科学技术与教育 2026年第8期
ISSN: 3079-4455

多视频流跟踪异步解耦机制设计与实验

崔洪祥[1,2]，王华朋[1]
[1]中国刑事警察学院，辽宁沈阳，中国
[2]辽宁省网络安全执法与视频侦查重点实验室，辽宁沈阳，中国

【摘要】随着智能监控、交通感知和多摄像头协同分析的发展，多视频流目标检测与跟踪对系统实时性和并发处理能力提出了更高要求。针对传统串行处理方式在多路视频输入下易出现阻塞、资源利用不足和吞吐量下降等问题，本文构建了基于 YOLO26n 检测器和 SORT 跟踪器的多视频流目标跟踪框架，并实现了单线程串行、普通多线程和异步解耦多线程三种处理方式。实验从吞吐率、单路帧率、延迟和阶段耗时等方面进行对比分析。结果表明，单线程方式能够满足单路视频实时跟踪需求；在多路视频并发场景下，异步解耦多线程方式能够有效减少各处理阶段之间的阻塞，提高系统吞吐能力并降低延迟；当视频流数量进一步增加时，系统性能逐渐受限于硬件资源。总体而言，本文提出的异步解耦方法能够在多视频流场景下提升系统实时性和整体处理效率。

【关键词】多视频流；目标跟踪；YOLO26n；异步解耦；多线程

1.绪论

随着视频监控系统的不断完善以及智慧交通平台和城市感知基础设施的发展，多摄像头协同感知逐渐成为目标检测及跟踪的一个重要应用方向[1]，相比单路视频任务，多视频流的目标跟踪除了需要进行每帧的目标检测以及跟踪更新以外，还需要考虑如何利用有限的计算资源来同时进行多个视频流之间的并行计算。由此产生的吞吐量、延迟以及资源调度等问题也成为影响到多视频流智能分析系统部署的主要障碍[2]。

近几年来由于其高效性和准确性，YOLO 系列单阶段检测器被大量应用于实时视觉应用中。而与其配套的 SORT 等轻量级多目标跟踪器则利用卡尔曼滤波以及匈牙利匹配进行在线关联[3]，在较低的计算开销下仍能达到良好的工程实用性。“高效检测器+轻量跟踪器”的组合方式也因此成为了兼顾实现难度与运行效率的一种途径[4]，但是这样的方式一旦应用于多个视频流的情况下就会出现一系列问题：串行的结构无法充分利用硬件的并行能力；普通多线程可以增加吞吐量，但是仍把检测以及跟踪放在同一条链路上，在一个阶段上花费过多时间会导致整个流程被卡住；而当并发度不断提升时，各个阶段间竞争以及资源争夺会对系统造成更大的影响。

对于上述问题，本文提出了针对多视频流目标跟踪的一种异步解耦方案。系统采用 YOLO26n 进行目标检测，使用 SORT 进行在线跟踪，在视频读取、目标检测以及目标跟踪之间分离并用线程及队列串联各个部分，减少阶段等待，提高多流处理效率[5]。

主要工作可概括如下：

1.以 YOLO26n 和 SORT 为基础，设计了一个单线程串行、普通多线程以及异步解耦多线程三种系统实现方案并对其相应的模块划分与组织方式进行说明。

2.提出一种针对检测、跟踪任务的异步解耦方法，在阶段划分以及队列串联的基础上减少阶段阻塞对整个系统的制约作用。

3.通过对多个实验对比分析，在系统整体吞吐量、平均单路 FPS、平均延迟等方面证明此方法可行性并且探讨不同并发量下表现变化趋势。

2.相关理论基础

2.1YOLO26 检测器原理

YOLO26n 是 Ultralytics YOLO26 系列的一个轻量化检测模型[6]，可以看作是一种针对边缘设备优化的新 YOLO 检测框架。其结构框架如图 1 所示，该模型具有端到端推断方式，在默认情况下可实现无须 NMS 的一对一预测结果输出，以降低后续处理开销以及提高部署速度。因此选择 YOLO26n 主要是因为它轻量级并且具有良好的推理速度，而不是对它的网络结构进行修改。

2.2 SORT 跟踪器原理

SORT 是一个经典的 tracking-by-detection 在线多目标跟踪的方法[7,8]，它并不直接从原

始图像里得到一个目标序列，它将检测器所输出的目标边界框作为输入，然后用卡尔曼滤波来进行状态预测，通过匈牙利匹配来连接当前帧的目标检测结果与其对应的跟踪目标的历史路径。与 DeepSORT、ByteTrack、OC-SORT 等方法相比[9]，SORT 更多地是利用目标的运动信息而不是复杂的外观特征来进行跟踪，所以它的计算量较小，可以方便地与实时检测器结合形成一个在线跟踪系统[10]。本文选择 SORT 可以更好地体现出不同的线程组织方式对于系统的吞吐量以及处理时延影响。

