

基于异步流水线与滑动窗口的交通效率诊断系统性能优化研究

赵欣宇¹, 张烁今¹, 叶豫桐², 王睿杰²

1. 北京航空航天大学软件学院, 北京市 100191;

2. 北京航空航天大学计算机学院, 北京市 100191

摘要

城市道路交叉口交通效率诊断需从监控视频中持续完成车辆检测、多目标跟踪、车道级特征提取和拥堵传播预测。针对现有系统前端视频处理串行等待与后端连续窗口重复构造等效率瓶颈, 围绕交通效率诊断场景构建了异步流水线与滑动窗口协同优化方案。前端将视频读取、车辆检测、多目标跟踪和结果输出拆分为衔接阶段, 通过线程安全队列实现并行流动; 后端维护固定长度窗口缓冲并复用历史窗口状态, 减少重复构造与调度开销。在约 109, 300 帧丁字路口监控视频上的实验表明, 系统总处理时间由 281.90 分钟缩短至 27.60 分钟, 平均帧率 (frames per second, FPS) 由 6.46 FPS 提升至 65.94 FPS, 整体效率提升约 10 倍。一致性实验表明, 滑动窗口机制不改变 ST-GCN 输出, 异步流水线不改变检测与跟踪结果。

关键词

智能交通系统; 异步流水线; 滑动窗口推理; ST-GCN; 外部流水线基线; 实时视频分析

中图分类号: TP391

文献标志码: A

Performance optimization of traffic efficiency diagnosis system based on asynchronous pipeline and sliding window

ZHAO Xinyu¹, ZHANG Shuojin¹, YE Yutong², WANG Ruijie²

1. School of Software, Beihang University, Beijing 100191, China;

2. School of Computer Science and Engineering, Beihang University, Beijing 100191, China

Abstract

Traffic efficiency diagnosis at urban intersections requires continuous processing of surveillance video streams for vehicle detection, multi-object tracking, lane-level feature extraction, and congestion

propagation prediction, combining high-frequency video processing with continuous temporal reasoning. Existing systems can complete the full workflow, but their practical implementations often suffered from serial waiting in front-end video processing and redundant window construction in back-end temporal inference, affecting system real-time

Key words

intelligent transportation system, asynchronous pipeline, sliding window inference, ST-GCN, external pipeline baseline, real-time video analysis

performance. To improve end-to-end efficiency, a collaborative optimization scheme based on an asynchronous pipeline and sliding-window inference was developed. The front-end stages were decomposed into connected stages for video reading, object detection, multi-object tracking, and result output, linked by thread-safe queues, so that different batches could flow through different stages in parallel. A fixed-length temporal buffer was maintained at the back-end, and the historical window buffer, resident adjacency tensor, and inference scheduling state were reused (while keeping the ST-GCN forward computation graph unchanged) to reduce redundant window construction, data copying, and scheduling overhead. It was shown by experimental results that, on the same T-intersection surveillance video containing approximately 109,300 frames, with a resolution of 1920x1080, a frame rate of 30 FPS, and a duration of 60 minutes and 43 seconds, the optimized system reduced the total processing time from 281.90 minutes to 27.60 minutes and improved the average processing frame rate from 6.46 frames

per second (FPS) to 65.94 FPS, achieving approximately 10x end-to-end efficiency improvement. Ultralytics StreamLoader was further evaluated as an open-source streaming inference pipeline baseline and ONNX Runtime as an inference backend baseline. Ultralytics StreamLoader achieved 51.54 FPS on the same video, while the frame rate was further improved by the proposed fully optimized method by about 1.28 times. In the single-sample ST-GCN inference backend baseline, the average inference latency was reduced by ONNX Runtime from 4.5584 ms to 2.1901 ms. It was further shown by consistency experiments that the sliding-window mechanism did not change ST-GCN outputs, and front-end detection and tracking results were not changed by the asynchronous pipeline.

1 引言

城市道路交叉口是路网运行中最容易出现瓶颈的位置。车辆在这里汇入、分流、排队、转向,运行状态变化快,局部拥堵还可能沿着相邻路段继续扩散。对交叉口通行效率进行持续监测,并进一步判断拥

堵是否正在形成、会向哪里扩散，对交通调度、拥堵治理以及智能交通系统建设都有现实价值。传统做法多依赖固定检测器和人工调查，这类方式能够提供一定交通信息，但也存在部署成本高、覆盖范围有限、数据更新速度慢等问题。近些年，随着计算机视觉和深度学习持续发展^[1-3]，基于监控视频的交通感知与运行分析逐渐成为重要研究方向。

从现有技术路线看，交通效率诊断已经形成了比较完整的处理链条。通常先借助标注工具定义路口车道结构，得到几何关系和拓扑连接；再使用YOLOv8（You Only Look Once v8）^[3-5]与DeepSORT（deep simple online and realtime tracking）^[6-8]完成车辆检测和多目标关联，进而得到轨迹、速度、位置及车道归属等信息；接着把车道抽象成图结构中的节点，交给时空图卷积网络（spatial-temporal graph convolutional network，ST-GCN）预测拥堵传播趋势^[9-11]。从功能角度看，这条链路已经较为完整，问题主要出在运行效率上。前端视频处理常按“读取—检测—跟踪—输出”的顺序逐帧推进，各模块之间会互相等待；后端预测面对连续输入时，又会反复处理大量重叠的时间窗口。结果是系统虽然能给出诊断结果，但运行速度不够快，离真实应用场景所要求的实时性仍有距离。

基于这一判断，提升交通效率诊断系统实时性的关键并不只是更换某一个模型，而是把整条处理链路重新梳理一遍，找出哪些环节适合并行、哪些数据组织过程适合复用，再有针对性地做重构。围绕这个思路，设计了两项系统级优化策略。第一项是异步流水线检测架构，把视频读取、车辆检测、多目标跟踪和结果输出拆成可衔接的阶段，让它们不再一帧一帧地排队

执行，而是尽量重叠工作。第二项是滑动窗口推理机制，针对连续时间序列中相邻窗口高度重合这一特点，维护固定长度输入缓冲区，并复用历史窗口缓冲、邻接矩阵驻留状态和推理调度状态，避免每次都从完整历史序列中重新切片、复制和调度。需要说明的是，该机制不改变ST-GCN模型结构，也不复用模型内部图卷积层或时间卷积层的中间激活。

为进一步引入公开流式推理管线和推理后端优化框架进行比较，补充了两类外部基线实验。第一类是Ultralytics StreamLoader外部开源流式推理管线基线，用于比较通用开源视频流式推理管线与该异步流水线设计之间的性能差异；第二类是ONNX Runtime（Open Neural Network Exchange Runtime）推理后端基线，用于分析仅替换ST-GCN推理后端时的单模型推理加速效果。通过这两类外部基线，进一步说明所提出方法的收益主要来自系统级执行组织优化，而不仅仅是单一推理引擎替换。

- 主要工作如下：
- 第一，面向视频处理链路中逐帧串行执行导致的吞吐不足问题，设计并实现了异步流水线检测架构。该架构将视频读取、车辆检测、多目标关联和结果输出划分为若干相互衔接的阶段，并通过线程安全队列实现阶段解耦，使不同批次数据能够在不同处理阶段上并行流动。
 - 第二，面向ST-GCN连续预测中相邻时间窗口高度重叠的问题，设计了滑动窗口推理机制。该机制维护固定长度的历史特征缓冲区，在新时间步到来时更新窗口内容，并复用已有的数据组织和状态管理结果，从而减少连续推理中的重复窗口构造与调度开销。
 - 第三，通过完整视频实验、消融实验、

预热行为分析、稳定性分析、外部开源流式推理管线基线实验、ONNX Runtime 推理后端基线实验以及结果一致性验证实验，系统评估了所提出方法的运行效果。实验表明，全优化方案能够显著提升系统平均 FPS；同时，外部基线实验表明，该方法相对于通用开源流式推理管线和单模型推理后端优化仍具有进一步性能优势。

2 国内外研究现状

2.1 车辆检测与多目标关联研究

在交通视频分析领域，基于车辆检测和多目标关联的状态感知方法近年发展很快。深度学习的发展为这一方向打下了重要基础。Krizhevsky 等提出 AlexNet，较早证明了深度卷积神经网络在图像分类任务上的显著优势^[1]；Simonyan 等提出 VGG，进一步讨论了网络深度与性能之间的关系^[2]；Szegedy 等提出 GoogLeNet，通过 Inception 模块强化多尺度特征提取^[12]；He 等提出 ResNet，利用残差连接缓解深层网络训练困难^[13]；Huang 等提出 DenseNet，通过密集连接提升特征复用效率^[14]。

在车辆检测任务中，YOLO（You Only Look Once）系列模型因速度快、部署方便而被广泛使用。Redmon 等提出 YOLOv3，实现了较好的实时检测效果^[15]；之后 YOLOv5^[16]、YOLOv7 以及 YOLOv8^[3-5,37-38]在结构设计、推理效率和检测效果上不断改进。针对复杂交通场景，不少研究又在 YOLOv8 基础上做了专门优化^[5,18-20]。这些工作说明，YOLO 系列已经成为交通目标检测研究中的主流基础框架。

在多目标关联方面，DeepSORT（deep simple online and realtime

tracking）因同时利用运动信息与外观特征，在复杂场景下具有较稳定的关联效果，已经成为这一方向的代表性方法之一^[6-8,21-22]。因此，“检测+跟踪”的基本框架已经较为成熟，也具备较强工程可用性。

2.2 交通预测与时空图神经网络研究

交通预测研究中，时空图卷积网络（spatial-temporal graph convolutional network，ST-GCN）因能够同时建模空间拓扑关系和时间演化规律，近年受到广泛关注。袁凯等对图神经网络在交通预测中的应用进行了全面综述，系统梳理了卷积图神经网络、图注意力网络、图自编码器和图时空网络等主流方法^[23]。Yu 等提出 ST-GCN 模型，较早把图卷积系统性引入交通预测任务^[9]，为后续研究奠定了基础。之后，不少工作通过加入注意力机制、门控结构或动态图建模方法，提升模型对复杂时空依赖的表达能力^[10-11,24-26]。王建龙进一步将图过程神经网络引入城市道路拥堵时空预测，在精细化交叉口级和路网级拥堵传播建模方面取得了进展^[27]。

