Middleware para Aplicações Distribuídas de Vídeo com Suporte à Computação na Borda na Indústria 4.0

Otacílio de A. Ramos Neto
otacilio.ramos@ifpb.edu.br
Instituto Federal da Paraíba (IFPB)
Guarabira/PB, Brasil

Rafael C. Chaves
rafael.chaves@polodeinovacao.ifpb.edu.br
Instituto Federal da Paraíba (IFPB)
João Pessoa/PB, Brasil

Alysson P. Nascimento
alysson.nascimento@polodeinovacao.ifpb.edu.br
Instituto Federal da Paraíba (IFPB)
Guarabira/PB, Brasil

Ruan D. Gomes
ruan.gomes@ifpb.edu.br
Instituto Federal da Paraíba (IFPB)
João Pessoa/PB, Brasil

ABSTRACT

Within the scope of Industry 4.0, computer vision is extensively employed for monitoring and control functions with stringent demands on performance and latency. A viable approach to meeting these requirements is distributed processing at the edge and in the cloud. In this context, this paper presents a middleware tailored for Industry 4.0 and distributed video applications using edge computing. A versatile communication protocol was developed, with support to both UDP and TCP, and incorporating two methods for frame delivery prioritization (either Last In, First Out or First In, First Out). Also, the protocol performs fragmentation, enabling the transmission of high-resolution images. Initial experiments have shown that the proposed middleware allows the distribution of high-resolution videos without significant overhead, while at the same time offering a high level of transparency for applications, which can be implemented as if getting the video stream from a locally connected camera.

KEYWORDS

Middleware, Distribuição de Vídeo, Edge Computing, Cloud Computing, Industria 4.0

1 INTRODUÇÃO

Nos últimos anos, a importância de sistemas de análise de vídeo tem aumentado significativamente. Um fator que contribui para esse crescimento é a adoção da visão computacional como primeiro passo para controle de qualidade em muitas plantas industriais [19]. O uso de visão computacional também se destaca no monitoramento das atividades humanas em ambientes de fábrica. Muitas vezes, a aplicação desse tipo de sistema se mostra mais eficiente do que o monitoramento feito por seres humanos, podendo auxiliar a alcançar uma maior segurança e precisão na gestão dos processos industriais [3, 9]. A identificação, rastreamento de elementos de interesse e classificação de imagens são apenas algumas das aplicações que reforçam a importância desses sistemas na Indústria 4.0.

In: Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia'2024). Juiz de Fora, Brazil. Porto Alegre: Brazilian Computer Society, 2024.

ISSN 2966-2753

Em ambientes industriais, os dispositivos de captura de vídeo frequentemente possuem capacidade de processamento insuficiente, o que torna necessário transmitir as imagens para dispositivos com recursos de *hardware* mais adequados. Nesse cenário, é essencial que a rede ofereça uma alta taxa de transmissão para permitir a distribuição adequada de vídeos em tempo real. Geralmente, as redes cabeadas são a opção mais viável. No entanto, uma alternativa menos invasiva, mais flexível e que pode atender às necessidades de taxa de bits e latência são as redes 5G privadas [17], que permitem reduzir a complexidade da implantação de novos dispositivos de vídeo, além de viabilizar a conexão de sistemas que possuem alguma mobilidade. Além disso, para aplicações de controle e classificação na indústria, há restrições na latência de respostas, exigindo que o processamento dos dados seja o mais próximo possível da borda da rede [13]. Dessa forma, é importante que a localização do equipamento responsável pelo processamento seja a mais próxima possível do local de atuação, dificultando o uso da computação em nuvem (*Cloud Computing*). Nesse contexto, a computação na borda (*Edge Computing*) se mostra uma alternativa mais adequada, sendo que a combinação de *Edge Computing* e redes 5G pode oferecer vantagens significativas na indústria 4.0, como altas taxas de bits, baixa latência, instalação pouco invasiva e computação próxima da aplicação, sem a necessidade de estar fisicamente integrada ao equipamento [17].

Para desenvolver um sistema de análise de vídeo que atenda aos requisitos de diferentes aplicações de visão computacional, é necessário implementar um sistema distribuído que integre os elementos de captura, transmissão e processamento. Para uma implantação adequada, esse sistema deve possuir características fundamentais, como transparência, abertura e escalabilidade [16]. Transparência, neste contexto, significa que os componentes do sistema são integrados de forma que haja uma ilusão de que todos operam em uma única maquina. A abertura, por sua vez, refere-se à capacidade do sistema de facilitar a integração de novos componentes com pouca ou nenhuma alteração. Uma forma de atingir a transparência em sistemas distribuídos é a partir da utilização de um *middleware*, que é uma camada de *software* que atua como intermediária entre múltiplos componentes do sistema. Dependendo da sua implementação, essa camada pode facilitar a integração de novas aplicações, promovendo um sistema distribuído mais aberto.

Nesse contexto, o *middleware* proposto neste trabalho apresenta diversas características importantes para sistemas de análise de

vídeo. Entre elas, destacam-se a capacidade de replicar o fluxo de vídeo de um produtor para múltiplos clientes, disponibilizar os vídeos para aplicações através de dispositivos virtuais (*/dev/video**) e utilizar um protocolo de transmissão próprio que permite a entrega de vídeos em altas resoluções, com diferentes lógicas de priorização para a entrega dos quadros.

Por meio do *middleware*, diferentes aplicações de análise de vídeo podem ser implementadas, seja consumindo um mesmo fluxo ou fluxos distintos de vídeo, sem que seja necessário lidar com a complexidade inerente à distribuição dos fluxos de vídeo e o acesso concorrente ao mesmo fluxo por diferentes aplicações, seja na mesma máquina ou em máquinas distintas. Além disso, o *middleware* oferece funcionalidades que permitem automatizar de maneira mais fácil a implantação ou migração das aplicações ao longo de uma infraestrutura de computação distribuída, de modo a atingir aos requisitos necessários de velocidade de processamento e latência. Nesse sentido, a aquisição dos fluxos de vídeo a partir das aplicações ocorre como se estas estivessem lendo a partir de um dispositivo de vídeo correspondente a um equipamento conectado localmente à máquina, independente do local onde a aplicação executa na infraestrutura de computação distribuída.

