UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

KARLLA BIANCA CHAVES RODRIGUES

Caracterização de Funções de Realidade Aumentada usando Computação de Borda

Goiânia
2025

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES

E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses e Dissertações disponibilizado na BDTD/UFG é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do material bibliográfico

[X] Dissertação [ ] Tese [ ] Outro*:_____

*No caso de mestrado/doutorado profissional, indique o formato do Trabalho de Conclusão de Curso, permitido no documento de área, correspondente ao programa de pós-graduação, orientado pela legislação vigente da CAPES.

Exemplos: Estudo de caso ou Revisão sistemática ou outros formatos.

2. Nome completo do autor

Karlla Bianca Chaves Rodrigues

3. Título do trabalho

Caracterização de Funções de Realidade Aumentada usando Computação de Borda

4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:
a) consulta ao(à) autor(a) e ao(à) orientador(a);
b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.
O documento não será disponibilizado durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

Obs. Este termo deverá ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Sand Luz Correa**, **Professora do Magistério Superior**, em 09/06/2025, às 16:19, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Karlla Bianca Chaves Rodrigues**, **Discente**, em 11/06/2025, às 14:04, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5415679** e o código CRC **312FB343**.

Referência: Processo nº 23070.021502/2025-11 SEI nº 5415679

KARLLA BIANCA CHAVES RODRIGUES

Caracterização de Funções de Realidade Aumentada usando Computação de Borda

Dissertação apresentada ao Programa de Pós–Graduação em Ciência da Computação, do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestra em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientadora: Profa. Sand Luz Corrêa

Co-Orientador: Prof. Kleber Vieira Cardoso

Goiânia
2025

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Rodrigues, Karlla Bianca Chaves
Caracterização de Funções de Realidade Aumentada usando Computação de Borda [manuscrito] / Karlla Bianca Chaves Rodrigues. - 2025.
LXXX, 80 f.

Orientador: Profa. Dra. Sand Luz Corrêa; co-orientador Dr. Kleber Vieira Cardoso.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2025.

Inclui siglas, gráfico, tabelas, lista de figuras, lista de tabelas.

1. Realidade estendida. 2. Computação de borda. 3. Offloading. 4. SLAM. 5. Detecção de objetos. I. Corrêa, Sand Luz, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE DISSERTAÇÃO

Ata nº **13** da sessão de Defesa de Dissertação de **Karlla Bianca Chaves Rodrigues**, que confere o título de Mestre em Ciência da Computação, na área de concentração em Ciência da Computação.

Aos dois dias do mês de junho de dois mil e vinte e cinco, a partir das catorze horas, via sistema de webconferência da RNP, realizou-se a sessão pública de Defesa de Dissertação intitulada **"Caracterização de Funções de Realidade Aumentada usando Computação de Borda"**. Os trabalhos foram instalados pela Orientadora, Professora Doutora Sand Luz Correa (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professor Doutor Kleber Vieira Cardoso (INF/UFG), coorientador; Professor Doutor Cristiano Bonato Both (Unisinos), membro titular externo; Professora Doutora Luciana de Oliveira Berretta (INF/UFG), membra titular interna. A realização da banca ocorreu por meio de videoconferência. Durante a arguição os membros da banca não fizeram sugestão de alteração do título do trabalho. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Dissertação, tendo sido a candidata **aprovada** pelos seus membros. Proclamados os resultados pela Professora Doutora Sand Luz Correa, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora, aos dois dias do mês de junho de dois mil e vinte e cinco.

TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **Kleber Vieira Cardoso**, **Professor do Magistério Superior**, em 02/06/2025, às 15:39, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Sand Luz Correa**, **Professora do Magistério Superior**, em 02/06/2025, às 15:39, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Luciana De Oliveira Berretta**, **Professora do Magistério Superior**, em 02/06/2025, às 15:39, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Karlla Bianca Chaves Rodrigues**, **Discente**, em 02/06/2025, às 16:07, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **CRISTIANO BONATO BOTH**, **Usuário Externo**, em 02/06/2025, às 18:30, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Referência: Processo nº 23070.021502/2025-11 SEI nº 5404866

Dedico este trabalho a todos que me acompanharam até aqui e que não me deixaram desistir.

Agradecimentos

Começo agradecendo, de todo o coração, à minha orientadora, Professora Sand Corrêa. Mesmo nos momentos mais difíceis, você me ensinou não apenas a ser uma cientista, mas a ser uma mulher cientista. Desde a graduação, tem apontado meus erros com carinho e sabedoria — e isso não apenas me transformou, como continua moldando a profissional que desejo me tornar. Agradeço também a todas as professoras e professores que contribuíram para que eu chegasse até aqui. Cada ensinamento deixou uma marca essencial nessa caminhada.

Sou profundamente grata à Universidade Federal de Goiás, que tem sido meu lar há dez anos. É nela que encontrei provas vivas de como o ensino público neste país tem o poder de transformar vidas. Toda a minha trajetória acadêmica é pensada com o compromisso de retribuir tudo o que o ensino público me proporcionou — com responsabilidade, dedicação e propósito.

Aos meus pais, meu mais sincero agradecimento. Com muito esforço, vocês sempre buscaram oferecer a melhor educação possível às filhas — e puderam ver ambas se formarem em uma universidade federal. O esforço de vocês não foi, e nunca será, em vão. Tudo o que conquistei até aqui carrega a marca da dedicação, dos sacrifícios e do amor que vocês sempre nos deram.

Por fim, mas não menos importante, dedico este trabalho ao meu companheiro, Marcos. Você esteve presente em todos os momentos — nos bons e nos difíceis — sempre oferecendo críticas construtivas, sugestões valiosas e, acima de tudo, incentivo constante para que eu me aperfeiçoasse e jamais desistisse dos meus sonhos. Saiba que este trabalho também é fruto do seu esforço, e que seguimos juntos, lado a lado, para conquistar ainda mais.

... a computação é muito importante para ser deixada aos homens.

Karen Spärck Jones,
Cientista da computação britânica.

Resumo

Rodrigues, Karlla Bianca Chaves. **Caracterização de Funções de Realidade Aumentada usando Computação de Borda**. Goiânia, 2025. 83p. Dissertação de Mestrado. Programa de Pós-Graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás.

O crescimento no uso de aplicações imersivas, como realidade aumentada, realidade virtual e realidade mista, tem gerado uma demanda crescente por recursos computacionais mais eficientes para o processamento de dados em tempo real. Embora a computação em nuvem ofereça capacidade para lidar com tais demandas, ela frequentemente introduz latências elevadas, comprometendo a experiência do usuário. A computação de borda surge como uma alternativa promissora ao aproximar os recursos computacionais dos dispositivos finais, reduzindo a latência, aprimorando a imersão e viabilizando a execução dessas aplicações em dispositivos móveis. Compreender os componentes funcionais dessas aplicações e seus respectivos perfis de desempenho é essencial para um descarregamento (*offloading*) eficiente entre dispositivos móveis e servidores de borda. Este trabalho tem como objetivo caracterizar duas tarefas centrais em aplicações de realidade aumentada móvel: a Localização e Mapeamento Simultâneo (SLAM) e a detecção de objetos. Para isso, foi utilizado o protótipo MR-Leo, que integra o ORB-SLAM2 e incorpora uma nova funcionalidade de detecção de objetos baseada na arquitetura YOLO. A pesquisa avalia o desempenho dessas tarefas em diferentes configurações de hardware, considerando execuções em CPU e GPU. Os resultados mostraram que, embora o SLAM seja computacionalmente intensivo, ele apresenta desempenho aceitável em arquiteturas baseadas em CPU. Em contrapartida, a detecção de objetos requer paralelismo massivo para um desempenho satisfatório, sendo fortemente dependente do uso de GPU. Com base na caracterização das tarefas, foi desenvolvido um modelo estatístico de carga de trabalho, com o objetivo de viabilizar a criação de geradores de carga capazes de emular o comportamento de aplicações de realidade aumentada sob diferentes cenários e arquiteturas computacionais.