在 SORT 中，卡尔曼滤波用于对目标位置和运动状态进行预测与更新，为当前帧的检测关联提供先验信息[11]。即使目标发生短时遮挡，系统仍可依据历史运动状态维持轨迹连续性。匈牙利算法用于求解轨迹与检测框之间的一对一最优匹配，是 SORT 完成在线关联的关键步骤之一[12]。

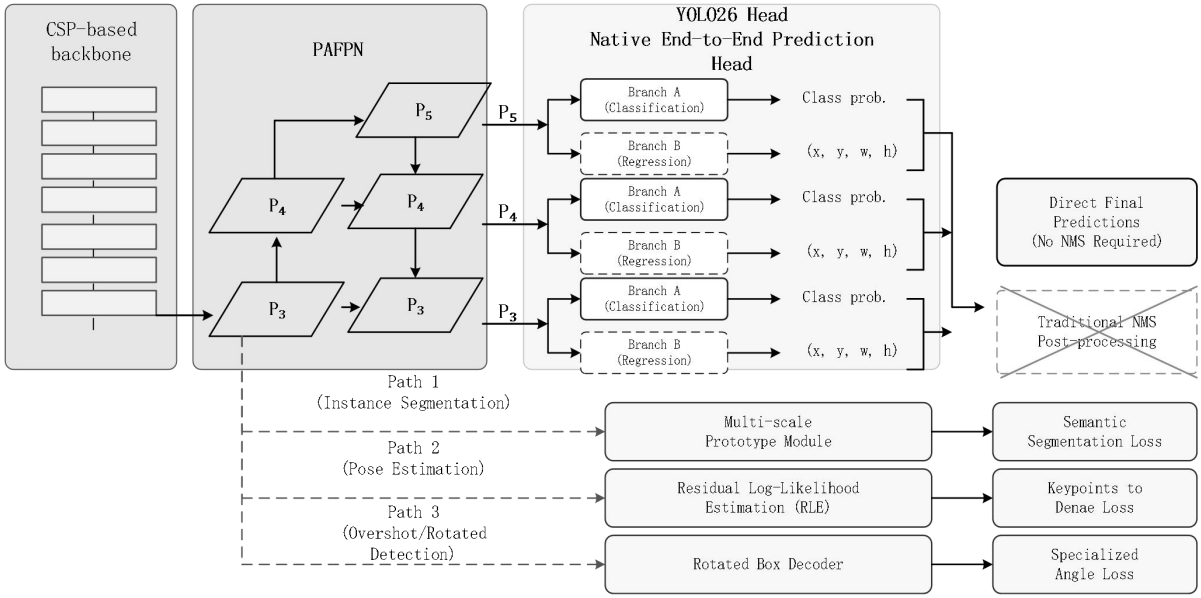

图 1.YOLO26 模型的网络结构

3.系统设计与方法实现

3.1 系统总体架构

本文提出的多视频流目标跟踪系统采用“视频输入—目标检测—目标跟踪—结果输出”的流程进行工作，在同一台计算机上能够同时接收多个视频流并对其进行每帧的目标检测以及目标追踪任务。

$$V_i = I_i^1, I_i^2, \ldots, I_i^{(}T_i), i = 1,2,\ldots,N \quad (1)$$

其中，I_i^t表示第i路视频流中的第t帧图像，T_i表示该路视频总帧数，N 表示系统并发处理的视频流数量。

首先从输入视频流中逐帧读取图像数据并送入 YOLO26n 进行目标检测，然后将检测结果送至 SORT 进行跟踪，根据运动状态对目标进行关联从而完成目标跟踪。最后输出带有检测框以及跟踪 ID 的结果。

$$S_i^t = G(D(I_i^t), S_i^{(}t-1)) \quad (2)$$

其中，$D(\cdot)$表示目标检测过程,$G(\cdot)$表示跟踪状态更新过程，$S_i^{(}t-1)$和S_i^t分别表示第i路视频在相邻时刻的轨迹状态。

因为检测阶段计算量大而跟踪阶段相对较轻，所以不同的结构的区别在于各个部分之间相互联系紧密程度以及同时处理多个视频的情况下的调度方式。

3.2 单线程串行模式

单线程串行模式是最基本的一种方式，在读取一帧视频之后，进行目标检测以及目标跟踪，直到这一帧所有操作完成后才开始下一帧。因为其简单性和清晰性，可以作为衡量整个系统的标准。

