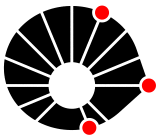

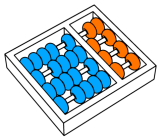

Realidade Estendida e Computação na Névoa: Estudo com Ciclismo Assistido

E. Gama *L. Bittencourt* *L. Vettori* *M. Hatzlhoffer* *P. César*

Relatório Técnico - IC-PFG-23-20
Projeto Final de Graduação
2023 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Realidade Estendida e Computação na Névoa: Estudo com Ciclismo Assistido

Eduardo Gama* Luiz Fernando Bittencourt * Lucca Vettori*
Matheus Hatzlhoffer* Piethro César*

Resumo

Este projeto de fim de curso visa explorar de maneira empírica possibilidades de aplicação de XR em óculos a partir do uso de diferentes arquiteturas e tecnologias para avaliar seus desempenhos no que tange o delay end-to-end de captura de frames a partir do cliente, processamento em névoa e display no cliente. Exploramos diversas opções disponíveis de processamento tanto em CPU quanto em GPU para verificar qual desempenhava melhor quando aplicado em um estudo de caso para ciclismo assistido. O documento apresenta resultados e análises dos diferentes métodos testados durante o trabalho.

1 Introdução

Um movimento que vem sendo percebido a partir da volta da pandemia é um aumento do número de ciclistas ativos nas grandes cidades como São Paulo. Junto com o aumento do número de ciclistas, infelizmente também há um crescimento expressivo do número de acidentes que os envolvem. Para ilustrar esse fato, a Folha de São Paulo apurou que entre janeiro e setembro de 2023 o número foi 8,33 pontos percentuais maior que no mesmo período em 2022 [1].

Em paralelo, o aumento da popularidade de realidade estendida criou novas possibilidades para aumentar a percepção humana em diversos domínios. No mercado ótico grandes empresas de tecnologia competem para se diferenciar e encontrar grandes demandas da população e do mercado que poderiam ser endereçadas a partir do uso de realidade aumentada.

Nesse sentido, com objetivo de atacar esse problema vivida pelos ciclistas com o emprego de tecnologia de realidade aumentada e computação em névoa, o projeto visa explorar possibilidades de softwares e arquiteturas que possam ser usados em aplicações de óculos de realidade aumentada para ciclistas, oferecendo maior segurança ao transitar pelas cidades Brasil a fora.

Por esse motivo, o foco desse trabalho está em explorar Extended Reality (XD) com olhar voltado à latência end-to-end do processamento de frames de stream transmitidos por ciclistas considerando um processamento feito em névoa para garantir baixa latência e um

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

hardware leve no usuário final. A baixa latência é necessário porque nesse tipo de serviço, porque de nada importa reconhecer os elementos presentes nas ruas se não for rápido o suficiente para que o ciclista possa tomar alguma ação em tempo hábil. Já o hardware leve é importante principalmente porque a aplicação é pensada para funcionar em um óculos que por padrão deve ser confortável e não muito complexo.

É extremamente importante frisar que essa aplicação só se torna viável a partir do offloading dos processamentos dos devices que é enviado para a névoa. Esse offloading é responsável por aumentar a duração da bateria dos dispositivos (nesse caso, dos óculos) e pela possível simplificação do hardware e, consequentemente, das dimensões do produto. Além disso, o processamento em névoa permite alcançar requisitos de baixa latência ligadas ao uso rotineiro de aplicações XR [2]. É relevante também dizer que a expansão do 5G também é elemento crucial porque gera uma rede que seja suficiente para lidar com a aplicação em questão.

2 Referencial Teórico

Primeiro precisávamos decidir como abordar o problema, como a ideia era trabalhar com medições de latência, processamento e memória para avaliar a viabilidade do software, a primeira discussão era como simular a rede, o cenário de computação em névoa e decidir quais os protocolos de redes que seriam usados no nosso código.

2.1 computação em névoa

A computação em névoa emerge como uma extensão paradigmática da computação em nuvem, buscando abordar as crescentes demandas por serviços computacionais eficientes e de baixa latência em ambientes distribuídos e heterogêneos. Em sua essência, o conceito de computação em névoa refere-se à descentralização do processamento e armazenamento de dados, transferindo essas operações para dispositivos mais próximos dos usuários ou dos locais de origem dos dados na borda da rede. Isso implica na distribuição de recursos computacionais em uma malha interconectada de dispositivos, como roteadores, servidores de borda e gateways, formando uma "névoa"que se estende desde a nuvem até os dispositivos finais. As vantagens dessa abordagem incluem a redução da latência, a melhoria na eficiência do uso de largura de banda e a capacidade de oferecer serviços em tempo real. Contudo, a implementação bem-sucedida da computação em névoa enfrenta desafios consideráveis, tais como a necessidade de lidar com a heterogeneidade de dispositivos, a gestão eficiente de recursos em ambientes dinâmicos e a garantia de segurança e privacidade dos dados. A otimização desses aspectos demanda o desenvolvimento de algoritmos sofisticados de alocação de recursos, políticas de segurança robustas e arquiteturas flexíveis capazes de se adaptar a mudanças na carga de trabalho e nas condições da rede. Nesse contexto, a compreensão aprofundada das potencialidades e limitações da computação em névoa torna-se essencial para explorar seu pleno potencial em diversos domínios, desde a Internet das Coisas até aplicações críticas em tempo real.

2.2 Processamento de Imagem

O processamento de imagem é um campo interdisciplinar que se concentra na manipulação e análise de imagens digitais, buscando extrair informações significativas e aprimorar a qualidade visual. Esta disciplina abrange uma série de técnicas e algoritmos aplicados a imagens estáticas ou sequências temporais, desempenhando um papel crucial em diversas áreas, incluindo visão computacional, diagnóstico médico, reconhecimento de padrões e automação industrial.

O YOLO (You Only Look Once) [3] é um framework de detecção de objetos em imagens e vídeos. Diferenciando-se de técnicas convencionais, o YOLO aborda o problema de detecção de objetos como uma tarefa de regressão, capacitando um único modelo a predizer simultaneamente as coordenadas das caixas delimitadoras e as probabilidades associadas a várias classes em uma única passagem pela imagem. Esse design confere ao YOLO mais velocidade, tornando-o especialmente adequado para aplicações em tempo real, como em sistemas de vigilância, veículos autônomos e interações em tempo real. O YOLO não só possui uma boa eficiência computacional, mas também sua capacidade de lidar com detecções de objetos em diversas escalas, tornando-o versátil para cenários onde objetos de diferentes tamanhos coexistem.

O SSD (Single Shot Multibox Detector) [4] é outro framework de detecção de objetos em imagens, projetado para otimizar a eficiência e precisão na identificação de objetos em tempo real. Distingue-se por sua abordagem única de prever simultaneamente múltiplas caixas delimitadoras, incorporando características de diferentes escalas extraídas de várias camadas da rede neural convolucional. Essa característica confere ao SSD uma notável vantagem ao lidar com objetos de diferentes tamanhos, proporcionando uma detecção mais robusta e adaptável a variadas condições de cena. Além disso, o SSD destaca-se por sua eficiência computacional, permitindo a realização de detecções em tempo real sem comprometer a acurácia.

O Google Cloud Vision [5] é um serviço de visão computacional fornecido pela Google Cloud Platform, projetado para realizar análise avançada de imagens e vídeos. No contexto acadêmico, o Google Vision utiliza modelos de aprendizado de máquina pré-treinados para executar tarefas como detecção de objetos, reconhecimento facial, leitura de texto, e classificação de conteúdo visual. A aplicação desses modelos é possibilitada por meio de uma API que permite aos desenvolvedores integrar funcionalidades de visão computacional em seus aplicativos e sistemas. O Google Vision utiliza algoritmos sofisticados para interpretar e extrair informações contextuais de dados visuais, contribuindo significativamente para a automação de tarefas relacionadas ao processamento de imagem e vídeo em diversos campos, desde automação industrial até aplicações em saúde e análise de mídias sociais. Sua natureza escalável e a capacidade de processar grandes volumes de dados visuais de maneira eficiente fazem do Google Cloud Vision uma ferramenta valiosa para pesquisas e implementações práticas que demandam análise visual avançada.