Foram realizados experimentos para verificar o funcionamento do *middleware* em um cenário inicial de computação na borda. Verificou-se que ele consegue distribuir fluxos com resoluções de até 4K em tempo real, sem provocar sobrecarga significativa, mantendo estabilidade na taxa de quadros entregue para as aplicações responsáveis por analisar o vídeo. No experimento, foram avaliados cenários utilizando UDP e TCP e também considerando as duas lógicas distintas suportadas pelo *middleware* para a priorização da entrega dos quadros (i.e. *Last In, First Out* e *First In, First Out*).

O restante deste artigo está organizado da seguinte forma: na Seção 2 é descrita uma breve fundamentação teórica, abordando conceitos de *Edge Computing* e protocolos pra transmissão de vídeo; na Seção 3 são descritos os trabalhos relacionados a este artigo encontrados na literatura; na Seção 4 o *middleware* proposto neste artigo é descrito; na Seção 5 são descritos os resultados obtidos a partir de experimentos para avaliar o desempenho e o funcionamento do *middleware*; finalmente, na Seção 6 são descritas as conclusões e as perspectivas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A implementação do conceito de *Cloud Computing* viabilizou o uso sob demanda da infraestrutura de *data centers*, permitindo o processamento intensivo de dados, provendo escalabilidade e disponibilizando *hardware* robusto [4]. Nesse contexto, a computação em nuvem tem sido aplicada no processamento massivo de dados, principalmente em sistemas de análise de vídeo que fornecem *insights* valiosos a partir da aplicação de técnicas de visão computacional. No entanto, essa abordagem enfrenta desafios significativos em razão da quantidade de dados gerados. Como exemplo, uma câmera com resolução HD (*High Definition*) pode produzir até 200 GB de vídeo não comprimido por dia [8]. Essa grande quantidade de dados, se transmitida em larga escala, pode levar a problemas mais frequentes de sobrecarga de rede e de *hardware* nas centrais de processamento [18]. Além disso, a computação em nuvem impõem limitações para aplicações de análise de vídeo devido à latência adicionada durante a transmissão dos dados e questões relacionadas à privacidade [7, 13].

O surgimento do conceito de *Edge Computing* representa uma evolução desse paradigma, atuando como um complemento à computação em nuvem e provendo recursos para processamento intensivo na borda da rede [13]. Essa estratégia é especialmente eficiente para o processamento de dados de mídia em tempo real, pois alivia a pressão sobre os recursos da nuvem e reduz a latência de transmissão [2, 7]. Por outro lado, também existem desafios ao utilizar a computação na borda. Os dispositivos mais adequados para o processamento de vídeo são as *Graphics Processing Units* (GPUs) [8], projetadas principalmente para execução em servidores ou *desktops*. Embora existam GPUs para aplicações embarcadas, como as da linha Jetson da NVIDIA[1], sua capacidade limitada (quando comparada com outras GPGPUs para servidores) e custo ainda impedem que sejam uma solução definitiva. Como alternativa, o processamento pode ser transferido para servidores de borda, o que exige a transmissão dos vídeos através da rede local. Isso exige que a rede de acesso dos dispositivos seja capaz de prover a taxa de bits necessária e não adicione uma latência excessiva.

Em relação às tecnologias de transmissão de vídeo, diversos protocolos têm sido estudados e desenvolvidos, se diferenciando quanto aos objetivos e funcionalidades implementadas. Protocolos como o DASH (*Dynamic Adaptive Streaming over HTTP)* [14] e o HLS (*HTTP Live Streaming*) [10] foram construídos sobre o protocolo HTTP, com o objetivo de priorizar a escalabilidade e a qualidade da experiência do usuário, ambos apresentando funcionalidades de adaptação da taxa de bits de acordo com o estado da rede. Por outro lado, o WebRTC [5] é um conjunto de protocolos e APIs, desenvolvido para oferecer suporte completo a navegadores e permitir a realização de videoconferências. Embora esses protocolos sejam amplamente utilizados, eles foram projetados para a transmissão e exibição de vídeos na internet, e não são adequados para o cenário considerado neste artigo, que possui foco na distribuição de vídeo para processamento de sinais e visão computacional.

Quanto aos protocolos utilizados para transmissão em tempo real, pode-se destacar o RTMP (*Real Time Messaging Protocol)* [15], que é construído sobre o TCP e, devido a isso, herda as características de controle de congestionamento e confiabilidade, o tornando ideal para cenários de transmissão para a nuvem ou em redes locais sobrecarregadas; no entanto, pode não aproveitar todo o potencial em outros cenários. Nesse contexto, surgem os protocolos construídos sobre UDP, que priorizam transmissões de baixa latência. Entre eles, pode-se citar o RTP (*Real-time Transport Protocol*) [11], um protocolo bem conhecido e amplamente utilizado, e o SRT (*Secure Reliable Transport*) [12], que é um protocolo recente e bastante promissor. O SRT é implementado sobre o protocolo UDP e provê mecanismos para controle de fluxo e controle de congestionamento, entre outras funcionalidades.

Apesar de RTP e SRT serem alternativas interessantes, optou-se neste trabalho pela implementação e utilização de um protocolo próprio com o objetivo de permitir maior flexibilidade na configuração do protocolo de transporte utilizado, bem como na estratégia de priorização na entrega dos quadros, seja FIFO (*First-In, First-Out*) ou

[1] https://www.nvidia.com/pt-br/autonomous-machines/embedded-systems/

LIFO (*Last-In, First-Out*). Essa flexibilização permitiu avaliar o impacto de cada uma das configurações no desempenho do *middleware* em um cenário inicial de computação na borda. Em trabalhos futuros pretende-se realizar um estudo experimental para comparar o desempenho das estratégias implementadas no *middleware* descrito neste artigo com os protocolos RTP e SRT.

3 TRABALHOS RELACIONADOS

Com a crescente adoção do conceito de Indústria 4.0 nos processos produtivos, diversas pesquisas têm abordado a integração de sistemas de visão computacional nas linhas de produção. Além disso, o surgimento e utilização de novos paradigmas como *Edge Computing* e *Cloud Computing* têm viabilizado o desenvolvimento de sistemas de análise de vídeo de forma distribuída. Nesse sentido, nos parágrafos seguintes são descritos alguns trabalhos que foram considerados relevantes e que reforçam a importância do tema abordado no presente artigo.

