South China University of Technology

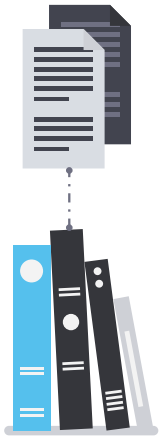
未来技术学院
FUTURE TECH

Smart Code Document Generator

——Overseas Masterclass Course Design

Group6 Members:
Zhang Yue, Chen Sihan,
Huang Ruiqi, Zhang Yimeng, Gu Haoyu

Table of contents



01

Introduction

Background

✅ Problem Statement:

Manual documentation is

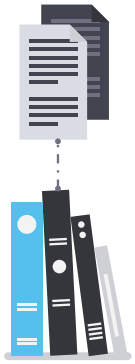
Time-consuming,
Repetitive,
and often Neglected.

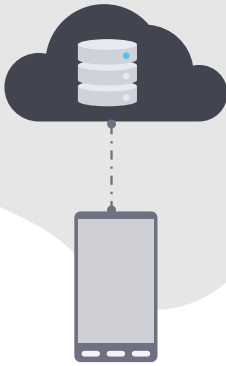
🚀 Our Motivation:

We aim to build an

Intelligent Documentation Generator

that automatically generate structured and readable documentation for code repositories.





South China University of Technology

Problem Statement

✅ Input

Complete code files,
Pasted code snippets,
Code from GitHub repositories.

✅ Output:

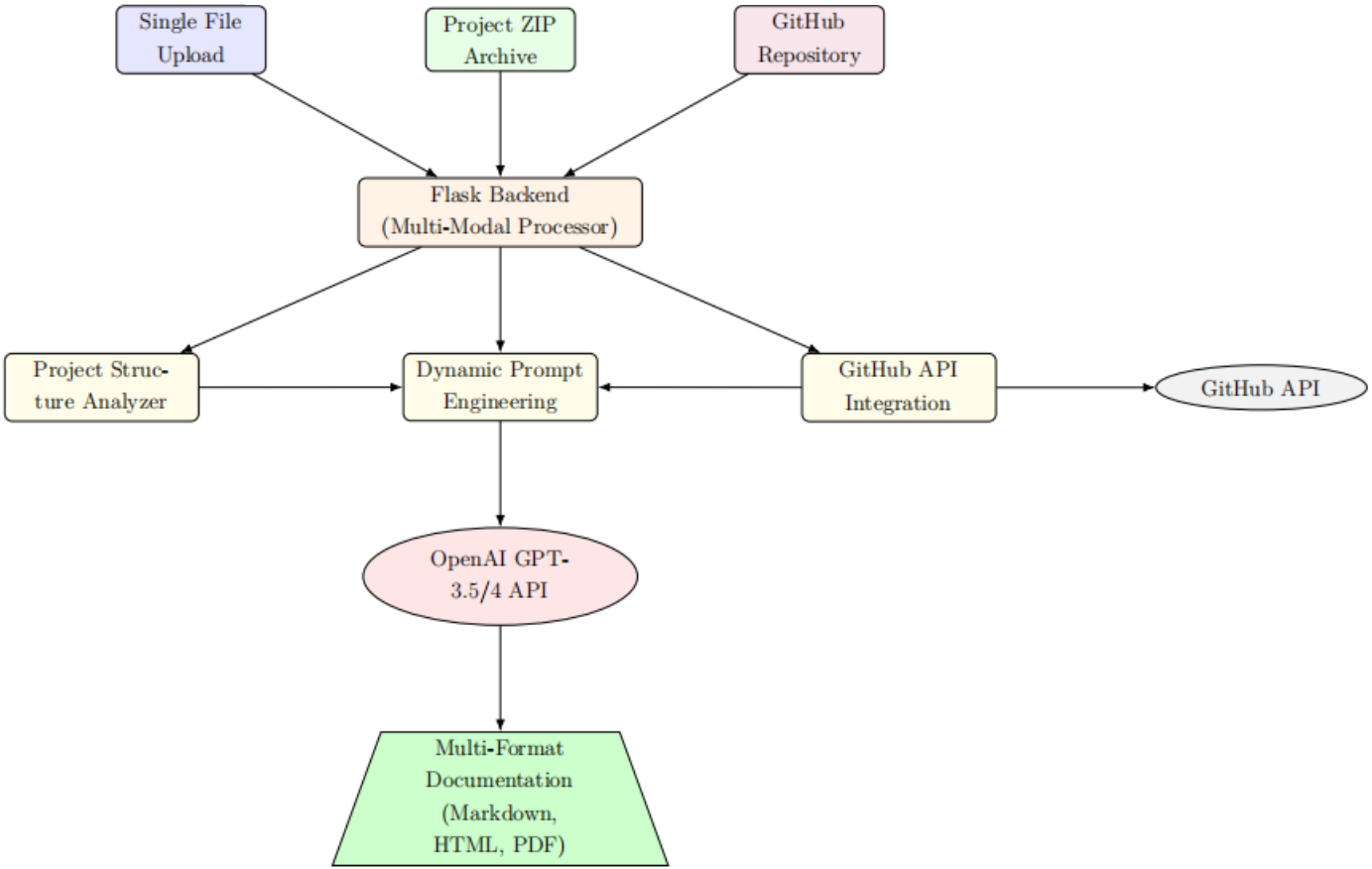
Clear and Structured
Code Documentation

02

System Design and Methodology

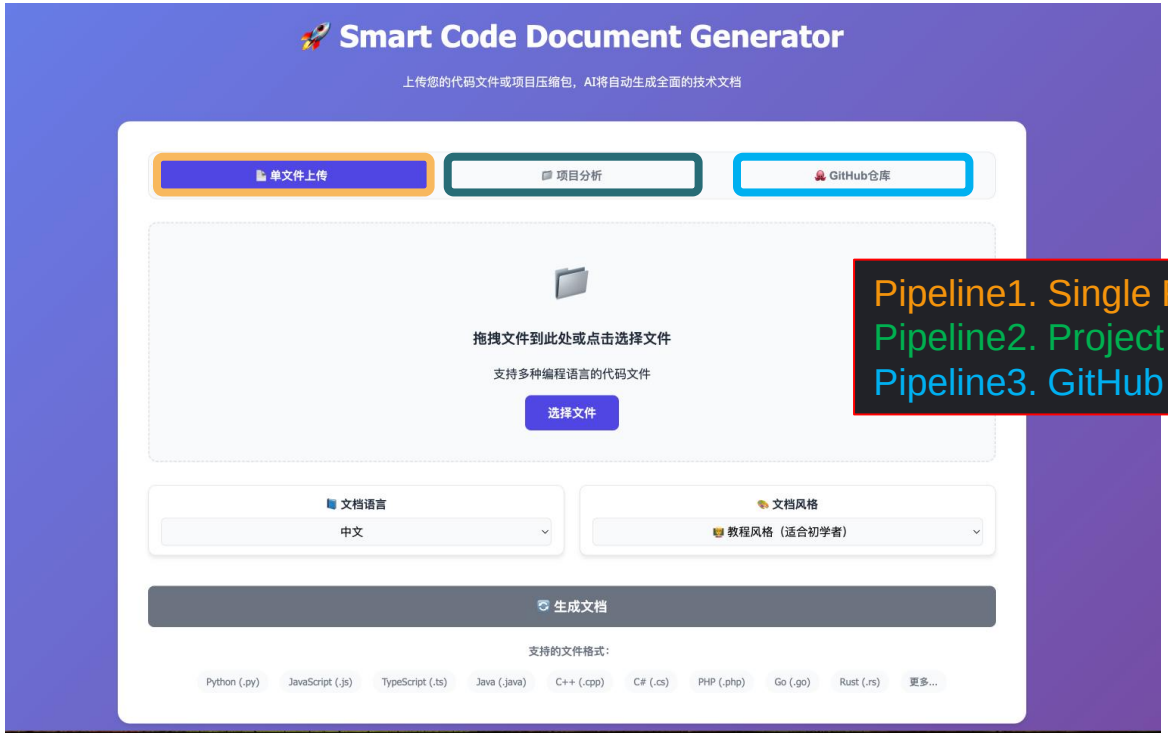
South China University of Technology

Enhanced Multi-Modal System Architecture



South China University of Technology

Advanced Multi-Modal Processing Pipeline



Pipeline1. Single File Processing
Pipeline2. Project Archive Analysis
Pipeline3. GitHub Repository Integration