No domínio das estruturas de detecção de objetos, YOLO (You Only Look Once) e SSD (Single Shot Multibox Detector) se destacam com características e vantagens distintas. O SSD, apesar de apresentar uma ligeira redução na precisão para objetos menores, prova ser econômico e eficiente em termos de recursos, especialmente em cenários com recursos

limitados. Os modelos SSD300 e SSD500 demonstraram desempenho louvável a 46 FPS com mAP 74,3% e 19 FPS com mAP 76,8% [6]. No entanto, a precisão do SSD pode diminuir ao lidar com modelos extremamente grandes, impactando a velocidade.

Por outro lado, YOLO é conhecido por sua velocidade notável, atingindo 45 quadros por segundo para redes maiores e impressionantes 150 quadros por segundo para redes menores. A capacidade do YOLO de generalizar imagens sem sobrecarregar a memória de processamento contribui para sua eficiência. No entanto, o YOLO enfrenta desafios, especialmente na precisão da localização e na identificação de objetos próximos.

No contexto da detecção de objetos, o SSD revela-se vantajoso em aplicações que demandam um reconhecimento preciso de objetos, como análise forense de vídeo, investigações legais e identificação de pontos de referência. Em contrapartida, o YOLO destaca-se em cenários nos quais a velocidade assume primazia sobre imprecisões mínimas, abrangendo áreas como monitoramento de tráfego em tempo real, detecção de formas de vida em regiões remotas, acompanhamento de produtos agrícolas, veículos autônomos e técnicas de reconhecimento de câncer. Este projeto emprega o processamento de imagem para identificar elementos no vídeo, tais como carros, pedestres e placas. Foram utilizados cenários com ambas as abordagens, YOLO e SSD, a fim de realizar uma comparação abrangente, analisando as discrepâncias em termos de precisão e recursos computacionais empregados.

2.3 Sockets e TCP

O socket é uma interface de comunicação que permite a troca de dados entre processos distintos em uma rede de computadores. Trata-se de uma abstração de programação que facilita a comunicação cliente-servidor, possibilitando a transmissão de informações através de diferentes protocolos, como o TCP (Transmission Control Protocol) ou UDP (User Datagram Protocol). A utilização de sockets é fundamental para o desenvolvimento de sistemas distribuídos, permitindo a criação de aplicações que envolvem a troca de dados em tempo real, como mensagens instantâneas, jogos online, e transferência de arquivos.

O Transmission Control Protocol (TCP) é um componente crucial nas comunicações de rede, operando na camada de transporte do modelo OSI. Este protocolo orientado à conexão garante uma transmissão confiável e ordenada de dados entre sistemas computacionais. No âmbito acadêmico, o estudo do TCP abrange desde seus princípios teóricos até sua implementação prática, considerando suas fases de estabelecimento, transferência e finalização de conexões. Reconhecido por sua capacidade de lidar com desafios como perda de pacotes e congestão, o TCP é comparado ao UDP, destacando-se pela confiabilidade, embora com um custo associado de maior overhead. Compreender as características distintivas do TCP e seus mecanismos de controle é essencial para profissionais e pesquisadores interessados em aprofundar seus conhecimentos em redes de computadores.

2.4 RTMP

O Real-Time Messaging Protocol (RTMP) é um protocolo de comunicação desenvolvido pela Adobe Systems, concebido para viabilizar a transmissão instantânea de áudio, vídeo e dados entre um servidor e um cliente em tempo real. Operando sobre a camada de transporte

TCP, o RTMP oferece uma comunicação confiável, sendo frequentemente empregado em aplicações de streaming ao vivo na internet. Sua notável capacidade de adaptação dinâmica às flutuações de largura de banda permite ajustes na qualidade da transmissão, otimizando o desempenho em ambientes de rede variáveis. Embora previamente difundido, o RTMP observou um declínio em sua popularidade, sendo gradualmente substituído por protocolos contemporâneos, como o HTTP Live Streaming (HLS) e o Dynamic Adaptive Streaming over HTTP (DASH), devido a sua maior versatilidade e desempenho superior em contextos de larga escala.

2.5 Flask

O Flask [7] é um framework web leve e modular para o desenvolvimento de aplicações em Python. Projetado para ser simples e fácil de entender, o Flask oferece uma abordagem descomplicada para a construção de aplicativos web, permitindo que os desenvolvedores foquem na lógica de negócios em vez de lidar com complexidades estruturais. Sua arquitetura minimalista não impõe estruturas rígidas, proporcionando flexibilidade ao desenvolvedor na escolha de ferramentas e bibliotecas adicionais. Com uma comunidade ativa e suporte extensivo de documentação, o Flask é amplamente adotado para o desenvolvimento de projetos web, desde pequenas aplicações até sistemas mais complexos. Sua simplicidade, extensibilidade e integração harmoniosa com outras bibliotecas Python fazem do Flask uma escolha popular para aqueles que buscam uma solução eficiente e elegante para o desenvolvimento web em Python.

2.6 Qualidade de Experiência (QoE)

A Qualidade de Experiência (QoE) é um conceito central no domínio das tecnologias da informação e comunicação, especificamente no contexto de serviços e aplicações que impactam diretamente a experiência do usuário. Refere-se à percepção subjetiva do usuário sobre a qualidade geral de interação com um determinado sistema, serviço ou aplicativo. A QoE transcende simples métricas técnicas de desempenho, incorporando fatores subjetivos como satisfação, usabilidade e resposta emocional. A avaliação da QoE é essencial para entender como os usuários percebem e valorizam a eficácia, eficiência e agradabilidade de uma experiência de uso. Para proporcionar uma experiência positiva, é fundamental considerar não apenas aspectos técnicos, mas também aspectos psicológicos e emocionais que influenciam a percepção do usuário em relação a um determinado serviço ou produto digital. A QoE desempenha um papel crucial na otimização e no aprimoramento contínuo de sistemas e aplicativos, visando atender às expectativas e necessidades dos usuários finais.

2.7 MobfogSim: Vs Sistemas de Docker

O MobfogSim [8] é um simulador avançado que se destina a possibilitar a modelagem da mobilidade de dispositivos e a migração de serviços no contexto da computação em névoa. A complexidade inerente à mobilidade de um usuário pode resultar em desafios significativos no processamento de tarefas nos cloudlets, uma vez que a distância entre a entidade responsável pelo processamento e o usuário final está diretamente relacionada à latência expe-

rimentada. Simultaneamente, a migração de containers e processos entre cloudlets constitui um procedimento oneroso, demandando, portanto, uma análise cuidadosa da relação entre a distância entre o cloudnet e o usuário, e a execução do handover de máquinas virtuais e processos. Estabelecer um equilíbrio eficaz entre esses fatores torna-se imperativo para otimizar o desempenho do sistema, pois a minimização da latência deve ser ponderada em relação aos custos associados à migração de recursos computacionais.

Uma outra alternativa plausível reside na utilização de um conjunto de containers com recursos restritos com o propósito de emular o cenário de estudo em questão. A fim de alcançar configurações mais específicas e resultados mais fidedignos à realidade, o procedimento envolveria a execução de uma diversidade de processos distribuídos em distintos containers, os quais teriam a responsabilidade de emular a dinâmica da mobilidade e realizar transições de processos de um para o outro mediante a prática do handover. Este enfoque se revela crucial para a obtenção de dados mais precisos e aprofundados, uma vez que a simulação envolveria uma gama de fatores realistas que capturam de maneira mais autêntica as nuances inerentes à mobilidade, redes e à transição de processos no contexto da computação em névoa.

A distinção fundamental entre simuladores e emuladores no contexto da computação reside na natureza dos ambientes que cada um reproduz e nas metas específicas que buscam alcançar. Em geral, um simulador é um sistema que replica o comportamento de um determinado sistema ou processo, proporcionando uma representação abstrata e simplificada para análise ou experimentação. Em contrapartida, um emulador procura reproduzir o ambiente operacional exato de um sistema, replicando suas características funcionais e, muitas vezes, até mesmo sua arquitetura interna. Enquanto simuladores frequentemente simplificam certos aspectos para permitir análises mais eficientes para simular cenários com mais clientes, os emuladores procuram reproduzir fielmente o comportamento do sistema original e por isso executam cenários menores. Os simuladores mais adequados para análises de alto nível e os emuladores para replicação precisa de ambientes operacionais específicos.