现有工作在预测效果上已经做得较充分，但对连续输入条件下的推理效率关注还不够多。对需要长时间连续运行的实际系统来说，预测准确固然重要，推理组织效率同样重要；如果相邻窗口高度重叠，却依旧在每个时间步重复完成窗口构造、数据拷贝和调度，整体效率就会被明显拖住。

2.3 系统加速部署与外部优化框架

围绕系统加速与部署优化，异步流水线、批处理以及推理引擎优化都已被证明能够显著提高深度学习系统的吞吐能力。已有研究指出，异步处理和流水线结构可

以减少模块等待时间，并提升异构计算资源利用率^[28]。朱昌宏等进一步设计了面向多路视频流的边缘端处理框架，通过多智能体强化学习方法对批大小、帧率和并发度进行联合调控，提升了资源受限条件下的视频分析吞吐能力^[29]，基于边缘计算的视频分析系统也在类似场景中取得了工程实效^[39]。同时，TensorRT、ONNX Runtime（Open Neural Network Exchange Runtime）、DeepStream、OpenVINO（Open Visual Inference and Neural Network Optimization）、GStreamer 以及 Ultralytics StreamLoader 等推理或视频流处理工具，也常用于实际系统部署和性能优化^[30-35]。

其中，ONNX Runtime 是一类较通用的推理优化框架，能够通过图优化、算子融合和后端执行器优化降低模型推理开销；TensorRT（NVIDIA TensorRT）更偏向于通过层融合、精度优化和内核选择提升 NVIDIA GPU 上的推理效率；DeepStream（NVIDIA DeepStream SDK）更适合多路视频流的工业级部署；OpenVINO 主要面向 Intel CPU、iGPU 和 NPU 等平台优化；Ultralytics StreamLoader 则提供与 YOLO 生态兼容的开源流式推理接口。与这些框架相比，该方法关注的是不依赖特定推理引擎的系统级执行组织优化，包括前端阶段解耦、异步流水线并行以及后端连续窗口状态复用。因此，选择 Ultralytics StreamLoader 作为外部开源流式推理管线基线，并选择 ONNX Runtime 作为推理后端优化基线，用于验证该方法相对于通用流式管线和单模型推理后端替换的额外收益。

3 算法总体设计

3.1 现有功能流程与效率瓶颈

交通效率诊断系统的现有功能流程可以分为两个部分：前端负责从监控视频中提取车辆级和车道级信息，后端负责根据这些信息完成拥堵传播预测。系统先读取道路监控视频帧，再利用目标检测模型识别车辆目标，接着通过多目标跟踪算法为同一车辆建立跨帧关联，随后结合车道区域信息提取速度、位置、排队长度等特征，最后将这些特征按时间顺序组织起来，送入 ST-GCN 进行时空建模与预测。

前端的主要瓶颈来自串行等待。按照传统写法，视频读取、车辆检测、多目标跟踪和结果输出往往一帧一帧地依次执行。上一阶段没结束，下一阶段就只能等待。后端的主要瓶颈则来自重复窗口组织与调度。ST-GCN 需要在固定时间窗口上做预测，而在连续输入场景中，相邻两个时间窗口往往只相差一个时间步，绝大部分历史数据其实相同。前后两个瓶颈叠加起来，造成系统整体实时性不足。

3.2 协同优化思路

前端瓶颈和后端瓶颈并不是同一类问题。前端更像"流程组织不合理"，关键在于各阶段如何配合；后端更像"连续输入下窗口组织重复太多"，关键在于已有窗口状态能否继续利用。基于这样的判断，没有简单追求某一个模块的单点加速，而是把系统拆成前后两段分别优化。

前端部分采用异步流水线检测架构。它的目标不是改变检测和跟踪任务本身，而是改变任务被组织和调度的方式。通过阶段解耦、线程并发和批处理，系统可以让视频读取、检测、跟踪和输出彼此错开并重叠执行，减少无意义等待。

后端部分采用滑动窗口推理机制。它

关注的是连续推理时能否减少重复窗口构造、数据拷贝和推理调度。由于相邻窗口高度重合，系统可以维护固定长度输入缓冲区，并在新时间步到达时通过滑动更新窗口内容。

3.3 系统整体结构

基于上述思路，系统被重构为"异步视频分析前端+滑动窗口时序预测后端"的整体框架，整体结构如图1、图2和图3所示。

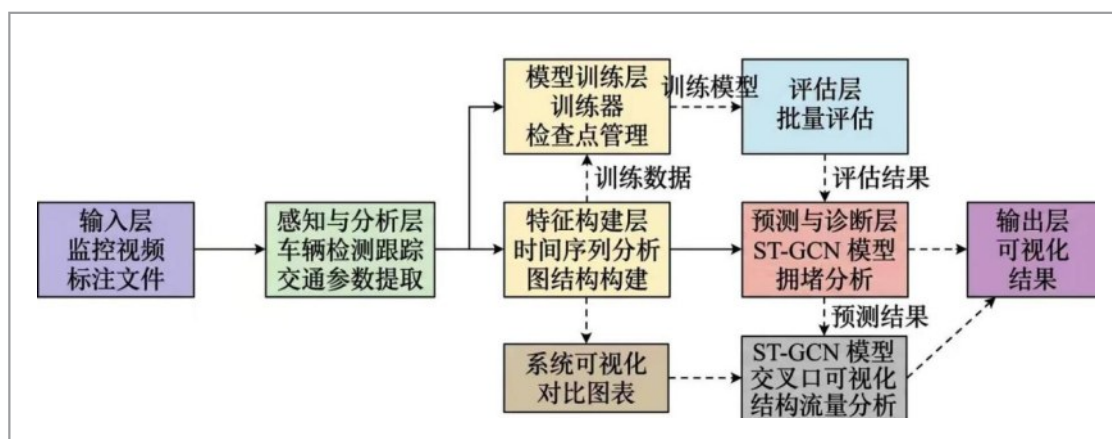


图1 现有功能流程结构图

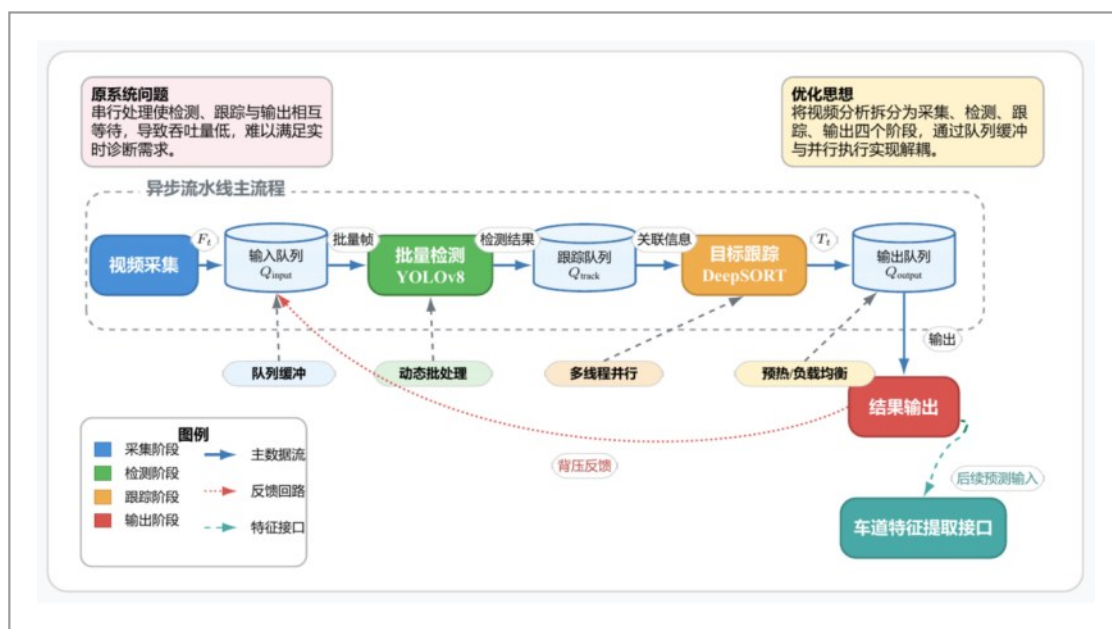


图2 异步流水线检测架构图

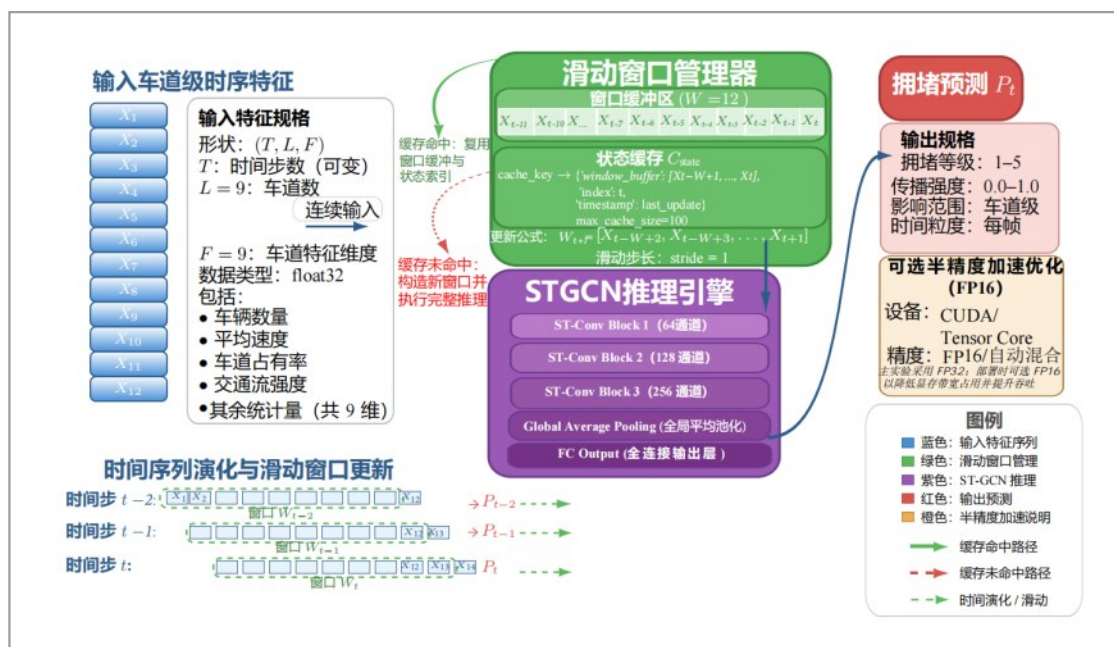


图3 滑动窗口推理架构图

4 异步流水线检测架构

4.1 设计动机

传统串行流程里, 很多时间不是花在计算本身, 而是花在等待上一环节结束。例如, 视频读取通常由 CPU 完成, 车辆检测主要依赖 GPU, 多目标关联又回到 CPU 侧。如果这些阶段被严格串成一条线, 那么某一时刻真正工作的往往只有一个模块, 其他模块处于等待状态。异步流水线的本质是重组执行顺序, 让不同类型的工作更紧凑地拼接起来。已有研究将流水线技术引入 CNN 并行计算, 在视频分析场景中显著提升了训练与推理效率^[36]。