Em [20] é apresentada uma revisão da literatura sobre o crescente papel da visão computacional nos processos da indústria de manufatura. Os autores descrevem o fluxo de aplicação dessas tecnologias na linha de produção, abrangendo etapas como aquisição de imagens, aplicação de algoritmos e atuação com base nas informações obtidas. Além disso, são discutidas as principais técnicas utilizadas, tais como detecção de características, reconhecimento de objetos, segmentação e modelagem 3D. Por fim, são abordadas diversas aplicações nas áreas de modelagem de produtos, planejamento, produção, controle de qualidade, montagem e transporte. Embora o artigo não trate diretamente sobre aspectos de implementação de sistemas de análise de vídeo, ele contribui para o reforçar a importância de tais sistemas na indústria atual.

Em [1] é descrito um conjunto de aplicações que formam um *pipeline* para análise de vídeos em tempo real. O sistema é composto por diversos módulos que gerenciam os fluxos das câmeras conectadas e realizam diversas ações, tais como identificar os melhores parâmetros (p.ex: taxa de quadros, resolução e posição das câmeras) e gerenciar os recursos disponíveis por meio da redistribuição do processamento entre a nuvem e a borda. Para realizar a prova de conceito o sistema foi aplicado no monitoramento de cruzamentos de ruas, identificando o volume de tráfego e alertando sobre padrões anômalos de trânsito. Como resultado, os autores observaram que o sistema identificou com 95% de acurácia o volume de tráfego nos locais monitorados, utilizando menos de 15% do poder de processamento de um servidor *quad-core* para cada fluxo criado. Apesar de abordar desafios como transmissão de fluxos de vídeo e redistribuição de processamento, o artigo não descreve como tais funcionalidades foram implementadas. Porém, a apresentação de propostas de aplicações nesse sentido reforça a necessidade de *middlewares* que facilitem a construção de sistemas de análise de vídeo de forma distribuída.

Em [6] é descrito um *middleware* para distribuição de vídeo para aplicações de processamento de vídeo na borda, baseado no paradigma produtor-consumidor, com armazenamento de dados. Uma característica importante do *middleware* é o acesso através de uma API simples, permitindo que a aplicação consumidora configure parâmetros e resoluções dos fluxos de vídeo, equilibrando a latência de transmissão e a acurácia dos métodos utilizados. Para caracterizar o comportamento da latência na rede, os autores implementaram um *testbed* utilizando duas NVIDIA Jetson TX2, um servidor de borda x86 equipado com uma GPU e um ponto de acesso Wi-Fi (802.11ac 5GHz). Durante os testes, identificaram que a latência cresce quase linearmente em relação ao tamanho dos *frames* e que a transmissão de dois fluxos simultâneos pode aumentar a latência em até 164%. O *middleware* proposto por [6] se destaca por possibilitar o ajuste da qualidade da transmissão e acesso por meio de uma API simplificada. Por outro lado, o *middleware* apresentado no presente artigo oferece a transmissão de forma transparente, permitindo que as aplicações consumidoras recebam os fluxos como se fossem câmeras conectadas fisicamente à máquina que realiza o processamento do fluxo.

Em [8] é descrito o sistema *Vision Edge IoT* (VEI), um *Edge gateway multi-cloud* projetado para aplicações de análise e processamento de vídeo na borda. O VEI atua como um *middleware* de distribuição de vídeo e também como um *gateway* para a publicação dos resultados em serviços de nuvem, por meio de uma API simplificada. O *middleware* do VEI é baseado em um sistema de publicação/assinatura (Pub/Sub), permitindo a replicação de um fluxo de vídeo para múltiplas aplicações clientes. Os autores realizaram diversos experimentos para avaliar o desempenho do VEI utilizando diferentes aplicações de processamento de vídeo, incluindo testes com múltiplos clientes simultâneos. Os resultados demonstraram que a latência permaneceu estável mesmo com a carga adicional de múltiplos clientes.

O *middleware* apresentado em [8] possui semelhanças com a solução proposta neste trabalho, porém, difere no formato da interface disponibilizada. Enquanto o VEI permite sua utilização através de chamadas de API no código da aplicação cliente, a solução proposta no presente artigo permite o acesso ao fluxo de vídeo da rede como uma câmera diretamente conectada à máquina consumidora. Uma vantagem dessa abordagem é permitir a utilização de aplicações clientes sem que seja necessária nenhuma modificação, não importando se ela está localizada no dispositivo de captura, na nuvem ou no servidor de borda. Além disso, a replicação dos vídeos por meio de dispositivos virtuais oferece outra vantagem, uma vez que é possível instanciar múltiplos clientes para processamento do vídeo em uma mesma máquina, sem que haja a necessidade de transmitir o fluxo individualmente para todos os clientes. No *middleware* proposto no presente artigo apenas um fluxo é transmitido para cada máquina, e o *middleware* replica esse fluxo para os dispositivos virtuais que são acessados pelos clientes distintos.

As soluções encontradas em nosso levantamento não implementam os conceitos de transparência de localização e transparência de concorrência em um sistema distribuído. De forma sucinta, a transparência de localização é a capacidade que um sistema distribuído possui de ocultar a localização geográfica de um recurso fazendo parecer que ele é local. Já a transparência de concorrência é a capacidade de ocultar que um recurso é compartilhado por vários usuários ao mesmo tempo [16]. Para isso, as fontes de vídeo remotas devem se apresentar como se fossem câmeras locais na mesma máquina onde executa a aplicação cliente, sendo esse o principal diferencial da solução proposta no presente artigo. Da mesma forma, para que o sistema de distribuição de vídeo possua transparência de concorrência, vários clientes devem poder acessar

o fluxo de vídeo ao mesmo tempo, seja em uma mesma máquina ou em máquinas diferentes.

4 DESCRIÇÃO DO *MIDDLEWARE* PROPOSTO

O primeiro passo para a criação da solução foi decidir se a funcionalidade de distribuição de vídeo seria integrada aos *drivers* dos dispositivos de captura ou implementada como um *software* separado. Considerando que adicionar essa funcionalidade aos *drivers* contraria seu propósito principal (funcionar como interface entre o *hardware* e o sistema operacional) e que outros recursos de rede, como sistemas de arquivos, são compartilhados por meio de *software* externo (p.ex: o *Network File System* - NFS), optou-se por criar um *middleware*, que captura as imagens a partir das interfaces fornecidas pelos *drivers* e as distribui para as máquinas clientes usando interfaces padronizadas do sistema operacional.