对于单路视频而言，在单线程串行方式下无需进行线程切换以及队列间的交互，因此带来额外调度成本较低；检测及跟踪都是以一帧接一帧的方式进行，也方便计算每帧所需时间和整个流程总耗时。但是，这种方式如果应用于多个视频，则每一个检测跟踪链都会变长进而造成帧间等待时间增加，从而影响整个系统的吞吐量，因此不适合大量并发情况下的在线处理。

3.3 普通多线程模式

普通的多线程方法，在这种模式下每一帧单独对应一个线程，但是每个线程内部还是采用“读帧—检测—跟踪”的顺序方式进行工作，相比于单线程串行方式可以使得多个视频同

时进行到较高层次上的并发，从而能够提高整个系统的整体吞吐量。

但是普通的多线程方式所具有的并发能力也受到线程内部串行链路的影响，在检测与跟踪被绑在一个线程的情况下，当前帧检测未完成前下一帧的跟踪无法进行；而跟踪未完成之前下一流水线也不能开始新的检测工作。工作流程如图 2 所示，也就是说这种方式虽然可以做到“流水线上并发”，但是不能达到“阶段上并发”。如果检测是主要开销，则这种紧密联系会带来很大的问题，即在线程中因为对检测或者跟踪的竞争而导致耦合与阻塞从而降低了系统的并行性。

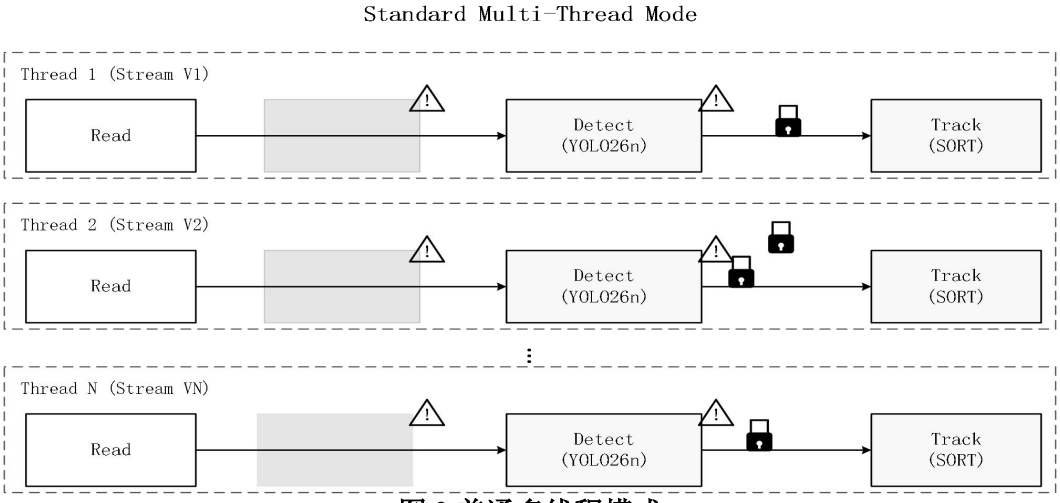

图 2.普通多线程模式

3.4 异步解耦多线程模式

为了解决普通多线程模式下因检测与跟踪强耦合而造成的阻塞问题，我们提出一种异步解耦多线程设计方案。主要思路是：将视频读取、目标检测以及目标跟踪分开，在不同的线程中进行，使用缓冲区传递中间结果。即对每个视频都设置一个读取线程、一个检测线程以及一个跟踪线程：读取线程不断获取视频帧并存放在输入队列里；检测线程从输入队列里面取得一张图片之后就进行 YOLO26n 推理并将推断的结果放入到检测结果队列内；而跟踪线程则是从结果队列里面获得检测输出然后使用 SORT 来更新目标轨迹。

相比于普通的多线程方式，异步解耦的方式在两方面改善系统的运行方式。第一，它把检测和跟踪之间的串行关系变为并行的关系，使得不同的部分可以更像流水线一样同时进行；第二，它利用队列作为缓冲降低各个部分之间的时间上的差异给整个流程带来的影响，也就减少了前后部分之间需要等待的概率。而对于多个视频的情况，这种方式可以有效缓解一部分被卡住的问题，提高硬件的使用效率以及在中等并发的情况下提升整个系统的吞吐量。