4.2 基本结构与运行过程

前端视频分析链路被划分为四个阶段:

视频读取阶段、车辆检测阶段、多目标关联阶段以及结果输出阶段。各阶段之间通过线程安全队列传递数据。假设输入视频被划分为若干批次, 记第 i 个输入批次为 B_i , 其流水线执行过程可表示为:

$$B_i \xrightarrow{\text{Read}} R_i \xrightarrow{\text{Detect}} D_i \xrightarrow{\text{Track}} T_i \xrightarrow{\text{Output}} O_i (1)$$

其中, B_i 表示第 i 个输入批次, R_i 表示读取后的原始帧批次, D_i 表示检测结果, T_i 表示多目标关联结果, O_i 表示最终输出结果。虽然公式按顺序写出各阶段, 但在真实运行时, 多个批次会在不同阶段同时流动。

若记四个阶段单批次处理时间分别为 t_r 、 t_d 、 t_t 和 t_o , 总批次数为 N , 则传统串行总时间近似为:

$$T_{\text{serial}} = N \cdot (t_r + t_d + t_t + t_o). \quad (2)$$

采用异步流水线后, 在忽略启动和收尾阶段额外开销的理想情况下, 总时间更接近于:

$$T_{\text{pipe}} \approx t_r + t_d + t_l + t_o + (N - 1) \cdot \max(t_r, t_d, t_l, t_o).$$

(3)

流水线加速比为：

$$S_{\text{pipe}} = \frac{T_{\text{serial}}}{T_{\text{pipe}}}.$$

(4)

4.3 关键实现策略

要让流水线稳定运行，需要配套阶段解耦、批处理和负载均衡策略。各阶段通过独立队列交换数据，而不是直接相互调用，以减少线程之间的硬耦合。检测阶段采用批处理方式，将若干帧打包后统一送入YOLO模型，提高GPU利用率。与此同时，通过调整批大小、线程数量以及队列容量，使各阶段在长期运行中保持较为接近的节奏，避免某一队列持续堆积。

需要说明的是，异步流水线优化不改变车辆检测模型和多目标跟踪算法本身。对于同一输入帧，在检测阈值、输入分辨率和跟踪器参数保持一致的条件下，流水线版本与串行版本执行相同的检测和跟踪逻辑。其性能提升主要来自阶段解耦、批处理调度以及CPU与GPU工作重叠，而不是来自检测精度或跟踪策略的改变。

4.4 异步流水线处理算法

5 滑动窗口推理机制

5.1 设计动机

滑动窗口推理机制的核心思想，是利用连续时序输入之间天然存在的窗口重叠关系，减少重复窗口构造、数据拷贝和推理调度开销。当前实现并不复用ST-GCN内部图卷积层或时间卷积层的中间激活，

Algorithm 1:异步流水线视频处理算法	
输入:输入视频流V,批处理大小B,阶段队列集合{Qr, Qd, Qt}	
输出:处理结果序列O	
1	初始化读取线程、检测线程、跟踪线程和输出线程
2	初始化线程安全队列Qr, Qd, Qt
3	while 视频流V未结束 do
4	读取线程从V中取出B帧,组成批次Ri,写入Qr
5	检测线程从Qr中取出Ri,执行YOLO检测,得到Di,写入Qd
6	跟踪线程从Qd中取出Di,执行多目标关联,得到Ti,写入Qt
7	输出线程从Qt中取出Ti,完成可视化和结果写出,得到Oi
8	等待所有线程完成剩余任务
9	返回处理结果序列O

而是在保持模型前向计算图不变的条件下优化连续输入组织方式。传统实现里，每来一个新的时间步，系统通常会从完整历史序列中重新切片，构造长度固定的输入窗口，再送入模型。由于连续交通预测任务中相邻两个窗口往往只差最后一个时间步，前面大部分历史内容相同，因此重复切片、复制和调度会带来额外开销。滑动窗口机制通过维护固定长度窗口缓冲区，使新时间步到达时只更新窗口内容和相关状态管理信息，从而降低连续推理过程中的输入组织成本。

5.2 窗口更新与缓冲区管理

设时间窗口大小为W=12，输入特征序列为{X1, X2,..., XT}，其中Xt∈R L×F表示时刻t的车道级特征矩阵，L=9表示车道数，F=9表示每条车道的特征维度。系统在时刻t维护窗口缓冲区Bt；当新特征Xt+1到达时，窗口按如下方式更新：

$$\mathcal{B}_{t+1} = [X_{t-W+2}, X_{t-W+3}, \dots, X_t, X_{t+1}]. \quad (5)$$

在窗口长度达到 W 后，ST-GCN 预测过程可表示为：

$$\hat{Y}_{t+1} = f_{\theta}(\mathcal{B}_{t+1}, \mathbf{A}), \quad (6)$$

其中， $\mathcal{B}_{t+1}$ 表示更新后的时间窗口缓冲区， $\mathbf{A}$ 表示车道拓扑对应的邻接矩阵， $f_{\theta}(\cdot)$ 表示 ST-GCN 预测模型， $t+1$ 表示 $t+1$ 时刻预测结果。

对于任意时刻 t ，传统整窗推理和滑动窗口推理输入到模型中的窗口张量均为：

$$\mathbf{X}_t^W = [\mathbf{X}_{t-W+1}, \mathbf{X}_{t-W+2}, \dots, \mathbf{X}_t]. \quad (7)$$

传统方法每次从完整历史序列中重新切片得到窗口张量 $\mathbf{X}$ ，而滑动窗口方法维护同一长度为 W 的窗口内容。当滑动窗口缓冲区 $\mathcal{B}_t$ 与传统切片得到的窗口张量 $\mathbf{X}$ 在时间步顺序、特征维度和邻接矩阵输入上完全一致时，有：

$$\hat{\mathbf{Y}}_t^{\text{serial}} = f_{\theta}(\mathbf{X}_t^W, \mathbf{A}), \hat{\mathbf{Y}}_t^{\text{slide}} = f_{\theta}(\mathcal{B}_t, \mathbf{A}). \quad (8)$$

$$\hat{\mathbf{Y}}_t^{\text{serial}} = \hat{\mathbf{Y}}_t^{\text{slide}}. \quad (9)$$

这也是结果一致性验证实验的理论依据。需要强调的是，当前实现并不复用 ST-GCN 内部图卷积层或时间卷积层的中间激活，而是在保持 ST-GCN 前向计算图不变的条件下，复用连续输入中的窗口缓冲、邻接矩阵驻留和推理状态管理结果。因此，只要输入窗口、模型权重和邻接矩阵一致，传统整窗推理与滑动窗口推理的输出应保持一致。

5.3 状态缓存更新算法

5.4 内存优化与可选半精度推理

Algorithm 2: 滑动窗口输入缓冲更新算法

输入: 新特征 X_{new} , 历史窗口缓冲区 B , 窗口大小 W , 邻接矩阵 A

输出: 更新后的窗口缓冲区 B 与预测结果

- 1 将 X_{new} 追加到窗口缓冲区 B 尾部
- 2 if $|B| > W$ then
- 3 删除 B 中最早的时间步特征
- 4 if $|B| < W$ then
- 5 返回当前缓冲区, 不执行 ST-GCN 预测
- 6 由 B 构造固定长度窗口张量 XW
- 7 if 邻接矩阵 A 尚未驻留在 GPU 显存中 then
- 8 将 A 传输至 GPU 并缓存
- 9 在保持 ST-GCN 前向计算图不变的条件下执行预测: $= f_{\theta}(XW, A)$
- 10 返回 B 与 $\hat{\mathbf{Y}}_t$

除窗口缓冲复用外，半精度推理也是实际部署中可以考虑的一项优化方向。在 GPU 上，FP16 的计算吞吐通常高于 FP32，同时显存带宽占用更低，比较适合实时推理场景。主实验和一致性验证均采用全精度推理，以保证传统整窗推理与滑动窗口推理的数值可比性。在实际部署中，若硬件支持 Tensor Core，也可以进一步引入 FP16 或自动混合精度推理，以降低显存带宽占用并提升吞吐能力。

设原始 FP32 格式的模型权重和激活值分别为 P_{fp32} 与 H_{fp32} ，转换后的 FP16 表示分别为 P_{fp16} 与 H_{fp16} ，则有：

$$\begin{aligned} W_{fp16} &= \text{cast}_{fp32 \rightarrow fp16}(W_{fp32}), \\ A_{fp16} &= \text{cast}_{fp32 \rightarrow fp16}(A_{fp32}). \end{aligned} \quad (10)$$

使用 PyTorch 自动混合精度机制实现低精度推理的示例如下：

6 预热行为与缓存策略

```
with torch.cuda.amp.autocast(enabled=True):  
    with torch.no_grad():  
        prediction=model(window_tensor, adj_tensor)
```