Palavras–chave

Realidade Estendida, Computação de Borda, *Offloading*, SLAM, Detecção de Objetos

Abstract

Rodrigues, Karlla Bianca Chaves. **Characterization of Augmented Reality Functions using Edge Computing**. Goiânia, 2025. 83p. MSc. Dissertation. Programa de Pós-Graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás.

The growing use of immersive applications—such as augmented reality, virtual reality, and mixed reality—has led to increasing demand for more efficient computational resources capable of real-time data processing. While cloud computing can meet these demands, it often introduces high latency, negatively affecting the user experience. Edge computing emerges as a promising alternative by bringing computational resources closer to end-user devices, reducing latency, enhancing immersion, and enabling the deployment of such applications on mobile devices. Understanding the functional components of these applications and their performance profiles is essential for efficient offloading between mobile devices and edge servers. This work aims to characterize two core tasks in mobile augmented reality applications: Simultaneous Localization and Mapping (SLAM) and object detection. To this end, the MR-Leo prototype was used, integrating ORB-SLAM2 and incorporating a new object detection functionality based on the YOLO architecture. The research evaluates the performance of these tasks under different hardware configurations, considering execution on both CPUs and GPUs. The results show that although SLAM is computationally intensive, it performs acceptably on CPU-based architectures. In contrast, object detection requires massive parallelism for satisfactory performance and is heavily dependent on GPU usage. Based on the task characterization, a statistical workload model was developed to support the creation of workload generators capable of emulating the behavior of augmented reality applications under different computational architectures and scenarios.

Keywords

Extended Reality, Edge Computing, Offloading, SLAM, Object Detection

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

3GPP *3rd Generation Partnership Project*

5G Quinta Geração de Rede Móvel Celular

6-DoF *Six Degrees of Freedom*

AMF *Access and Mobility Management Function*

APIs *Application Programming Interface*

AR *Augmented Reality*

AUSF *Authentication Server Function*

AVR *Augmented Virtuality*

CDF *Cumulative Distribution Function*

CNN *Convolutional Neural Network*

CPU *Central Processing Unit*

CUDA *Compute Unified Device Architecture*

CU *Central Unit*

DTAM *Dense Tracking and Mapping in Real-Time*

DU *Distributed Unit*

eMBB *Enhanced Mobile Broadband*

ETSI *European Telecommunications Standards Institute*

FPS *Frames Per Second*

FRTT *Frame Round Trip Time*

gNBs *Next-Generation Base Station*

GPU *Graphics Processing Unit*

GSMA *Global System for Mobile Communications Association*

HMD *Head-Mounted Display*

ISG *Industry Specification Group*

LiDAR *Light Detection and Ranging*

MAR *Mobile Augmented Reality*

MEC *Multi-access Edge Computing*

MR *Mixed Reality*

NRF *Network Repository Function*

OPG *Operator Plataform Group*

ORB *Oriented FAST and Rotated BRIEF*

PCF *Policy Control Function*

QoE *Quality of Experience*

QoS *Quality of Service*

RAN *Radio Access Network*

RU *Radio Unit*

SLAM *Simultaneous Localization and Mapping*

SMF *Session Management Function*

SVO *Semi-direct Visual Odometry*

TCP *Transmission Control Protocol*

TIC Tecnologia da Informação e Comunicação

UDM *Unified Data Managemen*

UDP *User Datagram Protocol*

UE *User Equipement*

UPF *User Plane Function*

URLLC *Ultra Reliable Low Latency Communications*

VRAM *Video Random Access Memory*

VR *Virtual Reality*

XR *Extended Reality*

YOLO *Yolo Only Look Once*

CAPÍTULO 1

Introdução

A realidade aumentada, ou *Augmented Reality* (AR), onde o mundo físico é enriquecido por objetos virtuais, é um campo da Ciência da Computação que tem despertado muito interesse nas últimas décadas [Carmigniani et al. 2011], principalmente devido à melhorias no poder de processamento em dispositivos móveis e às novas tecnologias de comunicação sem fio, especialmente as capazes de oferecer serviços com latência ultra baixa e com taxas de dados extremamente altas [Chatzopoulos et al. 2017]. Para ilustrar, em 2017, tanto o Google quanto a Apple lançaram, respectivamente, o ARCore[1] e o ARKit[2], arcabouços de desenvolvimento de aplicações AR para as plataformas Android e iOS. Recentemente, consumidores passaram a ter uma variedade de opções de dispositivos imersivos portáteis, como óculos de AR ou capacetes de realidade virtual, também conhecidos como *Head-Mounted Display* (HMD). Esses dispositivos, juntamente com novas aplicações de Realidade Aumentada Móvel, ou *Mobile Augmented Reality* (MAR), estão promovendo novas formas de educação, aprimorando o ambiente de trabalho online, melhorando a interface de interação em sistemas de saúde, além de outros casos de uso inovadores [Cao et al. 2023, Moro et al. 2021, Siriwardhana et al. 2021].

De fato, aplicações MAR não apenas combinam conteúdo real e virtual, mas também requerem que ambos funcionem de maneira sincronizada. Isso geralmente exige que o ambiente real seja analisado com a maior precisão possível e em tempo real, demandando múltiplos algoritmos computacionalmente intensivos, como SLAM, reconstrução 3D, compreensão semântica e renderização de quadros. Embora existam algumas implementações de última geração em tempo real desses algoritmos [Mur-Artal e Tardós 2017, Redmon et al. 2016, Dai et al. 2017], elas exigem hardware moderno, que geralmente não é portátil. Além disso, para uma boa experiência do usuário, o conteúdo de AR deve ser renderizado com uma latência de movimento para fóton (*motion-to-photon delay*)[3] abaixo de 100 ms para *smartphone* [Toczé et al. 2019] e

[1] https://developers.google.com/ar

[2] https://developer.apple.com/augmented-reality/arkit/

[3] tempo que leva desde o momento em que o usuário faz um movimento físico (ex: mover a cabeça, mexer o controle, girar a câmera) até o momento em que a consequência visual desse movimento aparece

15 ms para HMD [Morín, Pérez e Armada 2022].