对于队列，在异步解耦的多线程中，视频读取、目标检测以及目标跟踪都在不同的线程进行，而它们之间是通过队列来进行通信的，即输入帧队列为Q_f，检测结果队列为Q_d，那么第 i 个视频第 j 帧的操作可以表示为：

$$I_i^t \xrightarrow{Reader} Q_f \tag{3}$$

$$Q_f \xrightarrow{Detector} B_i^t = D(I_i^t) \tag{4}$$

$$B_i^t \xrightarrow{Tracker} S_i^t = G(B_i^t, S_i^{t-1}) \tag{5}$$

队列状态可表示为：

$$Q_f(t) = I_i^a, I_i^{a+1}, \ldots, I_i^b \tag{6}$$

$$Q_d(t) = B_i^c, B_i^{c+1}, \ldots, B_i^d \tag{7}$$

其中，$Q_f(t)$和$Q_d(t)$分别表示时刻 t 下输入帧队列和检测结果队列中的元素集合。

在异步模式下，一帧端到端延迟不再是仅仅只是检测所需时间和跟踪所需时间两者的总和，还会有额外的排队等待时间。

$$T_{i,t}^{async} = T_{i,t}^{wait,f} + T_{i,t}^{det} + T_{i,t}^{wait,d} + T_{i,t}^{trk} \tag{8}$$

其中，$T_{i,t}^{wait,f}$是该帧在输入帧队列中等待时间，$T_{i,t}^{wait,d}$是检测结果在被追踪之前所等待的时间。如果队列等待时间较短并且各个部分可以良好地重叠，则系统的总体吞吐量会更高。

对于异步流水线结构，由于整个系统的效率由最慢一个环节决定，所以可以写成如下形式。

$$FPS_{pipe} \approx \frac{1}{\max(\bar{T}_{read}, \bar{T}_{det}, \bar{T}_{trk})} \tag{9}$$

其中，$\bar{T}_{read}$、$\bar{T}_{det}$和$\bar{T}_{trk}$分别为读取、检测以及跟踪所花费的时间，在本实验中检测耗时

远大于跟踪耗时，因此有：

$$\bar{T}_{det} \gg \bar{T}_{trk} \quad (10)$$

所以，异步解耦并不是减少检测计算量，在阶段并行以及使用队列缓解阶段之间阻塞的情况下，其目的是提高系统的整体性能而不是改进底层检测和跟踪的方法本身。因此，它对系统的有效性主要表现在吞吐量、延迟以及并发性等方面。图 3 展现的是异步解耦模式的工作框架示意图。

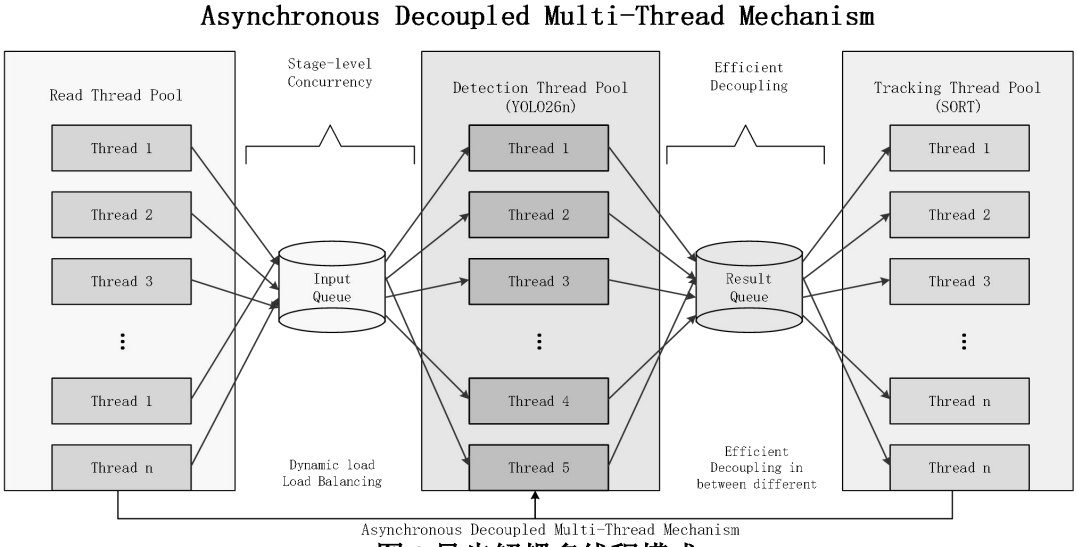

图 3.异步解耦多线程模式

3.5 三种处理模式的比较

从系统结构上看，这三种方法是从低并发到高并发、从强耦合到弱耦合的关系。对比图 2 和图 3，单线程串行方式是最简单的实现方式，但是只能用于单一或者少量的任务；普通的多线程可以将多个视频分配给不同的线程去处理，从而提高并发性；而异步解耦多线程可以在每个线程内部打破串行的流程，在各步骤之间使用队列进行串联，以此来提高整体的流水线程度。因此，在接下来的测试中我们将会根据不同的视频数目对它们做对比分析，以考察异步解耦所带来的优势以及它的适用范围。