6.1 预热现象描述

实验中可以观察到，异步流水线优化方法与全优化方法在运行初期都存在较明显的预热行为：前 100 帧附近的 FPS 偏低，处理数百帧后，性能才迅速上升并逐渐稳定。这说明优化带来的性能优势不是在程序启动那一刻就完全释放出来的，而是需要经历模型加载、队列填充、缓存建立以及硬件状态调整等过程之后，系统才会进入真正的高效工作状态。

6.2 预热产生的原因

预热的根源在于系统中的多层缓存尚未建立，计算图和硬件执行环境也尚未完成自适应调整。启动时，YOLO 和 ST-GCN 的模型权重需要从磁盘加载到 GPU 显存。模型第一次前向传播时，PyTorch 还需要完成计算图构建、中间缓冲区分配以及 CUDA 内核初始化。除此之外，GPU 动态频率调节、滑动窗口缓冲区初始化、轨迹缓存初始化和队列填充过程，也都会影响启动阶段的 FPS 表现。

6.3 主动预热策略

为缩短过渡过程，设计了主动预热策略。具体做法包括：使用与真实输入尺寸一致的样本数据执行若干次前向推理，以提前完成 CUDA 内核初始化；预先申请显存缓冲区，减少运行中频繁动态分配内存带来的额外开销；利用视频开头的一小段数据预填充滑动窗口缓冲区与轨迹缓存，

使系统在进入连续处理时具备一定缓存基础。

6.4 性能监控机制

为观察系统性能变化，设计了监控模块，每处理 100 帧记录一次关键指标，包括当前 FPS、平均 FPS、累计 FPS 和处理延迟等。监控器采用环形缓冲区保存最近一段时间内的 FPS 历史，并通过移动平均方式平滑瞬时波动，为后续性能对比、稳定性分析和预热行为研究提供数据支撑。

7 实验结果与分析

7.1 实验设置

为验证协同优化方案是否有效，实验选取真实道路监控视频作为测试数据。所用视频来自"复杂场景下河谷型城市兰州市交通路口视频流数据集"。实验从该数据集中选取一段丁字路口监控视频作为统一测试样本。该视频时长为 60 分钟 43 秒，分辨率为 1920×1080，帧率为 30.00 帧/秒，总帧数为 109,300 帧，编码格式为 H.264，总比特率为 1746 kbps。需要说明的是，后续表格中的"日志统计帧数"并非重新选取不同视频，而是同一原始视频在不同实现下的日志计数结果。由于批处理取整、尾帧处理方式以及日志记录粒度不同，个别方法的统计帧数存在百帧量级差异；该差异占总帧数比例约为 0.1%，不影响同一视频条件下的整体性能比较。

为分析不同优化策略对系统性能的影响，设置了传统方法、异步流水线优化、滑动窗口优化和全优化四种内部对比方法。其中，传统方法采用串行视频处理流程和传统整窗 ST-GCN 推理；异步流水线优化

仅重构前端视频处理链路；滑动窗口优化仅重构后端连续时序推理组织方式；全优化方法同时启用异步流水线检测架构和滑动窗口推理机制。

补充了 Ultralytics StreamLoader 外部开源流式推理管线基线^[31]。该基线使用 Ultralytics 提供的 stream=True 接口完成视频读取、预处理和 YOLO 推理调度，并在其输出检测框基础上使用相同 DeepSORT 跟踪器。实验中，视频输入、YOLO 模型权重、输入分辨率、置信度阈值、IoU 阈值、最大检测数和跟踪器主要参数均与该方法尽量保持一致。同时尝试构建 GStreamer-Queue 基线，但当前 Windows 实验环境中的 OpenCV 未启用 GStreamer 支持，因此未将其纳入正式对比。

进一步设置了 ONNX Runtime 推理基线实验。该实验以 ST-GCN 为对象，在相同模型权重、相同输入规模和相同邻接矩阵条件下，对比 PyTorch 后端与 ONNX Runtime 后端的推理延迟、FPS 和输出数值差异。该实验用于分析仅替换推理后端时的单模型推理加速效果。

此外，为验证优化机制是否改变系统输出，补充了两类一致性实验。第一类是后端 ST-GCN 滑动窗口一致性与预测精度保持实验，用于验证传统整窗推理和滑动窗口推理是否产生相同输入、相同输出和相同预测指标。第二类是前端检测与跟踪

输出稳定性实验，用于验证优化前后车辆检测数量、边界框匹配结果、置信度和轨迹 ID 是否保持一致。

实验运行平台为一台笔记本电脑，硬件环境如下：处理器为 Intel(R)Core (TM)Ultra 9 275HX，独立显卡为 NVIDIA GeForce RTX 5060 Laptop GPU，显存容量为 8GB，系统内存为 32GB DDR5，操作系统为 Windows 11 64 位。软件环境如下：Python 3.10, PyTorch 2.6, CUDA 12.8, ultralytics、OpenCV、ONNX Runtime 等依赖均配置在同一 conda 虚拟环境中，以保证各方法在相同条件下运行。

7.2 端到端性能对比结果

表 1 给出了四种内部方法在同一段约 109,300 帧视频上的总体性能对比结果。表中帧数为各实现对应的日志统计帧数，受批处理取整、尾帧处理和日志记录粒度影响，不同方法之间存在百帧量级差异，但所有实验均基于同一原始视频完成。从结果可以看出，传统方法共耗时 281.90 分钟，日志平均 FPS 为 6.46；异步流水线优化耗时 28.10 分钟，日志平均 FPS 达到 64.76；滑动窗口优化耗时 294.02 分钟，日志平均 FPS 为 6.20；全优化方法耗时 27.60 分钟，日志平均 FPS 达到 65.94。

表1 四种方法总体性能对比

方法	日志统计帧数	总时间/min	日志平均 FPS	性能层级
传统方法	109,300	281.90	6.46	低性能组
异步流水线优化	109,200	28.10	64.76	高性能组
滑动窗口优化	109,314	294.02	6.20	低性能组
全优化	109,200	27.60	65.94	高性能组

可以看到，单独启用滑动窗口优化时，端到端平均FPS并未明显提升，甚至略低于传统方法。这主要是因为该实验的主要耗时瓶颈集中在前端视频读取、检测和跟踪阶段，而滑动窗口机制主要优化后端ST-GCN连续推理中的窗口组织与调度开销。当后端推理耗时在全链路中占比较低时，单独优化后端难以显著改变端到端吞吐；同时，滑动窗口维护、状态检查和日志统计也会引入少量额外开销。因此，滑动窗口机制的主要作用体现在与前端异步流水线结合后的全优化方案中，而不是单独决定系统端到端性能。

图4展示了全量数据累计处理时间对比结果。传统方法和滑动窗口优化的累计时间曲线斜率接近，说明只在后端做滑动窗口优化时，对整条链路累计耗时的影响有限；异步流水线优化和全优化的累计时间曲线斜率明显更小，表示在前端完成流水线改造后，系统累计处理时间被大幅压缩。

图5展示了全量数据的FPS变化对比结果。传统方法与滑动窗口优化的FPS曲线基本维持在6 FPS左右；异步流水线优化和全优化的FPS曲线在开始阶段快速上升，之后稳定在较高区间，处理能力明显更高。

7.3 预热行为与综合性能分析

为更直观地展示四种方法在“总处理时间”和“平均FPS”两个单维度上的差异，图6以柱状图形式给出了四种方法的总处理时间对比，图7给出了对应的平均FPS对比结果。

图8进一步在双轴上同时呈现总处理时间与平均FPS，以综合反映四种方法的性能差异。

7.4 稳定性分析

为分析不同方法在稳定运行阶段的波动情况，统计了处理帧数超过5000帧后的性能数据，并计算相应FPS的标准差、变异系数和稳定性评级，结果如表2所示。

高性能组出现波动，主要与视频内容复杂度、GPU功耗和温度控制、操作系统线程调度等因素有关。需要强调的是，这类波动是在高吞吐前提下出现的，属于可以接受的工程现象。

7.5 外部开源流式推理管线基线实验

为进一步比较公开流式推理管线在相同任务条件下的性能水平，增加了Ultralytics StreamLoader外部开源流式推理管线基线实验^[31]。该基线使用Ultralytics提供的stream=True接口完成视频流读取、预处理和YOLO推理调度，并在其输出检测框基础上继续使用DeepSORT进行多目标跟踪。该设置用于分析公开开源流式推理管线在相同任务条件下能够达到的性能水平。

实验结果如表3所示。Ultralytics StreamLoader基线在处理109,300帧视频时，总耗时为35.34分钟，平均FPS为51.54，相比传统串行方法的6.46 FPS获得了显著提升，说明公开流式推理管线能够有效减少视频处理流程中的部分串行开销。然而，与全优化方法相比，Ultralytics StreamLoader基线仍存在一定差距。全优化方法平均FPS为65.94，相比Ultralytics StreamLoader基线提升约1.28倍；总处理时间由35.34分钟进一步缩短到27.60分钟，减少约21.9%。这表明，通用开源流式推理管线虽然能够提供较强基线性能，但针对交通效率诊断链路设计的阶段解耦、队列并行和前后端协同