A demanda por hardware avançado juntamente com requisitos restritos de latência justificam a ideia de descarregar (*offload*) tarefas MAR computacionalmente intensivas para infraestruturas remotas, porém localizadas o mais próximo possível do dispositivo do usuário, ou *User Equipement* (UE). Nesse sentido, computação de borda (*edge computing*) [Cruz, Achir e Viana 2022] desempenha um papel importante por trazer maior poder computacional para mais perto dos usuários. Para o usuário final, o descarregamento de tarefas para a borda significa economia de energia e redução de aquecimento no dispositivo. O descarregamento de tarefas críticas para uma infraestrutura de borda pode também aumentar a qualidade da experiência do usuário, fornecendo às tarefas acesso a aceleradores de hardware como *Central Processing Unit* (CPU)s. Ao mesmo tempo, o descarregamento pode trazer benefícios aos desenvolvedores de MAR, que podem aproveitar um serviço de *offloading* para tratar questões de privacidade e otimizar o desempenho com base nos requisitos do usuário e em fatores contextuais [Yadhav et al. 2024].

Nos últimos anos, diversos trabalhos têm investigado como o paradigma de computação de borda pode beneficiar as aplicações MAR, com ênfase na redução de latência e no alívio do processamento em dispositivos móveis. Estudos como os de [Chen et al. 2017], [Zhang et al. 2022] e [Toczé et al. 2019] demonstram os ganhos obtidos com o descarregamento de tarefas como detecção de objetos e SLAM para servidores de borda. No entanto, para que o descarregamento de tarefas seja eficaz, é fundamental compreender o perfil de consumo de recursos de cada tarefa em diferentes arquiteturas de hardware. Tarefas como SLAM e detecção de objetos apresentam comportamentos computacionais distintos dependendo da plataforma em que são executadas, podendo variar significativamente em termos de uso de CPU, *Graphics Processing Unit* (GPU), memória e largura de banda. Essa variabilidade torna essencial a realização de análises de desempenho que considerem não apenas o tempo de execução e o consumo energético, mas também a compatibilidade com aceleradores específicos, como unidades de processamento gráfico ou motores de inferência neural. Um mapeamento preciso desses perfis permite que mecanismos de descarregamento tomem decisões sobre quando, onde e como executar uma determinada tarefa, maximizando o desempenho e a eficiência energética, ao mesmo tempo em que restrições contextuais sejam respeitadas, como latência e conectividade de rede. Assim, o conhecimento detalhado sobre o comportamento das tarefas em diferentes arquiteturas se torna um componente estratégico no desenvolvimento de aplicações MAR adaptativas e eficientes.

Apesar de importante, na literatura, apenas o trabalho em [Toczé et al. 2020]

na tela.

provê um modelo de carga para aplicações MAR extraída de um protótipo real, denominado MR-Leo. Ainda assim, apenas a tarefa de SLAM é representada neste modelo e apenas a carga de trabalho de CPU é caracterizada. De fato, a existência de poucas plataformas de realidade aumentada de código aberto contribuem para essa situação. Tais plataformas envolvem diversas tecnologias, o que as tornam difíceis de configurar para diferentes tarefas, arquiteturas e cenários de carga. Essa lacuna, por outro lado, dificulta a investigação de estratégias de alocação de recursos e orquestração de serviços em computação de borda, essenciais para melhorar o *Quality of Experience* (QoE) dos usuários MAR.

Para preencher essa lacuna, neste trabalho, pretendemos responder às seguintes perguntas de pesquisas:

- Quais são as principais tarefas que compõem um fluxo típico de aplicações MAR?
- Existem implementações reais de código aberto representativas dessas tarefas?
- Em termos de latência fim-a-fim e vazão, como a computação de borda pode impactar no desempenho dessas tarefas, considerando diferentes arquiteturas de hardware (CPU e GPU)?
- Qual é o perfil de consumo de recursos dessas tarefas em diferentes arquiteturas de hardware?
- Como derivar modelos de cargas de trabalho para aplicações MAR levando em consideração a infraestrutura de borda que se mostrou mais adequada para a execução de cada tarefa?

Para responder essas perguntas, a maior parte desta dissertação envolveu pesquisa bibliográfica, implementação de protótipos e coleta e análise de dados. Particularmente, partindo do trabalho apresentado em [Toczé et al. 2019], estendemos o MR-Leo, adicionado uma tarefa de SLAM acelerada por GPU. Adicionamos também ao protótipo uma tarefa de detecção de objetos que pode ser executada tanto em CPU quanto em GPU. Em seguida, usando as implementações desenvolvidas, coletamos e analisamos o desempenho das tarefas de SLAM e detecção de objetos em duas infraestruturas de borda distintas: básica, apenas com CPU; e avançada, com CPU e GPU. Por fim, para cada tarefa, derivamos um modelo de carga de trabalho levando em consideração a infraestrutura que se mostrou mais adequada para a sua execução.

A seguir, apresentamos a organização do trabalho.

- O Capítulo 2 introduz os principais conceitos e fundamentos necessários para a realização deste trabalho. Este capítulo discute também os principais trabalhos relacionados e como este trabalho se distingue da literatura existente.
- O Capítulo 3 apresenta os principais componentes do MR-Leo e discute como o protótipo foi estendido para incluir aceleração em GPU e a tarefa de detecção de

objetos. São apresentadas também as principais ferramentas, bibliotecas e plataformas utilizadas nesta implementação.

- Capítulo 4 apresenta e discute a avaliação de desempenho das tarefas de SLAM e de detecção de objetos em diferentes infraestruturas de borda. Este capítulo discute também a metodologia utilizada para coletar e analisar os dados obtidos durante a avaliação de desempenho, bem como a metodologia usada para extrair o modelo de carga das tarefas.
- Por fim, o Capítulo 5 discute as principais conclusões obtidas neste trabalho e aponta para perspectivas futuras.

CAPÍTULO 2

Fundamentos e Trabalhos Relacionados

Neste capítulo serão discutidos os principais conceitos que fundamentam este trabalho. A Seção 2.1 introduz alguns conceitos importantes para a compreensão sobre computação de borda e realidade aumentada. Em seguida, a Seção 2.2 discute os trabalho relacionados que tratam sobre o descarregamento de realidade mista para a borda e também sobre caracterização de carga de trabalho de realidade mista.

2.1 Fundamentos

2.1.1 Introdução à computação de borda

Usuários de dispositivos móveis demandam capacidades de processamento cada vez maiores. No entanto, esses dispositivos enfrentam limitações significativas de recursos computacionais e energéticos [Cruz, Achir e Viana 2022]. Como consequência, as aplicações móveis dependem cada vez mais do suporte de infraestruturas externas, como a computação em nuvem. Um exemplo adicional de cenário que se beneficia desse suporte é a Internet das Coisas (IoT), inicialmente proposta para a gestão de cadeias de suprimento [Gubbi et al. 2013], mas que passou a gerar um volume massivo de dados. Inicialmente, o processamento dessas informações foi direcionado à nuvem, devido à sua capacidade de lidar com grandes volumes de dados. Contudo, a nuvem geralmente não consegue atender as aplicações sensíveis à latência, *Virtual Reality* (VR) e AR, onde atrasos na comunicação podem comprometer significativamente a qualidade da experiência dos usuários [Ren et al. 2019].