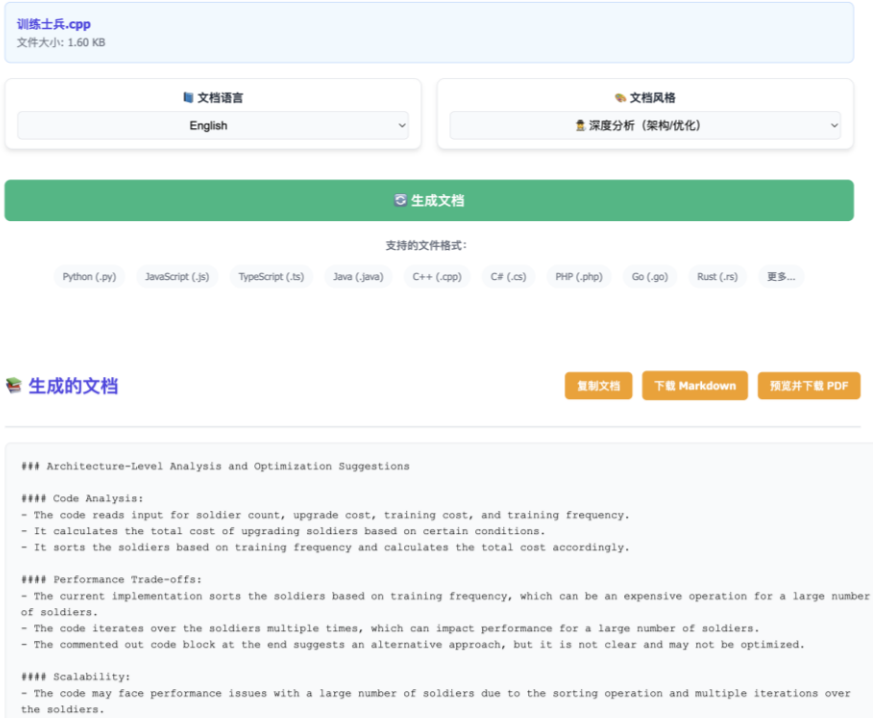
South China University of Technology

Advanced Multi-Modal Processing Pipeline

1. Single File Processing Pipeline

Listing 1: Single File Processing Implementation

```
def generate_documentation(self, code, filename, lang='zh', ...
    style='manual'):
    try:
        language = self.get_language_from_extension(filename)

        # Style-specific prompt generation
        if style == 'tutorial':
            prompt = self.create_tutorial_prompt(code, language, ...
                filename, lang)
        elif style == 'api':
            prompt = self.create_api_prompt(code, language, filename, lang)
        elif style == 'insight':
            prompt = self.create_insight_prompt(code, language, filename, ...
                lang)
        else:
            prompt = self.create_manual_prompt(code, language, filename, ...
                lang)

        response = self.client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": self.get_system_prompt(lang)},
                {"role": "user", "content": prompt}
            ],
            max_tokens=4000,
            temperature=0.3
        )
        return response.choices[0].message.content
    except Exception as e:
        raise Exception(f"Documentation generation error: {str(e)}")
```