Como será melhor explicado na próxima seção, como nosso objetivo é entender as necessidades e requisitos dessa aplicação e validar esse cenário de uso em grandes centro urbanos, decidimos modelar nossos problemas e medições de acordo com o emulador usando uma rede de containers.

3 Objetivos

A investigação em tela concentra-se na análise do emprego da computação em névoa (computação em névoa) em conjunto com o processamento de vídeo, em um contexto inerente à mobilidade urbana, com o propósito de capacitar ciclistas a explorarem as potencialidades e capacidades da realidade aumentada. Este enfoque visa facultar a identificação eficiente de elementos urbanos como postes, placas, veículos, e outros objetos corriqueiros nas vias urbanas, almejando, assim, otimizar a experiência de deslocamento e prevenir incidentes. O escopo do projeto se estende à validação dessa concepção sob uma perspectiva técnica, empregando medidas e definições de requisitos de rede abarcando aspectos como velocidade de download e upload, latência, utilização de memória RAM e capacidade de processamento.

Este escrutínio compreende tanto o lado do cliente, encarregado de transmitir as imagens e receber o retorno com os objetos identificados, quanto o servidor, responsável por receber o vídeo e realizar o processamento de forma meticulosa.

- Empreender esforços na exploração de configurações de resolução de transmissão que otimizem a eficiência do uso da rede, sem comprometer a experiência do usuário.
- Analisar a discrepância no desempenho do consumo de recursos em distintas configurações de hardware, incluindo a utilização de GPU em comparação com a utilização exclusiva da CPU, bem como a variação entre execuções locais e execuções em rede.
- Investigar como diferentes modelos de reconhecimento de imagem influenciam a latência e a demanda por recursos, visando compreender de maneira aprofundada os efeitos dessas variações na implementação da solução.

4 Metodologia

Durante a execução, empregamos abordagens específicas para conduzir e analisar o experimento, visando alcançar nossos objetivos definidos. Utilizando um vídeo que exemplifica o ponto de vista de um ciclista como uma representação típica de transmissão neste cenário de mobilidade urbana, adotamos estratégias específicas para assegurar uma transmissão de imagem aprimorada, buscando, assim, aproximar os resultados obtidos a condições mais realistas de aplicação.

No transcurso do projeto, diversificamos os ambientes e cenários de teste, alterando variáveis como os sistemas operacionais empregados, a utilização de GPU versus CPU, a qualidade da transmissão e a capacidade dos modelos de identificar corretamente os objetos durante o processo de transmissão. Essa abordagem permitiu uma abrangente compreensão dos requisitos essenciais para a viabilidade prática da concepção proposta.

Algumas métricas de recursos, tais como uso de memória e capacidade de processamento, fundamentam-se nos relatórios inerentes ao sistema operacional, tanto em relação à máquina como um todo quanto aos processos individuais. Por outro lado, os recursos de rede foram avaliados de maneira independente, empregando a própria execução do código como parâmetro referencial.

4.1 Arquitetura do sistema

Abaixo temos uma representação de como o sistema deve estar estruturado, de maneira a permitir o aprendizado de máquina de acordo com os dados coletados pelos usuários. Esta implementação também prioriza os tempos de resposta individuais para cada usuário, e na usabilidade do hardware como uma peça vestível do sistema. Concentrar o poder de processamento na borda, possibilita tanto a redução do hardware para o ciclista, quanto bons tempos de processamento e resposta, além de permitir a melhoria do modelo de visão por meio de uma nuvem voltada para o treinamento e aperfeiçoamento da parte de visão computacional.

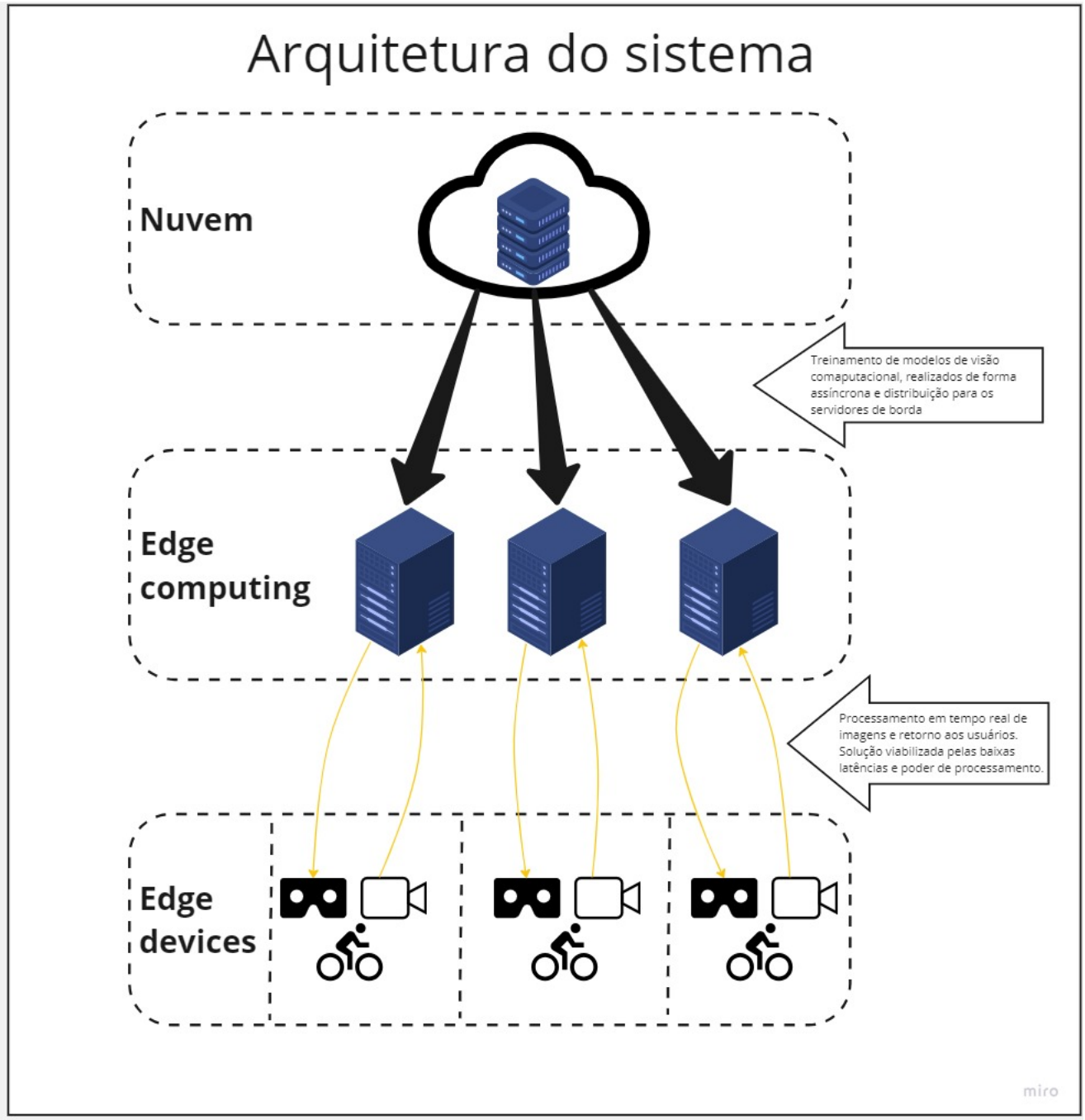

Figura 1: Arquitetura do sistema

4.2 Primeiras Abordagens

Para chegar até o resultado final, foram explorados diferentes caminhos e tecnologias. Aqui eles serão apenas expostos em ordem cronológica de trabalho e posteriormente os detalhes serão apresentados.

Em um primeiro momento, a abordagem desenhada consistia em duas frentes de trabalho. Uma seria responsável por achar uma maneira de processar os frames e a outras por fornecer a estrutura que enviaria e receberia o frame processado. Para processar os frames, chegamos primeiro ao uso de API utilizando o serviço Cloud Vision IA fornecido pelo Google enquanto para fazer a transmissão utilizamos um ambiente em Docker [9] e o protocolo RTMP.

Essa primeira abordagem foi descartada por dois principais motivos: o desempenho de processamento dos frames e a dificuldade que tivemos em criar uma aplicação que funcionasse o protocolo RTMP em um docker.

