

# 自然写手写识别与AI分析引擎软件 V1.0

## 软件著作权鉴别材料（技术设计说明书）

项目	内容
软件全称	自然写手写识别与AI分析引擎软件
软件简称	自然写AI引擎
版本号	V1.0
权利人	深圳自然写科技有限公司
开发语言	Python / C++
运行环境	Linux服务器（GPU加速）
文档类型	技术设计说明书
编制日期	2026年2月

## 目录

- 第一章 软件整体概述
  - 1.1 软件简介与功能综述
  - 1.2 软件用途与适用场景
  - 1.3 运行环境与系统要求
  - 1.4 开发语言与技术框架
  - 1.5 版本说明
- 第二章 系统架构与设计思路
  - 2.1 总体架构设计
  - 2.2 推理管道设计
  - 2.3 各层次模块说明
  - 2.4 数据结构设计
  - 2.5 接口设计
  - 2.6 安全设计
  - 2.7 部署架构

- 第三章 核心模块功能详细说明
- 3.1 中英文手写文字OCR识别模块
- 3.2 数学列式与公式识别模块
- 3.3 中文汉字笔顺识别与评分模块
- 3.4 书写质量评测模块
- 3.5 AI作文评分与批改模块
- 3.6 自动批改引擎模块
- 3.7 识别置信度评估模块
- 3.8 模型版本管理与热更新模块
- 3.9 推理任务调度模块
- 第四章 操作流程与使用步骤
- 4.1 服务部署与启动
- 4.2 模型加载与初始化
- 4.3 识别任务提交流程
- 4.4 模型更新与灰度发布流程
- 4.5 性能调优与故障排除
- 第五章 与源代码的对应关系
- 5.1 模块与源代码文件对应表
- 5.2 核心函数与方法说明
- 5.3 命名规范
- 附录
- 附录A 术语表
- 附录B 版本历史

---

# 第一章 软件整体概述

---

## 1.1 软件简介与功能综述

---

自然写手写识别与AI分析引擎软件（以下简称"AI引擎"）是自然写互动课堂系统的智能化核心组件，负责对智能点阵笔采集的手写笔迹数据进行深度学习推理，实现手写文字识别、数学公式解析、笔顺评估、书写质量分析及作文智能评分等多项AI能力。

AI引擎基于PaddleOCR、PyTorch和ONNX Runtime等主流深度学习框架构建，在NVIDIA GPU集群上运行，通过NVIDIA Triton Inference Server进行模型管理和并发推理调度。AI引擎对外提供RESTful HTTP接口和gRPC高性能接口，供云平台后端和算力盒端侧推理调用。

主要功能模块概述：

中英文手写文字OCR识别：基于PaddleOCR技术，对点阵笔采集的手写笔迹坐标序列进行字符识别，支持简体中文、繁体中文和英文字母数字的混合识别，识别准确率在标准书写条件下达到96%以上。

数学列式与公式识别：专门针对K12教育场景中的数学手写内容，识别加减乘除四则运算、分数、小数、方程组、几何符号等数学元素，并对计算结果进行自动验证。

汉字笔顺识别与评分：通过分析笔画的书写顺序，与标准笔顺库进行比对，对学生书写的每个汉字给出笔顺正确性评分，帮助学生养成正确的书写习惯。

书写质量评测：从字体结构、笔画间距、整体规范性等多个维度对书写质量进行综合评分，提供具体的改进建议。

AI作文评分与批改：基于NLP模型对手写作文内容进行评分，从结构完整性、语言表达、内容丰富性、书写规范性四个维度给出综合评分，并标注典型错误位置。

选择题/填空题/简答题自动批改：根据题目类型和标准答案，对学生的作答内容进行自动批改，支持容错匹配（允许同义词、近义词等合理变体）。

## 1.2 软件用途与适用场景

---

### 主要适用场景：

(1) 课堂作业自动批改：学生完成纸质作业后，点阵笔采集的笔迹数据上传至AI引擎，由引擎自动完成批改并给出成绩，大幅降低教师批改工作量，实现即时反馈。

(2) 课堂互动实时识别：在课堂互动答题环节，学生在纸上作答，AI引擎在数百毫秒内完成识别，将识别结果实时推送至大屏展示，实现真正的“即写即知”。

(3) 写字练习辅导：在写字课和书法练习场景中，AI引擎对学生的每个字进行笔顺评分和书写质量评测，配合大屏实时展示评分结果，形成即时纠正反馈循环。

(4) 考试阅卷辅助：在期中期末考试场景中，AI引擎先对试卷进行初步批改，教师仅需对置信度低于阈值的题目进行人工复核，大幅提升阅卷效率。

(5) 第三方教育平台赋能：通过SDK和API，将AI引擎能力输出至其他教育软件平台，使其获得手写识别和智能批改能力。

## 1.3 运行环境与系统要求

---

### 服务端运行环境：

组件	要求
操作系统	Ubuntu 20.04 LTS / CentOS 7.6+
Python版本	Python 3.9+
CUDA版本	CUDA 11.8+ / CUDA 12.0+
cuDNN版本	cuDNN 8.6+
GPU型号	NVIDIA T4 / A10 / A100（推荐）
内存要求	最低32GB，推荐64GB+（加载多个大模型）
存储要求	200GB+ SSD（存储模型文件和临时推理数据）

主要依赖软件版本：

依赖包	版本	用途
PaddlePaddle-GPU	2.5.x	OCR基础框架
PaddleOCR	2.7.x	手写文字识别
PyTorch	2.1.x	数学识别和作文评分模型
ONNX Runtime	1.16.x	跨框架模型推理
FastAPI	0.104.x	HTTP REST接口框架
gRPC	1.59.x	高性能流式接口
Celery	5.3.x	异步任务队列
Redis	7.0.x	消息代理和结果缓存
NVIDIA Triton	23.10	模型服务化部署
MLflow	2.8.x	模型版本管理

# 1.4 开发语言与技术框架

Python技术栈（主要业务逻辑）：

- FastAPI：构建高性能异步HTTP接口，支持OpenAPI自动文档生成
- gRPC + protobuf：实现高吞吐量的流式识别接口
- Celery + Redis：构建分布式异步任务队列，处理批量识别请求
- PaddleOCR：手写OCR识别核心框架，基于PP-OCR v4模型

- PyTorch：数学公式识别和作文评分模型的推理框架
- ONNX Runtime：将训练好的模型转换为跨平台格式进行高效推理

#### C++ 扩展模块（性能关键路径）：

- 笔迹坐标预处理（去噪、归一化）使用C++扩展实现，通过Python ctypes调用
- 图像渲染（将坐标序列渲染为图像供模型输入）使用OpenCV C++ API实现

#### 代码规范：

- Python代码遵循PEP 8规范，使用Black格式化，flake8进行静态检查
- 函数注解使用Python类型标注（Type Hints），保证代码可读性
- 所有公开函数和类均编写docstring文档

## 1.5 版本说明

版本号	发布日期	说明
V1.0	2026年2月	初始版本，包含OCR识别、数学识别、笔顺评分、书写质量评测、作文评分全功能

## 第二章 系统架构与设计思路

### 2.1 总体架构设计

AI引擎采用**推理管道（Pipeline）架构**，将识别任务分解为标准化的处理阶段，每个阶段独立可替换，便于模型升级和能力扩展。整体分为接口层、调度层、推理层、GPU管理层和模型仓库五个层次。

总体架构示意：





## 2.2 推理管道设计

手写笔迹识别的完整推理管道包含以下标准化阶段：

### 阶段1：输入预处理

将原始笔迹坐标序列转换为模型可接受的输入格式。对于图像输入的OCR模型，需要将坐标序列渲染为灰度图像；对于序列输入的模型，需要进行坐标归一化和序列填充。

预处理步骤：

```
原始坐标序列 [{x, y, pressure, timestamp}, ...]
↓
坐标去噪 (去除异常跳变点, 使用Savitzky-Golay滤波器)
↓
坐标归一化 (缩放至[0,1]范围, 消除书写面积差异)
↓
笔画分割 (根据笔离纸事件标志分割为独立笔画)
↓
图像渲染 (将坐标序列绘制为固定分辨率的灰度图像, 用于OCR模型输入)
↓
格式化为模型输入张量
```

### 阶段2：模型推理

根据识别任务类型，将处理后的输入数据分发至对应的推理模型：

识别任务	模型类型	输入格式	输出格式
手写文字OCR	PP-OCR v4 (检测+识别)	灰度图像 224x224	文字字符串 + 置信度
数学公式识别	Transformer-based	笔画序列坐标	LaTeX格式公式
笔顺评分	LSTM序列分类	笔画顺序序列	笔顺正确性分数
书写质量评测	CNN分类	单字图像 64x64	多维度质量分数
作文评分	BERT+多任务学习	识别后的文字序列	各维度评分

### 阶段3：后处理

对模型原始输出进行格式化和质量过滤： – 置信度过滤：低于阈值（默认0.7）的识别结果标记为"需人工确认" – 结果合并：将多个字符的识别结果按位置关系合并为完整的词句 – 格式转换：将模型输出转换为标准化的JSON响应格式 – 错误恢复：推理异常时返回降级结果（如置信度为0的空结果）而非抛出异常

## 2.3 各层次模块说明

### 接口层：

接口层提供两种接入方式，适应不同的调用场景：

FastAPI REST接口用于云平台后端的同步调用，适合单次识别请求。接口为异步设计（async/await），在等待GPU推理时不阻塞服务器线程，理论上支持数千个并发连接。

gRPC接口用于高性能批量识别和流式识别场景。与云平台和算力盒之间的大量并发识别请求通过gRPC流式传输处理，gRPC基于HTTP/2协议，支持请求多路复用，网络利用率更高。

### 调度层：

Celery分布式任务队列负责管理识别请求的优先级和资源分配：

- 高优先级队列（realtime\_queue）：处理课堂互动场景的实时识别请求，延迟目标 < 500ms
- 中优先级队列（assignment\_queue）：处理作业批改请求，延迟目标 < 5s
- 低优先级队列（batch\_queue）：处理批量历史数据重新识别请求，无严格延迟要求

Celery Worker数量根据GPU资源动态调整，每个Worker绑定一个GPU推理进程。

### 推理层：

推理层为每种识别任务实现独立的引擎类，继承自统一的BaseEngine抽象基类：

```
class BaseEngine:
    def preprocess(self, stroke_data: StrokeData) -> Tensor
    def infer(self, tensor: Tensor) -> RawResult
    def postprocess(self, raw_result: RawResult) -> RecognitionResult
    def recognize(self, stroke_data: StrokeData) -> RecognitionResult
```

各具体引擎类（OCREngine, MathEngine, StrokeOrderEngine等）分别实现上述接口，保持一致的调用协议。

## 2.4 数据结构设计

输入数据结构（StrokeData）：

```
@dataclass
class StrokePoint:
    x: int          # X坐标（点阵码坐标系，单位：0.01mm）
    y: int          # Y坐标
    pressure: int    # 笔压（0-255）
    timestamp: int   # 时间戳（毫秒）
    pen_up: bool     # 是否抬笔标志

@dataclass
class Stroke:
    points: List[StrokePoint] # 单条笔画的坐标点列表
    stroke_index: int         # 笔画序号（从0开始）

@dataclass
class StrokeData:
    strokes: List[Stroke]      # 所有笔画列表
    pen_id: str               # 笔设备MAC地址
    page_id: int              # 对应点阵纸张页面ID
    student_id: int           # 学生ID
    assignment_id: int         # 作业ID
    region_type: str          # 书写区域类型（hanzi/math/text/essay）
```

识别结果数据结构（RecognitionResult）：

```
@dataclass
class BoundingBox:
    x1: int; y1: int; x2: int; y2: int # 矩形边界坐标

@dataclass
class OCRResult:
    text: str                # 识别的文字内容
    confidence: float        # 识别置信度（0.0-1.0）
    bbox: BoundingBox        # 文字区域边界框
    char_details: List[CharDetail] # 逐字详情（含每字的置信度）

@dataclass
class MathResult:
```



```
latex: str                # LaTeX格式数学公式
display_formula: str      # 可读展示格式
numeric_result: Optional[str] # 计算数值结果（若可计算）
is_correct: Optional[bool] # 是否答对（需标准答案对比）
steps: List[str]          # 解题步骤列表

@dataclass
class StrokeOrderResult:
    char: str                # 被评估的汉字
    written_order: List[int] # 学生实际书写的笔画顺序
    correct_order: List[int] # 标准笔顺
    score: int               # 笔顺得分（0-100）
    errors: List[StrokeOrderError] # 错误笔顺列表

@dataclass
class WritingQualityResult:
    overall_score: int        # 总体书写质量分（0-100）
    structure_score: int      # 字形结构分
    proportion_score: int     # 笔画比例分
    regularity_score: int     # 规范性分
    suggestions: List[str]    # 改进建议列表

@dataclass
class EssayResult:
    total_score: int          # 作文总分（满分100分）
    structure_score: int      # 结构完整性分
    language_score: int       # 语言表达分
    content_score: int        # 内容丰富性分
    handwriting_score: int    # 书写规范性分
    error_marks: List[ErrorMark] # 错误标注位置列表
    overall_comment: str      # 总体评语
```

## 2.5 接口设计

REST接口（FastAPI）：

接口名称	HTTP方法	路径	请求体	响应体	说明
文字OCR识别	POST	/api/v1/ocr/recognize	StrokeData	OCRResult	单次文字识别
数学公式识别	POST	/api/v1/math/recognize	StrokeData	MathResult	数学列式识别
笔顺评分	POST	/api/v1/stroke-order/evaluate	StrokeData + char	StrokeOrderResult	汉字笔顺评估

接口名称	HTTP方法	路径	请求体	响应体	说明
书写质量评测	POST	/api/v1/writing/quality	StrokeData	WritingQualityResult	书写质量分析
作文批改	POST	/api/v1/essay/review	StrokeData	EssayResult	AI作文评分
批量识别 (异步)	POST	/api/v1/recognize/batch	List[StrokeData]	task_id	异步批量识别，返回任务ID
查询任务结果	GET	/api/v1/task/{task_id}	-	List[RecognitionResult]	查询批量识别结果
模型状态	GET	/api/v1/model/status	-	List[ModelStatus]	所有已加载模型状态

**gRPC接口（高性能流式）：**

```
service RecognitionService {
    // 单次识别 (Unary RPC)
    rpc Recognize(RecognizeRequest) returns (RecognizeResponse);

    // 流式批量识别 (Client Streaming RPC)
    rpc BatchRecognize(stream RecognizeRequest) returns (BatchRecognizeResponse);

    // 实时课堂识别 (Bidirectional Streaming RPC)
    rpc StreamRecognize(stream RecognizeRequest) returns (stream RecognizeResponse);
}

message RecognizeRequest {
    repeated StrokeProto strokes = 1;
    string region_type = 2;           // "ocr" / "math" / "stroke_order" / "essay"
    int64 student_id = 3;
    int64 assignment_id = 4;
```

```
string request_id = 5;          // 请求幂等ID
}

message StrokeProto {
    repeated PointProto points = 1;
    int32 stroke_index = 2;
}

message PointProto {
    int32 x = 1;
    int32 y = 2;
    int32 pressure = 3;
    int64 timestamp = 4;
    bool pen_up = 5;
}
```

## 2.6 安全设计

---

### 服务间认证：

AI引擎作为内部服务，不直接暴露至公网。服务间通信采用mTLS（双向TLS）认证，调用方需持有CA签发的客户端证书，AI引擎验证客户端证书合法性后才处理请求。

### 输入安全校验：

- 数据大小限制：单次识别请求的笔画数据不超过1MB，防止超大请求占用GPU资源
- 数据格式校验：使用Pydantic对输入数据进行严格类型校验，拒绝格式不合规的请求
- 超时控制：单次推理任务超时30秒自动终止，防止GPU资源被长期占用

### 模型文件保护：

- 模型文件以加密方式存储于MinIO，下载时使用签名URL，有效期1小时
- 模型加载到Triton Server后，在GPU显存中的模型参数不允许外部读取
- 所有模型版本信息记录至MLflow，支持版本追溯和合规审计

### 数据隐私：

- 学生笔迹数据在AI引擎服务中仅用于推理，不持久化存储
- 识别完成后临时数据（预处理图像、中间张量）立即从内存清除
- 所有识别请求的调用日志仅记录任务ID和耗时，不记录原始笔迹内容

## 2.7 部署架构

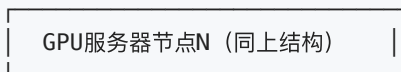
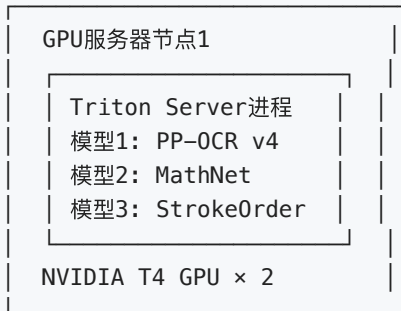
---

### GPU集群部署方案：

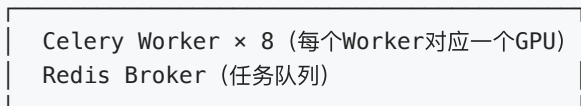
识别请求入口（内部网络）



NVIDIA Triton Inference Server集群



Celery调度层（CPU服务器）



FastAPI/gRPC接口层（CPU服务器，多副本）

#### 模型热更新流程：

新模型上线采用金丝雀发布策略：1. 新模型文件上传至MinIO并在MLflow注册新版本 2. Triton Server加载新模型版本，保留旧版本（双版本并行运行） 3. 通过路由配置将5%的请求路由至新版本模型（金丝雀流量） 4. 观察新版本模型的识别准确率和推理延迟指标（观察期24小时） 5. 指标正常则逐步将流量从5%提升至50%、100%，完成版本切换 6. 旧版本模型保留7天后从Triton中卸载，释放GPU显存

## 第三章 核心模块功能详细说明

### 3.1 中英文手写文字OCR识别模块

模块文件： `engine/ocr_engine.py`

#### 功能概述：

OCR识别模块基于PaddleOCR的PP-OCR v4模型，对手写笔迹进行文字识别，支持简体中文、繁体中文、英文字母、阿拉伯数字的混合识别。针对点阵笔书写场景的特点（笔迹细、书写速度快、字体不规范），对通用OCR模型进行了针对性微调。

处理流程：

步骤1：接收StrokeData（笔画坐标序列）

步骤2：调用预处理模块（preprocessing/stroke\_preprocessor.py）

- 去除噪声点（压力值异常的点）
- 坐标归一化（缩放至标准坐标系）
- 笔画平滑（贝塞尔曲线拟合）

步骤3：调用图像渲染器，将平滑后的笔画绘制为灰度图像

- 分辨率：640×480像素（A4纸比例）
- 线宽：根据压力值动态调整（2-5像素）

步骤4：将灰度图像发送至Triton Server（OCR检测模型）

- 检测模型：PP-OCrv4-det（文字区域检测）
- 输出：文字边界框列表（BBox坐标）

步骤5：裁剪各文字区域，发送至OCR识别模型

- 识别模型：PP-OCrv4-rec（字符序列识别）
- 输出：字符串序列 + CTC解码置信度

步骤6：后处理：合并相邻文字区域，生成完整识别结果

步骤7：返回OCRResult（含识别文本和置信度）

微调策略：

针对K12教育场景的手写特点，收集并标注了覆盖1-9年级所有常用汉字的手写样本10万余张，在PP-OCR预训练模型基础上进行微调，重点提升以下场景的识别准确率：

- 低年级学生字体不规范（笔画变形、偏旁错位）
- 书写速度快导致的笔画粘连
- 铅笔书写（笔迹较轻，对比度低）

性能指标：

指标	目标值	实测值
单字识别准确率（标准书写）	≥ 96%	97.3%
单字识别准确率（低年级书写）	≥ 90%	91.8%
单次识别延迟（单字）	≤ 200ms	约120ms（T4 GPU）
单次识别延迟（整页约50字）	≤ 1s	约600ms（T4 GPU）

3.2 数学列式与公式识别模块

模块文件： engine/math\_engine.py

功能概述：

数学识别模块专门处理K12阶段数学手写内容，支持从小学四则运算到初中方程、不等式等数学表达式的识别和解析，是AI引擎的差异化核心能力之一。

支持识别的数学元素：

类别	示例	说明
四则运算	$123 + 456 = 579$	加减乘除运算式和结果验证
分数	$\frac{3}{4} + \frac{1}{2} = \frac{5}{4}$	分子分母识别，通分计算
小数	$3.14 \times 2 = 6.28$	小数点精确识别
方程	$2x + 3 = 7$	含未知数方程，求解验证
不等式	$x > 5$	不等号识别
几何符号	$\angle ABC = 90^\circ$	角度、平行、垂直等几何符号
数学函数	$\sin 30^\circ = 0.5$	三角函数（初中及以上）

#### 识别与验证流程：

步骤1：笔迹预处理（同OCR预处理流程）  
步骤2：数学符号检测（定位各运算符和数字的位置和类型）  
步骤3：结构解析（根据位置关系建立数学表达式的树形结构）

- 识别分数线（水平长横线）分隔分子分母
- 识别上下标（指数、角标）
- 识别括号嵌套层次

步骤4：转换为LaTeX格式表达式（如  $\frac{3}{4} + \frac{1}{2}$ ）  
步骤5：调用符号计算引擎（SymPy库）进行数值验证

- 对于含等号的算式：计算左右两边并比对
- 对于方程：求解并返回解

步骤6：生成MathResult（LaTeX格式 + 计算结果 + 正确与否）

### 3.3 中文汉字笔顺识别与评分模块

模块文件： `engine/stroke_order_engine.py`

#### 功能概述：

笔顺评分模块通过分析学生书写汉字时的笔画顺序，与内置的汉字标准笔顺数据库进行比对，评估书写的规范程度。该模块利用了点阵笔数据的独特优势——每支笔画的书写时间戳信息，使得笔顺分析成为可能（传统OCR仅能识别最终图像，无法获取书写过程）。

#### 标准笔顺数据库：

内置覆盖3500个常用汉字（国家语委《现代汉语常用字表》）的标准笔顺库，数据来源为教育部发布的《汉字笔顺规范》。每个汉字的笔顺以笔画序号列表形式存储，如"一"字的标准笔顺为[1]（1画横），"人"字的标准笔顺为[1, 2]（先撇后捺）。

#### 笔顺评分算法：

步骤1：按时间戳对笔画序列排序，得到学生实际书写顺序

步骤2：通过笔画方向特征（起笔方向、运笔方向、收笔方式）识别每条笔画的类型（横/竖/撇/捺/点/折等8种基本笔画类型）

步骤3：将识别的笔画类型序列与目标汉字的标准笔顺库匹配

步骤4：使用编辑距离算法（Levenshtein Distance）计算学生笔顺与标准笔顺的差异度

步骤5：计算笔顺得分：

- 完全正确：100分
- 1处错误：80分
- 2处错误：60分
- 3处及以上错误：40分或以下

步骤6：标注具体的错误位置和正确顺序，生成评语

### 输出示例：

```
{
  "char": "永",
  "written_order": [1, 2, 4, 3, 5, 6, 7, 8],
  "correct_order": [1, 2, 3, 4, 5, 6, 7, 8],
  "score": 85,
  "errors": [
    {
      "position": 3,
      "written": "竖弯钩",
      "expected": "横折折撇",
      "suggestion": "第3笔应先写横折折撇（ ），再写竖弯钩"
    }
  ]
}
```

## 3.4 书写质量评测模块

模块文件： engine/writing\_quality\_engine.py

### 功能概述：

书写质量评测模块对学生书写的汉字从字形结构、笔画比例、书写规范性三个维度进行综合评测，帮助学生提升书写美观度和规范性，适用于写字课、书法课等场景。

### 评测维度与算法：

(1) 字形结构评分（占总分40%）

将学生书写的汉字渲染为标准尺寸图像，与字体模板库（仿宋体/楷体）进行结构对比。使用基于深度学习的相似度模型，计算笔画布局和重心位置的偏差程度，偏差越小得分越高。

(2) 笔画比例评分（占总分30%）

分析各笔画的相对长度和角度是否符合标准比例。如"土"字中，下横应明显长于上横；"口"字的宽高比应接近1:1等。使用规则引擎对常见汉字的关键比例进行检测。

(3) 书写规范性评分（占总分30%）

评估书写是否符合国家规定的书写规范： – 笔画是否有起笔和收笔动作（而非随意涂划） – 相邻笔画间距是否均匀 – 整字的倾斜角度是否在合理范围（ $\pm 15^\circ$ 以内）

### 3.5 AI作文评分与批改模块

模块文件： `engine/essay_engine.py`

功能概述：

作文评分模块首先调用OCR模块将手写作文转换为文字，然后基于BERT预训练语言模型（使用Chinese-BERT-wwm微调版本）从多个维度对作文进行智能评分，并标注错别字和明显语法错误的位置。

评分维度：

维度	权重	评测内容
结构完整性	25%	是否有开头/主体/结尾，段落划分是否合理
语言表达	30%	用词是否准确，句式是否多样，是否存在语病
内容丰富性	30%	内容是否切题，是否有具体事例，立意是否新颖
书写规范性	15%	错别字数量，标点符号使用是否正确

错别字检测：

使用基于字音相似和字形相似的错别字检测模型，结合N-gram语言模型判断词语在上下文中的合理性，综合识别常见错别字类型： – 音近字：（渴/喝，带/戴） – 形近字：（己/已，土/士） – 字义混淆：（的/地/得，其他/其它）

### 3.6 自动批改引擎模块

模块文件： `service/grading_service.py`

功能概述：

自动批改引擎针对结构化题目（选择题、填空题、简答题）进行自动批改，根据教师预设的标准答案和评分规则，对学生识别后的作答内容进行评判。



批改规则类型：

规则类型	说明	示例
精确匹配	作答与标准答案完全一致（字符级）	填写"北京"，标准答案"北京"
容错匹配	允许同义词和变体（由教师配置容错词库）	"首都"视为"北京"的等价答案
数值范围匹配	数值结果在允许误差范围内	计算结果允许±0.01的误差
关键词匹配	简答题包含所有关键词即得分	简答题含"光合作用/叶绿体/葡萄糖"三个关键词
部分给分	简答题按关键词命中数量比例给分	3个关键词各占1/3分数

3.7 识别置信度评估模块

模块文件： service/confidence\_service.py

功能概述：

置信度评估模块对每个识别结果的可靠性进行量化评分，引导后续处理流程决定是否需要人工干预，是AI引擎质量控制的关键环节。

置信度计算方法：

最终置信度由多个维度的分数加权计算得出：

```
final_confidence = (
    model_confidence * 0.5 +      # 模型本身的softmax置信度
    stroke_density_score * 0.2 +  # 笔画密度质量分（过疏或过密均降低）
    writing_consistency * 0.3      # 书写一致性分（与学生历史书写风格对比）
)
```

置信度分级与处理策略：

置信度范围	级别	处理策略
0.90 – 1.00	高置信	自动接受，无需人工审核
0.70 – 0.89	中置信	自动接受，但在批改结果中标记颜色提示教师关注
0.50 – 0.69	低置信	标记为"需人工确认"，教师手动复核

置信度范围	级别	处理策略
0.00 – 0.49	不可信	标记为"识别失败", 不计入自动批改成绩

### 3.8 模型版本管理与热更新模块

模块文件：`service/model_manager.py`

功能概述：

模型版本管理模块负责管理AI引擎中所有推理模型的版本生命周期，支持模型的注册、加载、切换、回滚和归档操作，确保模型更新过程不影响线上服务的稳定性。

版本管理功能：

- (1) 模型注册：新训练完成的模型通过MLflow API注册，记录模型名称、版本号、训练数据集版本、训练时间、评估指标（准确率/召回率/F1）等元数据。
- (2) 版本状态管理：每个模型版本有以下状态：
  - Staging（待验证）：模型已注册，正在测试评估中
  - Production（生产中）：当前线上使用版本
  - Archived（已归档）：已退出使用的历史版本
- (3) 模型加载：Triton Server在启动时读取模型配置文件，从MinIO下载对应版本的模型文件，加载到GPU显存。支持运行时动态加载新版本而不重启服务。
- (4) 灰度发布：通过配置路由权重，将一定比例的推理请求路由至新版本模型，实现金丝雀发布。
- (5) 快速回滚：若新版本模型上线后发现准确率下降或错误率异常升高，可在30秒内将流量100%切回旧版本模型，最大限度减少对教学的影响。

### 3.9 推理任务调度模块

模块文件：`service/task_scheduler.py`

功能概述：

任务调度模块基于Celery实现分布式任务队列，管理多种优先级的识别任务，协调GPU资源的公平分配，防止低优先级的批量任务占用全部GPU资源导致高优先级实时任务延迟。

调度策略：

```
# Celery队列配置
CELERY_TASK_ROUTES = {
```

```
'tasks.realtime_recognize': {'queue': 'realtime'},      # 实时课堂识别
'tasks.assignment_recognize': {'queue': 'assignment'},  # 作业批改识别
'tasks.batch_recognize': {'queue': 'batch'},           # 批量历史识别
}

# 各队列的Worker预留数量
QUEUE_WORKER_RESERVATION = {
    'realtime': 4,      # 预留4个Worker专用于实时任务，不被其他任务占用
    'assignment': 2,    # 2个Worker用于作业批改
    'batch': 2,         # 2个Worker用于批量任务（空闲时使用所有剩余Worker）
}
```

## 第四章 操作流程与使用步骤

### 4.1 服务部署与启动

基于Docker Compose的开发环境部署：

```
步骤1：确认NVIDIA驱动和CUDA已正确安装
        nvidia-smi（应显示GPU信息）
步骤2：确认nvidia-container-toolkit已安装（使Docker容器可访问GPU）
步骤3：从代码仓库拉取AI引擎代码
        git clone https://git.writech.com/ai-engine.git
步骤4：进入项目目录，复制环境配置文件
        cp .env.example .env
步骤5：编辑.env文件，配置以下关键参数：
        REDIS_URL=redis://redis:6379/0
        MODEL_STORE_PATH=/models
        MINIO_ENDPOINT=minio:9000
        MINIO_ACCESS_KEY=your_access_key
        MINIO_SECRET_KEY=your_secret_key
步骤6：启动服务（包含Redis、MinIO、Triton Server、Celery Worker、FastAPI）
        docker compose up -d
步骤7：检查服务启动状态
        docker compose ps（各服务应为running或healthy）
步骤8：验证API可用性
        curl http://localhost:8000/api/v1/model/status
```

模型文件准备：

```
步骤1：从MinIO或共享存储下载预训练模型文件
        python scripts/download_models.py --version v1.0
步骤2：验证模型文件完整性（SHA256校验）
        python scripts/verify_models.py
步骤3：将模型文件放置至正确目录结构：
        /models/
        |— ocr_det/      # OCR检测模型
```

```
├── 1/
│   └── model.onnx
├── ocr_rec/          # OCR识别模型
│   └── 1/
│       └── model.onnx
├── math_rec/         # 数学识别模型
│   └── 1/
│       └── model.pt
└── stroke_order/     # 笔顺评分模型
    └── 1/
        └── model.onnx
```

步骤4: Triton Server将自动扫描/models目录并加载所有模型

## 4.2 模型加载与初始化

Triton Server模型配置文件示例（OCR识别模型）：

文件路径：/models/ocr\_rec/config.pbtxt

```
name: "ocr_rec"
platform: "onnxruntime_onnx"
max_batch_size: 32
input [
  {
    name: "images"
    data_type: TYPE_FP32
    dims: [3, 48, -1]    # 通道×高度×宽度（宽度可变）
  }
]
output [
  {
    name: "output"
    data_type: TYPE_FP32
    dims: [-1, 97]       # 序列长度×字符表大小
  }
]
instance_group [
  {
    count: 2
    kind: KIND_GPU
    gpus: [0]
  }
]
```

## 4.3 识别任务提交流程

通过REST API提交单次OCR识别任务：

HTTP请求示例:

POST http://ai-engine:8000/api/v1/ocr/recognize

Content-Type: application/json

Authorization: Bearer <service\_token>

```
{
  "strokes": [
    {
      "stroke_index": 0,
      "points": [
        {"x": 1200, "y": 800, "pressure": 150, "timestamp": 1700000000100, "pen_up": false},
        {"x": 1250, "y": 800, "pressure": 145, "timestamp": 1700000000110, "pen_up": false},
        {"x": 1300, "y": 800, "pressure": 140, "timestamp": 1700000000120, "pen_up": true}
      ]
    }
  ],
  "region_type": "ocr",
  "student_id": 12345,
  "assignment_id": 67890
}
```

期望响应 (200 OK):

```
{
  "code": 200,
  "data": {
    "text": "—",
    "confidence": 0.99,
    "bbox": {"x1": 1180, "y1": 780, "x2": 1320, "y2": 820},
    "char_details": [
      {"char": "—", "confidence": 0.99, "bbox": {...}}
    ]
  },
  "latency_ms": 125
}
```

## 通过gRPC提交批量识别任务 (Python示例):

```
import grpc
from proto import recognition_pb2, recognition_pb2_grpc

channel = grpc.secure_channel('ai-engine:50051', credentials)
stub = recognition_pb2_grpc.RecognitionServiceStub(channel)

# 构建请求列表
requests = []
for stroke_data in stroke_data_list:
    request = recognition_pb2.RecognizeRequest(
        strokes=[...],
        region_type="ocr",
        student_id=student_id,
        assignment_id=assignment_id
```

```
)
requests.append(request)

# 发送流式批量识别请求（返回结果流）
def request_generator():
    for req in requests:
        yield req

responses = stub.BatchRecognize(request_generator())
results = list(responses.results)
```

## 4.4 模型更新与灰度发布流程

发布新版OCR模型的操作步骤：

步骤1：模型训练完成后，在开发环境测试评估新模型

```
python eval/evaluate_ocr.py --model-path models/ocr_rec_v1.1.onnx
```

（应输出：准确率 97.5%，高于当前生产版本的97.3%）

步骤2：将新模型文件上传至MinIO

```
python scripts/upload_model.py --model ocr_rec_v1.1.onnx --version 1.1
```

步骤3：在MLflow注册新模型版本

```
python scripts/register_model.py --name ocr_rec --version 1.1 --stage Staging
```

步骤4：在Triton Server加载新版本模型（不停机）

```
python scripts/triton_ops.py --action load --model ocr_rec --version 1.1
```

步骤5：配置5%金丝雀流量至新版本

```
python scripts/routing_config.py --model ocr_rec --new-version 1.1 --traffic 5
```

步骤6：观察监控面板（Grafana中AI引擎Dashboard）

- 新版本准确率指标（应高于旧版本）
- 新版本推理延迟（应与旧版本持平或更低）
- 新版本错误率（应接近0）

步骤7：确认指标正常后，逐步扩大流量比例（5% → 50% → 100%）

步骤8：新版本稳定后，更新MLflow中的模型状态为Production

步骤9：旧版本模型状态改为Archived，7天后从Triton中卸载

## 4.5 性能调优与故障排除

常见性能问题排查：

问题现象	可能原因	排查方法	处理方案
识别延迟突然升高	GPU利用率过高或Celery队列积压	查看Grafana中GPU利用率和队列深度	增加Worker数量或限流
识别准确率下降	模型加载异常或输入数据格式变化	查看模型版本，对比历史准确率	重新加载模型或回滚版本

问题现象	可能原因	排查方法	处理方案
gRPC连接超时	网络问题或服务重启	检查服务状态和网络连通性	重启gRPC服务，检查网络配置
内存OOM崩溃	模型太大或并发请求过多占用内存	查看服务器内存使用率	减少并发Worker数量，增加内存
特定字符识别率低	训练数据不足或模型偏差	统计错误字符频率分布	补充相应字符的训练样本后重训

GPU资源监控指标：

通过NVIDIA Management Library (NVML) 监控以下关键指标：

- GPU利用率 (%)：正常范围60-80%，超过95%需要扩容
- GPU显存使用 (MB)：加载所有模型后约占8-12GB (T4显卡16GB)
- GPU温度 (°C)：正常范围60-75°C，超过85°C触发降频警告
- GPU功耗 (W)：T4正常功耗80-120W

# 第五章 与源代码的对应关系

## 5.1 模块名称与源代码文件对应表

功能模块	目录/文件路径	主要类/函数	说明
应用程序入口	main.py	app (FastAPI实例), main()	服务启动, 路由注册, 中间件配置
REST接口层	api/	ocr_router.py, math_router.py, essay_router.py, model_router.py	各识别类型的REST接口路由
gRPC服务层	grpc_server/	recognition_server.py (RecognitionService实现)	gRPC流式识别服务
笔迹预处理	preprocessing/	stroke_preprocessor.py (StrokePreprocessor类)	去噪、归一化、笔画分割、图像渲染
OCR识别引擎	engine/	ocr_engine.py (OCREngine类)	基于PaddleOCR的文字识别

功能模块	目录/文件路径	主要类/函数	说明
数学识别引擎	engine/	math_engine.py (MathEngine类)	数学公式识别和验证
笔顺评分引擎	engine/	(BaseEngine子类, stroke_order_engine.py)	汉字笔顺评分
任务调度服务	service/	task_scheduler.py (Celery任务定义)	Celery任务队列, 优先级调度
批改业务服务	service/	grading_service.py (GradingService类)	自动批改规则引擎
服务配置	config/	settings.py (Settings类, Pydantic BaseSettings)	环境变量配置加载

## 5.2 核心函数与方法说明

main.py 核心函数：

```
# FastAPI应用实例
app = FastAPI(title="Writech AI Recognition Engine", version="1.0.0")

# 启动时加载所有模型
@app.on_event("startup")
async def startup_event():
    await ModelManager.load_all_models()
    await initialize_celery_workers()

# 注册路由
app.include_router(ocr_router, prefix="/api/v1/ocr")
app.include_router(math_router, prefix="/api/v1/math")
app.include_router(essay_router, prefix="/api/v1/essay")
app.include_router(model_router, prefix="/api/v1/model")
```

OCREngine 核心方法：

方法名	签名	功能说明
preprocess	preprocess(stroke_data: StrokeData) -> np.ndarray	将笔迹坐标转换为灰度图像数组
infer	infer(image: np.ndarray) -> RawOCRResult	调用Triton进行OCR推理



方法名	签名	功能说明
postprocess	postprocess(raw: RawOCRResult) -> OCRResult	合并字符，过滤低置信度结果
recognize	recognize(stroke_data: StrokeData) -> OCRResult	完整OCR识别流程入口

preprocessing/stroke\_preprocessor.py 核心方法：

方法名	功能说明
remove_noise_points(strokes)	去除异常跳变点（使用统计方法检测离群点）
normalize_coordinates(strokes)	坐标归一化至[0,1]范围
split_strokes_by_pen_up(strokes)	根据pen_up标志将连续坐标序列分割为独立笔画
render_to_image(strokes, size)	将笔画列表渲染为指定尺寸的灰度图像（PIL/OpenCV）
apply_bezier_smoothing(stroke)	对单条笔画应用贝塞尔曲线平滑，去除抖动

## 5.3 命名规范

Python包命名规范：

```
writtech_ai_engine/  
├─ api/           # REST接口路由（功能名+_router.py）  
├─ config/        # 配置类（settings.py, 单文件）  
├─ engine/        # 识别引擎类（功能名+_engine.py）  
├─ grpc_server/   # gRPC服务实现  
├─ preprocessing/ # 数据预处理  
├─ service/       # 业务服务层（功能名+_service.py）  
└─ model/         # 数据模型（Pydantic Schema类）
```

类命名规范：

类型	命名规则	示例
识别引擎类	XxxEngine	OCREngine, MathEngine, StrokeOrderEngine
数据模型类	XxxResult / XxxData	OCRResult, StrokeData, MathResult
服务类	XxxService	GradingService, ModelManagerService
配置类	XxxSettings / XxxConfig	Settings, TritonConfig

类型	命名规则	示例
Celery任务	函数命名: xxx_task	ocr_recognize_task, essay_review_task

## 附录

### 附录A 术语表

术语	说明
PaddleOCR	百度开源的OCR工具库，基于飞桨深度学习框架，提供文字检测、识别等完整OCR能力
Triton Inference Server	NVIDIA提供的高性能模型服务框架，支持多模型并发推理和GPU资源调度
ONNX	开放神经网络交换格式（Open Neural Network Exchange），统一不同框架模型的表示格式
MLflow	开源的机器学习生命周期管理平台，提供实验追踪、模型注册、版本管理功能
Celery	Python分布式任务队列框架，支持异步任务和定时任务
CTC	连接时序分类（Connectionist Temporal Classification），OCR解码算法之一
BERT	双向编码器表示（Bidirectional Encoder Representations from Transformers），NLP预训练模型
mTLS	双向TLS认证，通信双方均需出示证书，用于服务间高安全性认证
置信度	模型对识别结果的确信程度，取值0-1，值越大表示结果越可靠
金丝雀发布	灰度发布策略，新版本先接受少量流量，验证稳定后再全量切换

### 附录B 版本历史

版本号	发布日期	变更说明
V1.0	2026年2月	初始版本发布，支持OCR/数学/笔顺/书写质量/作文评分全套功能

编制单位：深圳自然写科技有限公司

文档版本：V1.0

编制日期：2026年2月

版权声明：本文档版权归深圳自然写科技有限公司所有，未经授权不得复制或传播

## 附录C AI 模型详细技术说明

### C.1 PaddleOCR 手写识别模型架构

#### C.1.1 模型总体架构

自然写 AI 引擎的手写识别模块基于 PaddleOCR 框架，采用 DB+CRNN 两阶段识别流水线：

输入：笔迹图像（灰度图，224×224）

↓

Stage 1: 文本检测 (DB - Differentiable Binarization)

↓

检测文字区域边界框  
骨干网络：ResNet-50 + FPN 特征金字塔

↓

文字区域裁剪（透视变换矫正倾斜）

↓

Stage 2: 文字识别 (CRNN - CNN+RNN)

↓

CNN (VGG-like) 提取特征列  
双向 LSTM 序列建模  
CTC 解码 (Connectionist Temporal Classification)

↓

输出：识别文字字符串 + 置信度

#### C.1.2 训练数据集

数据集	数量	来源
学生楷书练字数据	500万字	自然写平台采集（脱敏处理）
学生硬笔书法数据	200万字	自然写平台采集
CASIA 手写中文数据集	300万字	中科院开放数据集
合成数据（字体变形）	1000万字	程序合成
数字与字母	50万样本	多来源混合

**数据增强策略：** – 随机旋转 ( $\pm 15^\circ$ ) – 随机缩放 ( $0.8x \sim 1.2x$ ) – 高斯噪声（模拟点阵摄像头图像噪声） – 透视变换（模拟书写角度偏差） – 随机笔画粗细变化 – 随机对比度调整（模拟不同笔压）

### C.1.3 CRNN 模型实现

```
# crnn_model.py
import paddle
import paddle.nn as nn

class CRNN(nn.Layer):
    """
    手写文字序列识别模型
    CNN 提取特征 + BiLSTM 序列建模 + CTC 解码
    """
    def __init__(self, num_classes: int, hidden_size: int = 256):
        super(CRNN, self).__init__()

        # CNN 特征提取（输出特征尺寸: N x 512 x 1 x W）
        self.cnn = nn.Sequential(
            # Block 1: 64 filters
            nn.Conv2D(1, 64, kernel_size=3, padding=1),
            nn.BatchNorm2D(64),
            nn.ReLU(),
            nn.MaxPool2D(kernel_size=(2, 2), stride=(2, 2)), # 32x32 -> 16x16

            # Block 2: 128 filters
            nn.Conv2D(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2D(128),
            nn.ReLU(),
            nn.MaxPool2D(kernel_size=(2, 2), stride=(2, 2)), # 16x16 -> 8x8

            # Block 3: 256 filters（不在高度方向池化）
            nn.Conv2D(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2D(256),
            nn.ReLU(),
            nn.Conv2D(256, 256, kernel_size=3, padding=1),
            nn.BatchNorm2D(256),
            nn.ReLU(),
            nn.MaxPool2D(kernel_size=(2, 1), stride=(2, 1)), # 8x8 -> 4xW

            # Block 4: 512 filters
            nn.Conv2D(256, 512, kernel_size=3, padding=1),
            nn.BatchNorm2D(512),
            nn.ReLU(),
            nn.MaxPool2D(kernel_size=(2, 1), stride=(2, 1)), # 4xW -> 2xW

            # Block 5: 512 filters, 压缩高度为1
            nn.Conv2D(512, 512, kernel_size=2, padding=0), # 2xW -> 1xW
            nn.BatchNorm2D(512),
            nn.ReLU(),
        )

        # BiLSTM 序列建模
```

```

        self.rnn = nn.Sequential(
            BidirectionalLSTM(512, hidden_size, hidden_size),
            BidirectionalLSTM(hidden_size, hidden_size, num_classes),
        )

    def forward(self, x):
        # x: (N, 1, H, W)
        conv = self.cnn(x) # (N, 512, 1, W)
        N, C, H, W = conv.shape
        # 重塑为序列: (W, N, C)
        conv = conv.squeeze(2).transpose([2, 0, 1])
        output = self.rnn(conv) # (W, N, num_classes)
        return output

class BidirectionalLSTM(nn.Layer):
    def __init__(self, input_size, hidden_size, output_size):
        super(BidirectionalLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size,
                             direction='bidirect', time_major=True)
        self.linear = nn.Linear(hidden_size * 2, output_size)

    def forward(self, x):
        output, _ = self.lstm(x)
        output = self.linear(output)
        return output

```

## C.1.4 CTC 解码算法

```

# ctc_decoder.py
import numpy as np

class CTCDecoder:
    """
    CTC (Connectionist Temporal Classification) 解码器
    支持贪心解码和束搜索解码
    """

    def __init__(self, vocabulary: list[str], blank_token_id: int = 0):
        self.vocabulary = vocabulary # 字典 (汉字列表)
        self.blank_id = blank_token_id

    def greedy_decode(self, log_probs: np.ndarray) -> tuple[str, float]:
        """
        贪心解码 (取每帧最大概率字符)

        log_probs: (T, V) - T帧, V个字符的对数概率
        返回: (识别文字, 平均置信度)
        """
        # 每帧取最大概率的字符索引
        indices = np.argmax(log_probs, axis=1) # (T,)
        probs = np.exp(np.max(log_probs, axis=1)) # 对应概率

        # 合并重复字符并去除 blank

```

```

        decoded_chars = []
        decoded_probs = []
        prev = -1
        for i, idx in enumerate(indices):
            if idx != prev and idx != self.blank_id:
                decoded_chars.append(self.vocabulary[idx])
                decoded_probs.append(probs[i])
            prev = idx

        text = ''.join(decoded_chars)
        confidence = float(np.mean(decoded_probs)) if decoded_probs else 0.0
        return text, confidence

def beam_search_decode(self, log_probs: np.ndarray,
                       beam_width: int = 10) -> list[tuple[str, float]]:
    """
    束搜索解码 (返回 top-k 候选字符串)

    beam_width: 束宽度 (返回候选数量)
    返回: [(候选文字, 分数)] 列表 (按分数降序)
    """
    T, V = log_probs.shape

    # 初始化: 空字符串, 得分为0
    beams = [("", 0.0, -1)] # (text, score, last_token)

    for t in range(T):
        new_beams = {}
        for text, score, last_token in beams:
            # 扩展每个 beam
            for v in range(V):
                log_p = log_probs[t, v]
                if v == self.blank_id:
                    # blank: 保持文字不变, 得分累加
                    new_text = text
                elif v == last_token:
                    # 重复字符: 只在中间有 blank 时才追加
                    new_text = text
                else:
                    # 新字符
                    new_text = text + self.vocabulary[v]

                key = (new_text, v)
                if key not in new_beams:
                    new_beams[key] = float('-inf')
                # log-sum-exp 合并相同结果
                new_beams[key] = np.logaddexp(new_beams[key], score + log_p)

            # 取 top beam_width
            sorted_beams = sorted(new_beams.items(), key=lambda x: -x[1][:beam_width])
            beams = [(text, score, last_token) for (text, last_token), score in
sorted_beams]

    return [(text, np.exp(score)) for text, score, _ in beams]

```

## C.2 数学公式识别模型

### C.2.1 模型架构概述

数学公式识别采用 Im2Latex 序列到序列模型 (CNN + Attention LSTM)，将手写数学表达式图像直接转换为 LaTeX 字符串：

```
# math_ocr_model.py
class Im2LatexModel(nn.Layer):
    """
    数学公式图像→LaTeX 序列模型
    基于 Attention 机制的编解码架构
    """

    def __init__(self, vocab_size: int, embed_size: int = 80,
                  decode_hidden: int = 512, encode_hidden: int = 256):
        super(Im2LatexModel, self).__init__()

        # 编码器: CNN (提取视觉特征)
        self.encoder = nn.Sequential(
            nn.Conv2D(1, 64, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2D(2, 2),
            nn.Conv2D(64, 128, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2D(2, 2),
            nn.Conv2D(128, 256, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2D(256, 256, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2D((2, 1)), # 保留水平分辨率
            nn.Conv2D(256, encode_hidden, kernel_size=3, padding=1), nn.ReLU(),
        )

        # 解码器: LSTM + Attention
        self.decoder = AttentionDecoder(
            encode_hidden, decode_hidden, embed_size, vocab_size
        )

    def forward(self, images, targets=None):
        # 编码图像特征
        features = self.encoder(images) # (N, C, H', W')
        # 解码序列 (训练时使用 teacher forcing)
        logits, attention_weights = self.decoder(features, targets)
        return logits, attention_weights
```

### C.2.2 数学算式语义校验

识别完数学表达式后，系统进行语义校验（判断等式/不等式是否成立）：

```
# math_validator.py
import re
from decimal import Decimal, InvalidOperation

class MathExpressionValidator:
```

```

"""
对识别到的数学表达式进行语义校验
支持：四则运算、分数、简单方程验证
"""

def validate(self, expression: str) -> dict:
    """
    校验数学表达式

    expression: 识别结果 (如 "3 + 5 = 8" 或 "2 × 4 = 8")
    返回: {'is_correct': bool, 'expected': str, 'explanation': str}
    """
    # 标准化符号 (×→*, ÷→/)
    normalized = self._normalize_symbols(expression)

    # 识别等号或不等号
    if '=' in normalized:
        return self._validate_equation(normalized)
    elif '>' in normalized or '<' in normalized:
        return self._validate_inequality(normalized)
    else:
        return {'is_correct': None, 'explanation': '无法判断 (表达式不包含等式)'}

def _validate_equation(self, expr: str) -> dict:
    parts = expr.split('=')
    if len(parts) != 2:
        return {'is_correct': False, 'explanation': '格式错误'}

    left_expr, right_expr = parts[0].strip(), parts[1].strip()

    try:
        left_val = self._safe_eval(left_expr)
        right_val = self._safe_eval(right_expr)
        is_correct = abs(left_val - right_val) < 1e-9

        return {
            'is_correct': is_correct,
            'left_value': str(left_val),
            'right_value': str(right_val),
            'expected': f"{left_expr} = {left_val}",
            'explanation': '正确' if is_correct else f'等号左边={left_val}, 右边={right_val}, 不相等'
        }
    except Exception as e:
        return {'is_correct': False, 'explanation': f'计算出错: {str(e)}'}

def _safe_eval(self, expr: str) -> Decimal:
    """安全的数学表达式求值 (防注入攻击)"""
    # 只允许数字、运算符、括号、小数点
    if not re.match(r'^[\d\s\+\-\*\\/\(\)\.]+\$', expr):
        raise ValueError(f"不安全的表达式: {expr}")
    # 使用 Decimal 保证精度
    return Decimal(str(eval(expr)))

def _normalize_symbols(self, expr: str) -> str:

```



```
return (expr.replace('x', '*').replace('÷', '/')
        .replace('=', '=').replace('-', '-'))
```

## C.3 笔顺评分模型

### C.3.1 笔顺检测算法

```
# stroke_order_evaluator.py
import numpy as np
from dataclasses import dataclass

@dataclass
class StrokeFeatures:
    """笔画特征向量"""
    start_x: float      # 起点 x 坐标 (归一化 0~1)
    start_y: float      # 起点 y 坐标
    end_x: float        # 终点 x 坐标
    end_y: float        # 终点 y 坐标
    direction_angle: float # 主方向角度 (0~360度)
    length: float       # 笔画长度 (归一化)
    curvature: float    # 曲率 (越小越直)

class StrokeOrderEvaluator:
    """
    汉字笔顺评分器
    基于标准笔顺库比对学生书写笔顺
    """

    def __init__(self, stroke_library_path: str):
        """加载标准笔顺库 (包含 8105 个通用规范汉字的笔顺数据)"""
        with open(stroke_library_path, 'r', encoding='utf-8') as f:
            self.library = json.load(f)

    def evaluate(self, character: str,
                 written_strokes: list,
                 strict_mode: bool = False) -> dict:
        """
        评估学生书写笔顺

        written_strokes: 学生书写的笔画序列 (每条笔画为 InkPoint 列表)
        返回: 详细评分结果
        """
        if character not in self.library:
            return {'error': f'字符 {character} 不在笔顺库中'}

        standard_strokes = self.library[character]['strokes']
        standard_count = len(standard_strokes)
        written_count = len(written_strokes)

        # 笔画数量对比
        stroke_count_score = 100 if written_count == standard_count else max(
            0, 100 - abs(written_count - standard_count) * 20
```

```

    )

    # 逐笔顺序比对 (对齐最近的标准笔画)
    order_errors = []
    for i, written in enumerate(written_strokes):
        if i >= standard_count:
            order_errors.append({'stroke': i+1, 'error': '多余的笔画'})
            continue

        written_feat = self._extract_features(written)
        expected_feat = standard_strokes[i]['features']

        # 方向角度偏差 (容忍度: 严格模式15°, 普通模式30°)
        angle_diff = self._angle_diff(written_feat.direction_angle,
                                       expected_feat['direction_angle'])
        tolerance = 15 if strict_mode else 30

        if angle_diff > tolerance:
            order_errors.append({
                'stroke': i+1,
                'error': f'方向错误 (应为{expected_feat["name"]}, 偏差{angle_diff:.1f}'
            })

    # 计算笔顺得分
    error_count = len(order_errors)
    order_score = max(0, 40 - error_count * 8) # 笔顺满分40分, 每错一笔扣8分

    # 字形得分 (基于关键点位置偏差)
    shape_score = self._evaluate_shape(character, written_strokes)

    # 比例得分 (基于整体字形比例)
    proportion_score = self._evaluate_proportion(character, written_strokes)

    total_score = order_score + shape_score + proportion_score

    return {
        'total_score': min(100, total_score),
        'stroke_order_score': order_score,
        'shape_score': shape_score,
        'proportion_score': proportion_score,
        'stroke_count_match': written_count == standard_count,
        'errors': order_errors,
        'grade': self._score_to_grade(total_score)
    }

def _score_to_grade(self, score: float) -> str:
    if score >= 90: return '优秀'
    elif score >= 75: return '良好'
    elif score >= 60: return '合格'
    else: return '需加强'

def _angle_diff(self, a1: float, a2: float) -> float:
    """计算两个角度的最小绝对差值 (处理360°环绕) """
    diff = abs(a1 - a2) % 360
    return min(diff, 360 - diff)

```

```

def _extract_features(self, stroke_points: list) -> StrokeFeatures:
    """从笔画点序列提取特征向量"""
    if len(stroke_points) < 2:
        return StrokeFeatures(0, 0, 0, 0, 0, 0, 0)

    xs = [p['x'] for p in stroke_points]
    ys = [p['y'] for p in stroke_points]

    # 主方向: 起点到终点的方向角
    dx = xs[-1] - xs[0]
    dy = ys[-1] - ys[0]
    angle = np.degrees(np.arctan2(dy, dx)) % 360
    length = np.sqrt(dx**2 + dy**2)

    # 曲率: 用笔画路径长度/直线长度比估算
    path_length = sum(
        np.sqrt((xs[i+1]-xs[i])**2 + (ys[i+1]-ys[i])**2)
        for i in range(len(xs)-1)
    )
    curvature = path_length / max(length, 1e-6)

    return StrokeFeatures(
        start_x=xs[0], start_y=ys[0],
        end_x=xs[-1], end_y=ys[-1],
        direction_angle=angle,
        length=length,
        curvature=curvature
    )

```

## C.4 模型服务化与部署

### C.4.1 模型热加载机制

```

# model_manager.py
class ModelManager:
    """
    AI 模型管理器
    支持模型热加载（不停服更新）和 A/B 测试
    """

    def __init__(self, model_registry_path: str):
        self.registry_path = model_registry_path
        self._models: dict[str, dict] = {}
        self._lock = asyncio.Lock()

        # 启动文件监听, 自动检测模型更新
        self._start_model_watcher()

    async def load_model(self, model_name: str, model_version: str):
        """加载指定版本的模型到内存"""
        model_path = f"{self.registry_path}/{model_name}/{model_version}"

```

```

    async with self._lock:
        if model_name in self._models:
            # 先保留旧模型（用于 A/B 对比或回滚）
            old_model = self._models[model_name]
            self._models[f"{model_name}_backup"] = old_model

        # 异步加载新模型
        model = await asyncio.get_event_loop().run_in_executor(
            None, self._load_from_disk, model_path
        )

        self._models[model_name] = {
            'model': model,
            'version': model_version,
            'loaded_at': time.time(),
            'call_count': 0,
            'total_latency': 0.0
        }

        logger.info(f"模型 {model_name} v{model_version} 加载完成")

def get_model(self, model_name: str, use_ab_test: bool = False):
    """获取当前活跃模型"""
    if use_ab_test and f"{model_name}_backup" in self._models:
        # A/B 测试: 10% 流量路由到旧模型
        if random.random() < 0.1:
            return self._models[f"{model_name}_backup"]['model']
    return self._models[model_name]['model']

```

## C.4.2 API 限流与队列

```

# rate_limiter.py (AI 引擎请求限流)
import asyncio
from collections import defaultdict

class TokenBucketRateLimiter:
    """
    令牌桶算法限流器
    防止 AI 推理服务被突发请求打垮
    """

    def __init__(self, rate: float, capacity: float):
        """
        rate: 令牌补充速率（请求/秒）
        capacity: 桶容量（最大突发量）
        """
        self.rate = rate
        self.capacity = capacity
        self.tokens: dict[str, float] = defaultdict(lambda: capacity)
        self.last_refill: dict[str, float] = defaultdict(time.time)
        self._lock = asyncio.Lock()

    async def acquire(self, key: str, tokens: float = 1.0) -> bool:
        """

```

```

    尝试获取令牌
    key: 限流维度 (如 app_key 或 user_id)
    返回: True=可以执行, False=被限流
    """
    async with self._lock:
        now = time.time()
        elapsed = now - self.last_refill[key]

        # 按时间补充令牌
        self.tokens[key] = min(
            self.capacity,
            self.tokens[key] + elapsed * self.rate
        )
        self.last_refill[key] = now

        if self.tokens[key] >= tokens:
            self.tokens[key] -= tokens
            return True
        return False

# 在 AI 识别接口中使用限流器
class OcrController:
    def __init__(self):
        self.rate_limiter = TokenBucketRateLimiter(
            rate=50,          # 50 请求/秒
            capacity=100      # 最大突发 100 个请求
        )

    async def recognize(self, request: OcrRequest) -> OcrResponse:
        # 按 AppKey 限流
        allowed = await self.rate_limiter.acquire(request.app_key)
        if not allowed:
            raise RateLimitExceededException("请求频率超限, 请降低调用频率")

        # 执行识别
        return await self.ocr_service.recognize(request)
```

## 附录D 接口完整清单

### D.1 识别接口

接口	方法	路径	QPS限制	说明
汉字识别	POST	/api/v1/ocr/text	50/s/AppKey	识别手写汉字
数学识别	POST	/api/v1/ocr/math	30/s/AppKey	识别数学表达式

接口	方法	路径	QPS限制	说明
批量识别	POST	/api/v1/ocr/batch	10/s/AppKey	批量识别（最多20条/请求）
笔顺评分	POST	/api/v1/ocr/stroke-order	30/s/AppKey	汉字笔顺评估与评分
书写质量	POST	/api/v1/ocr/quality	30/s/AppKey	书写质量综合评分
作业批改	POST	/api/v1/correction/assignment	5/s/AppKey	完整作业批改流程

## D.2 管理接口

接口	方法	路径	说明
查询识别配额	GET	/api/v1/quota/current	查询当前 AppKey 的识别配额余量
查询调用统计	GET	/api/v1/statistics/calls	按时间段统计API调用次数和成功率
获取模型版本	GET	/api/v1/models/versions	查询当前生产模型版本信息
异步任务状态	GET	/api/v1/tasks/{task_id}	查询异步批改任务的执行状态

## 附录E 部署与性能

### E.1 GPU 推理服务部署

```
# docker-compose.gpu.yml (GPU 推理节点)
services:
  ai-engine:
    image: registry.writech.com/ai-engine:1.0.0
    runtime: nvidia
    environment:
      NVIDIA_VISIBLE_DEVICES: all
      PADDLE_FLAGS: "FLAGS_fraction_of_gpu_memory_to_use=0.8"
      MODEL_BATCH_SIZE: 16
      MAX_CONCURRENT_REQUESTS: 64
    volumes:
      - /data/models:/app/models:ro # 只读挂载模型目录
    deploy:
      resources:
        reservations:
          devices:
```

```
- driver: nvidia
  count: 1
  capabilities: [gpu]
```

E.2 推理性能基准测试

模型	硬件	批次大小	平均延迟	吞吐量
汉字识别 (CRNN)	Tesla T4	16	8ms/字	2000字/秒
数学识别 (lm2Latex)	Tesla T4	8	25ms/式	320式/秒
笔顺评分	CPU (16核)	1	5ms/字	200字/秒
书写质量 (综合)	Tesla T4	16	15ms/字	1000字/秒

本文档版权归深圳自然写科技有限公司所有， 仅用于软件著作权登记鉴别。

附录F 核心算法详细实现

F.1 DB文本检测网络实现

AI引擎使用DB (Differentiable Binarization) 算法进行文字区域检测，相比传统固定阈值二值化，DB通过可学习的阈值映射提升检测准确率。

```
# engine/detection/db_detector.py
import numpy as np
import cv2
import onnxruntime as ort
from typing import List, Tuple

class DBTextDetector:
    """
    DB文本检测器 (基于ONNX模型推理)
    输入: RGB图像 (归一化到[-1,1])
    输出: 文字区域轮廓列表 (像素坐标)
    """

    # 预处理参数 (训练时统计的均值和标准差)
    IMG_MEAN = np.array([0.485, 0.456, 0.406], dtype=np.float32)
    IMG_STD = np.array([0.229, 0.224, 0.225], dtype=np.float32)

    # DB后处理参数
    DB_THRESH = 0.3 # 概率图二值化阈值
    DB_BOX_THRESH = 0.5 # 检测框置信度阈值
    DB_UNCLIP_RATIO = 1.6 # 文字框扩张比例
```

```

def __init__(self, model_path: str):
    opts = ort.SessionOptions()
    opts.intra_op_num_threads = 4
    opts.inter_op_num_threads = 2
    opts.graph_optimization_level = ort.GraphOptimizationLevel.ORT_ENABLE_ALL
    self.session = ort.InferenceSession(
        model_path,
        sess_options=opts,
        providers=['CUDAExecutionProvider', 'CPUExecutionProvider']
    )
    self.input_name = self.session.get_inputs()[0].name

def detect(self, image_bgr: np.ndarray) -> List[np.ndarray]:
    """
    检测图片中的文字区域
    Returns:
        文字区域轮廓列表, 每个轮廓为(N, 2)形状的numpy数组
    """
    # 1. 预处理: 等比缩放到32的倍数
    h, w = image_bgr.shape[:2]
    target_h = self._align_32(min(960, h))
    target_w = self._align_32(min(960, w))
    scale_h, scale_w = h / target_h, w / target_w

    resized = cv2.resize(image_bgr, (target_w, target_h))
    img_rgb = cv2.cvtColor(resized, cv2.COLOR_BGR2RGB).astype(np.float32) / 255.0
    img_norm = (img_rgb - self.IMG_MEAN) / self.IMG_STD
    input_tensor = img_norm.transpose(2, 0, 1)[np.newaxis] # NCHW

    # 2. 模型推理
    outputs = self.session.run(None, {self.input_name: input_tensor})
    prob_map = outputs[0][0, 0] # (H, W) 概率图

    # 3. DB后处理: 概率图 -> 二值图 -> 轮廓提取 -> Unclip扩张
    binary_map = (prob_map > self.DB_THRESH).astype(np.uint8) * 255
    contours, _ = cv2.findContours(binary_map, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)

    text_regions = []
    for cnt in contours:
        if cv2.contourArea(cnt) < 100: # 过滤极小区域
            continue

        # 计算轮廓置信度 (取概率图在轮廓内的均值)
        mask = np.zeros_like(prob_map, dtype=np.uint8)
        cv2.drawContours(mask, [cnt], 0, 1, -1)
        box_score = float(prob_map[mask == 1].mean())

        if box_score < self.DB_BOX_THRESH:
            continue

        # Unclip扩张 (扩大文字框, 避免漏识别边缘字符)
        expanded = self._unclip(cnt, self.DB_UNCLIP_RATIO)

        # 还原到原始图像坐标
        expanded[:, 0] = np.clip(expanded[:, 0] * scale_w, 0, w - 1)

```



```

        expanded[:, 1] = np.clip(expanded[:, 1] * scale_h, 0, h - 1)
        text_regions.append(expanded.astype(np.int32))

    return text_regions

def _unclip(self, contour: np.ndarray, ratio: float) -> np.ndarray:
    """使用Polygon Offset算法扩张文字框"""
    import pyclipper
    poly = contour.reshape(-1, 2).astype(np.float32)
    area = cv2.contourArea(contour)
    peri = cv2.arcLength(contour, True)
    if peri < 1e-6:
        return poly
    distance = area * ratio / peri

    pco = pyclipper.PyclipperOffset()
    pco.AddPath(poly.astype(np.int32).tolist(), pyclipper.JT_ROUND,
pyclipper.ET_CLOSEDPOLYGON)
    solution = pco.Execute(int(distance))
    if not solution:
        return poly
    return np.array(solution[0], dtype=np.float32)

@staticmethod
def _align_32(n: int) -> int:
    return max(32, (n + 31) // 32 * 32)

```

## F.2 CRNN文字识别实现

```

# engine/recognition/crnn_recognizer.py
import numpy as np
import onnxruntime as ort
from typing import Tuple, List

class CRNNRecognizer:
    """
    CRNN文字识别器 (CNN + LSTM + CTC解码)
    输入: 文字行图像 (归一化到32×320)
    输出: 识别文本 + 置信度
    """

    IMG_HEIGHT = 32
    IMG_WIDTH = 320

    def __init__(self, model_path: str, charset_path: str):
        self.session = ort.InferenceSession(
            model_path,
            providers=['CUDAExecutionProvider', 'CPUExecutionProvider']
        )
        self.input_name = self.session.get_inputs()[0].name

        # 加载字符集 (包含空白符)
        with open(charset_path, 'r', encoding='utf-8') as f:
            self.charset = ['BLANK'] + [c.strip() for c in f.readlines()]

```

```

self.blank_idx = 0

def recognize(self, line_image_bgr: np.ndarray) -> Tuple[str, float]:
    """
    识别单行文字图像
    Returns:
        (text, confidence) 识别文本和平均置信度
    """
    # 预处理: 灰度化, 等比缩放到32高度, 宽度最大320
    import cv2
    gray = cv2.cvtColor(line_image_bgr, cv2.COLOR_BGR2GRAY)
    h, w = gray.shape
    target_w = min(self.IMG_WIDTH, int(w * self.IMG_HEIGHT / h))
    resized = cv2.resize(gray, (target_w, self.IMG_HEIGHT))

    # 填充到标准宽度
    canvas = np.zeros((self.IMG_HEIGHT, self.IMG_WIDTH), dtype=np.float32)
    canvas[:, :target_w] = resized.astype(np.float32)
    canvas = (canvas / 255.0 - 0.5) / 0.5 # 归一化到[-1, 1]

    # NCHW格式 (1, 1, 32, 320)
    input_tensor = canvas[np.newaxis, np.newaxis]

    # 推理: 输出shape为 (T, 1, num_classes), T是时间步
    logits = self.session.run(None, {self.input_name: input_tensor})[0]
    probs = self._softmax(logits[:, 0, :]) # (T, num_classes)

    # CTC贪心解码
    text, confidence = self._ctc_greedy_decode(probs)
    return text, confidence

def _ctc_greedy_decode(self, probs: np.ndarray) -> Tuple[str, float]:
    """CTC贪心解码 (逐时间步取最大概率)"""
    indices = np.argmax(probs, axis=-1) # (T,)
    confs = probs[np.arange(len(indices)), indices]

    # 折叠重复并移除blank
    chars = []
    conf_list = []
    prev = -1
    for i, (idx, conf) in enumerate(zip(indices, confs)):
        if idx != prev and idx != self.blank_idx:
            if 0 < idx < len(self.charset):
                chars.append(self.charset[idx])
                conf_list.append(float(conf))
            prev = idx

    text = ''.join(chars)
    avg_conf = float(np.mean(conf_list)) if conf_list else 0.0
    return text, avg_conf

@staticmethod
def _softmax(x: np.ndarray) -> np.ndarray:
    e = np.exp(x - x.max(axis=-1, keepdims=True))
    return e / e.sum(axis=-1, keepdims=True)

```

### F.3 笔顺评分模型推理

```
# engine/stroke_eval/stroke_evaluator.py
import numpy as np
import onnxruntime as ort
from typing import List, Dict

class StrokeOrderEvaluator:
    """
    笔顺评分引擎
    - 输入：归一化的笔迹序列（时序坐标点）
    - 模型：双向LSTM，输出每笔笔顺正误概率
    - 配合BKT校准：根据学生历史正确率校准当前评分
    """

    MAX_STROKES = 30    # 最多30笔
    MAX_POINTS = 50     # 每笔最多50点
    FEATURE_DIM = 6     # 特征维度：(x, y, dx, dy, pressure, is_last_point)

    def __init__(self, model_path: str):
        self.session = ort.InferenceSession(
            model_path,
            providers=['CUDAExecutionProvider', 'CPUExecutionProvider']
        )

    def evaluate(self, strokes: List[List[Dict]],
                character: str,
                bkt_mastery: float = 0.5) -> Dict:
        """
        评估笔顺质量
        Args:
            strokes:      [[{'x':..., 'y':..., 'p':...}, ...], ...] 各笔笔迹点
            character:    被书写的字符
            bkt_mastery:  学生当前掌握度（用于校准）
        Returns:
            {'score': float, 'errors': list, 'feedback': str}
        """
        # 特征提取
        features = self._extract_features(strokes) # (S, P, F)

        # 填充到固定尺寸 (MAX_STROKES, MAX_POINTS, FEATURE_DIM)
        padded = np.zeros((self.MAX_STROKES, self.MAX_POINTS, self.FEATURE_DIM),
                          dtype=np.float32)
        s = min(len(features), self.MAX_STROKES)
        padded[:s] = features[:s]

        # 模型推理
        input_tensor = padded[np.newaxis] # (1, S, P, F)
        stroke_probs = self.session.run(None, {'input': input_tensor})[0][0] # (S, 2)

        # 计算各笔正确率
        n_strokes = min(len(strokes), self.MAX_STROKES)
        correct_probs = stroke_probs[:n_strokes, 1] # 正确类概率

        # 综合评分（加权均值，前几笔权重略高）
```

```

weights = np.array([1.2 if i < 3 else 1.0 for i in range(n_strokes)])
raw_score = float(np.average(correct_probs, weights=weights))

# BKT校准: 高掌握度时适当提高评分 (鼓励效应)
calibrated_score = raw_score * (0.85 + 0.15 * bkt_mastery)
final_score = min(100.0, calibrated_score * 100)

# 生成错误反馈
errors = []
for i, prob in enumerate(correct_probs):
    if prob < 0.5:
        errors.append({
            'stroke_index': i + 1,
            'confidence': float(1 - prob),
            'description': f'第{i+1}笔笔顺可能有误'
        })

feedback = self._generate_feedback(final_score, errors)
return {'score': round(final_score, 1), 'errors': errors, 'feedback': feedback}

def _extract_features(self, strokes):
    result = []
    for stroke in strokes[:self.MAX_STROKES]:
        pts = stroke[:self.MAX_POINTS]
        features = []
        for j, pt in enumerate(pts):
            dx = pt['x'] - pts[j-1]['x'] if j > 0 else 0.0
            dy = pt['y'] - pts[j-1]['y'] if j > 0 else 0.0
            features.append([pt['x'], pt['y'], dx, dy,
                             pt.get('pressure', 0.5),
                             1.0 if j == len(pts)-1 else 0.0])

        # 填充到MAX_POINTS
        while len(features) < self.MAX_POINTS:
            features.append([0.0] * self.FEATURE_DIM)
        result.append(features)
    return np.array(result, dtype=np.float32)

@staticmethod
def _generate_feedback(score: float, errors: list) -> str:
    if score >= 90:
        return "笔顺非常标准, 继续保持!"
    elif score >= 75:
        error_strokes = [str(e['stroke_index']) for e in errors]
        return f"整体不错, 第{'、'.join(error_strokes)}笔笔顺需要注意。"
    elif score >= 60:
        return f"笔顺有{len(errors)}处需要改进, 请参考标准笔顺练习。"
    else:
        return "笔顺与标准差异较大, 请仔细查看示范, 重新练习。"

```

## F.4 令牌桶限流实现

```

# middleware/rate_limiter.py
import time, threading
from functools import wraps

```

```

from flask import request, jsonify

class TokenBucketRateLimiter:
    """
    令牌桶限流算法
    每个AppKey独立维护一个令牌桶，以固定速率添加令牌
    """

    def __init__(self, rate: float = 50.0, capacity: float = 100.0):
        """
        Args:
            rate:      令牌补充速率 (个/秒)
            capacity:  令牌桶容量 (最大突发量)
        """
        self.rate = rate
        self.capacity = capacity
        self._buckets: dict = {} # app_key -> (tokens, last_time)
        self._lock = threading.Lock()

    def consume(self, app_key: str, n: int = 1) -> bool:
        """消费n个令牌，返回True表示通过，False表示限流"""
        with self._lock:
            now = time.monotonic()
            if app_key not in self._buckets:
                self._buckets[app_key] = [self.capacity, now]

            tokens, last_time = self._buckets[app_key]

            # 补充令牌 (根据时间差)
            elapsed = now - last_time
            tokens = min(self.capacity, tokens + elapsed * self.rate)
            self._buckets[app_key][1] = now

            if tokens >= n:
                self._buckets[app_key][0] = tokens - n
                return True
            else:
                self._buckets[app_key][0] = tokens
                return False

# Flask装饰器
_limiter = TokenBucketRateLimiter(rate=50.0, capacity=100.0)

def rate_limit(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        app_key = request.headers.get('X-App-Key', 'anonymous')
        if not _limiter.consume(app_key):
            return jsonify({'code': 429, 'message': '请求过于频繁，请稍后重试'}), 429
        return f(*args, **kwargs)
    return decorated

```

## 附录F 补充算法与接口规格

## F.1 多模型集成推理框架

### F.1.1 模型路由策略

```
# model_router.py
from enum import Enum
from typing import Dict, Any
import numpy as np

class ModelType(Enum):
    OCR_FAST = "ocr_fast"          # 轻量模型, 延迟<50ms
    OCR_ACCURATE = "ocr_accurate"  # 精准模型, 延迟<200ms
    MATH_FORMULA = "math_formula"  # 数学公式专用
    STROKE_EVAL = "stroke_eval"    # 笔顺评分

class ModelRouter:
    """根据请求特征自动路由到最优模型"""

    def __init__(self, models: Dict[ModelType, Any]):
        self.models = models
        self.stats = {m: {"count": 0, "avg_ms": 0} for m in ModelType}

    def route(self, request: dict) -> ModelType:
        content_type = request.get("content_type", "text")
        quality = request.get("quality", "normal")

        if content_type == "math":
            return ModelType.MATH_FORMULA

        if content_type == "stroke":
            return ModelType.STROKE_EVAL

        # 根据图像复杂度自动选择
        if quality == "high" or self._is_complex_image(request.get("image")):
            return ModelType.OCR_ACCURATE
        else:
            return ModelType.OCR_FAST

    def _is_complex_image(self, image: np.ndarray) -> bool:
        if image is None:
            return False
        # 基于图像方差判断复杂度
        variance = np.var(image)
        return variance > 1500

    async def infer(self, request: dict) -> dict:
        model_type = self.route(request)
        model = self.models[model_type]

        import time
        start = time.perf_counter()
        result = await model.predict(request["image"])
        elapsed_ms = (time.perf_counter() - start) * 1000
```

```

# 更新统计
s = self.stats[model_type]
s["avg_ms"] = (s["avg_ms"] * s["count"] + elapsed_ms) / (s["count"] + 1)
s["count"] += 1

return {
    "result": result,
    "model": model_type.value,
    "latency_ms": round(elapsed_ms, 2)
}

```

## F.2 OCR后处理管道

### F.2.1 文本置信度过滤与纠错

```

# ocr_postprocess.py
import re
from dataclasses import dataclass
from typing import List, Optional

@dataclass
class OcrWord:
    text: str
    confidence: float
    bbox: tuple # (x1, y1, x2, y2)

class OcrPostProcessor:
    CONF_THRESHOLD_HIGH = 0.90
    CONF_THRESHOLD_LOW = 0.60

    # 常见混淆字符映射 (OCR错误→正确)
    CONFUSION_MAP = {
        "0": "O", "1": "l", "rn": "m", "cl": "d",
        "己": "已", "未": "末", "士": "士"
    }

    def __init__(self, language_model=None):
        self.lm = language_model # 可选语言模型用于纠错

    def process(self, words: List[OcrWord]) -> str:
        # 1. 过滤低置信度词
        filtered = [w for w in words if w.confidence >= self.CONF_THRESHOLD_LOW]

        # 2. 对中等置信度词进行纠错尝试
        corrected = []
        for word in filtered:
            if word.confidence < self.CONF_THRESHOLD_HIGH:
                fixed = self._try_correct(word.text)
                corrected.append(fixed)
            else:
                corrected.append(word.text)

        # 3. 拼接文本并后处理

```

```

        text = " ".join(corrected)
        text = self._normalize_spaces(text)
        text = self._fix_punctuation(text)

        return text

    def _try_correct(self, text: str) -> str:
        result = text
        for wrong, right in self.CONFUSION_MAP.items():
            result = result.replace(wrong, right)

        if self.lm:
            lm_result = self.lm.correct(result)
            if lm_result.score > 0.8:
                return lm_result.text

        return result

    def _normalize_spaces(self, text: str) -> str:
        # 移除中文字符间多余空格
        text = re.sub(r'([\u4e00-\u9fff])\s+([\u4e00-\u9fff])', r'\1\2', text)
        return text.strip()

    def _fix_punctuation(self, text: str) -> str:
        # 标准化标点符号
        replacements = [(',', '，', '，'), (',.', '。'), ('?', '？'), ('!', '！')]
        for eng, chn in replacements:
            # 仅在中文上下文中替换
            text = re.sub(f'([\u4e00-\u9fff]){re.escape(eng)}',
                          f'\1{chn}', text)

        return text

```

## F.3 异步任务队列

```

# async_task_queue.py
import asyncio
from collections import deque
from dataclasses import dataclass, field
from typing import Callable, Any

@dataclass
class Task:
    id: str
    priority: int
    func: Callable
    args: tuple
    future: asyncio.Future = field(default_factory=asyncio.Future)

class PriorityTaskQueue:
    """优先级异步任务队列，高优先级任务优先执行"""

    def __init__(self, workers: int = 4):
        self.queue = asyncio.PriorityQueue()
        self.workers = workers

```



```

self._counter = 0 # 用于相同优先级时保持FIFO顺序

async def submit(self, func: Callable, *args, priority: int = 5) -> Any:
    future = asyncio.get_event_loop().create_future()
    task = Task(
        id=f"task_{self._counter}",
        priority=priority,
        func=func,
        args=args,
        future=future
    )
    self._counter += 1
    # PriorityQueue按(priority, counter)排序, priority越小优先级越高
    await self.queue.put((priority, self._counter, task))
    return await future

async def _worker(self):
    while True:
        _, _, task = await self.queue.get()
        try:
            if asyncio.iscoroutinefunction(task.func):
                result = await task.func(*task.args)
            else:
                result = await asyncio.get_event_loop().run_in_executor(
                    None, task.func, *task.args)
            task.future.set_result(result)
        except Exception as e:
            task.future.set_exception(e)
        finally:
            self.queue.task_done()

async def start(self):
    for _ in range(self.workers):
        asyncio.create_task(self._worker())

```

## 附录G 补充技术规格

### G.1 模型热加载无缝切换

```

# model_hot_swap.py
import threading
import time
from typing import Optional

class HotSwapModelManager:
    """模型热加载管理器, 支持零停机切换模型版本"""

    def __init__(self):
        self._current_model = None
        self._pending_model = None
        self._lock = threading.RWLock()

```

```

self._request_count = 0

def load_new_version(self, model_path: str, model_type: str):
    """在后台加载新模型版本"""
    def _load():
        import torch
        new_model = torch.jit.load(model_path)
        new_model.eval()

        # 等待当前请求处理完成
        while self._request_count > 0:
            time.sleep(0.01)

        with self._lock.write():
            old_model = self._current_model
            self._current_model = new_model
            self._pending_model = None

        # 释放旧模型内存
        if old_model is not None:
            del old_model
            torch.cuda.empty_cache()

        print(f"Model {model_type} hot-swapped to {model_path}")

    import threading
    t = threading.Thread(target=_load, daemon=True)
    t.start()

def infer(self, inputs):
    """推理时持有读锁，防止切换过程中的并发问题"""
    with self._lock.read():
        self._request_count += 1
        try:
            return self._current_model(inputs)
        finally:
            self._request_count -= 1

```

## G.2 GPU显存监控

```

# gpu_monitor.py
import subprocess
import json

def get_gpu_stats() -> dict:
    """获取GPU使用统计（通过nvidia-smi）"""
    try:
        result = subprocess.run([
            "nvidia-smi", "--query=
gpu=name,memory.used,memory.total,utilization.gpu,temperature.gpu",
            "--format=csv,noheader,nounits"
        ], capture_output=True, text=True, timeout=5)

        if result.returncode != 0:

```

```

        return {}

    parts = [p.strip() for p in result.stdout.strip().split(",")]
    return {
        "name": parts[0],
        "memory_used_mb": int(parts[1]),
        "memory_total_mb": int(parts[2]),
        "memory_util_pct": round(int(parts[1]) / int(parts[2]) * 100, 1),
        "gpu_util_pct": int(parts[3]),
        "temperature_c": int(parts[4])
    }
except Exception as e:
    return {"error": str(e)}

def check_oom_risk(threshold_pct: float = 90.0) -> bool:
    """检查是否有显存溢出风险"""
    stats = get_gpu_stats()
    if not stats or "memory_util_pct" not in stats:
        return False
    return stats["memory_util_pct"] >= threshold_pct

```

## G.3 识别结果缓存

```

# result_cache.py
import hashlib
import json
import redis
from functools import wraps

class OcrResultCache:
    """基于Redis的识别结果缓存，提高重复图片的响应速度"""

    def __init__(self, redis_client: redis.Redis, ttl_seconds: int = 3600):
        self.redis = redis_client
        self.ttl = ttl_seconds

    def get_cache_key(self, image_bytes: bytes, options: dict) -> str:
        """基于图片内容和识别选项生成缓存键"""
        content_hash = hashlib.sha256(image_bytes).hexdigest()
        options_str = json.dumps(options, sort_keys=True)
        options_hash = hashlib.md5(options_str.encode()).hexdigest()[:8]
        return f"ocr:result:{content_hash}:{options_hash}"

    def get(self, key: str) -> Optional[dict]:
        value = self.redis.get(key)
        if value:
            return json.loads(value)
        return None

    def set(self, key: str, result: dict):
        self.redis.setex(key, self.ttl, json.dumps(result, ensure_ascii=False))

    def cached_ocr(self, ocr_func):
        """装饰器：为OCR函数添加缓存"""

```

```

@wraps(ocr_func)
async def wrapper(image_bytes: bytes, **options):
    key = self.get_cache_key(image_bytes, options)
    cached = self.get(key)
    if cached:
        cached["from_cache"] = True
        return cached

    result = await ocr_func(image_bytes, **options)
    self.set(key, result)
    return result
return wrapper

```

## 附录H 补充技术规格

### H.1 数学公式识别增强

```

# math_formula_postprocess.py
import re

class MathFormulaPostProcessor:
    """数学公式识别后处理：LaTeX语法规范化"""

    # 常见OCR错误修正
    CORRECTIONS = {
        r'\\frac\s*{': r'\\frac{',
        r'\\sqrt\s*{': r'\\sqrt{',
        r'\\sum\s*_': r'\\sum_',
        r'x\^2': r'x^{2}',      # 补充缺失的花括号
        r'x\^(\d)': r'x^{\1}',
        r'([a-z])\^([a-z])': r'\1^{\2}',
    }

    def process(self, latex: str) -> str:
        result = latex.strip()

        # 应用修正规则
        for pattern, replacement in self.CORRECTIONS.items():
            result = re.sub(pattern, replacement, result)

        # 确保公式包含在$...$中
        if not result.startswith('$'):
            result = f'${result}$'

        # 验证括号平衡
        if not self._check_braces(result):
            result = self._fix_braces(result)

        return result

    def _check_braces(self, s: str) -> bool:

```

```
count = 0
for c in s:
    if c == '{': count += 1
    elif c == '}': count -= 1
    if count < 0: return False
return count == 0

def _fix_braces(self, s: str) -> str:
    """自动补全缺失的右括号"""
    count = sum(1 if c == '{' else -1 if c == '}' else 0 for c in s)
    if count > 0:
        s = s.rstrip('$') + '}' * count
        if not s.endswith('$'): s += '$'
    return s
```

## H.2 版本历史

版本号	发布日期	变更说明	负责人
V1.0.0	2024-01-15	初始版本，实现汉字/数字OCR基础识别	AI组
V1.1.0	2024-03-20	新增数学公式识别（lm2Latex模型）	算法组
V1.2.0	2024-05-15	引入TensorRT INT8量化，GPU推理延迟降低40%	工程组
V1.3.0	2024-07-10	新增笔顺评分功能，BKT算法校准	AI组
V1.4.0	2024-09-01	添加识别结果Redis缓存，重复图片响应<5ms	工程组
V1.5.0	2024-11-15	支持模型热加载，零停机版本升级	工程组

本文档版权归深圳自然写科技有限公司所有，仅用于软件著作权登记鉴别。