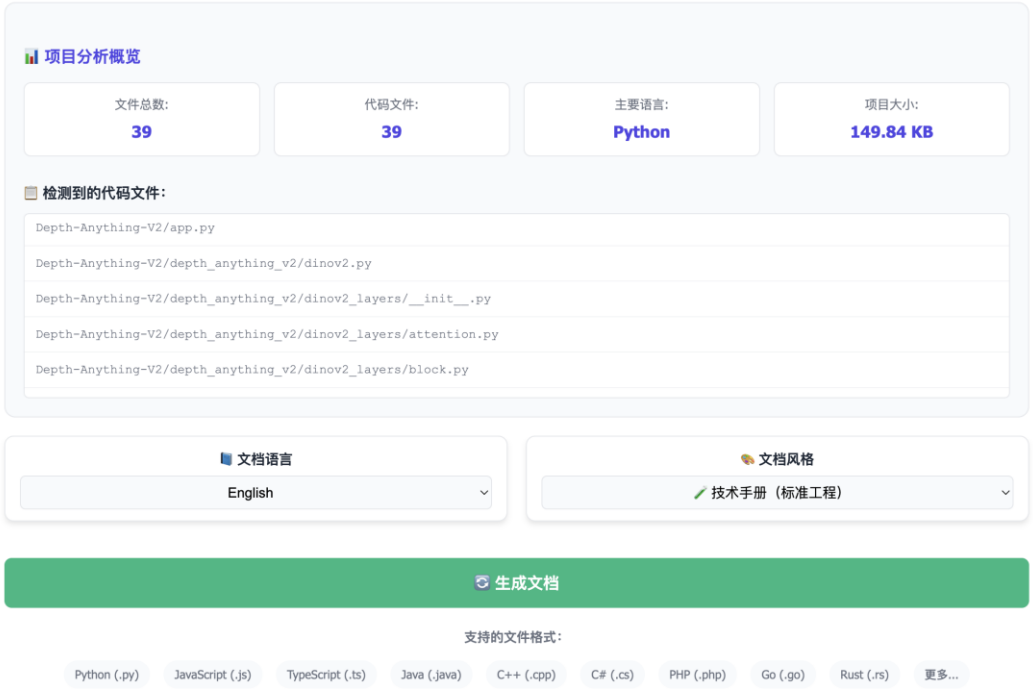
South China University of Technology

Advanced Multi-Modal Processing Pipeline

2. Project Archive Analysis Pipeline

Listing 2: Project Structure Analysis Implementation

```
def analyze_project_structure(self, file_contents):
    analysis = {
        'total_files': len(file_contents),
        'languages': {},
        'directories': set(),
        'file_types': {},
        'main_files': [],
        'config_files': [],
        'test_files': []
    }

    for filepath, content in file_contents.items():
        # Directory structure analysis
        path_parts = Path(filepath).parts
        if len(path_parts) > 1:
            analysis['directories'].add(path_parts[0])

        # Language distribution mapping
        language = self.get_language_from_extension(filepath)
        analysis['languages'][language] = ...
            analysis['languages'].get(language, 0) + 1

        # File categorization by importance and role
        filename = Path(filepath).name.lower()
        if filename in ['main.py', 'app.py', 'index.js', 'main.js', ...
            'index.html']:
            analysis['main_files'].append(filepath)
        elif filename.startswith('test_') or 'test' in filename:
            analysis['test_files'].append(filepath)
        elif filename in ['package.json', 'requirements.txt', 'pom.xml', ...
            'Cargo.toml']:
            analysis['config_files'].append(filepath)

    return analysis
```