Nossa segunda abordagem foi desenvolver uma arquitetura cliente-servidor onde o cliente final e a câmera que fornecia os frames estavam no mesmo device (assim como em um óculos) e o servidor estava rodando em outra máquina fazendo o processamento. Nesse trabalho utilizamos sockets TCP para fazer a comunicação entre os devices. Com a infraestrutura cliente-servidor já montada, encontramos o YOLO (You Only Look Once), um método de detecção de objetos de passada única que utiliza uma rede neural convolucional (Darknet) como extrator de características para processar os frames recebidos do vídeo. Em seguida, como uma otimização, passamos a fazer o processamento dos frames em GPU com uma placa de vídeo NVIDIA 1050TI 4GB de notebook. E, embora o resultado tenha melhorado, ainda não parecia uma prova de conceito boa o suficiente considerando a sua aplicação final.

Buscando por outras maneiras de otimizar o delay end-to-end utilizando o YOLO, foi apresentado um novo método que parecia ser promissor. O método em questão é o (SSD) Single Shot Detection, um algoritmo baseado em rede neural convolucional profunda que assim como o YOLO faz a detecção de diversas classes. Com o SSD, já rodando em CPU tivemos o resultado final já melhor que os demais métodos. A diferença foi ainda maior quando utilizamos GPU para processar os frames.

Uma vez introduzidos os diferentes caminhos que foram explorados, agora serão apresentados de maneira detalhada os resultados de delay end-to-end, de processamento e de rede para cada uma das soluções.

4.3 Google Vision

Dado que o projeto concentra-se na análise e observação dos recursos de rede, e não na fase de reconhecimento de imagens, nossa abordagem primordial consistiu em pesquisar e avaliar frameworks que já possuíssem modelos pré-treinados, analisando sua eficácia no contexto da aplicação.

A primeira ferramenta avaliada foi o Google Vision. A Google, detentora de um ecossistema integrado que combina produtos para processamento de imagem com inteligência artificial, demonstrou notável desempenho. Utilizando o kit de desenvolvimento e submetendo diversas imagens, observaram-se resultados excepcionais em termos de eficiência no

reconhecimento de imagem. Cenas contendo diversos objetos foram identificadas de maneira precisa, independentemente da distância em relação ao frame como podemos ver na imagem 2.

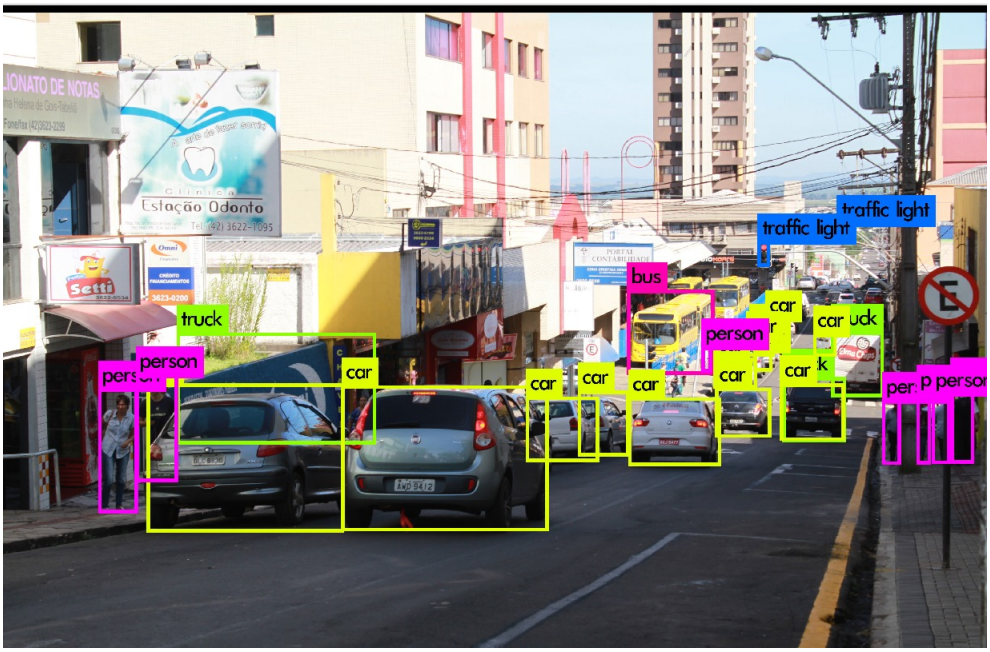

Figura 2: Google vision example

O desafio enfrentado reside na demora significativa no processamento de uma imagem, com cada frame demandando aproximadamente de 10 a 20 segundos para ser analisado. Embora essa latência seja consideravelmente baixa em termos de detalhes e precisão na análise, torna-se substancial no contexto específico de mobilidade urbana. Nesse intervalo de tempo, um ciclista pode ter mudado de direção, uma pessoa pode ter surgido em sua trajetória, ou um veículo pode ter realizado uma manobra de frenagem. Essa considerável demora torna a solução inviável para o escopo do projeto.

4.4 Servidor RTMP e Flask

Quanto à transmissão do vídeo em si, adotamos inicialmente uma abordagem que empregava um protocolo moderno especializado em transmissão de vídeo em tempo real. Optamos pelo RTMP devido à sua eficiência na inicialização do vídeo em comparação com outros protocolos, como o MPEG DASH. Além disso, a escolha do RTMP foi facilitada pela sua implementação acessível a partir de imagens disponibilizadas no Docker.

Conforme ilustrado na figura 3, a arquitetura adotada compreendia um cliente, representado pelo ffmpeg, encaminhando uma stream de vídeo via RTMP para um container do Docker contendo um servidor. Este servidor, por sua vez, realizava a retransmissão para um cliente-servidor Flask, o qual disponibilizava a transmissão no navegador utilizando HTTP. Uma vantagem notável dessa arquitetura é a acessibilidade dos recursos, uma vez que a

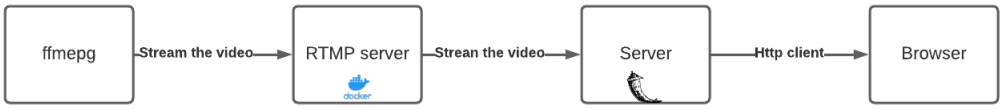

Figura 3: Arquitetura proposta usando RTMP e Flask

conexão com um navegador é generalizada e viabilizada em uma ampla gama de dispositivos. Adicionalmente, o uso de containers Docker proporciona flexibilidade na configuração e orquestração, facilitando o dimensionamento e a implementação eficiente da solução.

Esta abordagem foi descontinuada devido a diversas questões técnicas que apresentou. Primeiramente, os frames transmitidos por HTTP eram compactados, resultando em uma significativa degradação na qualidade do vídeo. A elevada latência, decorrente das múltiplas transmissões, resultava em notáveis discrepâncias entre o fluxo de vídeo original e o apresentado no navegador. Além disso, a transferência por HTTP não era ordenada, resultando na perda de frames e na interrupção da percepção de um vídeo completo e contínuo, o que substancialmente impactava a Qualidade de Experiência (QoE).

4.5 Sockets e TCP

A segunda abordagem consistiu em adotar um protocolo menos especializado em vídeo em tempo real, com o intuito de simplificar a criação e experimentação da transmissão, além de reduzir a quantidade de redirecionamentos. Essa arquitetura foi completamente implementada em Python 3, utilizando sockets.

O cliente transmissor encapsula os frames no formato de arquivos binários e os transmite por meio de sockets para o servidor. O servidor, por sua vez, recebe esses dados pelo protocolo TCP, desencapsula o frame em formato binário, realiza o processamento de imagem (detalhado nas seções subsequentes) e transmite o resultado para o cliente receptor. Este último, por sua vez, retransmite o vídeo modificado, destacando as caixas contendo as identificações de placas, carros e pedestres identificados.