优化仍能够带来额外收益。

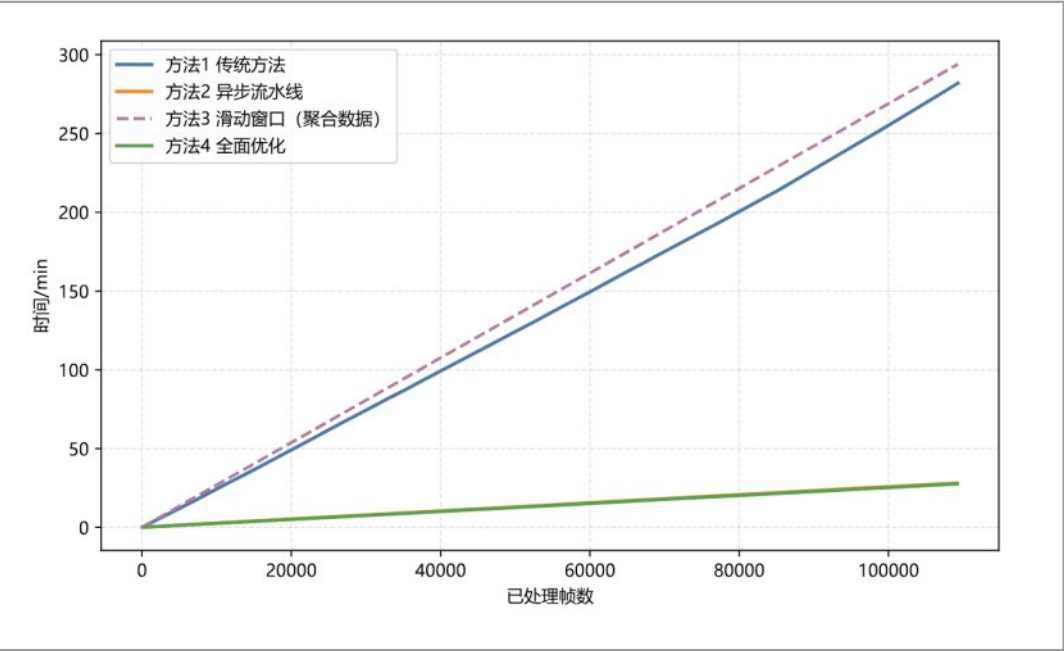


图4 全量数据累计处理时间对比

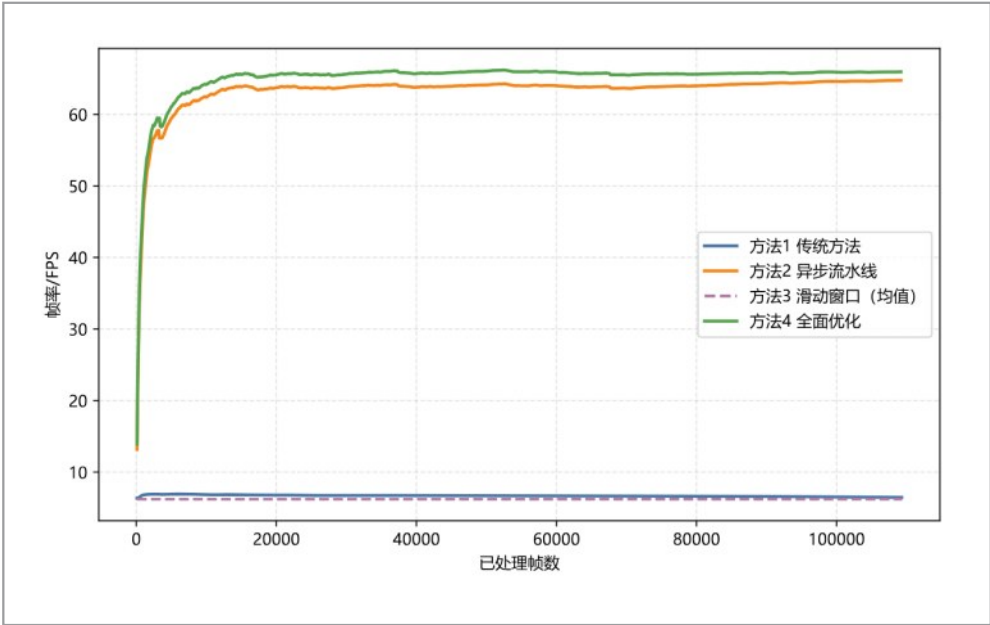


图5 全量数据帧率变化对比

表2 稳定阶段性能稳定性对比

方法	稳定FPS	标准差	变异系数	稳定性评级
传统方法	6.55	0.17	2.6%	非常稳定
异步流水线优化	65.71	6.21	9.5%	中等波动
滑动窗口优化	6.68	0.10	1.5%	极其稳定
全优化	66.86	6.13	9.2%	中等波动

需要说明的是，也尝试构建GStreamer-Queue外部视频流水线基线，但当前Windows实验环境中的OpenCV未启用GStreamer支持，因此未将其纳入正式对比。为保证实验可复现性，选择在当前环境下稳定可运行、且与YOLOv8生态兼容的Ultralytics StreamLoader作为外部开源流式推理管线基线^[31,35]。

7.6 ONNX Runtime 外部推理后端基线实验

为进一步分析外部推理优化框架对后端推理性能的影响，补充了ONNX Runtime基线实验^[32]。实验采用已训练的ST-GCN模型权重，模型配置为：车道节点数为9，输入特征维度为9，预测输出维度为3，时间窗口长度为12，ST-GCN模块数为2，隐藏通道数为32，dropout为0.3，并启用图注意力结构。实验输入张量形状为 $1 \times 12 \times 9 \times 9$ ，邻接矩阵形状为 9×9 ，邻接矩阵采用单位矩阵。模型权重与网络结构完全匹配。

需要说明的是，本小节中的推理延迟与表6和表10中的ST-GCN单窗耗时统计口径不同。表6和表10中的0.089944 ms/window与0.079667 ms/window来自2,000个连续窗口批量一致性验证，是按窗口数摊销后的平均耗时，主要用于比较传统整窗组织方式和滑动窗口组织方式的相对差异；而本小节中的4.5584 ms和2.1901 ms是在batch size为1的条件下，