Uma solução para fornecer aos usuários de aplicações móveis a QoE requerida é oferecer recursos semelhantes aos da nuvem, porém localizados na borda da rede [Satyanarayanan 2017]. Para os usuários, os recursos de borda estão mais próximos que os recursos da nuvem. Portanto, os dispositivos móveis podem consumir recursos de borda (ou seja, de processamento e de armazenamento) com uma latência de comunicação menor que a oferecida pela nuvem. Denomina-se computação de borda (*edge computing*)

um sistema próximo dos usuários móveis e capaz de executar aplicações na borda da rede [Khan et al. 2019].

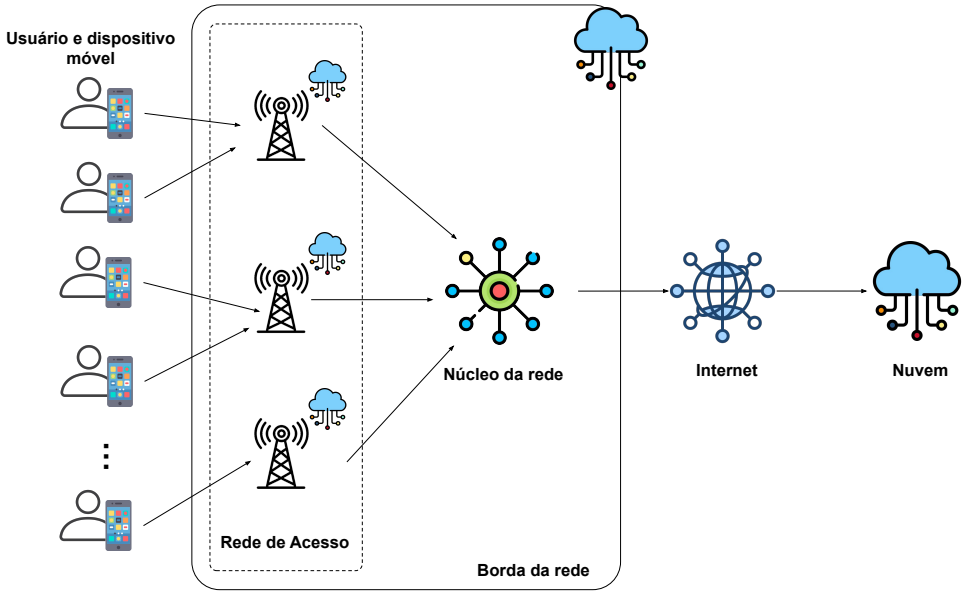


Figura 2.1: Arquitetura genérica para computação de borda. Adaptado de [Cruz, Achir e Viana 2022].

De fato, como ilustrado na Figura 2.1, o caso de uso típico da computação de borda são as aplicações sensíveis à latência. Como os dispositivos móveis possuem restrições de processamento, armazenamento e de energia, eles precisam "descarregar" tarefas computacionalmente intensivas para servidores externos. Quando uma infraestrutura de borda não está disponível, recursos da nuvem são utilizados, fazendo com que a requisição do dispositivo móvel atravesse diversas redes até chegar no servidor destino. Neste último, a requisição é processada e a resposta é enviada de volta para o dispositivo móvel. A latência fim-a-fim é, portanto, o tempo decorrido desde o momento em que o dispositivo faz uma requisição para a nuvem até o momento em que ele recebe a resposta da requisição. Neste cenário, a latência de comunicação e, portanto, a latência fim-a-fim, pode ser muito alta e degradar, ou mesmo inviabilizar, uma aplicação que possui requisitos estritos de latência.

Quando uma infraestrutura de borda está disponível, a requisição do dispositivo móvel é atendida por um servidor de borda, o qual pode estar localizado na rede de acesso ou na rede de núcleo da operadora de celular. Como ilustrado na Figura 2.1, requisições enviadas para essas redes não atingem a Internet. Portanto, a infraestrutura de borda provê uma latência de comunicação menor que a nuvem. Além das considerações de latência, a descarga para a borda da rede é também preferível do ponto de vista energético, uma vez que as redes de telecomunicações têm a maior parcela do

consumo geral de energia do setor de Tecnologia da Informação e Comunicação (TIC) [Global System for Mobile Communications Association (GSMA) 2019]. Como a computação de borda consiste em um tráfego local, ela pode oferecer maior eficiência energética ao evitar o uso de enlaces de transporte [Ahvar, Orgerie e Lebre 2022].

A Figura 2.1 mostra uma arquitetura típica de uma rede móvel celular. Contudo, a computação de borda pode ser usada com outras tecnologias de rede, como o WiFi. No entanto, desde a Quinta Geração de Rede Móvel Celular (5G), as redes móveis celulares oferecem diversas funcionalidades que facilitam a implantação e uso de uma infraestrutura de borda [Hassan, Yau e Wu 2019], como discutido a seguir.

2.1.2 O sistema 5G e a computação de borda

Embora a quinta geração de redes móveis (5G) introduza transformações significativas em termos de requisitos, casos de uso, paradigmas e tecnologias, sua estrutura lógica mantém a divisão tradicional em três domínios principais: rede de acesso, ou *Radio Access Network* (RAN), rede de núcleo e rede de transporte [3GPP 2020]. A RAN é a responsável por conectar os UEs à rede da operadora e é composta por estações base denominadas *Next-Generation Base Station* (gNBs). Para explorar os benefícios da virtualização, essas estações podem ser logicamente desagregadas em três entidades funcionais, que podem ser vistas na Figura 2.2: *Radio Unit* (RU), *Distributed Unit* (DU) e *Central Unit* (CU), conforme diferentes tipos de divisão funcional (*splits*) definidos pelo órgão padronizador *3rd Generation Partnership Project* (3GPP) [Larsen, Checko e Christiansen 2019].

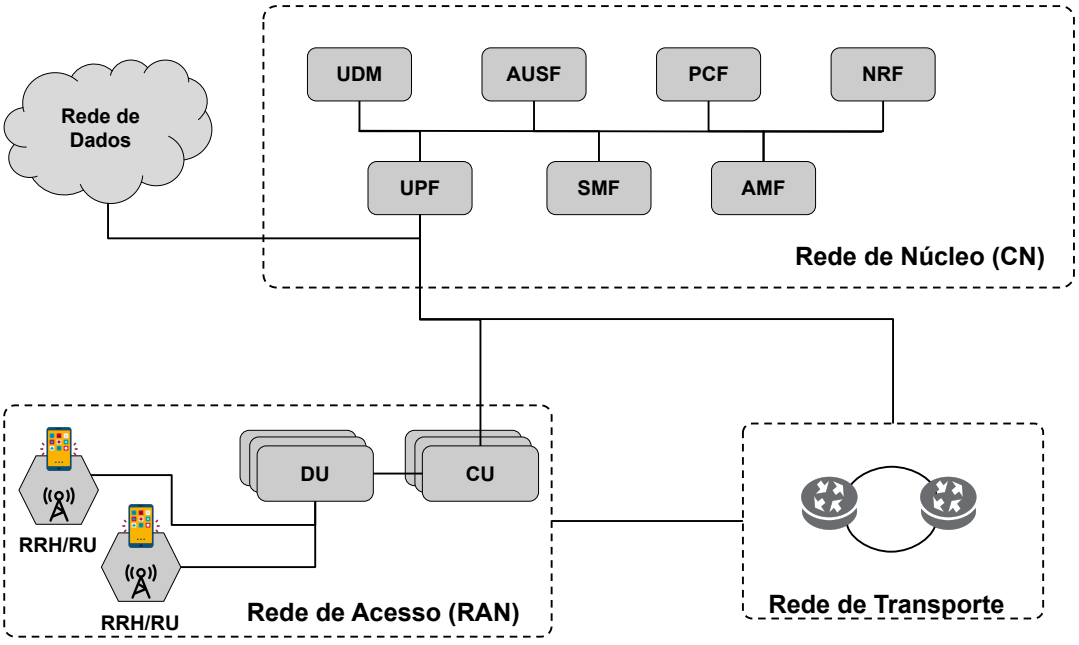


Figura 2.2: Arquitetura básica do sistema 5G.

A rede de núcleo, por sua vez, foi projetada de forma modular para permitir a separação entre o plano de controle e o plano de usuário. Entre as principais funções do plano de controle destacam-se a *Access and Mobility Management Function* (AMF), responsável por registrar, autenticar e gerenciar a mobilidade dos UEs; a *Session Management Function* (SMF), que gerencia sessões de dados; a *Unified Data Managemen* (UDM) e a *Authentication Server Function* (AUSF), responsáveis por dados e autenticação; a *Policy Control Function* (PCF), que define políticas de acesso e mobilidade; e a *Network Repository Function* (NRF), que permite a descoberta de funções de rede. Complementando essa arquitetura, a *User Plane Function* (UPF) atua no plano de dados, processando e encaminhando o tráfego dos usuários. Por fim, a rede de transporte fornece conectividade fim-a-fim, ou seja da RU até a rede de dados.

O 5G oferece diversos serviços que facilitam a implantação da computação de borda, como o acesso local à rede de dados, a continuidade de sessões e serviços, o roteamento otimizado de tráfego, o suporte a redes de dados locais e o provisionamento de parâmetros externos [Kim et al. 2023].

Ao permitir que o tráfego de dados de uma aplicação seja direcionado para uma rede de dados local, a infraestrutura 5G viabiliza o envio das informações do dispositivo do usuário para servidores de borda geograficamente próximos, reduzindo significativamente a latência e possibilitando o processamento em tempo real — um requisito essencial para aplicações de AR. Além disso, o 5G implementa mecanismos que asseguram a continuidade das sessões mesmo com a movimentação física do usuário, permitindo que os serviços executados na borda mantenham-se operacionais durante a mobilidade, sem comprometer a imersão da experiência.

Outro recurso importante das redes 5G é o roteamento otimizado de tráfego, que possibilita que a aplicação solicite o redirecionamento dinâmico de seus dados para uma rede local específica. Esse maior controle por parte da aplicação favorece a tomada de decisões adaptativas, como o redirecionamento de tráfego em situações de congestionamento. A rede pode ser configurada para manter o tráfego restrito a uma determinada área, viabilizando a implementação de serviços locais. Além disso, o 5G permite que terceiros configurem diretamente parâmetros de aplicação — como requisitos de *Quality of Service* (QoS) e políticas de rede — por meio de **API!** (**API!**)s expostas pela própria infraestrutura.

Todas essas possibilidades oferecidas pelo 5G promovem a computação de borda, ao permitir a redução da latência, a otimização do uso da rede e, consequentemente, a melhoria da experiência do usuário.

2.1.3 A computação de borda e os órgãos de padronização

Na computação de borda, vários órgãos de padronização e consórcios colaboram para definir padrões e promover a interoperabilidade entre dispositivos, redes e serviços. A *European Telecommunications Standards Institute* (ETSI) é uma organização responsável pela padronização de sistemas de telecomunicações e TIC. Um de seus grupos de estudo, o *Industry Specification Group* (ISG) é responsável pela criação e elaboração da arquitetura *Multi-access Edge Computing* (MEC), que oferece aos desenvolvedores de aplicações e aos provedores de conteúdo um padrão para a implementação em ambientes de borda. Assim, as operadoras podem disponibilizar a borda de suas redes para que terceiros autorizados possam implantar seus serviços. O padrão MEC define arquiteturas e interfaces padronizadas para os terceiros, além de especificar *Application Programming Interface* (APIs) para facilitar a interação entre as aplicações e a borda da rede [ETSI ISG MEC 2016].

O 3GPP também propôs uma padronização complementar ao padrão MEC definido pela ETSI. Para isso, foram estabelecidas algumas especificações técnicas [3GPP 2022] com o objetivo de viabilizar a implantação de aplicações de borda baseadas nos padrões da 3GPP. Essas especificações definem uma arquitetura de borda, denominada EDGEAPP, projetada para ser implantada próxima ao usuário e de forma integrada aos componentes da rede 5G. O EDGEAPP define um conjunto de entidades e interfaces que viabilizam a descoberta de instâncias de aplicações de borda pelo cliente que executa noo UE, expõem APIs da rede para as aplicações e também permitem a solicitação de políticas de qualidade de serviço por meio da rede 5G, a fim de garantir requisitos como latência e largura de banda.

Outro órgão padronizador que também reuniu esforços para especificar questões voltadas para a computação de borda foi a *Global System for Mobile Communications Association* (GSMA). Ao contrário da ETSI e do 3GPP, a GSMA não cria padrões e especificações detalhadas, e foca na definição de diretrizes e requisitos de alto nível. Para tal foi criado um grupo de trabalho chamado *Operator Plataform Group* (OPG) que especifica os requisitos necessários para se criar uma plataforma unificada de implementação para as operadores e para os provedores. Foi elaborado o GSMA *Telco Edge Cloud* [GSM Association 2021], uma arquitetura que oferece interoperabilidade entre as operadoras, APIs padronizadas e a integração com as arquiteturas MEC da ETSI e com as redes 5G, padronizadas pela 3GPP. Isso garante que, independente do padrão utilizado, é possível realizar uma integração sem problemas maiores no decorrer da implementação da rede.

2.1.4 O continuum realidade-virtualidade

O aumento crescente do poder de computação nos últimos anos trouxe de volta o interesse pelo *continuum* realidade-virtualidade [Bekele et al. 2018]. Este compreende um conjunto de conceitos envolvendo, de um lado, a realidade do mundo físico e, de outro, a VR. Denomina-se VR, um ambiente gerado por recursos computacionais que reúne e apresenta elementos como espaços, objetos e seres que simulam o que existe na realidade vivenciada pelos cinco sentidos físicos. Entre os dois extremos desse *continuum*, encontram-se a AR e a virtualidade aumentada, ou *Augmented Virtuality* (AVR). A primeira (AR) consiste em sobrepor à percepção que se tem do mundo real elementos visuais e sonoros que existem apenas virtualmente, ou seja, que são gerados e apresentados usando o computador.

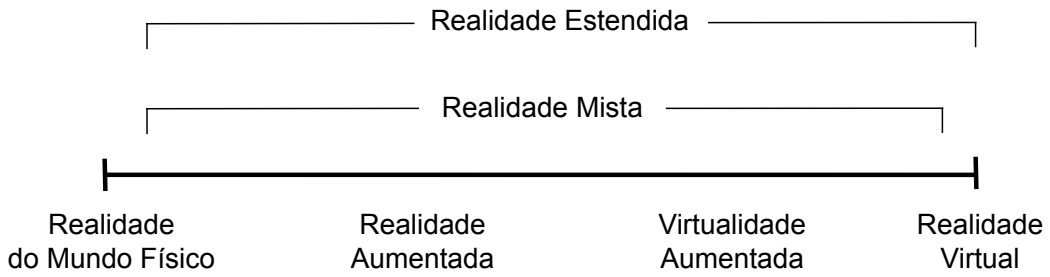


Figura 2.3: Espectro do conceito de *continuum* realidade-virtualidade. Adaptado de [Siriwardhana et al. 2021].

A segunda (AVR) consiste em incluir, em um ambiente de VR, imagens e sons que advém do mundo real. O termo realidade mista, ou *Mixed Reality* (MR), designa, de forma genérica, qualquer tecnologia de AR ou *AVR*. O termo realidade estendida, ou *Extended Reality* (XR), por sua vez, representa qualquer abordagem que usa elementos virtuais, englobando, portanto, AR, AVR e VR. A Figura 2.3 ilustra todo o espectro do *continuum* realidade-virtualidade. Aplicações do tipo AR serão o foco no restante deste trabalho.

2.1.5 Realidade Aumentada Móvel

O princípio fundamental da AR consiste na sobreposição de objetos virtuais ao ambiente físico, de modo que os mundos real e virtual operem de forma integrada e sincronizada [Westphal 2017]. Esses objetos virtuais são ativos digitais dinâmicos renderizados e podem incluir modelos tridimensionais, vídeos bidimensionais e rótulos de texto. O objetivo dessa sobreposição é enriquecer a percepção e a experiência do usuário e facilitar a interação humano-computador.

O termo AR surgiu em 1990, cunhado por Thomas Caudell na Boeing [Thomas e David 1992], ao desenvolver um sistema para auxiliar mecânicos. Em 1992, Louis Rosenberg criou o primeiro sistema AR totalmente imersivo para a Força Aérea dos EUA [Rosenberg 1992] e em 1999 foi criado o ARToolKit, biblioteca open-source conhecida amplamente pela comunidade que trabalha com AR [Malta, Farinha e Mendes 2023]. Avanços significativos ocorreram durante todas as primeiras décadas do ano 2000, com o lançamento do Google Glass em 2013 e HoloLens em 2016, consolidando a AR como uma tecnologia promissora e em constante evolução.

Com toda essa evolução, uma nova tendência em AR é o consumo desse tipo de aplicação em dispositivos móveis, como *smartphones* e HMD. Essas aplicações são conhecidas como aplicações móveis de realidade aumentada (MAR) [Cao et al. 2023] e têm despertado o interesse de diversos setores da economia. Na navegação, por exemplo, aplicações MAR podem ser usadas em *smartphones* para reconhecer objetos no ambiente e sobrepor instruções de navegação para o usuário. Aplicações MAR também são importantes habilitadores para a implementação da Indústria 4.0 [Masood e Egger 2019], podendo ser utilizadas em atividades como manutenção e cooperação remotas e treinamento ou tutorial online para operadores no chão de fábrica. Aplicações MAR encontram diversos casos de uso também na medicina, oferecendo novas abordagens para relacionamentos paciente-médico, tratamentos e educação médica. Por exemplo, as modalidades atuais de imagens médicas, como ultrassom, podem ser significativamente aprimoradas superpondo-as sobre partes físicas do corpo do paciente, ao mesmo tempo em que o médico também visualiza informações como histórico médico anterior e outros sinais vitais [Moro et al. 2021].

A Figura 2.4 ilustra o fluxo típico de uma aplicação MAR. De modo geral, dois componentes principais formam esse tipo de aplicação: as operações de entrada e saída e as tarefas específicas de MAR [Cao et al. 2023]. As operações de entrada e saída envolvem o uso de sensores como câmeras, acelerômetros, giroscópios, sensores de profundidade, entre outros. Esses sensores funcionam como uma ponte entre o mundo físico e o ambiente virtual, capturando informações do ambiente físico em tempo real (passo 1 na Figura 2.4). Os dados provenientes desses sensores são então encaminhados para processamento pelas tarefas MAR (passo 2). Dois módulos principais realizam esse processamento: SLAM e detecção de objetos. O módulo de SLAM permite estimar simultaneamente a posição do UE no espaço e construir um mapa tridimensional do ambiente. O módulo de detecção de objetos, por sua vez, é responsável por identificar e classificar os elementos visuais presentes num quadro ou cena. Esses dois módulos podem operar de forma independente ou integrada, dependendo da arquitetura da aplicação. As saídas desses módulos são utilizados para criar objetos virtuais com diferentes formas — como modelos tridimensionais, caixas delimitadoras (*bounding boxes*), nuvens de pontos,

entre outros — e são então alinhados espacialmente com base na localização e orientação do usuário (passo 3). Em seguida, essa informação é sobreposta às imagens do mundo real capturadas (passo 4). Por fim, o conteúdo aumentado é encaminhado ao dispositivo de saída, permitindo que o usuário visualize a cena com elementos virtuais integrados ao ambiente físico (passo 5).

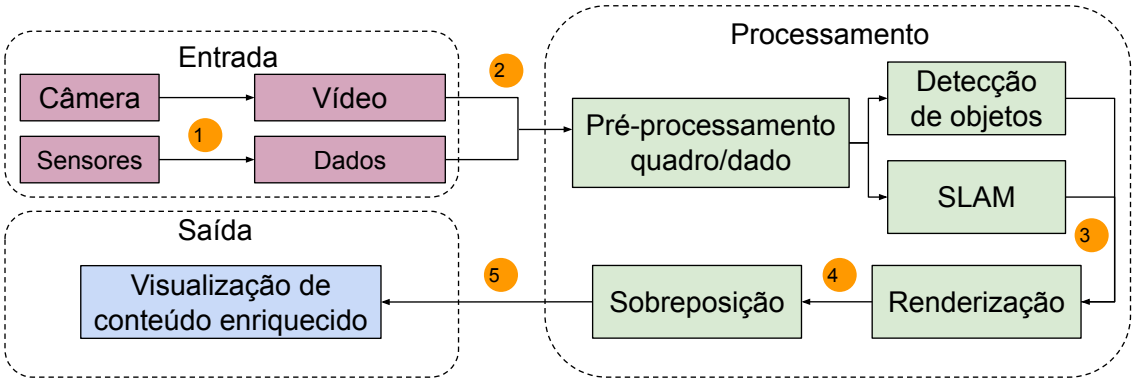


Figura 2.4: Arquitetura básica para aplicações MAR.

Em geral, as tarefas MAR utilizam algoritmos complexos de visão computacional, reconhecimento de imagem e renderização. Como consequência, elas exigem muito recurso de processamento [Chatzopoulos et al. 2017]. Além disso, o conteúdo de AR deve ser renderizado com um atraso de movimento para fóton abaixo de 15 ms [Morín, Pérez e Armada 2022]. Essas duas exigências reforçam a necessidade da combinação entre o descarregamento de tarefas e o uso de redes 5G.

Devido aos exigentes requisitos computacionais das tarefas MAR, executar toda a aplicação diretamente no dispositivo móvel pode ser inviável. Isso se deve ao fato de que a realização de operações intensivas nesses dispositivos consome rapidamente a bateria. Embora uma possível solução fosse equipá-los com baterias maiores, isso comprometeria a ergonomia, tornando-os mais pesados, volumosos e desconfortáveis para o uso prolongado. Ao mesmo tempo, as restrições severas de latência tornam o descarregamento dessas tarefas para a nuvem impraticável. Nesse contexto, a computação de borda surge como uma alternativa promissora para aplicações MAR [Chen et al. 2017]. Como discutido na subseção 2.1.1, os servidores de borda estão mais próximos do dispositivo de usuário e são capazes de atender às demandas computacionais das aplicações. Além disso, os serviços fornecidos por redes 5G, como o *Enhanced Mobile Broadband* (eMBB) e o *Ultra Reliable Low Latency Communications* (URLLC), possibilitam a transmissão eficiente de fluxos de dados provenientes de múltiplos sensores, bem como o recebimento dos resultados processados, mantendo a latência dentro dos limites exigidos por aplicações interativas e sensíveis ao tempo e permitindo o descarregamento das mesmas para os servidores de borda.

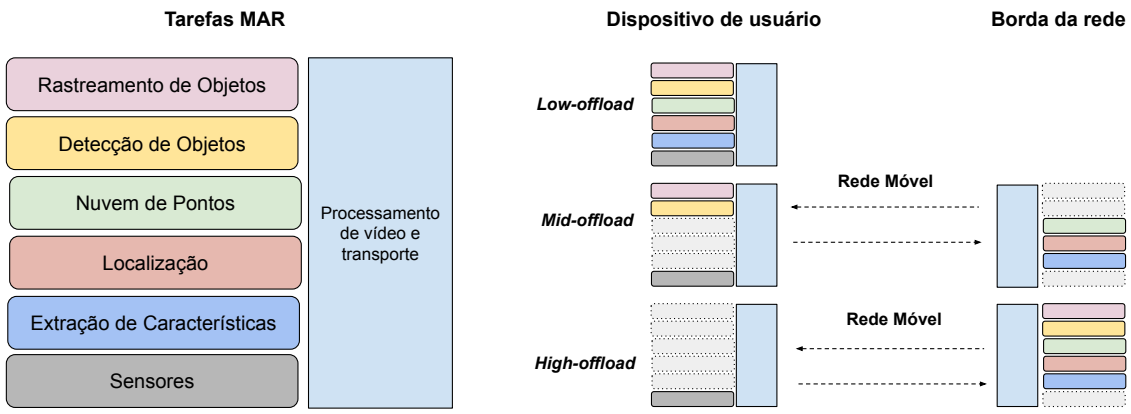


Figura 2.5: Tarefas MAR sem a realização de *offloading* e com a realização de *offloading*. Adaptado de [Alriksson et al. 2021]

O lado esquerdo da Figura 2.5 ilustra as tarefas MAR de forma detalhada. A tarefa mais básica consiste na captura de imagens do ambiente em torno do usuário, bem como dos dados de sensores do dispositivo móvel (UE). Em seguida, as imagens e dados capturados podem ser enviados para um algoritmo de SLAM, o qual executa três tarefas principais: extração de características, localização do dispositivo e geração de nuvem de pontos. A primeira tarefa, extração de características, identifica pontos de interesse numa imagem como esquinas, bordas ou outros elementos distintivos. Usando as características extraídas, a segunda tarefa, localização, estima a posição do dispositivo móvel no ambiente para, em seguida, gerar uma nuvem de pontos. Esta, por sua vez, representa os pontos tridimensionais do ambiente com base nas características extraídas. A nuvem de pontos é usada para ancorar o conteúdo virtual ao cenário real para que o movimento do dispositivo não afete a posição e o comportamento dos objetos virtuais.

Além do SLAM, algoritmos de detecção de objetos também são de grande importância para aplicações MAR. Eles obtêm informações semânticas da cena real como quais objetos reais estão na cena, juntamente com sua forma, posição e tamanho.

Por fim, uma vez que um algoritmo de SLAM cria uma nuvem de pontos ou um algoritmo de detecção de objetos identifica e classifica objetos nas imagens capturadas, objetos virtuais podem ser renderizados e sobrepostos nas imagens do mundo físico. Áudio e som também podem ser codificados para posterior transmissão. Essas funcionalidades compõem a tarefa de processamento de vídeo e transporte.

O lado direito da Figura 2.5 mostra como as tarefas MAR podem ser divididas entre o dispositivo móvel e uma infraestrutura de computação de borda [Alriksson et al. 2021]. Uma alternativa, denominada modelo *low-offload*, consiste em executar todas as tarefas no dispositivo móvel, sem nenhum descarregamento para a borda. Como discutido anteriormente, esse modelo exige que o dispositivo móvel

possua grande capacidade de processamento, o que, normalmente, não está disponível na grande maioria dos dispositivos. Outra alternativa, denominada *mid-offload*, consiste em descarregar as tarefas de SLAM e/ou de detecção de objetos para a borda da rede, enquanto as tarefas de captura de dados de entrada, rastreamento de objetos e renderização são realizadas no dispositivo móvel. Esse modelo se apoia em algoritmos estado-da-arte para alcançar bom desempenho em tempo real em dispositivos como *smartphones*. No entanto, a utilização de recursos nesse modelo ainda é elevada, podendo comprometer a acurácia dos algoritmos. Finalmente, o terceiro modelo, denominado *high-offload*, descarrega todas as tarefas computacionalmente intensivas para a borda da rede, executando apenas a tarefa de entrada de dados no dispositivo móvel. Este modelo, em geral, apresenta o melhor compromisso entre desempenho e consumo de energia no dispositivo [Pires et al. 2024]. A seguir, discutimos com maior detalhe as tarefas de SLAM e detecção de objetos.

2.1.6 SLAM

O processo de localização e mapeamento simultâneo (SLAM) tem por objetivo permitir a construção de um mapa de determinado ambiente e, ao mesmo tempo, utilizar este mapa construído para deduzir uma localização [Aulinas et al. 2008]. Esse conceito é amplamente utilizado na robótica, onde um robô precisa se deslocar em um ambiente desconhecido e, com a ajuda de seus sensores, deve ser capaz de escanear o espaço, prever o melhor caminho a seguir e determinar sua própria localização no mapa gerado.

Uma das maneiras de se dividir esse tipo de algoritmo é de acordo com a forma de captura dos dados para mapeamento e localização. Existem dois tipos: os algoritmos de SLAM baseados em visão (*Visual SLAM*) e os baseados em *Light Detection and Ranging* (LiDAR) [Alsadik e Karam 2021].