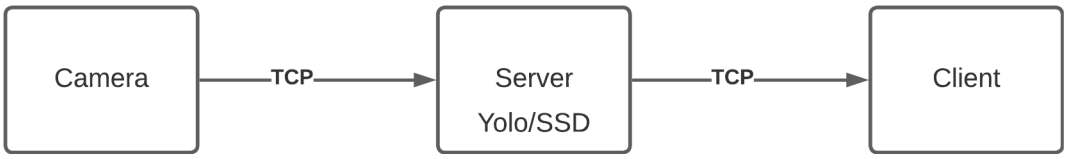

Figura 4: Arquitetura usando Sockets e TCP

Esta arquitetura da figura 4, ao contrário da anterior, não apresenta uma escalabilidade tão robusta. Algumas bibliotecas utilizadas são mais compatíveis com determinados sistemas operacionais, e a implementação em Python para exibir o stream com as imagens

processadas precisa ser adaptada para suportar dispositivos de realidade aumentada ou mista. No entanto, destaca-se por sua menor latência e consumo de recursos. A Qualidade de Experiência (QoE) é excelente para vídeos de menor resolução, proporcionando uma transmissão contínua com baixa latência entre o cliente receptor e o emissor, como será mais abordado na seção subsequente de Resultados. Com base nos resultados obtidos, foi possível dar continuidade e implementar frameworks de processamento de imagem, avaliando essa solução em diferentes ambientes, conforme previamente mencionado.

4.6 Yolo

Com a estabelecimento da comunicação entre clientes e servidor, a atenção foi direcionada para o processamento de imagem. Dado que a velocidade é um requisito crítico para a execução do projeto, optou-se por adotar um framework de detecção de objetos reconhecido por sua eficiência e precisão. O YOLO, notável por realizar um único estágio de processamento por meio de um modelo de detecção de objetos de tiro único (single-shot object detection), baseado em uma rede neural convolucional (CNN), foi o primeiro a ser testado.

O YOLO produz simultaneamente as coordenadas das caixas delimitadoras e a confiança associada em uma única inferência (como pode-se notar nas figuras 5 e 6. O framework opera recebendo uma imagem e emprega 24 camadas da rede convolucional, além de 2 camadas totalmente conectadas, para realizar a detecção de objetos. Conforme os objetos são identificados, várias predições são geradas, e a predição final é determinada pela que possui a maior sobreposição (IOU - Intersection over Union). Essa abordagem possibilita predições eficazes e rápidas, tornando-o uma escolha adequada para cenários em que a velocidade é crucial.

Figura 5: YOLO identificando semáforos, pessoas e carros

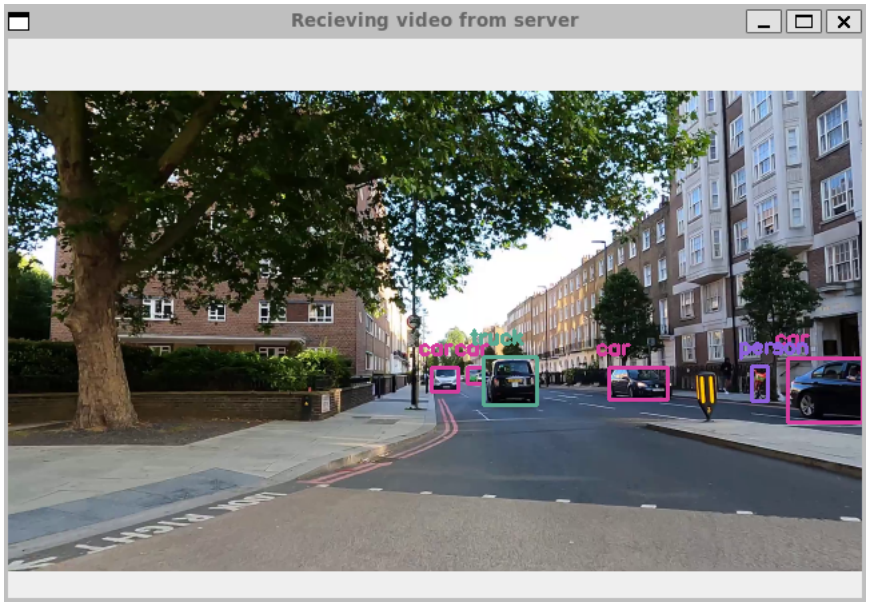

Figura 6: YOLO identificando uma pessoa

No código, essa metodologia é refletida por meio de um modelo pré-definido, contendo pesos e categorias específicas relacionadas à mobilidade urbana, que é executado conjuntamente com o programa. Ao longo das iterações das camadas, as caixas delimitadoras são centralizadas e armazenadas em um array, considerando apenas aquelas com uma confiança superior a 50%. Essa abordagem culmina na produção de resultados visuais, exemplificados nas figuras 5 e 6.

4.7 SSD

Prosseguindo com a abordagem de frameworks para detecção de objetos em um único estágio (Single-Shot Object Detection), o segundo framework analisado foi o SSD. O SSD é composto por duas partes principais: o modelo de base (backbone) e a cabeça (head). O modelo de base, geralmente pré-treinado, tem a responsabilidade de identificar os objetos, enquanto a cabeça consiste em camadas adicionais para definir as caixas delimitadoras dos objetos e suas classificações.

Seguindo uma abordagem semelhante à do YOLO, no código, o SSD também submete a imagem a várias camadas de uma rede neural convolucional (CNN). Essa abordagem fragmenta a imagem em seções e distribui a responsabilidade para diferentes processos. Após a identificação, ocorre uma filtragem das caixas com base em uma confiança mínima de 20%, resultando em saídas visuais, conforme ilustrado nas figuras abaixo:

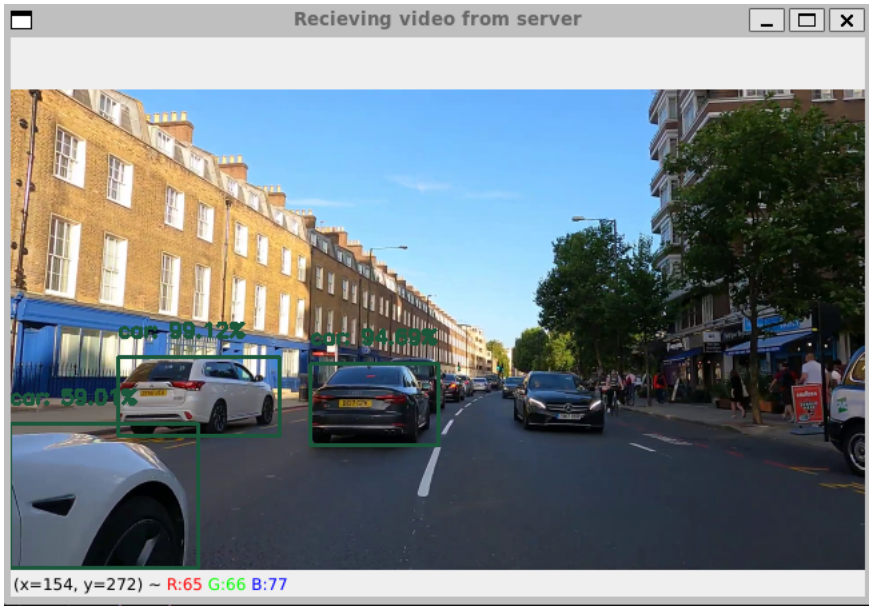

Figura 7: SSD identificando carros

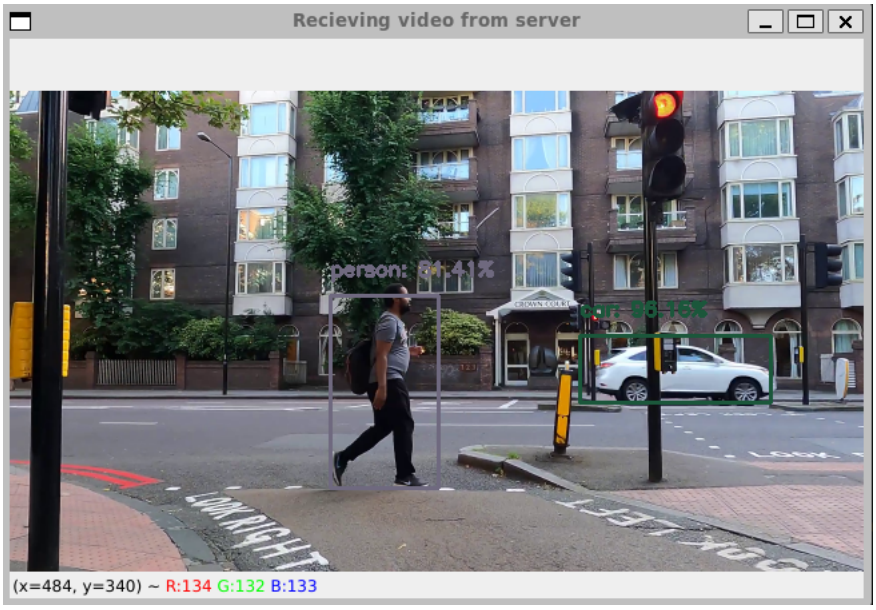

Figura 8: SSD identificando pessoa

5 Resultados

De acordo com nossos objetivos, nesta seção, apresentaremos os resultados obtidos por meio da exploração desse caso de uso. Todos os experimentos foram conduzidos utilizando a linguagem Python, uma escolha justificada pela sua ampla aceitação na indústria, adaptabilidade e robusto suporte da comunidade, permitindo implementações rápidas e precisas. Além disso, é relevante destacar que todos os testes foram realizados com a parte do cliente e o servidor em computadores distintos, conectados por cabos de rede e comunicando-se por meio de uma rede local (LAN). Nenhum sinal Wi-Fi ou redes móveis foi utilizado durante os experimentos, garantindo assim uma qualidade de rede e estabilidade de conexão superiores em relação a um cenário real com um ciclista nas ruas.

5.1 Como a resolução interfere no QoE do usuário

Neste trabalho, foram realizados testes em diversos cenários e resoluções para avaliar a comunicação em relação à qualidade de experiência para o usuário final e os recursos de rede envolvidos. Os indicadores-chave observados, considerados essenciais para a experiência, foram a taxa de quadros por segundo (FPS) e o tempo de resposta entre a captura, processamento de imagem e o resultado estar disponível na tela do usuário. Pois no contexto da mobilidade urbana, é crucial que as cenas registradas pelo ciclista sejam retornadas o mais próximo de tempo real quanto for possível, pois na prevenção de acidentes, cada fração de segundo pode ser decisiva.

Outro indicador fundamental para compreender a viabilidade dessa execução é o uso dos recursos de banda, tanto de download quanto de upload. é essencial pois a solução somente seria viável dentro do contexto das redes móveis disponíveis atualmente, como por exemplo 3G e 4G que são tecnologias já bem estabelecidas nas zonas urbanas.

Os comportamentos da taxa de quadros por segundo (FPS) e latência em diferentes cenários de resolução demonstram variações significativas dependendo do framework utilizado. Usando o YOLO na menor resolução que julgamos viável estar diferenciando objetos na tela, 240p, obtivemos uma média de 2 FPS e 500 ms de delay entre o envio da imagem pela câmera, e a captura desta imagem já processada pelo display. Este nível de desempenho e latência, inviabilizariam o uso do sistema. Entretanto, no caso do SSD, notamos que as taxas de transmissão para resoluções de 240p e 320p são excelentes, chegam ao display do usuário, entre 52 e 44 FPS, o que no contexto da realidade aumentada é próximo do mínimo necessário para uma boa usabilidade. No caso da recepção, o vídeo mantém, em sua maioria, a percepção de continuidade, mas já apresenta alguns sinais de descontinuidade e pequenos saltos entre algumas cenas.

Ao analisar essas resoluções menores, observamos baixas taxas de latência entre a câmera e o servidor, e entre o servidor e o cliente, com valores em torno de 64ms. Como o servidor processa uma imagem de cada vez, a perda na taxa de quadros ocorre devido à velocidade do processamento da imagem e as imagens são perdidas conforme chegam e o servidor já está processando outra imagem. No entanto, ao aumentar a resolução, percebemos que a dificuldade na transmissão da imagem aumenta de maneira que o gargalo deixa de ser o processamento da imagem e começa a ser a conectividade entre os processos e a continuidade

do vídeo é perdida, criando grande saltos entra as imagens, como podemos ver no gráfico 9:

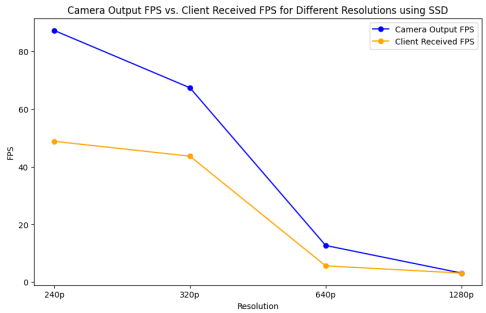

Figura 9: Output de FPS da camera vs FPS recebido pelo cliente usando SSD

O YOLO, por sua vez, apresenta um cenário completamente diferente. O gargalo permanece constante em todos os cenários, sendo o processamento da imagem. A taxa de quadros por segundo (FPS) se mantém em cerca de 3 durante todos os cenários, independentemente da taxa de transmissão da câmera. Esse resultado torna a qualidade de experiência do usuário praticamente impossível, como evidenciado na figura 11:

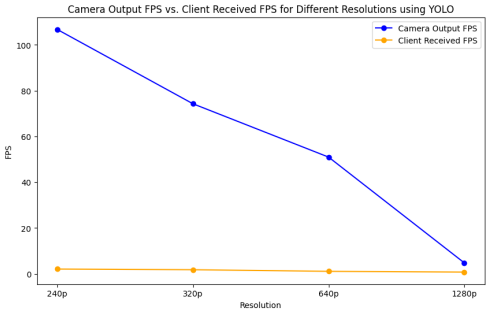

Figura 10: Output de FPS da camera vs FPS recebido pelo cliente usando YOLO

Esse comportamento indica uma limitação significativa do YOLO em termos de processamento em tempo real, comprometendo a eficácia da solução para aplicações que requerem uma transmissão contínua e rápida de dados visuais. Essas observações são cruciais para a compreensão das capacidades e limitações de cada framework em diferentes contextos de uso.

Comparação entre as taxas de transmissão e as resoluções para cada framework

	240p		320p		640p		1280p	
	Taxa de saída	Taxa de chegada	Taxa de saída	Taxa de chegada	Taxa de saída	Taxa de chegada	Taxa de saída	Taxa de chegada
SSD	74fps	52fps	90fps	43fps	13fps	5fps	3fps	3fps
YOLO	105fps	2fps	74fps	2fps	50fps	1fps	5fps	0.6fps

Figura 11: Comparação entre as taxas de transmissão e as resoluções para cada framework

Ao analisar diferentes cenários de resoluções, pudemos também compar diferentes situações de latência, entre as duas soluções. Nas figuras 12 e 13, tempos a latência em

diferentes resoluções, para o YOLO e para o SSD:

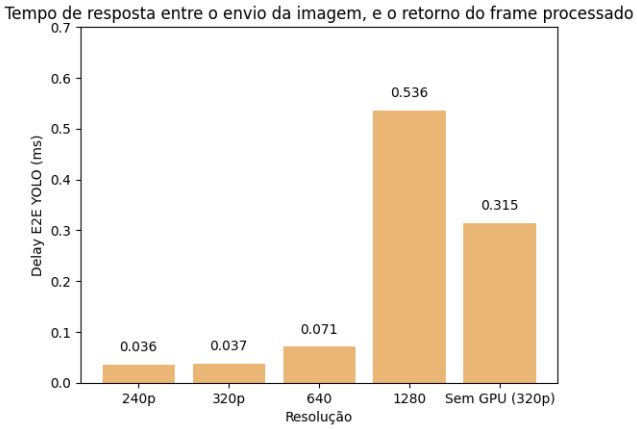

Figura 12: Latências para o usuário no YOLO

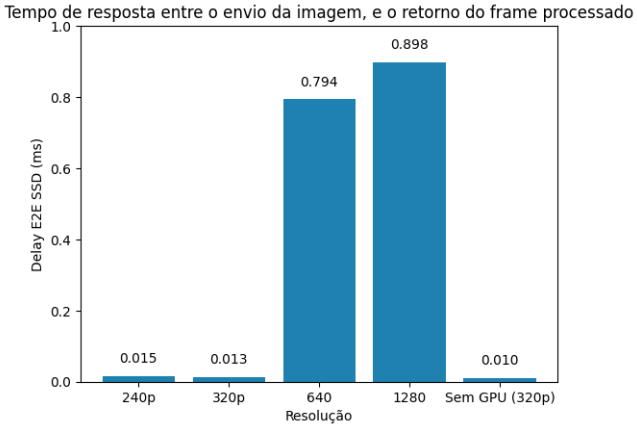

Figura 13: Latências para o usuário no SSD

As situações de latência refletem a mesma realidade que as de desempenho de FPS: Para resoluções pequenas, a solução é sustentável e perfeitamente funcional, entretanto, quando tentamos estender essa resolução para valores de maior qualidade de imagem, rapidamente encontramos uma barreira de desempenho que torna o sistema inviável.