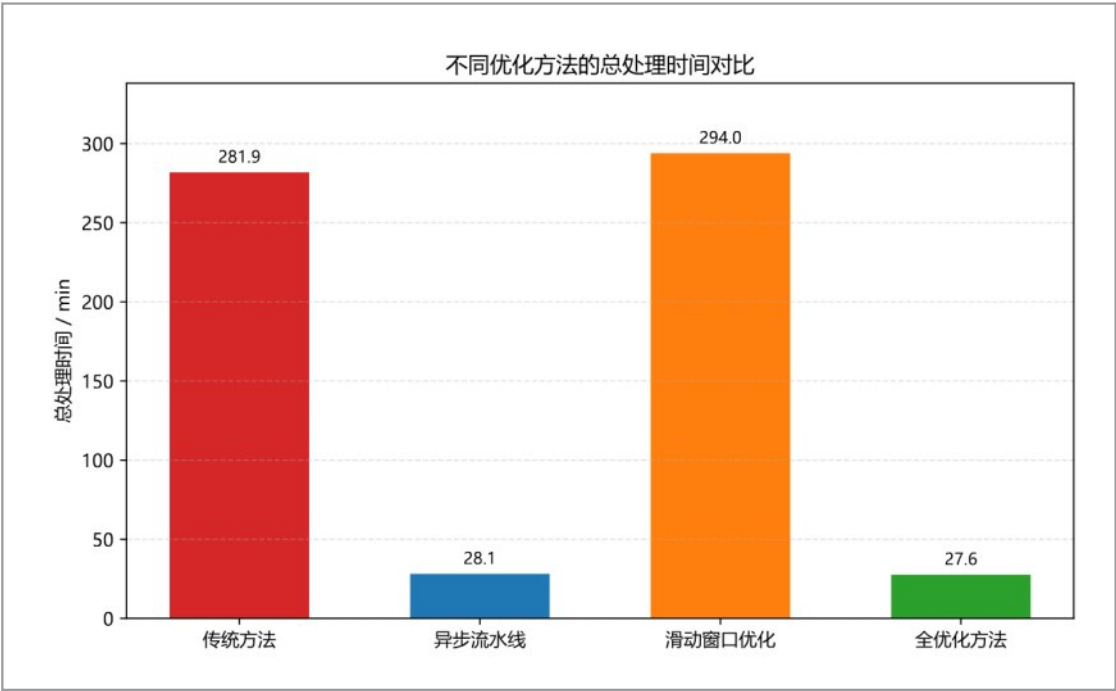


图6 不同优化方法的总处理时间对比

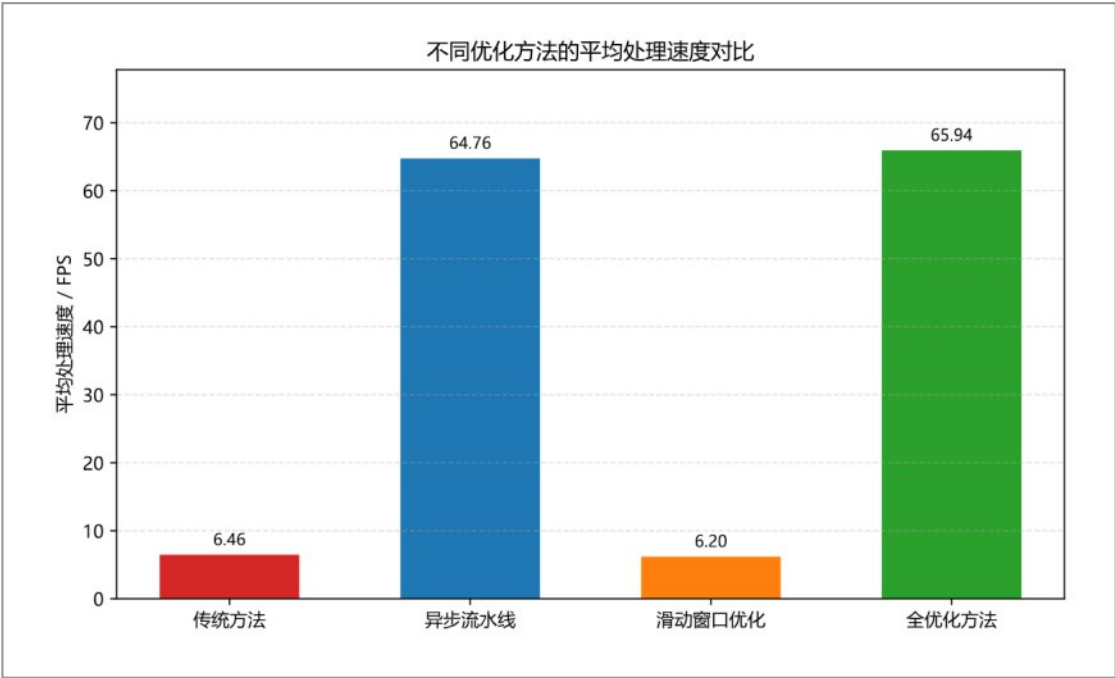


图7 不同优化方法的平均处理速度对比

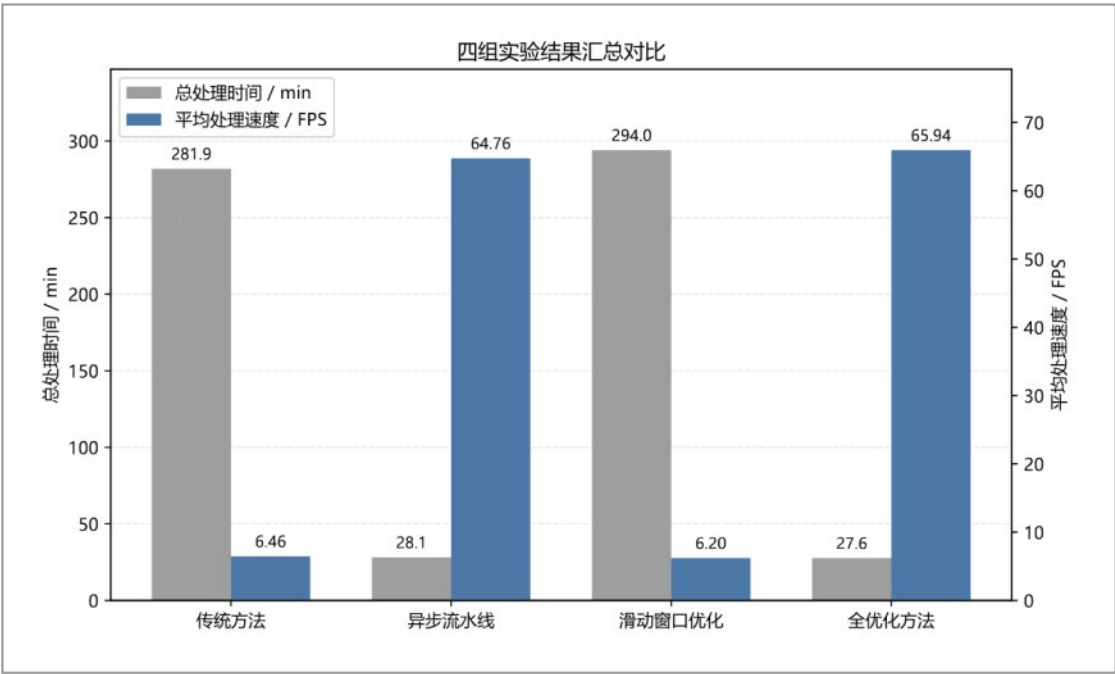


图8 总处理时间与平均FPS综合性能对比

对单个输入窗口进行多次独立调用得到的单样本推理后端延迟，包含单次调用调度、执行和同步等开销。因此，两组结果反映的是不同实验目的和统计口径，不应直接

表3 外部开源流式推理基线与该方法对比

方法	流水线来源	日志帧数	总时间/ min	平均 FPS	备注
传统方法	自实现串行流程	109,300	281.90	6.46	逐帧串行处理
Ultralytics StreamLoader	开源流式管线	109,300	35.34	51.54	stream=True
异步流水线优化	前端流水线	109,200	28.10	64.76	阶段解耦+队列并行
全优化	协同优化	109,200	27.60	65.94	异步流水线+滑动窗口

横向比较。

表 4 给出了 PyTorch 后端和 ONNX Runtime 后端的推理延迟与吞吐性能对比结果。可以看到，PyTorch 后端平均推理延迟为 4.5584 ms，对应 FPS 为 219.37；

ONNX Runtime 后端平均推理延迟为 2.1901 ms，对应 FPS 为 456.59。ONNX Runtime 相对 PyTorch 达到 2.0813 倍加速。

表4 ST-GCN PyTorch 与 ONNX Runtime 单样本推理性能对比(batch size=1)

推理后端	设备	平均延迟/ms	标准差/ms	中位数/ms	P95/ms	FPS	加速比
PyTorch	CUDA	4.5584	0.1595	4.5273	4.8995	219.37	1.0000
ONNX Runtime	CUDA	2.1901	0.0501	2.1864	2.2636	456.59	2.0813

表 5 进一步给出了 PyTorch 输出与 ONNX Runtime 输出之间的数值差异。两种后端输出张量形状均为[1,9,3]，最大绝对差为 1.1005×10^{-4} ，平均绝对差为 2.5352×10^{-5} ，均方根差为 3.9914×10^{-5} 。由于最大误差略高于 10^{-4} 阈值，因此在严格的 allclose(1e-5) 和 allclose(1e-4) 条件下均未通过。该现象主要来自不同推理后端在算子实现、浮点计算顺序和图优化策略上的差异。

上述结果说明，ONNX Runtime 能够显著降低 ST-GCN 单模型推理延迟，但它属于推理后端替换带来的单点加速。与系统级协同优化不同，ONNX Runtime 并不直接解决前端视频读取、检测、跟踪、输出之间的串行等待问题，也不直接解决连续窗口重复构造和调度问题。因此，

表5 ST-GCN PyTorch 与 ONNX Runtime 输出差异

指标	数值
输出形状	[1,9,3]
最大绝对差	1.1005×10^{-4}
平均绝对差	2.5352×10^{-5}
均方根差	3.9914×10^{-5}
allclose(1e-4)	False
allclose(1e-5)	False
权重是否完全匹配	True

ONNX Runtime 与该方法具有互补关系：前者侧重推理引擎优化，后者侧重系统执行组织优化。

7.7 ST-GCN 滑动窗口一致性验证

为验证滑动窗口推理机制不会改变

ST-GCN 的预测输出，进一步进行了优化前后结果一致性实验。实验构造 2,000 个连续窗口样本，窗口长度为 12，车道节点数为 9，每个节点包含 9 维特征。传统整窗推理方法和滑动窗口推理方法使用完全相同的模型权重、邻接矩阵和输入窗口内容。实验结果如表 6 所示。

表 6 传统整窗ST-GCN与滑动窗口ST-GCN批量窗口一致性验证

指标	数值
验证窗口数	2,000
窗口长度	12
车道节点数	9
节点特征维度	9
输出形状	[2000,9,3]
输入是否完全一致	True
输入最大绝对差	0.000000
输出是否完全一致	True
输出最大绝对差	0.000000
输出平均绝对差	0.000000
输出均方根差	0.000000
allclose(1e-5)	True
传统整窗平均耗时/ms/window	0.089944
滑动窗口平均耗时/ms/window	0.079667

从表 6 可以看出，在 2,000 个连续窗口上，传统整窗推理和滑动窗口推理的输入张量完全一致，模型输出也完全一致。输出最大绝对差、平均绝对差和均方根差均为 0，并满足 allclose(1e-5) 条件。这说明在保持模型结构、输入窗口内容和邻接矩阵一致的前提下，滑动窗口机制不会改变 ST-GCN 预测结果。

从推理组织时间看，传统整窗方法平均耗时为 0.089944 ms/window，滑动窗口方法平均耗时为 0.079667 ms/window。滑动窗口方法在该抽样实验中略快，主要原因是其减少了重复窗口构造和输入组织

开销。需要说明的是，该实验的核心目的不是评估 ST-GCN 模型本身的预测能力，而是验证优化前后推理结果是否保持一致。

7.8 ST-GCN 预测精度保持验证

除逐窗口输出一致性外，还补充了预测精度保持实验。实验使用 2,000 个评估样本，分别采用传统整窗 ST-GCN 推理和滑动窗口 ST-GCN 推理进行预测，并计算 MAE、RMSE 和 R² 指标。由于两种方法使用相同输入、相同模型权重和相同邻接矩阵，若滑动窗口机制未改变模型计算结果，则两者的预测指标应完全一致。实验结果如表 7 所示。

由表 7 可见，传统整窗推理和滑动窗口推理在 MAE、RMSE 和 R² 指标上完全一致，三项指标差值均为 0。这进一步说明，提出的滑动窗口推理机制只改变连续输入条件下的数据组织和调度方式，不改变 ST-GCN 模型本身的前向计算结果，也不会引入额外预测误差。需要说明的是，表中指标主要用于验证滑动窗口机制是否改变原 ST-GCN 模型输出，而不是用于重新评价 ST-GCN 模型本身的预测能力。

表 7 传统整窗ST-GCN与滑动窗口ST-GCN预测精度保持验证

方法	MAE	RMSE	R2
传统整窗推理	0.811063	1.017129	-0.020258
滑动窗口推理	0.811063	1.017129	-0.020258
绝对差值	0.000000	0.000000	0.000000

7.9 前端检测与跟踪输出稳定性验证

异步流水线优化主要改变前端视频处理链路的执行组织方式，而不改变车辆检

测模型、检测阈值、输入分辨率和多目标跟踪参数。因此，在输入视频、模型权重和跟踪器配置保持一致的情况下，优化前后前端输出应保持稳定。为验证这一点，在同一段视频上抽取 2,000 帧进行检测与跟踪输出稳定性实验。

实验分别生成基线前端输出文件和优

化前端输出文件，二者均使用相同 YOLO 权重、相同输入分辨率、相同置信度阈值、相同 IoU 阈值和相同 DeepSORT 跟踪器参数。输出文件采用 JSONL 格式记录每一帧的检测框、置信度、类别和轨迹 ID。表 8 给出了前端输出文件生成过程的运行记录。

表8 前端检测与跟踪输出文件生成记录

输出文件	处理帧数	分辨率	置信阈值	IoU 阈值	平均耗时/ms	推理 FPS
baseline.jsonl	2,000	640	0.25	0.45	9.1736	109.01
optimized.jsonl	2,000	640	0.25	0.45	6.9960	142.94

需要说明的是，表 8 中的耗时受程序启动状态、依赖安装、缓存预热和 GPU 状态影响，因此该表主要用于记录前端输出文件的生成过程，不单独作为严格的算法加速结论。前端算法稳定性应以两份输出文件的逐帧匹配结果为准。