4.实验设置与结果分析

4.1 实验设置与评价指标

为评估异步解耦对于多视频流目标跟踪的效果，在相同的软硬件条件下，分别用单线程串行、普通多线程以及异步解耦多线程进行实验比较。检测使用 YOLO26n 网络，而跟踪使用 SORT 算法。在整个实验中，模型权重、输入大小、阈值和参数均相同，只有不同实现方式和不同的视频流数量。

测试数据来源于同一段录像文件并且用复制方法产生多个并发输入，在每一路视频中其分辨率为 700×394，原始帧率为 29.97 FPS，共有 1150 帧。使用同源视频是为了在保持视频内容复杂性情况下比较各种方法自身效率。根据以上要求本文设计了五组实验，分别是单线程串行模式下的一路视频流处理、普通多线程模式下的两路和三路视频流处理、异步解耦多线程模式下的两路和三路视频流处理。

为了对不同的处理方式进行比较分析，本论文选择系统总吞吐量、每一路平均 FPS、平均延迟、平均检测时间和平均跟踪时间等作为评判标准。其中，系统总吞吐量是指：

$$FPS_{sys}=\frac{\sum_{i=1}^{N} F_i}{T_{wall}} \quad (11)$$

其中，F_i是第路视频实际处理完的帧数，T_wall 是系统从启动到所有视频流都被处理完毕所用的时间。这是衡量系统的并行度的一个重要指标。

对于端到端时延，在第 i 路视频第 j 帧进入处理到该帧处理完成所花费时间记作，则单帧时延为

$$L_{i,t}=t_{i,t}^{out}-t_{i,t}^{in} \quad (12)$$

进一步地，系统平均延迟定义为：

$$\bar{L}=\frac{1}{\sum_{i=1}^{N} F_i}\sum_{i=1}^{N}\sum_{t=1}^{F_i} L_{i,t} \quad (13)$$

该指标用于衡量系统对于一个输入帧的平均响应时间，在此之外还有几个指标来表示每一路的任务完成情况以及检测、跟踪这两个部分的时间消耗情况。

4.2 实验结果

不同处理模式下多视频流目标跟踪系统的性能测试结果如表 1 所示。

表 1.不同处理模式下多视频流目标跟踪系统性能对比

方法	视频流数	总吞吐率/FPS	平均单路FPS	平均延迟/ms	平均检测耗时/ms	平均跟踪耗时/ms
单线程串行	1	55.6419	55.6419	15.7984	15.6411	0.1565
普通多线程	2	49.7281	35.3986	35.7029	35.5060	0.1956
异步解耦多线程	2	77.0699	38.1594	24.3632	24.1820	0.1812
普通多线程	3	78.0564	25.6688	34.6016	34.4389	0.1624
异步解耦多线程	3	79.5629	25.2503	36.8340	36.6625	0.1715

从表 1 可以看出，不同的三种处理方式，在不同的并发情况下有较大的区别。单线程串行的方式，在 1 路视频的情况下可以达到 55.6419 FPS，平均延迟是 15.7984 ms，可以满足原始帧率为 29.97 FPS 的视频实时处理的要求。普通的多线程方式，在多个视频的情况下实现了流级别的并发，但是它的检测以及跟踪还是在线程内串行进行，所以在两个视频的情况下整个系统的吞吐量只有 49.7281 FPS。相比之下，异步解耦多线程模式在两路视频流情况下使整个系统的总体吞吐量提高到 77.0699 FPS 并使平均延迟下降为24.3632 ms从而具有更好的并发调度效果。而从整体上看，在一个视频流的情况下，单线程串行模式是比较稳定的；但是在两个视频流的情况下，异步解耦多线程模式表现出最大的优越性；而在三个视频流的情况下，异步解耦多线程模式的优势不再显著。

