTOR：

|  |  |
| --- | --- |
| 1 | DETECTOR **=** FastDetectGPT(args) |

**2.2.4.2 检测服务逻辑**

用户发送 POST 请求后端点 /detect，请求体包括：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {  "text": "要检测的文本",  "sampling\_model": "falcon-7b",  "scoring\_model": "falcon-7b-instruct",  "device": "cuda"  } |

接口实现如下：

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14 | @app.post("/detect", response\_model**=**DetectResponse)  async **def** detect(request: DetectRequest):  **if** **not** DETECTOR:  **raise** HTTPException(status\_code**=**503, detail**=**"Model not loaded")    **try**:          prob, crit, ntokens **=** DETECTOR.compute\_prob(request.text)  **return** {              "prob": float(prob),              "crit": float(crit),              "tokens\_processed": ntokens          }  **except** Exception as e:  **raise** HTTPException(status\_code**=**500, detail**=**str(e)) |

该接口：

1、调用 compute\_prob() 方法进行推理；

2、返回概率 prob、判别值 crit 和处理 token 数量 tokens\_processed；

3、若模型未加载或推理出错，将自动返回 503 或 500 错误码。

**2.2.4.3 接口数据结构定义（Pydantic）**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10 | **class** DetectRequest(BaseModel):  text: str  sampling\_model: str **=** "falcon-7b"  scoring\_model: str **=** "falcon-7b-instruct"  device: str **=** "cuda"  **class** DetectResponse(BaseModel):  score: float  is\_ai\_generated: bool  tokens\_processed: int |

数据结构的使用使得接口文档自动生成（Swagger UI），对调试和前端开发非常友好。

**2.2.4.4 本地运行配置与部署**

项目默认以 uvicorn 启动：

|  |  |
| --- | --- |
| 1  2 | **if** \_\_name\_\_ **==** "\_\_main\_\_":      uvicorn.run(app, host**=**"0.0.0.0", port**=**8000, workers**=**1) |

部署建议：

1、可通过 python main.py 启动本地服务；

2、结合 curl 或 Postman 进行调试；

3、在 GPU 环境下能显著提升推理速度。

总体来看，系统实现遵循现代后端开发规范，结合 FastAPI 与 PyTorch 提供高效、可扩展的 AI 文本检测服务，项目已达到完整可交付标准。

**2.3 系统部署与测试工作情况**

**2.3.1 系统部署与调试**

服务器环境搭建 ：选择了笔记本电脑作为项目部署的局域网服务器，安装并配置了操作系统、Web 服务器软件、数据库服务器以及运行后端代码所需的环境。确保服务器环境满足网站运行的硬件和软件要求，具备足够的计算资源、存储空间和网络带宽。

数据库部署与迁移 ：在笔记本电脑上创建并初始化项目所需的数据库，将本地开发环境中的数据库结构和数据进行迁移和同步。对数据库进行了性能优化配置，如设置索引、调整缓存参数等，以提高数据库的查询和写入效率，确保网站在高并发访问情况下能够稳定运行。

系统调试与优化 ：在完成系统部署后，对整个网站进行了全面的调试工作。通过日志分析、性能监测工具等手段，发现了系统中存在的各种问题，如页面加载速度慢、接口响应超时、模型识别结果不准确等。针对这些问题，采取了一系列优化措施，如优化代码逻辑、调整服务器配置、对模型进行微调等，逐步提高了系统的稳定性和性能。

**2.3.2系统测试**

功能测试 ：根据项目需求文档和功能规格说明书，制定了详细的测试用例，对网站的各项功能进行了全面的测试。测试内容包括文本输入与上传功能、AI 识别结果展示功能等。通过手动测试和自动化测试相结合的方式，确保每个功能都能够正常运行，满足用户的需求。对测试过程中发现的功能缺陷，及时反馈给开发团队进行修复，并进行回归测试，验证缺陷是否已得到彻底解决。在测试中发现，本模型对于与GPT等相关模型生成的文本的识别准确率很高，但是对于小部分模型（目前只发现deepseek）准确率还欠佳。