O mesmo padrão também pode ser observado quando olhamos as velocidades de Download e Upload (figura 14) necessárias para o tráfego de dados em tempo real:

Assim como apresentado na figura 14 de taxas de download e upload, pode-se perceber que independente de optar pelo YOLO ou pelo SSD, a solução é custosa em termos de rede. Dessa forma, acreditamos que sua viabilidade será alcançada primeiro em ambientes urbanos devido a chegada do 5G que pode garantir a latência crítica necessária para esse tipo de aplicações.

Comparação entre as taxas de download e upload								
	240p		320p		640p		1280p	
	Upload	Download	Upload	Download	Upload	Download	Upload	Download
SSD	18,01 MB/s	9 MB/s	23,70 MB/s	25,90MB/s	7,82 MB/s	6,66 MB/s	9,6MB/s	8,82 MB/s
YOLO	15,80 MB/s	15,02 MB/s	19,60 MB/s	17,20 MB/s	9,7 MB/s	7,9 MB/s	16,37 MB/s	15,36 MB/s

Figura 14: Tabela velocidade de Download e Upload

5.2 Como a CPU e GPU se comportam em diferentes cenários

Parte da solução e do trabalho que desenvolvemos passava por explorar a necessidade necessidade de processamento das opções disponíveis. Tendo isso em vista, fizemos um valor considerável de experimentações para entender de maneira empírica os trade-offs existentes na escolha de uma tecnologia/característica ou outra. Nessa sub-sessão serão apresentados os resultados obtidos relacionadas ao uso ou não de GPU e qual é o impacto relacionado.

O que ficou evidente na nossa investigação é que o uso de GPU está atrelado a um resultado muito mais eficiente com relação à necessidade de poder de processador. Testamos a solução tanto com YOLO quanto com SSD e o resultado geral foi o mesmo independente do método como pode ser visto nas figuras abaixo 15 e 16

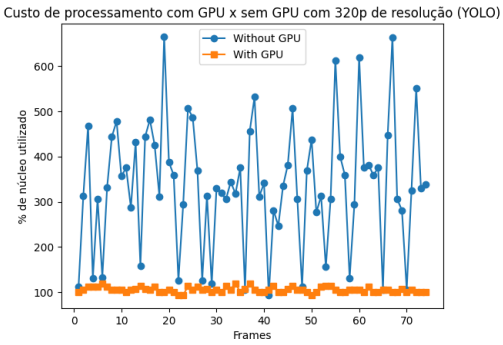

Figura 15: Custo de processamento com GPU x sem GPU com 320p com YOLO

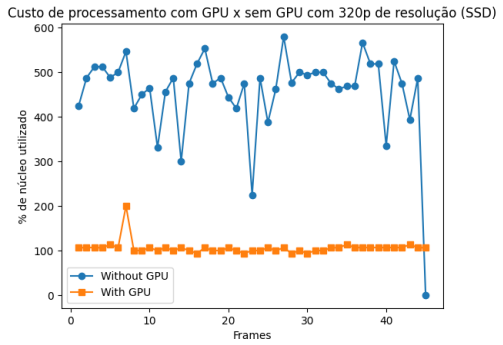

Figura 16: Custo de processamento com GPU x sem GPU com 320p com SSD

Como fica visível nos gráficos, o uso de GPU em conjunto com a CPU mantém os níveis de requerimento de poder computacional muito mais baixo do que quando ela não é utilizada. Apesar de isso já ser esperado, o que nos surpreendeu foi o fato de que a quantidade de GPU requerida para o processamento era inversamente proporcional à qualidade do frame. Ou seja, quanto maior a resolução da imagem, menos se requeria da GPU (fato que aconteceu usando tanto o YOLO quanto o SSD nas figuras 17 e 18).

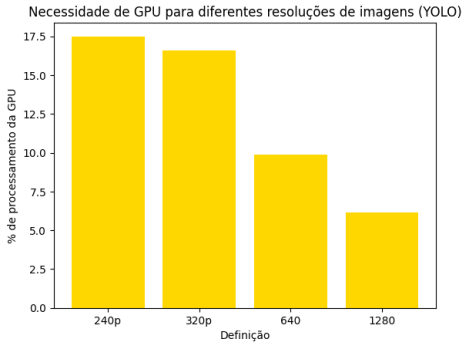

Figura 17: Necessidade de GPU para diferentes soluções de imagens com YOLO

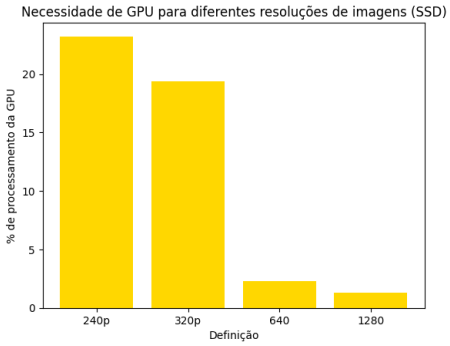

Figura 18: Necessidade de GPU para diferentes soluções de imagens com SSD

Acreditamos que esse fenômeno se dá porque por estarmos tratando frames de maior resolução, estamos processando menos frames e, por isso, o uso da GPU está mais baixo

5.3 Os diferentes resultados apresentados pelo YOLO e pelo SSD

Por fim, agora aprofundando mais nos resultados que obtivemos entre o YOLO e o SSD que foram as duas principais abordagens que seguimos. De maneira geral, o YOLO necessita de um processamento ligeiramente maior que o SSD, o que fica visivel em diferentes definições de frames na figure 19.

Além disso ele possui um aumento mais perceptível da necessidade de processamento à medida que a resolução do frame processado aumenta, enquanto o SSD possui um comportamento mais linear

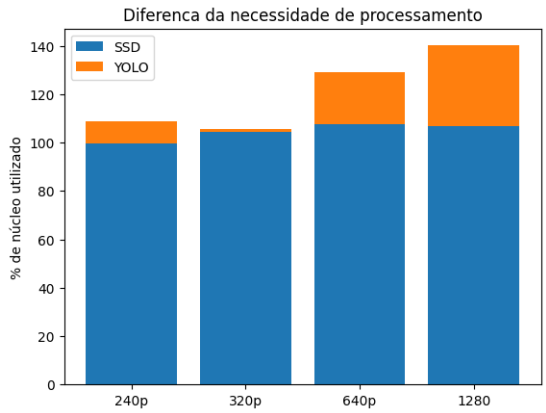

Figura 19: Comparação da necessidade de processamento

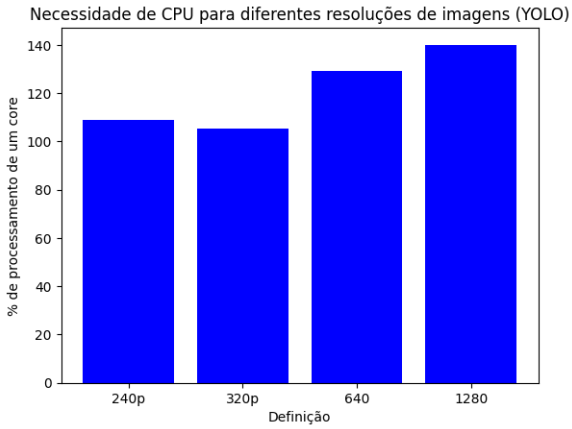

Figura 20: Necessidade de CPU para diferentes resoluções de imagens com YOLO

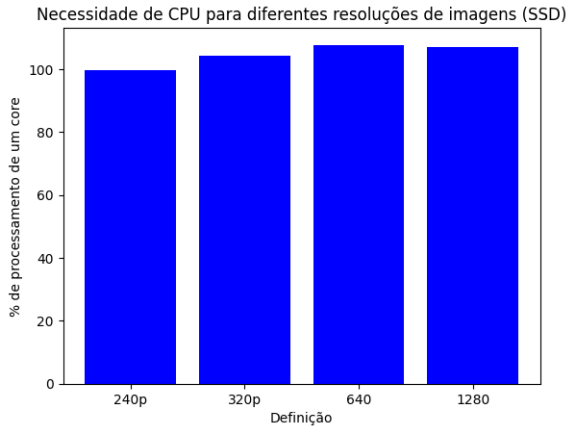

Figura 21: Necessidade de CPU para diferentes resoluções de imagens com SSD

Entretanto, essa velocidade de processamento vem acompanhada de um custo para a eficácia na identificação de objetos. Em diversos momentos nos cenários com o SSD, o framework não consegue identificar certos objetos, como alguns carros, dependendo do angulo ou profundidade do objeto na foto, ou falha ao identificar erroneamente alguns objetos, como um prédio sendo interpretado como um trem, ou uma calçada sendo confundida com uma mesa de jantar. Tais previsões que fogem do nosso cenário de uso específico, como pode ser observado na imagem 22.