O *middleware* proposto neste artigo foi criado com o objetivo de prover duas funcionalidades principais. A primeira é a transmissão e distribuição de vídeos satisfazendo os conceitos de transparência de localização e transparência de concorrência. A segunda é a flexibilidade com relação à entrega dos quadros, com a opção de priorizar a entrega do quadro mais recente em detrimento da retransmissão de quadros mais antigos (lógica LIFO) ou utilizando uma abordagem em que quadros antigos não são descartados de imediato quando um quadro novo torna-se disponível (lógica FIFO). Estas funcionalidades influenciaram as decisões de projeto tanto da arquitetura quanto do protocolo de comunicação.

4.1 Arquitetura e Modos de Funcionamento

O *middleware* é o responsável por integrar todos os componentes de um sistema de análise de vídeo, sendo sua principal característica a flexibilidade para se adaptar às diferentes estruturas de distribuição. Esse sistema é composto por três módulos fundamentais, como ilustrado na Figura 1: o *Capture*, instalado no dispositivo de captura para encapsular os quadros em pacotes e enviar o fluxo de vídeo ao servidor de distribuição; o *Decapture*, instalado na máquina de processamento para remontar os quadros e disponibilizar o fluxo de vídeo para as aplicações através de dispositivos de vídeo virtuais (*/dev/video**); e o servidor de distribuição, que funciona como o núcleo do *middleware*, e faz a distribuição dos fluxos de vídeo.

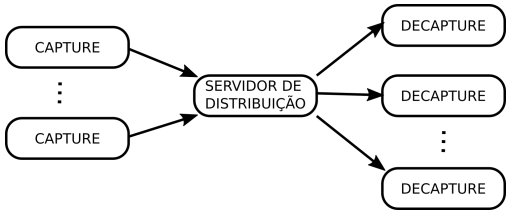

Figura 1: Arquitetura do *middleware*.

Por meio dos dispositivos de vídeo virtuais, as aplicações podem consumir os fluxos de forma simples, como se estivessem lendo de uma câmera local, devido à abstração provida pelo *middleware*. Diferentes aplicações também podem consumir o mesmo fluxo em simultâneo para realizar diferentes tipos de processamento. Como exemplo, a Figura 2 mostra um fluxo sendo recebido em um computador e aberto em simultâneo pelos *softwares* VLC e Scilab, por meio dos dispositivos de vídeo virtuais criados pelo *middleware*. Essa flexibilidade, aliada à possibilidade de replicação e distribuição dos fluxos de vídeo para diferentes máquinas, permite uma maior facilidade no desenvolvimento e execução das aplicações. Isto é importante, por exemplo, em cenários de Indústria 4.0, que empregam recursos de computação na borda para o processamento de vídeo.

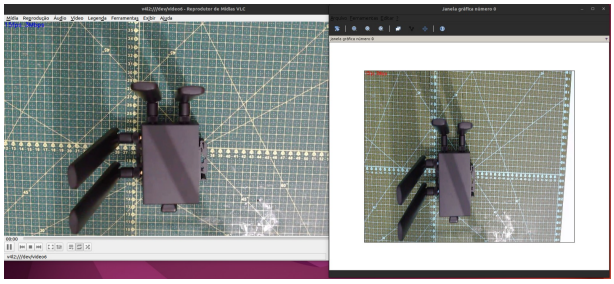

Figura 2: Fluxo sendo aberto em simultâneo por dois *softwares* distintos utilizando a abstração provida pelo *middleware*.

Em uma arquitetura de *Edge Video Analysis* (EVA), representada na Figura 3, parte do *middleware* (*Decapture* e Servidor de Distribuição) está instalada no servidor de borda para processamento, enquanto o *Capture* está instalado no dispositivo que provê o fluxo de vídeo. Por exemplo, em um cenário de redes 5G privadas, a máquina de borda poderia ser a mesma máquina que executa as funções de rede da RAN (*Radio Access Network*), o que permitiria minimizar a latência em cenários em que há a necessidade de integração com sistemas de controle. Na figura, considerou-se a execução do *Decapture* na mesma máquina que executa o servidor de distribuição. No entanto, o *Decapture* poderia ser executado em outras máquinas conectadas na mesma rede local, permitindo assim a execução distribuída das aplicações na infraestrutura de computação na borda.

Por outro lado, em uma arquitetura de *Cloud Video Analysis* (CVA), representada na Figura 4, o servidor de distribuição pode estar localizado tanto na nuvem quanto na rede local dos dispositivos de captura, enquanto o módulo de *Decapture* deve estar presente em uma máquina na nuvem. Também é possível realizar o processamento dos fluxos combinando infraestrutura de computação na borda, para tarefas que requerem baixa latência e geração de mensagens de volta para os dispositivos finais, e computação na nuvem para tarefas que não requerem baixa latência.

Como um exemplo de utilização do *middleware*, pode-se considerar uma aplicação para controle de qualidade da produção na indústria, em que um conjunto de câmeras são posicionadas ao longo do processo produtivo com o objetivo de identificar defeitos ocorridos durante o processo. Os fluxos obtidos a partir dessas câmeras podem ser processados em servidores na borda e gerar comandos de controle (p.ex: para separação automatizada de produtos defeituosos). Por meio da utilização do *middleware* descrito neste artigo, as aplicações que realizam a análise dos *frames* podem ser implementadas de maneira mais simples, sem se preocupar com a complexidade relacionada à captura do vídeo no dispositivo final e

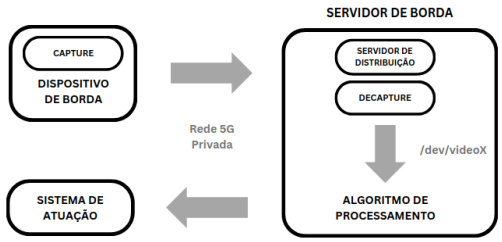

Figura 3: Arquitetura para soluções que utilizam *Edge Computing*.

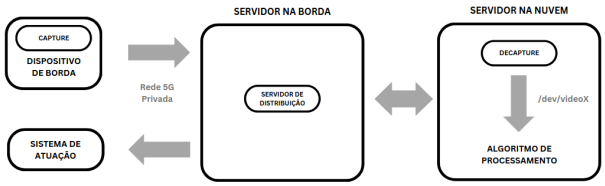

Figura 4: Arquitetura para soluções que utilizam *Cloud Computing*.