4.3 系统吞吐能力分析

不同处理模式下系统总吞吐率的对比结果如图 4 所示。

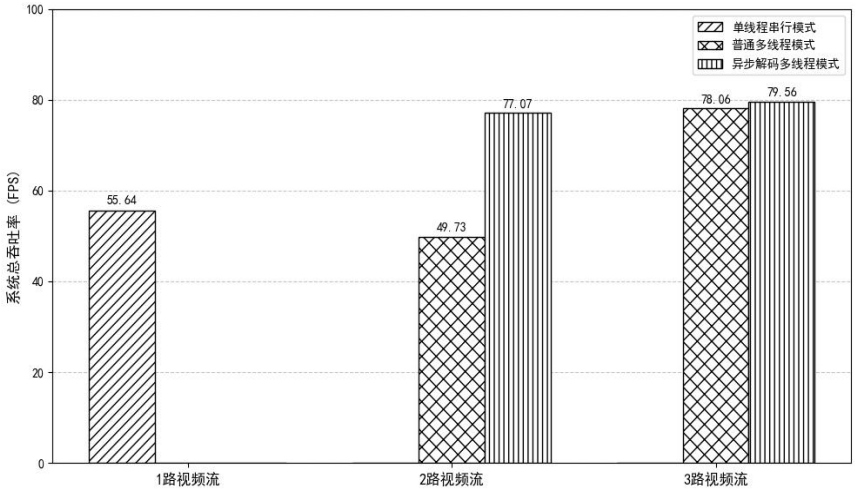

图 4.不同处理模式下系统总吞吐率对比

从图 4 可以看出，在单路视频流情况下，单线程串行模式下系统的总的吞吐量为 55.64 FPS，说明 YOLO26n 与 SORT 结合对于单路任务有足够的实时性。而在两路视频流的情况下，普通的多线程模式下的系统的总的吞吐量为 49.73 FPS，而异步解耦的多线程模式下为 77.07 FPS，比前者有较大提高，这是因为异步解耦可以使线程之间检测以及跟踪的耦合度降低，使系统不仅能在流上并行，而且可以在阶段上并行，所以能提升整个系统的吞吐量。

当视频流增至 3 路时，普通多线程方式以及异步解耦多线程方式下系统总体吞吐量分别为 78.06 FPS、79.56 FPS，二者之间差异较小，这是因为随着并发数上升，系统已接近检测推理资源限制，线程间优化带来的额外提升逐渐减少，所以异步解耦带来的收益具有一定的负载相关性，在 2 路视频流等中等并发情况下效果更佳。

不同处理模式下平均单路 FPS 的对比结果如图 5 所示，在单线程串行模式下，对于一路视频流来说其平均单路 FPS 达到 55.64，远大于视频原本帧率为 29.97 FPS，这表明在本实验所用设备及网络条件下，系统对单路视频处理有较大余量。

对于两个视频流而言，在普通多线程方式及异步解耦多线程方式下，它们各自的平均单路 FPS 分别是 35.40、38.16，而后者依然比前者有优势。这就表示异步带来的好处不仅仅是整个系统的吞吐量有所上升，而且是每个视频流自身处理速度有所提升。但是

对于三个视频流的情况来说，在这两种情况下，它们各自的平均单路 FPS 都是 25.67、25.25，都小于最初的视频帧率即 29.97 FPS，这就表示在这个负载下的情况下，每个视频流所能够得到的计算资源已经比较紧张，系统的实时性受到了影响。

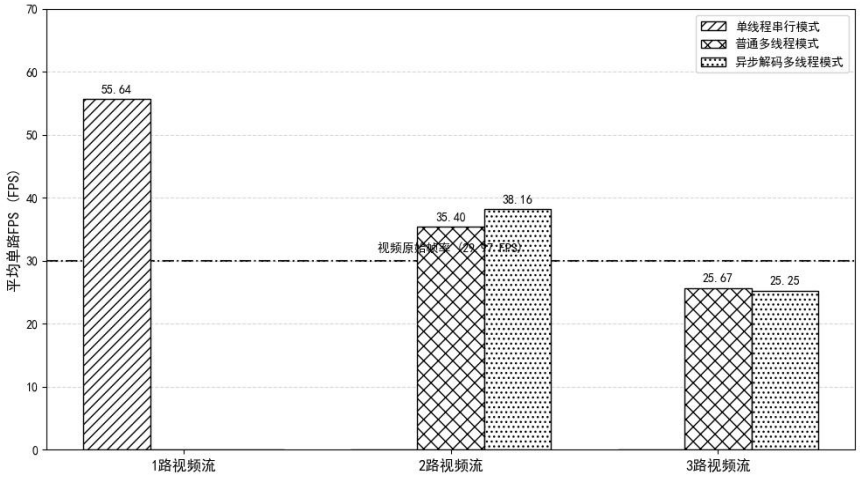

图 5.不同处理模式下平均单路处理帧率对比

4.4 系统延迟特性分析

对于多视频流目标跟踪系统而言，系统总吞吐量表示整个系统的大小，而平均延迟则表示系统对外界请求做出反应的速度。单线程串行模式下，对于 1 路视频流，平均延迟为 15.80ms，说明系统对单一任务可以做到较快地进行目标检测及跟踪；当增加到多路视频流时，平均延迟提高，这是由于并行处理增加了更多竞争以及调度开销所致。