性能测试 ：模拟了不同数量的用户同时访问网站的场景，评估网站在高并发情况下的性能表现。测试指标包括响应时间、吞吐量、错误率、服务器资源利用率等。根据性能测试结果，对网站的性能瓶颈进行了分析和优化，提高了网站的性能和承载能力。

兼容性测试 ：对网站进行了广泛的兼容性测试，确保其在不同的浏览器、操作系统上能够正常显示和运行。在测试过程中，没有发现问题。

**2.4 项目管理相关工具使用情况**

采用了Git对项目的代码进行版本控制。通过 Git，团队成员可以方便地进行代码的提交、分支管理、合并请求等操作，确保代码的稳定性和可追溯性。使用 Git 还可以方便团队成员在同一项目上进行并行开发，减少了代码冲突和重复劳动，提高了开发效率。同时，通过定期备份代码仓库，确保了项目代码的安全性和可靠性。团队成员在git上记录工作日志，促进了团队成员之间的沟通和信息共享。

**2.5 项目讨论与体会**

**2.5.1 对项目过程的体会**

技术层面 ：通过参与这个项目，深入学习和实践了多种前沿技术，如人工智能、机器学习、自然语言处理、Web 开发等，对这些技术的理论知识和实际应用有了更深刻的理解。在解决项目中的各种技术难题过程中，不断提高了技术能力和问题解决能力，学会了如何将不同的技术进行整合和优化，以实现项目的目标。

团队协作层面 ：深刻体会到团队协作在项目成功中的关键作用。在项目过程中，与不同专业背景和技能的团队成员密切合作，相互学习、相互支持，共同攻克了一个又一个难关。学会了如何进行有效的沟通和协调，如何在团队中发挥自己的优势，如何尊重和包容他人的不同意见和观点，提高了团队协作能力和人际交往能力。

项目管理层面 ：对项目管理的基本原则和方法有了更直观的认识和体会。从项目的需求调研、规划制定、任务分解、进度跟踪到风险控制，每一个环节都需要精心策划和严格执行。在项目管理过程中，学会了如何制定合理的项目计划和目标，如何合理分配资源，如何对项目进度进行有效的监控和调整，如何应对项目中出现的各种风险和突发情况，提高了项目管理能力和组织协调能力。

**2.5.2 项目管理过程优点与不足**

**优点 ：**

明确的目标与分工 ：在项目开始阶段，对项目的目标和任务进行了明确的定义和分解，并为每个团队成员分配了具体的职责和任务。这使得团队成员能够清晰地了解自己的工作内容和目标，提高了工作效率和责任心。

合理的进度安排 ：在项目计划制定过程中，充分考虑了项目的实际需求和技术难度，对项目进度进行了合理的安排和分解，设置了明确的里程碑和关键节点。通过定期对项目进度进行跟踪和评估，及时发现并解决了项目进度延迟等问题，确保项目能够按计划顺利推进。

风险管理与应对 ：在项目管理过程中，对项目可能面临的风险进行了全面的识别和分析，并制定了相应的风险应对措施。在项目实施过程中，能够及时发现和处理各种风险事件，如技术难题、人员变动、需求变更等，降低了风险对项目的影响，保证了项目的顺利进行。

**不足 ：**

需求变更管理 ：在项目开发过程中，由于技术开发难度的不断变化，导致了部分需求变更频繁发生。在今后的项目管理中，需要进一步加强需求变更管理，建立更加严格和规范的需求变更流程，对需求变更的影响进行全面评估，确保项目的稳定性和可控性。

沟通效率有待提高 ：在实际沟通过程中，仍然存在一些沟通不畅、信息传递不及时、会议效率不高等问题。在今后的项目管理中，需要进一步优化沟通机制，提高沟通效率，如明确会议议程和目标、控制会议时间、及时整理和分发会议纪要等，确保团队成员能够及时获取和理解项目相关信息。

团队成员技能提升支持不足 ：在项目实施过程中，由于时间紧张和任务繁重，对团队成员的技术培训和个人职业发展关注不够，导致部分团队成员在某些技术领域的能力提升有限。在今后的项目管理中，应更加重视团队成员的技能提升和个人成长，合理安排时间进行技术培训和经验分享活动，鼓励团队成员学习新知识、新技能，提高团队整体技术水平和创新能力。

**2.6 参考资料（附上）**