à distribuição do vídeo para processamento nos servidores de borda. Assim, o foco maior no desenvolvimento da aplicação se concentra nos algoritmos de visão computacional. Além disso, pode-se executar diferentes instâncias das aplicações em diferentes máquinas de modo a aumentar o desempenho (p.ex: cada instância processa uma região dos quadros ou aplica um algoritmo diferente para identificação de defeitos). Por meio do *middleware* torna-se possível também automatizar a instanciação das aplicações e a configuração necessária para que os fluxos de vídeo sejam encaminhados para cada instância, garantindo a criação dos dispositivos virtuais de vídeo a serem abertos pelas aplicações nas máquinas de processamento.

4.2 Protocolo de Comunicação

Optou-se pelo desenvolvimento e utilização de um protocolo próprio para avaliar a influência de diversas configurações no desempenho do *middleware*. Entre elas estão a estratégia de priorização na entrega dos *frames*, que pode ser importante para sistemas de controle em tempo real, e a flexibilidade na escolha de protocolo de transporte (UDP ou TCP), permitindo a adaptação do *middleware* para cenários diversos, a depender dos requisitos das aplicações e das características da rede.

Frequentemente, as ações dos dispositivos de controle devem ser baseadas nas informações mais recentes. Nesse contexto, um protocolo que realize a entrega dos quadros utilizando uma abordagem *First in, First Out* (FIFO) pode introduzir um atraso no sistema devido à entrega dos quadros sempre na mesma ordem em que foram capturados e enviados. Em contraste, existe a opção de recuperar os quadros com uma abordagem *Last In, First Out* (LIFO), o que garante que os quadros mais recentes disponíveis serão priorizados para envio às aplicações.

Em cenários de processamento na borda, usualmente em redes sem sobrecarga, a utilização do UDP pode prover baixa latência sem prejudicar a entrega dos pacotes. Por outro lado, em cenários onde o processamento é feito na nuvem ou onde há uma sobrecarga da rede, o TCP pode ser mais indicado, devido às características de controle de congestionamento.

O procolo de comunicação implementado possui dois tipos de pacotes com tamanho total de 1472 bytes. O pacote do Tipo 1 possui um cabeçalho com cinco campos: assinatura (2 bytes), canal de comunicação (1 byte), tipo do pacote (1 byte), tamanho da carga útil (2 bytes), número de sequência (8 bytes), segundo do instante de envio (8 bytes) e nanossegundo do instante de envio (8 bytes). O pacote do Tipo 2 possui os mesmos campos do pacote do Tipo 1 e mais dois campos: número do fragmento (2 bytes) e número total de fragmentos (2 bytes), que são utilizados no procedimento de remontagem de quadros que passaram pelo processo de fragmentação por serem maiores que a carga máxima de um pacote do Tipo 1. Uma ilustração do formato dos pacotes é apresentada na Figura 5.

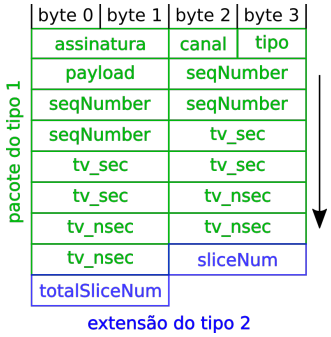

Figura 5: Campos do protocolo de comunicação.

O algoritmo de envio funciona da seguinte forma: inicialmente ele testa se o bloco de dados cabe em um único pacote do Tipo 1. Se sim, ele registra o instante de tempo e envia o pacote do Tipo 1, caso contrário ele calcula quantos pacotes do Tipo 2 são necessários e executa um laço para enviar todos os fragmentos do quadro em pacotes do Tipo 2. Os campos de sequência, tipo, canal, tv_sec, tv_nsec e assinatura também são preenchidos antes do envio.

A recepção dos pacotes é feita por uma *thread* que fica bloqueada aguardando a chegada de dados da rede. No momento em que ela recebe um pacote, ele é imediatamente lido, para evitar descarte, e adicionado na fila de recepção. A remontagem dos quadros é feita na *thread* principal, utilizando um algoritmo baseado em uma janela deslizante sobre o número de sequência dos quadros, à medida que os pacotes são entregues.

A fila usada para armazenamento durante a remontagem cresce conforme os pacotes de novos quadros são recebidos. Caso a fila atinja seu tamanho máximo, ela começa a "deslizar" descartando quadros antigos até liberar espaço suficiente para adicionar no final da fila o novo quadro cujo pacote inédito foi recebido. Se um pacote recebido for de um quadro mais antigo que o quadro mais antigo atualmente na fila ele é descartado. Se o pacote for de um quadro que está na janela da fila, ele é adicionado na posição do quadro correspondente. Quando um quadro é completado ele é enviado

para o decodificador e todos os mais antigos que ele são removidos. Por fim, caso o pacote seja de um quadro mais novo do que o mais recente, a fila cresce para acomodar o novo quadro. Se em algum momento a fila de remontagem entrar em um estado em que todas as suas posições estão ocupadas, então o quadro mais antigo é descartado para liberar espaço para a remontagem de um novo quadro.

A Figura 6 ilustra a janela de quatro quadros no processo de remontagem. Nela, os quadros 1 e 2 já foram remontados ou descartados e não estão na janela. Os quadros 3, 4 e 6 já receberam 3, 2 e 1 pacotes de dados, respectivamente. O quadro 7 ainda não teve nenhum pacote entregue. Caso algum seja entregue antes do quadro 3 ser completado, este será descartado para que a janela possa deslizar para a direita e acomodar o novo quadro cujo pacote foi recebido.

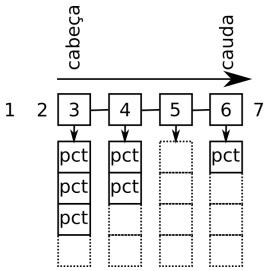

Figura 6: Ilustração da fila de remontagem deslizando sobre o número de sequência dos quadros.