在两路视频流的情况下，普通的多线程方式平均延迟是 35.70ms，而异步解耦的多线程方式是 24.36ms，这说明异步解耦不仅可以提高系统的吞吐量还可以降低端到端的响应时间，因为普通的多线程虽然每个视频流都分配一个线程，但是每一个视频流内部依然是“检测后才开始跟踪”，异步解耦是利用队列以及分段的方式来让检测和跟踪可以更加紧密地连接在一起从而减少由于等待带来的延迟累加。

为了定量描述异步解耦带来的好处相比于传统的多线程方式，我们引入系统总的吞吐量增加比例的概念。即：

$$R_{FPS}=\frac{FPS_{async}-FPS_{multi}}{FPS_{multi}}\times 100\% \quad (14)$$

其中，FPS_{async}表示异步解耦多线程模式下的系统总吞吐量，FPS_{multi}表示普通多线程模式下的系统总吞吐量，在两个视频流的情况下，它们分别是 77.0699 FPS、49.7281 FPS，所以有：

$$R_{FPS}=\frac{77.0699-49.7281}{49.7281}\times 100\%\approx 55.0\% \quad (15)$$

进一步定义平均延迟降低率为：

$$R_L=\frac{L_{multi}-L_{async}}{L_{multi}}\times 100\% \quad (16)$$

其中，L_{multi}和L_{async}分别表示普通多线程模式以及异步解耦多线程模式下的平均延时，在 2 路视频流的情况下，它们分别是 35.7029ms 和 24.3632ms，所以：

$$R_L=\frac{35.7029-24.3632}{35.7029}\times 100\%\approx 31.8\% \quad (17)$$

这表明在 2 路视频流情况下，异步解耦多线程相比普通多线程可获得约 55.0%系统吞吐量以及约 31.8%平均时延改善，说明阶段解耦对于中等并发数量是有效的。

而在 3 路视频流的情况下，普通多线程方式以及异步解耦的方式对应的平均延时是 34.60ms 和 36.83ms，异步解耦的方式并没有继续降低延时的优势。这是因为随着并发流的数量增多之后，系统已经由线程数量不足的问题转变为计算资源不足的问题，因此异步调度所带来的好处已经被更强的推理资源的竞争所掩盖。

4.5 系统延迟特性分析

检测所需的时间在整个测试过程中都远大于跟踪所花费的时间。例如，在单线程串行的情况下，平均检测时间为 15.6411 ms，而平均跟踪时间仅为 0.1565 ms，检测时间是跟踪时间的一百倍。即使对于多个视频流的情况，检测仍然是主要的负担，而 SORT 跟踪更新对于整个流程所占的比例较小。

为了对系统瓶颈进行形式化描述，设读取、检测以及跟踪所花费时间分别是$\bar{T}_{read}$、$\bar{T}_{det}$和$\bar{T}_{trk}$，可以表示为：

$$B=\arg\max_{k\in\{read,det,trk\}}\bar{T}^k \quad (18)$$

结合实验结果可知：

$$B=det \quad (19)$$

也就是说目标检测阶段是当前系统的主要瓶颈。这也能解释为什么不同的并发数量对于异步解耦的效果不同，在两路视频的情况下，由于系统还没有达到检测资源饱和的状态，所以阶段解耦可以有效地解决线程内的阻塞问题，从而提高系统的吞吐量并降低延迟；而当视频的数量上升到三路时，检测阶段的竞争激烈，系统的性能就主要取决于检测推理的能力了。