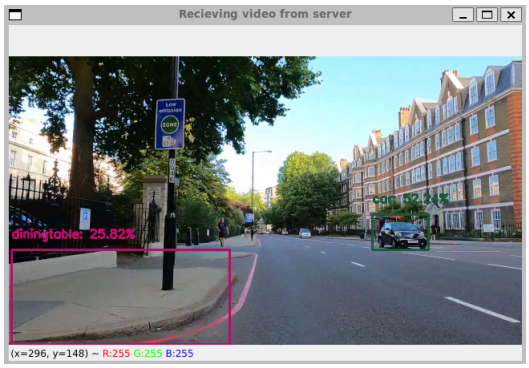

Figura 22: SSD reconhecendo erroneamente uma mesa

Por outro lado, o YOLO apresenta uma eficácia notavelmente superior, cometendo menos erros e com falhas mais alinhadas ao cenário real. Por exemplo, um carro grande pode ser confundido inicialmente com um caminhão, ou um ônibus pode começar a ser identificado como um caminhão até se aproximar, momento em que o YOLO corrige a predição para identificar corretamente como um ônibus. Além disso, o YOLO demonstra uma capacidade impressionante ao identificar com precisão objetos distantes e em diversos ângulos, tornando-o um framework mais robusto e confiável no sentido de precisão e acurácia como vemos na imagem 23.

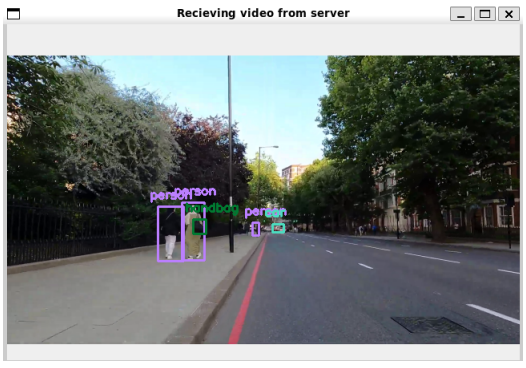

Figura 23: Exemplo de reconhecimento de objetos com YOLO

6 Conclusões e Trabalhos Futuros

Com base no exposto, acreditamos alcançamos nosso objetivo, concentrando-nos na identificação dos requisitos de memória, processamento e rede necessários para sustentar uma aplicação de vídeo em tempo real voltada para a identificação de imagens. Dessa maneira, evidenciamos a viabilidade prática desse cenário na vida cotidiana das pessoas. Contudo, há ainda vasto espaço para expandir e aprofundar a discussão em áreas subsequentes.

Para iniciar uma implementação com óculos de realidade mista ou aumentada que permita o uso das imagens da câmera por meio de APIs e kits de desenvolvimento, é importante observar que, atualmente, há limitações significativas nessa área. Infelizmente, a maioria dos óculos não habilita o acesso à câmera para fins de desenvolvimento, principalmente por questões de segurança. Exemplos notáveis incluem os Oculus Quest 2, Quest 2 Pro e Oculus Quest 3 da Meta, assim como o HoloLens da Microsoft. Nos óculos da NReal, embora haja a possibilidade de utilizar informações provenientes da Unidade de Medição Inercial (IMU), não há acesso direto à câmera.

Essas restrições impactam o desenvolvimento de soluções de realidade aumentada que dependem do processamento de imagens provenientes das câmeras embutidas nos óculos. É fundamental estar ciente dessas limitações ao considerar implementações práticas em dispositivos de realidade mista ou aumentada.

Outra alternativa promissora seria explorar o uso de protocolos mais modernos e especializados em transmissão de vídeo em tempo real, como o Real-Time Streaming Protocol (RTSP), o HTTP Live Streaming (HLS) e o Dynamic Adaptive Streaming over HTTP (DASH). A adoção desses protocolos tem o potencial de otimizar a transferência de dados e reduzir os recursos necessários para a utilização do sistema. A escolha entre esses protocolos dependerá das características específicas do cenário de implementação, tais como requisitos de latência, qualidade de vídeo e adaptabilidade às condições de rede. Essa exploração pode fornecer insights valiosos sobre como aprimorar ainda mais a eficiência do sistema.

Certamente, a consideração do handover do processamento e as mudanças entre cloudlets em um contexto de computação em névoa durante o deslocamento urbano apresenta um campo fértil para pesquisa e desenvolvimento. Explorar como esses processos podem ser otimizados para garantir a eficiência e a continuidade da experiência do usuário em ambientes urbanos dinâmicos é um desafio interessante. A integração de tecnologias Extended Reality (XR), como o video-streaming e as trocas de informações em rede, adiciona camadas adicionais de complexidade e oportunidades.

Dentro desse tema, considerações sobre a latência, a adaptabilidade dos serviços em nuvem distribuída, a segurança da comunicação e a eficiência na transição entre diferentes cloudlets são aspectos cruciais a serem explorados. Além disso, a interação entre dispositivos de realidade mista ou aumentada e o ambiente urbano oferece um vasto campo para a inovação, com potencial para melhorar significativamente a experiência do usuário e a segurança em deslocamentos urbanos.

Com uma abordagem interdisciplinar, que englobe conceitos de computação em névoa, redes móveis, processamento de vídeo em tempo real e tecnologias XR, é possível abrir novas perspectivas e oportunidades para aprimorar ainda mais a implementação de soluções práticas e eficazes nesse cenário dinâmico e desafiador.

Esperamos que esse trabalho posso ser um marco que apresenta algumas das bases que podem ser exploradas para que a tecnologia siga caminhando rumo ao progresso em XR.

Referências

[1] Folha de São Paulo. *Cresce o número de ciclistas mortos no trânsito de São Paulo*. URL: `https://www1.folha.uol.com.br/blogs/ciclocosmo/2023/11/cresce-o-numero-de-ciclistas-mortos-no-transito-de-sao-paulo.shtml`. (acesso: 15.12.2023).

[2] Ericson. *Technology Review Magazine 2023 – Spotlight on extended reality*. URL: `https://www.ericsson.com/49a623/assets/local/reports-papers/ericsson-technology-review/docs/2023/spotlight-on-xr.pdf`. (acesso: 15.12.2023).

[3] J. Redmon e tal. *You Only Look Once: Unified, Real-Time Object Detection*. (2015).

[4] Wey Lui e tal. *SSD: Single Shot MultiBox Detector*. Addison-Wesley (2015).

[5] Google. *Documentação do Cloud Vision*. URL: `https://cloud.google.com/vision/docs?hl=pt-br`. (acesso: 15.12.2023).

[6] João Machado. *Sistema de detecção de sinais de tráfego luminosos (semáforos) voltados para sistemas ADAS e veículos autônomos*. USP (2020).

[7] Flask. *User Guide*. URL: `https://flask.palletsprojects.com/en/3.0.x/`. (acesso: 15.12.2023).

[8] C. Puliafito e tal. *MobfogSim: Simulation of mobility and migration for computação em névoa*. (2019).

[9] Docker. *Overview of Docker Desktop*. URL: `https://docs.docker.com/desktop/`. (acesso: 15.12.2023).

7 Appendix

Repositório
Drive com análises