5 AVALIAÇÃO DE DESEMPENHO E FUNCIONAMENTO

A avaliação do sistema foi feita a partir de três métricas de desempenho: *quadros por segundo* (fps), *taxa de bits* do fluxo de vídeo e *latência* dos quadros. Escolhemos quadros por segundo por ser usada como métrica para *hardware* de processamento vídeo, taxa de bits por se usada como métrica de capacidade de transmissão de uma rede e latência por indicar quanto tempo o quadro levou para ser transmitido. Além dessas métricas, também foi realizada a contagem dos quadros descartados para verificar o funcionamento do algoritmo de priorização de quadros. A avaliação foi feita para as quatro configurações possíveis de transmissão do *middleware*: TCP-FIFO, TCP-LIFO, UDP-FIFO e UDP-LIFO, para 14 diferentes resoluções de imagem. O uso de FIFO ou LIFO diz respeito à forma como o algoritmo de remontagem retira os pacotes da fila de recepção. No caso do FIFO os quadros são remontados priorizando a ordem em que são enviados, já no caso do LIFO os quadros são remontados com priorização dos pacotes dos quadros mais recentes que foram recebidos.

A bancada de testes foi organizada para uma aplicação de EVA, conforme a Figura 7, em que as imagens foram capturadas a uma taxa de 28 quadros por segundo no formato MJPEG por uma câmera Logitech BRIO ligada a uma placa Jetson Nano, executando Ubuntu 18.04. Os quadros foram transmitidos para um notebook com arquitetura AMD64, executando o sistema operacional Ubuntu 22.04. O notebook faz o papel de servidor de processamento enquanto a placa Jetson Nano executa os componentes do *middleware* responsáveis pela captura e distribuição dos fluxos de imagens para os clientes.

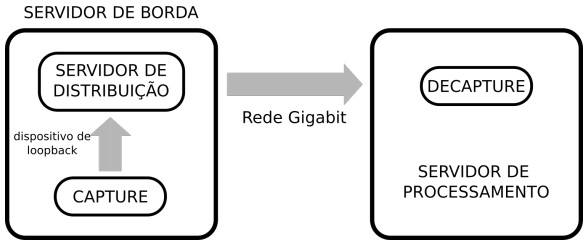

Figura 7: Arquitetura usada na avaliação da capacidade de distribuição e remontagem dos quadros.

As medições foram feitas durante 10 minutos para cada configuração, sendo que as máquinas mantiveram seus relógios ajustados por meio do *software ptp4l*. Para todos os cenários foi considerada uma captura a partir da câmera a 28 fps.

O resultado das medições para a métrica de quadros por segundo (fps) é apresentado na Figura 8. A partir desta figura, percebe-se que o sistema exibiu um desempenho muito próximo para todas as configurações, sem diferenças relevantes, desde a resolução mais baixa (160x120, QSIF) até a mais alta (4096x2160, DCI4K). Neste caso o sistema conseguiu entregar os quadros no destino com a mesma taxa de quadros em que elas foram capturadas, demonstrando que, para essa taxa de quadros, o *middleware* não adiciona sobrecarga significativa.

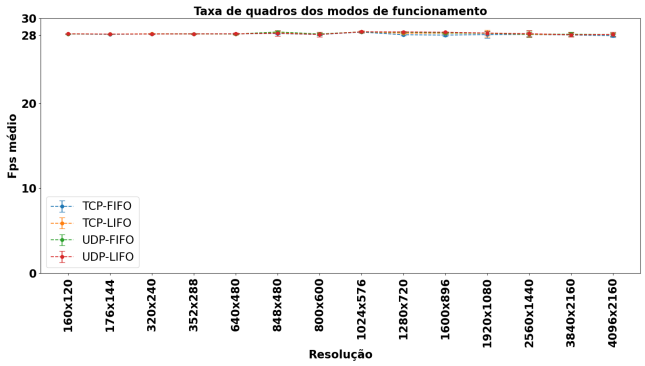

Figura 8: Taxa de quadros por segundo para cada configuração.

A Figura 9 exibe os resultados para a métrica de taxa de bits. Tal como na métrica de quadros por segundo, o *middleware* exibe desempenho semelhante para todas as configurações, sendo que as variações estão dentro da margem de tolerância calculada como sendo o desvio médio em relação ao valor médio da taxa de bits para uma mesma resolução.

Os resultados para a métrica de latência na entrega dos quadros são apresentados na Figura 10. Para este último caso, nota-se que existe uma diferença perceptível para a transmissão de quadros em resoluções iguais ou maiores que *Full-HD* nas configurações diferentes. Esta diferença advém do aumento considerável na quantidade

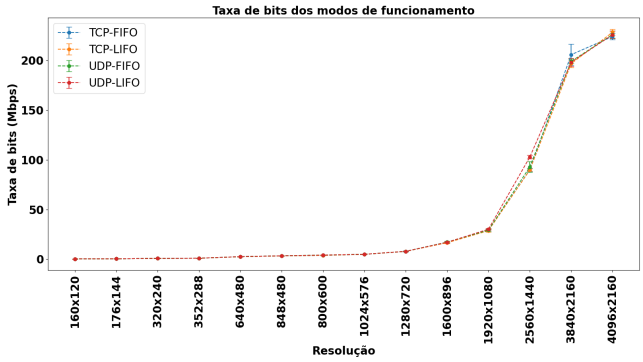

Figura 9: Taxa de bits por segundo para cada configuração.

de *pixels* de uma resolução para outra, impactando no algoritmo de remontagem dos quadros.

Comparando os resultados exibidos no gráfico, percebe-se que as configurações usando UDP apresentam uma menor latência até a resolução 2560x1440, com diferenças mais significativas para as resoluções 1920x1080 e 2560x1440. Para as resoluções 3840x2160 e 4096x2160, o TCP apresentou melhores resultados. Para as resoluções menores que 1920x1080 o uso de FIFO ou LIFO, tanto com TCP como UDP, não apresentou diferenças significativas na latência. Entretanto, para o uso de TCP, em resoluções maiores ou iguais a 1920x1080 a remoção por LIFO apresentou vantagens perceptíveis. Já para UDP o uso de FIFO nestas resoluções proporcionou uma latência média menor. No entanto, ao considerar o desvio padrão, não é possível afirmar que o uso de FIFO ou LIFO provoca diferença significativa na latência. A latência menor do TCP nas resoluções maiores é um resultado que foge do esperado, de modo que uma análise mais cuidadosa será realizada em trabalhos futuros para entender melhor este resultado.