表 9 给出了前端检测与跟踪输出稳定性比较结果。实验共比较 2,000 帧，基线输出和优化输出均包含 279 个检测目标，匹配检测数为 279。两者平均每帧检测数均为 0.1395，检测数量完全一致，边界框对称匹配率为 1.0000，平均匹配 IoU 为 1.0000，置信度平均绝对差为 0，同一轨迹 ID 匹配率为 1.0000。

由表 9 可见，在 2,000 帧抽样范围内，优化前后前端检测与跟踪输出完全一致。检测数量、边界框匹配、置信度、轨迹 ID 数量和平均轨迹长度均保持不变。这说明异步流水线优化没有改变前端感知结果，其性能提升主要来自执行组织方式的调整，而不是来自检测或跟踪策略的改变。

为更具体地量化各项优化技术的独立贡献，进行了消融实验，结果如表 10 所示。基线方法在无优化条件下端到端日志平均 FPS 为 6.46；只启用异步流水线后，端到端日志平均 FPS 达到 64.76；只启用滑动窗口后，在批量窗口一致性验证口径下，ST-GCN 单窗摊销耗时由 0.089944 ms/window 降至 0.079667 ms/window，端到端日志平均 FPS 为 6.20；当两项优化同时启用后，系统端到端日志平均 FPS 达到 65.94。

需要说明的是，表 10 中的 ST-GCN 单窗耗时为批量窗口一致性验证中的按窗口摊销耗时，与 ONNX Runtime 后端基线实验中的单样本独立推理延迟统计口径不同。

消融结果表明，端到端性能提升主要来自前端异步流水线。滑动窗口机制则在后端连续推理环节提供补充优化，并通过一致性实验验证不会改变预测输出。外部开源流式推理管线基线和 ONNX Runtime 实验进一步说明，通用流式推理管线和推理后端替换都可以带来性能收益，但系统级吞吐仍需要通过阶段解耦、流水线并行

7.10 消融实验

表9 前端检测与跟踪输出稳定性验证

指标	数值
抽样帧数	2,000
基线检测总数	279
优化后检测总数	279
匹配检测数	279
基线平均检测数/帧	0.139500
优化后平均检测数/帧	0.139500
平均检测数绝对差	0.000000
检测数量完全一致比例	1.000000
边界框对称匹配率	1.000000
基线召回匹配率	1.000000
优化后精确匹配率	1.000000
匹配框平均IoU	1.000000
置信度平均绝对差	0.000000
同一轨迹ID比例	1.000000
基线唯一轨迹ID数	13
优化后唯一轨迹ID数	13
基线平均轨迹长度	18.769231
优化后平均轨迹长度	18.769231

和连续窗口调度优化来进一步提升。

8 工程实践讨论

8.1 硬件资源与部署约束

这套优化方法的目标是提高系统实时处理能力。异步流水线检测需要CPU具备一定并发能力，才能支撑视频读取、车辆检测、多目标关联和结果输出形成稳定重叠；滑动窗口推理需要额外缓存空间，用

来保存历史时序特征和窗口状态。基于这个特点，该方法更适合部署在具备一定并行计算能力的GPU平台上。若放到算力和存储资源较为有限的边缘设备上，收益可能会受到限制。针对边缘场景下的实时视频分析，已有工作从模型适配、云边协同和资源分配等角度提出了系统优化方案^[37]。

8.2 参数配置与性能权衡

优化方案的表现不仅取决于算法结构本身，也和关键参数设置密切相关。对异步流水线而言，批处理大小、队列容量和线程协同方式会直接影响吞吐能力与启动延迟；对滑动窗口推理而言，窗口大小和缓存规模会影响状态复用程度以及内存占用。参数配置本质上反映的是吞吐效率、响应时延和资源消耗之间的平衡关系。

以批处理大小B、队列容量Q和窗口大小W为例，B增大通常有利于提高GPU利用率，但也可能带来更长的单批次等待时间；Q增大有助于缓冲短时波动，减轻阶段速度不匹配造成的阻塞，不过也会拉长流水线进入稳态所需的预热时间；W增大可以增强时序信息利用能力和缓存复用潜力，但也会带来更高显存占用以及状态维护开销。

8.3 优化机制的适用边界

表10 优化技术消融实验

配置	前端/系统日志FPS	ST-GCN单窗耗时/ms	端到端日志平均FPS
基线(无优化)	6.46	0.089944	6.46
异步流水线	64.76	0.089944	64.76
滑动窗口	6.20	0.079667	6.20
全优化	65.94	0.079667	65.94

所提出的优化方案属于系统级工程重构，其效果与具体任务负载密切相关。对于视频分析前端，当检测模型耗时明显高于读取和输出阶段时，异步流水线能够通过阶段重叠显著提升吞吐能力；但如果某一阶段长期成为绝对瓶颈，系统整体速度仍会受该阶段限制。对于ST-GCN后端，滑动窗口机制主要复用连续输入中的历史窗口状态和数据组织过程，在严格一致性设置下不改变模型前向计算图，因此能够保证预测输出完全一致。

此外，ONNX Runtime、TensorRT、DeepStream、GStreamer、OpenVINO和Ultralytics StreamLoader等外部推理或视频流处理框架也可用于进一步压缩模型推理或视频处理开销^[31-35]。其中ONNX Runtime更适合作为通用推理优化基线，TensorRT更偏向于通过层融合、精度优化和内核选择提升NVIDIA GPU上的推理效率，DeepStream更适合多路视频流的工业级部署，OpenVINO主要面向Intel CPU、iGPU和NPU等平台优化，Ultralytics StreamLoader则适合与YOLO生态结合进行开源流式推理。与这些框架相比，该方法关注的是不依赖特定推理引擎的系统级执行组织优化，因此与外部推理优化框架具有互补关系。

9 结论与展望

围绕交通效率诊断系统在长时监控视频处理过程中存在的效率瓶颈，重点分析了前端视频处理链路中的串行等待问题，以及后端时序预测链路中的重复窗口组织与调度问题，并在此基础上提出了异步流水线检测与滑动窗口推理相结合的协同优化方法。前者通过重组视频读取、车辆检

测、多目标关联和结果输出的执行方式，提升了前端视频处理的整体吞吐；后者通过利用连续时序输入之间的重叠信息，复用历史窗口缓冲、邻接矩阵驻留和推理状态管理结果，压缩了ST-GCN连续推理阶段的数据组织和调度开销。

实验结果表明，这套方法能够显著提升交通效率诊断系统的整体处理性能。在同一段约109,300帧的测试视频上，传统方法总时间为281.90分钟，日志平均FPS为6.46；全优化方法总时间缩短到27.60分钟，日志平均FPS提升到65.94，效率优势明显。不同方法的日志统计帧数存在百帧量级差异，主要来自批处理取整、尾帧处理和日志记录粒度差异，不影响总体性能比较结论。进一步分析发现，系统吞吐提升的主要来源是前端异步流水线重构，它显著压缩了阶段之间的等待时间；滑动窗口推理则在降低后端重复窗口构造、改善全链路协同性方面发挥了补充作用。

外部开源流式推理管线基线实验进一步表明，Ultralytics StreamLoader能够将系统平均FPS提升至51.54，明显优于传统串行方法，说明通用流式推理管线本身具有较强工程优化能力。但全优化方法仍达到65.94 FPS，相比该外部基线进一步提升约1.28倍，并将总处理时间从35.34分钟缩短至27.60分钟。这说明该方法的性能收益并不只是来自通用视频流读取或推理调度优化，而是来自面向交通效率诊断链路的阶段解耦、异步队列并行以及后端滑动窗口协同设计。

针对外部推理后端基线问题，补充引入了ONNX Runtime对ST-GCN后端推理进行对比实验。实验结果显示，在相同模型权重、相同输入规模和相同邻接矩阵条件下，ONNX Runtime将ST-GCN平均推理延迟由PyTorch的4.5584 ms降低

至2.1901 ms，推理FPS由219.37提升至456.59，获得2.0813倍加速。该结果说明，外部推理框架能够有效降低单模型推理延迟。不过，ONNX Runtime输出与PyTorch输出之间存在极小数值差异，最大绝对差为 1.1005×10^{-4} ，平均绝对差为 2.5352×10^{-5} 。因此，ONNX Runtime更适合作为推理后端加速基线，而提出的异步流水线与滑动窗口机制则进一步面向端到端系统执行组织进行优化。

此外，补充的ST-GCN一致性实验表明，在2,000个连续抽样窗口上，传统整窗推理与滑动窗口推理的输入张量和输出张量完全一致，平均绝对差、均方根差和最大绝对差均为0，预测误差指标MAE、RMSE和R²也完全一致。这说明后端优化机制在提升系统运行效率的同时，没有改变原有ST-GCN模型的预测输出。前端检测与跟踪稳定性实验进一步表明，在2,000帧抽样样本上，优化前后检测数量、边界框匹配率、置信度差异、轨迹ID数量和平均轨迹长度均保持一致，说明前端异步流水线优化没有改变检测与跟踪结果。

后续工作可以沿三个方向继续推进。第一，扩大实验数据规模和场景多样性，在更多路口、更多时段以及更多复杂环境下验证方法的泛化能力。第二，继续研究自适应批处理、动态调度和轻量化建模方法，在保证实时性的同时进一步降低资源消耗。第三，将单路视频优化扩展到多摄像头、多路口协同分析框架中，并进一步探索将流水线调度机制与TensorRT、DeepStream、GStreamer等工业级框架结合，以获得更高部署性能^[33,35]。

总的来说，异步流水线检测与滑动窗口推理协同优化方法，能够有效缓解交通效率诊断系统中的处理效率瓶颈，并为实时视频分析系统的性能优化提供一套具有

实际参考价值的设计思路。

参考文献：

[1] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90.

[2] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[C]//Proceedings of the 3rd International Conference on Learning Representations. San Diego: ICLR, 2015: 1-14.

[3] 董红召, 林少轩, 余翊妮. 交通目标YOLO检测技术的研究进展[J]. 浙江大学学报(工学版), 2025, 59(2): 245-260.

