

项目名称（暂定）

基于 RTX 5070 的多模态视觉模型推理加速与端到端性能优化

一、项目背景与目标

在多模态人机交互（如实时视觉理解、智能感知）中，
模型推理延迟与系统吞吐直接决定了用户体验的“流畅底线”。

本项目聚焦于 **单卡 GPU (RTX 5070) 场景下的 AI 推理加速**，
在不涉及嵌入式开发、不改动模型训练的前提下，通过推理与系统层优化：

- 降低端到端延迟
- 提升实时处理帧率 (FPS)
- 在同等精度下提高系统稳定性

核心目标不是做新算法，而是把已有模型跑得更快、更稳。

二、项目约束条件（现实情况）

- 团队背景：无嵌入式开发经验，AI/工程基础偏初级
- 硬件条件：单张 NVIDIA RTX 5070 GPU
- 经费预算：约 1000 RMB
- 项目目标：按时交付一个可运行、可展示、可量化对比的技术作品

因此，本项目选择：

- 纯软件层优化
 - 基于成熟模型与成熟推理框架
 - 追求“可交付成功率”，而非极限性能
-

三、具体任务定义（做什么）

任务类型（当前选择）

实时视觉推理任务（摄像头输入）

示例：

- 实时目标检测 (YOLO 系列)
- 单张图像推理 / 连续帧推理

输入：

- 摄像头或图片流

输出：

- 检测结果可视化
 - 实时性能指标 (FPS、延迟等)
-

四、Baseline (基线系统)

作为对照组，构建一个“未优化”的标准推理流程：

- 框架：PyTorch
- 精度：FP32
- 推理方式：同步、串行
- 输入尺寸：默认 (如 640×640)
- 运行设备：RTX 5070

基线系统用于回答一个问题：

“在不做任何工程优化的情况下，系统性能是多少？”

五、优化系统 (核心工作)

在 不改变模型结构与训练方式 的前提下，进行推理加速。

主要优化方向

1. 推理精度与引擎优化

- FP32 → FP16
- PyTorch → 推理引擎 (优先级)：
 - ONNX Runtime (CUDA + FP16)
 - TensorRT (如时间允许)

2. 系统层优化 (低风险)

- 固定输入尺寸，避免动态 shape
- 减少不必要的数据拷贝
- `model.eval() + torch.no_grad()`
- 简化预处理流程

3. 可选增强 (不强制)

- 微批处理 (micro-batch)
 - 简单异步处理 (如读取与推理解耦)
-

六、评估指标 (必须量化)

所有测试均遵循：

- 同一台机器
- 同一模型

- 同一输入数据
- 同一测试场景

核心指标

- FPS (Frames Per Second)
- 延迟 (Latency)：
 - 平均延迟
 - P95 / P99 延迟
- GPU 显存占用

对比方式

指标	Baseline (FP32)	Optimized (FP16)	提升幅度
FPS			
P95 延迟			
显存			

七、Demo 展示形式 (路演用)

- 同屏对比：
 - 左：Baseline (PyTorch FP32)
 - 右：Optimized (FP16 + 推理引擎)
- 实时显示：
 - FPS
 - 延迟
 - GPU 显存
- 使用摄像头输入，结果直观、可感知

目标：评委一眼就能看出“右边更快”。

八、项目交付物

- 可运行代码 (Baseline + Optimized)
- 一键运行脚本
- 性能对比表
- 简单架构示意图
- Demo 视频或现场展示

九、项目成功标准 (真实版本)

本项目的成功定义不是“做得多复杂”，而是：

- 能稳定运行
- 能清楚说明优化点

- 能用数据证明“确实更快”
 - 能按时交付
-

十、后续可扩展方向（如被问到）

- 引入 INT8 量化
- 引入多模态（视觉 + 文本）
- 拓展至端到端视频分析
- 拓展至边缘设备（未来工作）

项目名称（暂定）

基于 RTX 5070 的多模态视觉模型推理加速与端到端性能优化

一、项目背景与目标

在多模态人机交互（如实时视觉理解、智能感知）中，
模型推理延迟与系统吞吐直接决定了用户体验的“流畅底线”。

本项目聚焦于 **单卡 GPU (RTX 5070)** 场景下的 AI 推理加速，
在不涉及嵌入式开发、不改动模型训练的前提下，通过推理与系统层优化：

- 降低端到端延迟
- 提升实时处理帧率（FPS）
- 在同等精度下提高系统稳定性

核心目标不是做新算法，而是把已有模型跑得更快、更稳。

二、项目约束条件（现实情况）

- 团队背景：无嵌入式开发经验，AI/工程基础偏初级
- 硬件条件：单张 NVIDIA RTX 5070 GPU
- 经费预算：约 1000 RMB
- 项目目标：按时交付一个可运行、可展示、可量化对比的技术作品

因此，本项目选择：

- 纯软件层优化
 - 基于成熟模型与成熟推理框架
 - 追求“可交付成功率”，而非极限性能
-

三、具体任务定义（做什么）

任务类型（当前选择）

实时视觉推理任务（摄像头输入）

示例：

- 实时目标检测 (YOLO 系列)
- 单张图像推理 / 连续帧推理

输入：

- 摄像头或图片流

输出：

- 检测结果可视化
 - 实时性能指标 (FPS、延迟等)
-

四、Baseline (基线系统)

作为对照组，构建一个“未优化”的标准推理流程：

- 框架：PyTorch
- 精度：FP32
- 推理方式：同步、串行
- 输入尺寸：默认 (如 640×640)
- 运行设备：RTX 5070

基线系统用于回答一个问题：

“在不做任何工程优化的情况下，系统性能是多少？”

五、优化系统 (核心工作)

在 不改变模型结构与训练方式 的前提下，进行推理加速。

主要优化方向

1. 推理精度与引擎优化

- FP32 → FP16
- PyTorch → 推理引擎 (优先级)：
 - ONNX Runtime (CUDA + FP16)
 - TensorRT (如时间允许)

2. 系统层优化 (低风险)

- 固定输入尺寸，避免动态 shape
- 减少不必要的数据拷贝
- `model.eval() + torch.no_grad()`
- 简化预处理流程

3. 可选增强 (不强制)

- 微批处理 (micro-batch)
 - 简单异步处理 (如读取与推理解耦)
-

六、评估指标 (必须量化)

所有测试均遵循：

- 同一台机器
- 同一模型
- 同一输入数据
- 同一测试场景

核心指标

- FPS (Frames Per Second)
- 延迟 (Latency)：
 - 平均延迟
 - P95 / P99 延迟
- GPU 显存占用

对比方式

指标	Baseline (FP32)	Optimized (FP16)	提升幅度
FPS			
P95 延迟			
显存			

七、Demo 展示形式 (路演用)

- 同屏对比：
 - 左 : Baseline (PyTorch FP32)
 - 右 : Optimized (FP16 + 推理引擎)
- 实时显示：
 - FPS
 - 延迟
 - GPU 显存
- 使用摄像头输入，结果直观、可感知

目标：评委一眼就能看出“右边更快”。

八、项目交付物

- 可运行代码 (Baseline + Optimized)
- 一键运行脚本
- 性能对比表

- 简单架构示意图
 - Demo 视频或现场展示
-

九、项目成功标准（真实版本）

本项目的成功定义不是“做得多复杂”，而是：

- 能稳定运行
 - 能清楚说明优化点
 - 能用数据证明“确实更快”
 - 能按时交付
-

十、后续可扩展方向（如被问到）

- 引入 INT8 量化
- 引入多模态（视觉 + 文本）
- 拓展至端到端视频分析
- 拓展至边缘设备（未来工作）