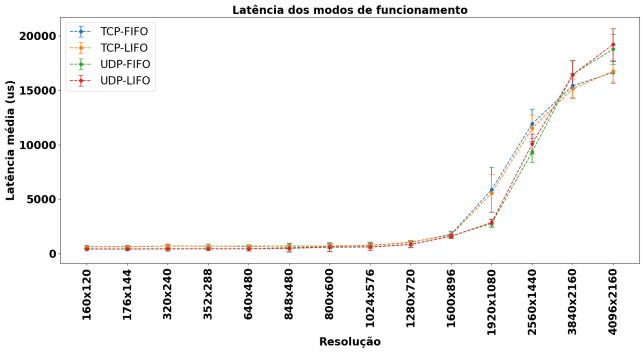

Figura 10: Latência para cada configuração.

No geral, percebe-se que no cenário avaliado, o *middleware* funcionou adequadamente e conseguiu entregar no destino um fluxo de vídeo com a mesma taxa de quadros capturada a partir da câmera, para todas as resoluções avaliadas, incluindo a DCI 4K. No entanto, existem algumas diferenças em termos de latência que são visíveis na Figura 10 para as diferentes configurações. Em cenários de distribuição mais complexos, considerando redes com possibilidades de rotas distintas e a distribuição dos vídeos para múltiplas máquinas, diferenças mais significativas entre os resultados obtidos para cada uma das quatro configurações possíveis poderão ser observadas. Isso será avaliado em trabalhos futuros.

O outro aspecto que o experimento de medição de desempenho analisou é a capacidade do *middleware* de descartar quadros que são mais antigos que os quadros mais novos e já remontados. Os resultados obtidos são apresentados nas figuras 11, 12 e 13, sendo que não existe gráfico correspondente para a configuração TCP-FIFO porque neste caso o controle de congestionamento do TCP, a retransmissão de pacotes e a retirada dos pacotes da fila de recepção na mesma ordem em que foram enviados, fazem com que o descarte de quadros pelo *middleware* seja zero em todas as resoluções, ficando a cargo da aplicação cliente realizar descartes caso seja necessário.

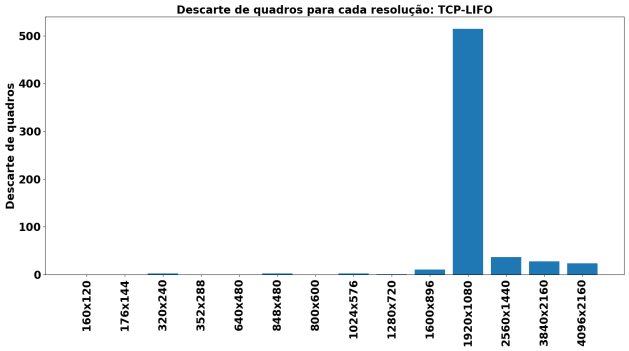

Figura 11: Número de quadros descartados para 10 minutos de transmissão usando TCP-LIFO.

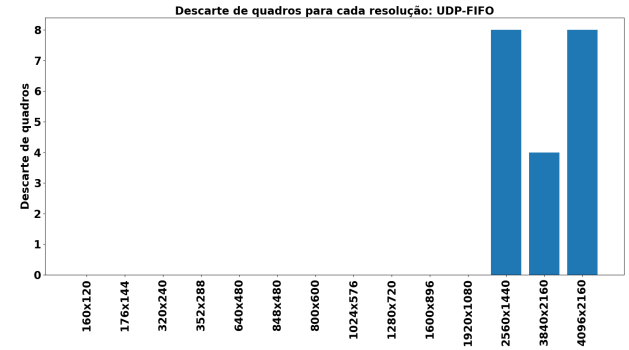

Figura 12: Número de quadros descartados para 10 minutos de transmissão usando UDP-FIFO.

Observando as figuras 11, 12 e 13 percebe-se claramente o funcionamento do mecanismo de descarte. Na configuração UDP-FIFO só houve descarte nas três maiores resoluções e o número foi muito baixo ($\approx 0,05\%$ de quadros descartados no período de 10 minutos). Isso mostra que nesse caso o descarte só ocorreu quando os quadros não puderam ser remontados devido à perda de pacotes na rede. Já nas configurações 11 e 13 tem-se um número maior de descartes, sendo o máximo ocorrido na resolução 1920x1080 no TCP-LIFO

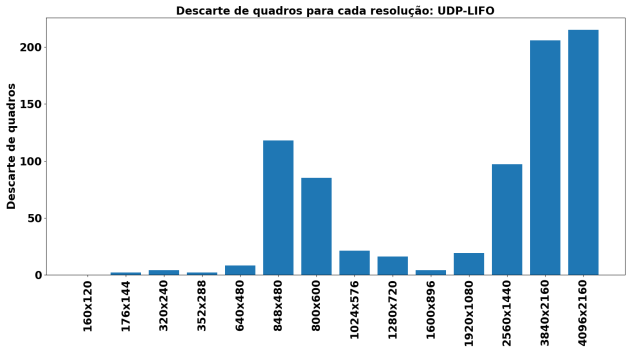

Figura 13: Número de quadros descartados para 10 minutos de transmissão usando UDP-LIFO.

(≈ 2, 98% de quadros descartados). Esse resultado se deve ao fato de que os pacotes são retirados da fila de recepção priorizando o mais recente, fazendo com que o sistema remonte o quadro mais recente primeiro e descarte os quadros mais antigos. Em geral, para esse cenário simples de experimentação, não foi possível estabelecer um padrão para o comportamento desta métrica de acordo com a resolução ou o modo de configuração. Diferenças mais significativas devem ser observadas em cenários mais complexos de avaliação, o que será alvo de trabalhos futuros.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este artigo apresenta um novo *middleware* para dar suporte ao desenvolvimento de aplicações distribuídas de vídeo, tendo como foco aplicações para Indústria 4.0 em um cenário de computação na borda. O *middleware* é capaz de capturar, replicar, distribuir e entregar fluxos de vídeo utilizando interfaces de vídeo virtuais genéricas (*/dev/video**) do sistema operacional Linux. Além disso, foram apresentadas as métricas de desempenho, como taxa de quadros por segundo, taxa de bits e latência, obtidas em uma bancada de testes com um cenário inicial de computação na borda.