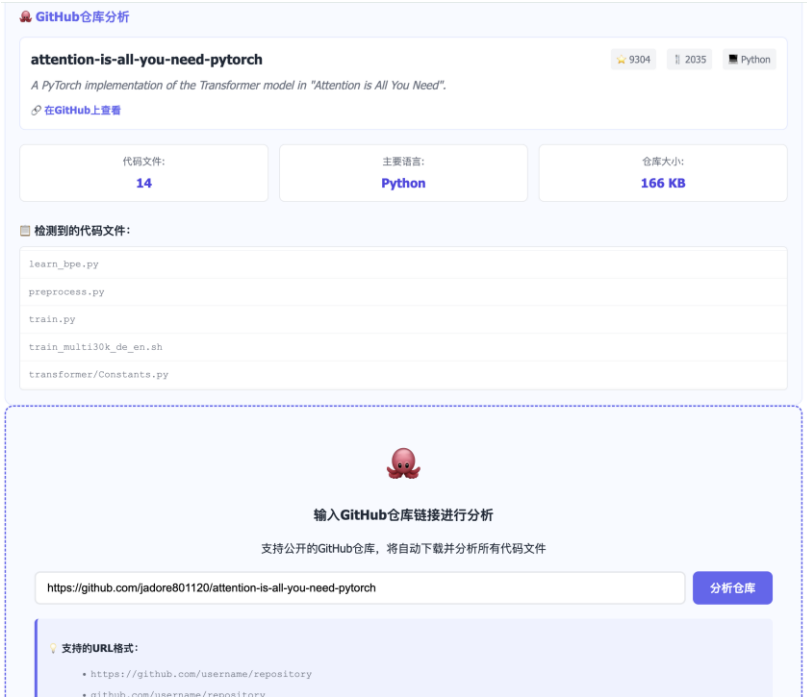
South China University of Technology

Advanced Multi-Modal Processing Pipeline

3. GitHub Repository Integration Pipeline

Listing 3: GitHub Repository Processing Implementation

```
def download_github_repo(self, github_url):
    try:
        # URL normalization and validation
        normalized_url = self.normalize_github_url(github_url)
        if not normalized_url:
            raise Exception("Invalid GitHub URL format")
        # Repository metadata extraction
        username, repo_name = self.extract_repo_info(normalized_url)
        api_url = f"https://api.github.com/repos/{username}/{repo_name}"

        # Repository information retrieval
        repo_response = requests.get(api_url, timeout=10)
        if repo_response.status_code == 404:
            raise Exception("Repository not found or private")

        repo_data = repo_response.json()

        # Multi-branch download strategy
        for branch in ['main', 'master']:
            download_url = ...
                f"https://github.com/{username}/{repo_name}/archive/refs/heads/{branch}.
            response = requests.get(download_url, timeout=30, stream=True)
            if response.status_code == 200:
                break

        if response.status_code != 200:
            raise Exception(f"Failed to download repository: HTTP ...
                {response.status_code}")

        # Secure file processing with size limitations
        file_contents = self.extract_code_files_from_github_zip(
            response.content, repo_name
        )

        return file_contents, self.format_repo_info(repo_data)

    except requests.exceptions.Timeout:
        raise Exception("Download timeout - check network connection")
    except Exception as e:
        raise Exception(f"GitHub processing error: {str(e)}")
```



03

Implementation Details

South China University of Technology

Technology Stack

Table 1: Enhanced Technology Stack and Implementation Details

Component	Technology	Implementation Details and Justification
Backend Framework	**Flask 2.x**	Production-grade WSGI application with comprehensive error handling, request validation, and multi-threaded processing capabilities. Implements RESTful endpoints with proper HTTP status codes and JSON responses.
AI Processing	**OpenAI API**	Integration with GPT-3.5-turbo for cost-effective processing and GPT-4 for complex analysis. Implements token management, rate limiting, and fallback strategies.
File Processing	**Python zipfile**	Native ZIP archive extraction with security validation, file type detection, and size limitations to prevent abuse and ensure system stability.
GitHub Integration	**Requests + GitHub API**	Direct repository access via GitHub's REST API v3, supporting both public repositories and authentication for private access. Implements intelligent branch detection (main/master).
Frontend Framework	**Vanilla JavaScript**	Modern ES6+ implementation with modular design, progressive enhancement, and comprehensive error handling. Avoids framework overhead for optimal performance.
UI Components	**CSS Grid/Flexbox**	Responsive design system with mobile-first approach, dark mode support, and accessibility compliance. Custom CSS variables enable consistent theming.
Document Rendering	**Markdown-it.js**	Client-side Markdown rendering with syntax highlighting support, table rendering, and HTML sanitization for security.
Export Capabilities	**html2pdf.js**	Client-side PDF generation with customizable styling, page breaks, and print optimization. Supports multiple output formats without server dependencies.

Environment and Dependencies

Python Virtual Environment

Isolate dependencies using Python 3 virtual environments for consistency

Core Library Selection Criteria

Choose key libraries based on maturity, performance, and community support

Security Configuration

Manage sensitive data via python-dotenv to prevent credential exposure

Environment and Dependencies

Library	Purpose	Key Features
Flask	Web Framework	Lightweight, unopinionated design
Flask-CORS	Cross-Origin Support	Automatic preflight request handling
openai	API Client	Official stable interface wrapper
markdown2	Format Conversion	Fast Markdown-to-HTML transformation

04

Discussion and Future Work

Technical Challenges & Mitigations

Core Difficulties:

Prompt Engineering

Code Ambiguity

·**Prompt Optimization**:

Modular prompt structure with four essential components

·**Context Enhancement**:

Implement additionalPrompt input field

·**Parameter Tuning**:

Low-temperature settings for output stability

·**Format Control**:

Strict Markdown syntax enforcement

Current Constraints

1. **Single-File Processing**:

Lacks project-level context understanding

2. **Static Analysis**:

Cannot capture runtime type information

3. **API Dependency**:

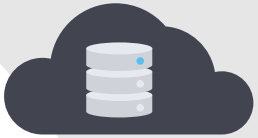
Fully reliant on OpenAI service availability

Future Roadmap

Area	Improvement	Expected Outcome
Context	RAG Framework	Project-level comprehension
Analysis	Hybrid Parsing	Reduce hallucinations
Deployment	Fine-Tuned Models	Decrease external dependency

華南理工大學
South China University of Technology

未来技术学院
FUTURE TECH



THE END

THANKS