为了更好地观察异步解耦机制对于不同并发规模所带来的额外增益，我们给出了它相比于普通多线程模式所提高的吞吐量的比例，见下图 6。

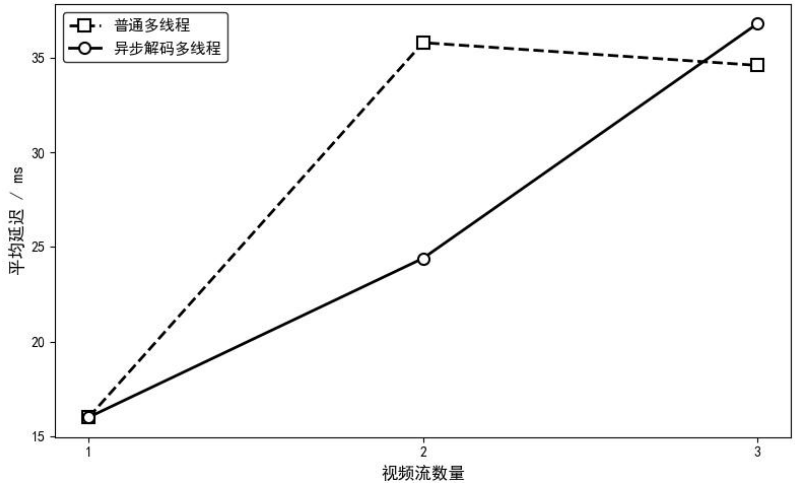

图 6.异步解耦模式相对普通多线程模式的吞吐率提升率对比

从图 6 可以看出，在 2 路视频流情况下，异步解耦多线程方式比普通多线程方式系统的整体吞吐量提高约 55.0%，而在 3 路视频流的情况下，其提高幅度仅为约 1.93%，两者差距较大说明异步解耦带来的优势并不是随着视频流的数量成比例增长，而是受限于检测时所分配的资源大小。

综上，本文实验结果如下：如果系统瓶颈在于阶段同步等待以及线程内阻塞，则采用异步解耦可以有效提高系统性能；但是如果检测推理资源接近饱和，则系统的性能受限于底层算力，在仅用线程进行优化的情况下已经很难达到同样的收益。

5.总结

本文主要针对多视频流目标跟踪问题，对比分析了单线程串行、普通多线程及异步解耦多线程这三种不同的方法。从实验结果可以看出，在一个视频流的情况下，单线程串行方式可以实现 55.6419 FPS，平均延迟为 15.7984 ms，可以满足单个视频流的需求；而在两个视频流情况下，采用异步解耦多线程比普通的多线程，可以使整个系统的总体吞吐量从原来的 49.7281 FPS 提高到 77.0699 FPS，提升了 55.0%，并且把平均延迟从之前的 35.7029 ms 下降到 24.3632 ms，减少了 31.8%，具有良好的并行性。对于 3 路视频流，在这两种多线程模式中，系统的吞吐量分别为 78.0564 FPS 以及 79.5629 FPS，差距已经很小，说明系统已经被检测部分所消耗计算资源所制约。

综上所述，异步解耦处理方式可以在一定程度上缓解中等规模多视频流情况下检测和跟踪之间同步造成的延时问题，在一定程度上提高系统的吞吐率以及及时性，但随着并发数量增加直到达到检测推理资源上限，其带来的增益逐渐趋近饱和状态。这为今后更大并发量下的视频目标跟踪系统改进提供了参考依据。

参考文献

[1]Redmon J, Divvala S, Girshick R, et al. You Only Look Once: Unified, Real-Time Object Detection [C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.
[2]Redmon J, Farhadi A. YOLO9000: Better, Faster, Stronger [C/OL]. 2017.
[3]Redmon J, Farhadi A. YOLOv3: An Incremental Improvement [J/OL]. 2018.
[4]Ge Z, Liu S, Wang F, et al. YOLOX: Exceeding YOLO Series in 2021 [J/OL]. 2021.
[5]Wang A, Chen H, Liu L, et al. YOLOv10: Real-Time End-to-End Object Detection [J/OL]. 2024.
[6]Ultralytics. Ultralytics YOLO26 [EB/OL]. Ultralytics YOLO Docs.
[7]Fang W, Wang L, Ren P. Tinier-YOLO: A

Real-Time Object Detection Method for Constrained Environments [J]. IEEE Access, 2020, 8: 1935-1944.
[8]Bewley A, Ge Z, Ott L, et al. Simple Online and Realtime Tracking [C]//2016 IEEE International Conference on Image Processing. 2016: 3464-3468.
[9]Wojke N, Bewley A, Paulus D. Simple Online and Realtime Tracking with a Deep Association Metric [C]//2017 IEEE International Conference on Image Processing. 2017: 3645-3649.
[10]Zhang Y, Wang C, Wang X, et al. FairMOT: On the Fairness of Detection and Re-Identification in Multiple Object Tracking [J]. International Journal of Computer Vision, 2021, 129(11): 3069-3087.
[11]Zhang Y, Sun P, Jiang Y, et al. ByteTrack: Multi-object Tracking by Associating Every Detection Box [C]//Computer Vision – ECCV 2022. Cham: Springer, 2022: 1-21.
[12]Cao J, Pang J, Weng X, et al. Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking [J/OL]. 2022.