Observou-se que para todas as resoluções, incluindo a resolução 4K, o *middleware* entregou para as aplicações uma taxa de quadros estável e muito próxima à capacidade máxima da câmera (≈ 28 fps), para todas as configurações de uso permitidas pelo *middleware* (i.e. TCP-FIFO, TCP-LIFO, UDP-FIFO e UDP-LIFO). Esses resultados indicam uma baixa sobrecarga causada pelo mecanismo de distribuição. Dessa forma, o *middleware* permite o desenvolvimento de aplicações de análise de vídeo de maneira simplificada, ao mesmo tempo em que garante a distribuição e o acesso simultâneo por múltiplas aplicações aos fluxos de vídeo gerados. As aplicações precisam apenas abrir o dispositivo de vídeo virtual criado pelo *middleware* para ter acesso ao fluxo de vídeo de entrada.

Como trabalhos futuros, o sistema será testado em duas novas bancadas de teste conectadas por meio de uma rede 5G privada e uma rede Wi-Fi 6, em cenários de Indústria 4.0. Por fim, também serão avaliados outros cenários de distribuição de processamento para investigar o desempenho e a escalabilidade do *middleware* em cenários mais complexos de distribuição, considerando múltiplos servidores de borda para processamento e redes de transporte mais complexas.

AGRADECIMENTOS

Este trabalho é parcialmente apoiado pela EMBRAPII e pelas empresas Cisco, Prysmian e MPT Cable. Os autores também agradecem ao CNPq (305536/2021-4), ao IFPB e ao Polo de Inovação do IFPB.

REFERÊNCIAS

[1] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *Computer* 50, 10 (2017), 58–67. https://doi.org/10.1109/MC.2017.3641638

[2] Yung-Yao Chen, Yu-Hsiu Lin, Yu-Chen Hu, Chih-Hsien Hsia, Yi-An Lian, and Sin-Ye Jhong. 2022. Distributed Real-Time Object Detection Based on Edge-Cloud Collaboration for Smart Video Surveillance Applications. *IEEE Access* 10 (2022), 93745–93759. https://doi.org/10.1109/ACCESS.2022.3203053

[3] Tamás Czimmermann, Gastone Ciuti, Mario Milazzo, Marcello Chiurazzi, Stefano Roccella, Calogero Maria Oddo, and Paolo Dario. 2020. Visual-Based Defect Detection and Classification Approaches for Industrial Applications—A SURVEY. *Sensors* 20, 5 (2020). https://doi.org/10.3390/s20051459

[4] Marios D. Dikaiakos, Dimitrios Katsaros, Pankaj Mehra, George Pallis, and Athena Vakali. 2009. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing* 13, 5 (2009), 10–13. https://doi.org/10.1109/MIC.2009.103

[5] W3C Working Draft. 2012. WebRTC 1.0: Real-Time Communication between Browsers. https://www.w3.org/TR/webrtc/

[6] Anjus George and Arun Ravindran. 2019. Distributed Middleware for Edge Vision Systems. In *2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT and AI (HONET-ICT)*. 193–194. https://doi.org/10.1109/HONET.2019.8908023

[7] Arpit Jain and Dharm Singh Jat. 2020. An Edge Computing Paradigm for Time-Sensitive Applications. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. 798–803. https://doi.org/10.1109/WorldS450073.2020.9210325

[8] Samantha Luu, Arun Ravindran, Armin Danesh Pazho, and Hamed Tabkhi. 2022. VEI: a multicloud edge gateway for computer vision in IoT. In *Proceedings of the 1st Workshop on Middleware for the Edge* (Quebec, Quebec City, Canada) *(MIDDLEWEDGE '22)*. Association for Computing Machinery, New York, NY, USA, 6–11. https://doi.org/10.1145/3565385.3565877

[9] Mahmoud Meribout, Asma Baobaid, Mohammed Ould Khaoua, Varun Kumar Tiwari, and Juan Pablo Pena. 2022. State of Art IoT and Edge Embedded Systems for Real-Time Machine Vision Applications. *IEEE Access* 10 (2022), 58287–58301. https://doi.org/10.1109/ACCESS.2022.3175496

[10] R. Pantos. 2009. *HTTP Live Streaming*. Internet-Draft draft-pantos-http-live-streaming-02. Apple Inc. https://datatracker.ietf.org/doc/pdf/draft-pantos-http-live-streaming-02.pdf

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. 2003. *RTP: A Transport Protocol for Real-Time Applications*. https://datatracker.ietf.org/doc/html/rfc3550

[12] M.P. Sharabayko, M.A. Sharabayko, J. Dube, JS. Kim, and JW. Kim. 2021. *The SRT Protocol*. https://datatracker.ietf.org/doc/html/draft-sharabayko-srt-01

[13] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[14] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia* 18, 4 (2011), 62–67. https://doi.org/10.1109/MMUL.2011.71

[15] Adobe Systems. 2012. *Adobe Real-Time Messaging Protocol (RTMP) Specification*. https://rtmp.veriskope.com/docs/spec/

[16] M. van Steen and A. S. Tanenbaum. 2023. *Distributed Systems* (4th ed.). distributed-systems.net. http://www.distributed-systems.net

[17] Pal Varga, Jozsef Peto, Attila Franko, David Balla, David Haja, Ferenc Janky, Gabor Soos, Daniel Ficzere, Markosz Maliosz, and Laszlo Toka. 2020. 5G support for Industrial IoT Applications— Challenges, Solutions, and Research gaps. *Sensors* 20, 3 (2020). https://doi.org/10.3390/s20030828

[18] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. 2016. Challenges and Opportunities in Edge Computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*. 20–26. https://doi.org/10.1109/SmartCloud.2016.18

[19] Raphael Wagner, Mario Matuschek, Philipp Knaack, Michael Zwick, and Manuela Geiß. 2023. IndustrialEdgeML - End-to-end edge-based computer vision system for Industry 5.0. *Procedia Computer Science* 217 (2023), 594–603. https://doi.org/10.1016/j.procs.2022.12.255 4th International Conference on Industry 4.0 and Smart Manufacturing.

[20] Longfei Zhou, Lin Zhang, and Nicholas Konz. 2023. Computer Vision Techniques in Manufacturing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 53, 1 (2023), 105–117. https://doi.org/10.1109/TSMC.2022.3166397