Dong H Z, Lin S X, She Y N. Research progress of YOLO detection technology for traffic targets[J]. Journal of Zhejiang University (Engineering Science), 2025, 59(2): 245-260.

[4] 张浩晨, 张竹林, 史瑞岩, 等. 基于YOLO-NPDL的复杂交通场景检测方法[J]. 山东交通学院学报, 2025, 33(2): 34-47.

Zhang H C, Zhang Z L, Shi R Y, et al. Complex traffic scene detection method based on YOLO-NPDL[J]. Journal of Shandong Jiaotong University, 2025, 33(2): 34-47.

[5] 谭金菁, 毛文逸, 付俊强, 等. 基于YOLOv8的道路交通目标检测优化算法[J]. 上海第二工业大学学报, 2025, 44(4): 78-89.

Tan J J, Mao W Y, Fu J Q, et al. Optimization algorithm for road traffic target detection based on YOLOv8[J]. Journal of Shanghai Second Polytechnic University, 2025, 44(4): 78-89.

[6] Wojke N, Bewley A, Paulus D. Simple online and realtime tracking with a deep association metric[C]//2017 IEEE International Conference on Image Process -

- ing (ICIP). Beijing: IEEE, 2017: 3645–3649.
- [7] 金立生, 华强, 郭柏苍, 等. 基于优化DeepSORT的前方车辆多目标跟踪[J]. 浙江大学学报(工学版), 2021, 55(6): 1056–1064.
Jin L S, Hua Q, Guo B C, et al. Multi-object tracking of preceding vehicles based on optimized DeepSORT[J]. Journal of Zhejiang University (Engineering Science), 2021, 55(6): 1056–1064.
- [8] 徐福良. 城市交通视域下基于YOLOv8和DeepSORT的检测跟踪算法研究[D]. 青岛: 青岛大学, 2025.
Xu F L. Research on detection and tracking algorithm based on YOLOv8 and DeepSORT in urban traffic scenario [D]. Qingdao: Qingdao University, 2025.
- [9] Yu B, Yin H, Zhu Z. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting[C]//Proceedings of the 27th International Joint Conference on Artificial Intelligence. Stockholm: IJCAI, 2018: 3634–3640.
- [10] 钟强. 基于图卷积神经网络的交通流预测研究[D]. 长沙: 中南大学, 2023.
Zhong Q. Research on traffic flow prediction based on graph convolutional neural network[D]. Changsha: Central South University, 2023.
- [11] 徐红飞, 李婧. 基于时空卷积神经网络GL-GCN的交通流异常检测算法[J]. 现代信息技术, 2021, 5(2): 70–75.
Xu H F, Li J. Traffic flow anomaly detection algorithm based on spatio-temporal convolutional neural network GL-GCN[J]. Modern Information Technology, 2021, 5(2): 70–75.
- [12] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Boston: IEEE, 2015: 1–9.
- [13] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition [C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016: 770–778.
- [14] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Honolulu: IEEE, 2017: 4700–4708.
- [15] Redmon J, Farhadi A. YOLOv3: An incremental improvement[J]. arXiv pre-print arXiv:1804.02767, 2018.
- [16] Ultralytics. YOLOv5: An improved version of YOLO[EB/OL]. 2021.
- [17] Wang C Y, Bochkovskiy A, Liao H Y M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. Vancouver: IEEE, 2023: 7464–7475.
- [18] 钱睿珂, 胡海霞. 基于改进YOLOv8的交通场景目标检测方法[J]. 商洛学院学报, 2025, 39(6): 52–61.
Qian R K, Hu H X. Traffic scene object detection method based on improved YOLOv8[J]. Journal of Shangluo University, 2025, 39(6): 52–61.
- [19] 张雨, 罗素云. YOLOv8-DSM: 高精度的小目标交通检测算法[J]. 传感器与微系统, 2025, 44(3): 145–152.
Zhang Y, Luo S Y. YOLOv8-DSM: High-precision small target traffic detection algorithm[J]. Transducer and Microsystem Technologies, 2025, 44(3): 145–152.
- [20] 王桂宝, 李鹏飞, 熊召新. YOLOv8-NTS: 一种针对交通标志检测的目标识别方法[J]. 陕西理工大学学报(自然科学版), 2025, 39(2): 45–51.
Wang G B, Li P F, Xiong Z X. YOLOv8-

NTS: A target recognition method for traffic sign detection[J]. Journal of Shaanxi University of Technology (Natural Science Edition), 2025, 39(2): 45–51.

[21] 顾立鹏, 孙韶媛, 李想, 等. 无人车驾驶场景下的多目标车辆与行人跟踪算法[J]. 小型微型计算机系统, 2021, 42(3): 542–549.

Gu L P, Sun S Y, Li X, et al. Multi-object vehicle and pedestrian tracking algorithm in unmanned vehicle driving scenarios[J]. Journal of Chinese Computer Systems, 2021, 42(3): 542–549.

[22] 金沙沙, 龙伟, 胡灵犀, 等. 多目标检测与跟踪算法在智能交通监控系统中的研究进展[J]. 控制与决策, 2023, 38(5): 1201–1210.

Jin S S, Long W, Hu L X, et al. Research progress of multi-object detection and tracking algorithms in intelligent traffic surveillance systems[J]. Control and Decision, 2023, 38(5): 1201–1210.

[23] 袁凯, 户佐安, 邓锦程, 等. 图神经网络在交通预测中的应用综述[J]. 交通运输工程学报, 2023, 23(5): 39–61.

Yuan K, Hu Z A, Deng J C, et al. Review on application of graph neural networks in traffic prediction[J]. Journal of Traffic and Transportation Engineering, 2023, 23(5): 39–61.

[24] 汤泽慧. 基于时空图卷积网络的道路行程车速预测模型[D]. 北京: 中国人民公安大学, 2025.

Tang Z H. Road travel speed prediction model based on spatio-temporal graph convolutional network[D]. Beijing: People's Public Security University of China, 2025.

[25] 张祥玉. 基于周期解耦的动态图卷积神经网络交通流预测[D]. 兰州: 兰州大学, 2024.

Zhang X Y. Traffic flow prediction based on periodic decoupled dynamic graph convolutional neural network[D]. Lanzhou: Lanzhou University, 2024.

[26] 张鹏程. 基于多尺度动态时空图卷积网络的交通流预测研究[D]. 兰州: 兰州理工大学, 2025.

Zhang P C. Research on traffic flow prediction based on multi-scale dynamic spatio-temporal graph convolutional network[D]. Lanzhou: Lanzhou University of Technology, 2025.

[27] 王建龙. 基于图过程神经网络的城市道路拥堵时空预测[D]. 武汉: 武汉大学, 2022.

Wang J L. Spatio-temporal prediction of urban road congestion based on graph process neural network[D]. Wuhan: Wuhan University, 2022.

[28] 吕甜甜. 深度学习流水线并行优化方法研究[D]. 济南: 齐鲁工业大学, 2024.

Lyu T T. Research on pipeline parallelism optimization methods for deep learning[D]. Jinan: Qilu University of Technology, 2024.

[29] 朱昌宏. 面向行人监测的多路实时视频流处理框架设计与应用[D]. 北京: 北京邮电大学, 2024.

Zhu C H. Design and application of multi-channel real-time video stream processing framework for pedestrian monitoring[D]. Beijing: Beijing University of Posts and Telecommunications, 2024.

[30] NVIDIA Corporation. NVIDIA TensorRT: A programmable inference accelerator[R]. Santa Clara: NVIDIA, 2024.

[31] Ultralytics. Ultralytics YOLO: Stream-Loader for real-time video inference [EB/OL]. 2023.

[32] Microsoft Corporation. ONNX Runtime: Cross-platform ML inference and training accelerator[EB/OL]. 2021.

[33] NVIDIA Corporation. NVIDIA Deep-Stream SDK developer guide[R]. Santa Clara: NVIDIA, 2024.

[34] Demidovskij A, Tarkhanov Y, Gorbachev E, et al. OpenVINO deep learning workflow toolkit: Analysis of improvements [C]//Proceedings of the 30th Interna-

tional Conference on Computer Graphics and Machine Vision. St. Petersburg: CEUR-WS, 2020: 1-7.

[35] Taymans W, Baker S, Wingo A, et al. GStreamer application development manual[M]. Boston: Free Software Foundation, 2023.

[36] 傅浩. 面向视频分析的CNN并行计算优化研究[D]. 天津: 天津大学, 2021.

Fu H. Research on CNN parallel computing optimization for video analysis [D]. Tianjin: Tianjin University, 2021.

[37] 张霞, 乔环宇, 曹峰. 基于改进YOLOv8的高分辨率遥感图像目标检测算法[J]. 大数据, 2025, 11(5): 152-169.

Zhang X, Qiao H Y, Cao F. High-resolution remote sensing image object

detection algorithm based on improved YOLOv8[J]. Big Data Research, 2025, 11(5): 152-169.

[38] 彭森, 张珂珂, 杨一群, 等. 基于改进YOLOv8的矿井复杂环境小目标检测算法[J]. 大数据, 2025, 11(6): 166-181.

Peng S, Zhang K K, Yang Y Q, et al. Small object detection algorithm for mine complex environment based on improved YOLOv8[J]. Big Data Research, 2025, 11(6): 166-181.

[39] 王亮. 基于边缘计算的高效实时视频分析系统研究[D]. 武汉: 华中科技大学, 2024.

Wang L. Research on efficient real-time video analysis system based on edge computing[D]. Wuhan: Huazhong University of Science and Technology, 2024.

作者简介



赵欣宇（2007年生），男，学士，北京航空航天大学软件学院，无职称，学生，主要研究领域为大模型与图机器学习。



张炼今（2007年生），男，学士，北京航空航天大学软件学院，无职称，学生，主要研究领域为大模型与图机器学习。



叶豫桐（1997年生），男，博士，北京航空航天大学计算机学院，无职称，博士后，主要研究领域为图数据管理与分析。



王睿杰（1997年生），男，博士，北京航空航天大学计算机学院，教授，教授，主要研究领域为开放动态环境下的可信高效人工智能。

录用日期: XXXX-XX-XX
通信作者:
基金项目:
Foundation Items: