

选题确认报告

组长：

组员：

1. 选题与目标

我们选择实现 **OCR（Optical Character Recognition）图像文字识别** 应用，目标是在**浏览器/移动端**对拍照或截图中的文字进行识别与导出，支持中英文与常见排版。项目为**计算密集且对时延敏感**的应用：用户希望在 1-2 秒内看到识别结果。

核心功能

- 图片输入：相机/本地上传/截图粘贴
- 预处理：旋转矫正、去噪、二值化、增强
- 文本检测：定位文字区域，支持多行、多语言
- 文本识别：按行/块识别为可编辑文本
- 后处理：版面重建（段落/表格粗对齐）、导出（TXT/JSON/Markdown）

典型场景：打印件、屏幕截图。

2. 总体架构与计算划分（Browser/Mobile ↔ Server/Cloud）

为兼顾时延与精度，采用**前后端协同**的分层架构：

- **端侧（Browser/Mobile）**：完成轻量级预处理、缩放、推理前后处理；可选运行**轻量检测模型**（WASM/WebGPU），并在网络较差时提供“仅端侧识别”的降级路径（低精度、快）。
- **服务端（Server/Cloud）**：提供**标准/高精度识别**（如大模型/Transformer）、版面分析与表格结构化；支持并发与批处理。
- **数据流**：图片 → 端侧预处理/（可选）检测 → 发送 ROI/特征到服务端 → 服务端识别 → 返回文本与版面 → 端侧渲染与导出。

阶段	端侧（浏览器/移动端）	服务端（云/远端）	说明
预处理	✓		降噪/缩放/纠偏，减少上传体积
文本检测	✓（轻量/可选）	✓（标准/高精度）	端侧先粗定位，服务端精修

阶段	端侧（浏览器/移动端）	服务端（云/远端）	说明
文本识别		✓	高精度模型、并发优化
版面重建	✓（渲染）	✓（结构推断）	服务端给结构，端侧负责呈现
导出	✓		TXT/JSON/Markdown

3. 源码模块结构

我们将采用单仓多包结构，区分端侧与服务端，并共享协议与工具。

ocr-app/

└─ client/ # 端侧应用（Web/移动）

└─ src/

└─ pages/ # 页面/路由

└─ components/ # UI 组件（上传、裁剪、结果展示）

└─ pipelines/ # 端侧推理流水线

└─ workers/ # WebWorker 线程 offload

└─ wasm/ # ONNX/WASM 运行时与模型

└─ utils/ # 图像/数学/计时/日志

└─ services/ # 调用 Go Gateway API

└─ tests/

└─ go-gateway/ # Go 编写的 API 网关

└─ cmd/

└─ server/

└─ main.go # 服务启动入口

└─ internal/

└─ handlers/ # HTTP 请求处理器（Gin/Echo）

└─ services/ # 业务逻辑层

```

├── └── └── rpc_client/                # gRPC 客户端 (调用 Python 服务)
├── └── pkg/                          # 项目内可共享的库
├── └── go.mod                        # Go 模块依赖
├── python-inference-service/        # Python 编写的推理服务
├── └── app/
├── └── └── __init__.py
├── └── └── server.py                 # gRPC 服务端启动入口
├── └── └── services/                # 实现 gRPC 接口
├── └── └── └── detector.py           # 文本检测 (DBNet/EAST)
├── └── └── └── recognizer.py         # 文本识别 (CRNN/TrOCR)
├── └── └── └── layout.py             # 版面分析
├── └── └── └── models/               # 模型权重 (ONNX/FP16/INT8)
├── └── └── └── utils/                # 预处理、切片等
├── └── └── benchmarks/              # 压测脚本
├── └── requirements.txt
├── └── protos/                      # Protobuf 协议定义
├── └── ocr.proto                     # 定义 RPC 服务和消息体
├── └── shared/
├── └── └── schemas/                  # 前端与 Go 网关的协议 (TypeScript types)
├── └── └── metrics/                  # 指标埋点与打点约定
├── └── scripts/                     # 数据准备/量化/导出工具
├── └── docs/                         # 架构图/接口说明/演示清单
├── README.md

```

```

ocr-app/
├── client/                          # 端侧应用 (Web/移动)
├── └── src/

```

带格式的: 正文
米

- shared/schemas: 请求（图片/ROI/参数）与响应（文本/坐标/版面）结构体，确保强类型对齐。
- benchmarks: 压测与 A/B 脚本，输出延迟分布、吞吐与召回/准确率指标。

4. 关键技术栈

- 端侧: TypeScript + React/Next.js, WebWorker + WebAssembly/WebGPU;
- 服务端: Python? + FastAPI, 推理框架 (PyTorch);
- 模型路线: 轻量检测 (DBNet-lite/EAST) + 识别 (CRNN/TrOCR/...);
- 质量与度量: 端侧/后端耗时 (TTI、检测/识别分阶段耗时)、端到端时延、网络传输体积。