A evolução dos algoritmos de SLAM baseados em visão reflete diferentes estratégias para capturar e interpretar informações do ambiente. O MonoSLAM [Davison et al. 2007] introduziu a ideia de utilizar uma única câmera para construir mapas probabilísticos a partir de pontos de interesse, utilizando o Filtro de Kalman Estendido para atualizar suas estimativas. Posteriormente, o *Dense Tracking and Mapping in Real-Time* (DTAM) propôs uma abordagem mais densa e detalhada, reconstruindo o ambiente considerando todos os pixels da imagem e exigindo maior capacidade computacional. Em seguida, o *Semi-direct Visual Odometry* (SVO) [Forster, Pizzoli e Scaramuzza 2014] surgiu como uma solução híbrida, combinando métodos diretos e baseados em características para melhorar a eficiência e reduzir o custo computacional.

Ao longo dos anos, novas técnicas continuaram a aprimorar a capacidade

dos sistemas SLAM, buscando equilibrar precisão, robustez e eficiência de recursos. Algoritmos como o ORB-SLAM [Mur-Artal et al. 2015] e seu sucessor, ORB-SLAM2 [Mur-Artal e Tardós 2017], exemplificam essa evolução ao introduzir estratégias como a seleção eficiente de quadros-chave e o uso de descritores compactos para pontos de interesse. Ao mesmo tempo em que o ORB-SLAM2 foi desenvolvido, surgiu também o CNN-SLAM [Tateno et al. 2017], uma das primeiras ferramentas de SLAM a utilizar redes neurais convolucionais. A principal contribuição oferecida pelo CNN-SLAM é a capacidade de combinar a predição de profundidade por redes neurais convolucionais, ou *Convolutional Neural Network* (CNN), com medições de profundidade obtidas a partir de um sistema de SLAM monocular. A evolução ao longo dos anos destes algoritmos de SLAM pode ser observada na Figura 2.6.

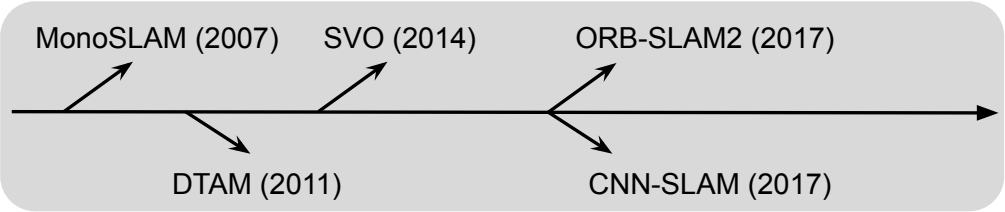


Figura 2.6: Evolução dos algoritmos de SLAM ao longo do tempo. Adaptado de [Barros et al. 2022].

2.1.7 Detecção de objetos

Os algoritmos de detecção de objetos são fundamentais em visão computacional e têm sido amplamente aplicados em áreas como segurança, veículos autônomos e saúde, com o objetivo de identificar e localizar objetos em imagens e vídeos. Eles surgem como solução para o desafio de classificar e posicionar objetos em cenas visuais, tarefa que ganhou avanços expressivos nas últimas duas décadas com o desenvolvimento da inteligência artificial e do aprendizado profundo [Zaidi et al. 2022].

Em [Zou et al. 2023] e [Wang et al. 2023], os autores dividem a evolução dos algoritmos de detecção em duas grandes eras: a era dos métodos tradicionais e a era dos métodos baseados em aprendizagem profunda. Historicamente, a detecção evoluiu da era dos métodos tradicionais, como Viola-Jones [Viola e Jones 2001], HOG [Dalal e Triggs 2005] e DPM [Felzenszwalb, McAllester e Ramanan 2008] — que utilizaram abordagens baseadas em características manuais e classificadores simples — para a era da aprendizagem profunda, com redes neurais convolucionais. Esses métodos tradicionais foram fundamentais para estabelecer as bases da visão computacional, cada um trazendo avanços como o uso de imagens integrais, histogramas de gradientes e modelagem por partes deformáveis.

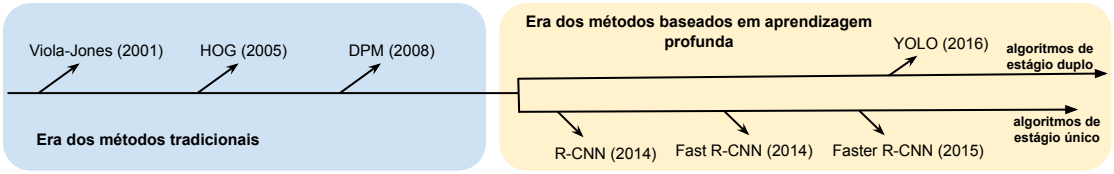


Figura 2.7: Evolução dos algoritmos de detecção de objetos. Adaptado de [Zou et al. 2023]

Com os avanços das redes neurais, especialmente as convolucionais, surgiram algoritmos como o R-CNN [Girshick et al. 2014], que utiliza regiões de interesse combinadas com CNNs, marcando o início da transição para métodos mais robustos. Em seguida, modelos como o YOLO [Redmon et al. 2016] propuseram uma abordagem de detecção em estágio único, oferecendo grande velocidade ao analisar toda a imagem de uma vez só. A evolução dos algoritmos de detecção está apresentada na Figura 2.7.

Apesar de ser possível realizar a detecção de objetos em arquiteturas sem aceleração de hardware, os algoritmos de detecção baseados em redes neurais convolucionais exigem grande poder de processamento, especialmente em tempo real, para alcançar níveis aceitáveis de desempenho e acurácia [Mansoub, Abri e Yarıcı 2019], [Shin e Kim 2022].

A tarefa de detecção de objetos é um componente importante em aplicação de realidade mista e aumentada. Para dispositivos como os óculos de realidade aumentada, a tarefa de selecionar e identificar itens do mundo real com a mesma rapidez de um humano é uma tarefa complicada mas necessária, já que ela traz segurança – por exemplo em sistemas de direção autônoma que precisam identificar pedestres – e eficácia – por exemplo em sistemas de reconhecimento facial [Liu, Li e Gruteser 2019, Hu et al. 2020].

2.2 Trabalhos Relacionados

Nesta seção, são discutidos trabalhos relacionados a esta dissertação, organizados em duas categorias principais: o primeiro grupo abrange estudos que tratam do descarregamento de tarefas de aplicações MAR para a borda da rede; o segundo concentra-se em estudos que realizam a caracterização dessas tarefas, com foco específico em suas propriedades e comportamentos.

2.2.1 Descarregamento de realidade mista para a borda

Um dos primeiros trabalhos a investigar os benefícios da computação de borda para aplicações MAR foi realizado por [Chen et al. 2017]. Nesse estudo, os autores propuseram e avaliaram protótipos de aplicações MAR voltados à assistência cognitiva. A

pesquisa incluiu experimentos que variaram os tipos de dispositivos móveis utilizados, as tecnologias de rede empregadas (4G e Wi-Fi) e diferentes configurações de servidores de borda. Os protótipos implementados eram baseados em técnicas de visão computacional, abrangendo desde algoritmos simples de detecção de bordas até modelos mais complexos de detecção de objetos. Como um dos principais resultados, o trabalho propôs um sistema baseado em múltiplos algoritmos, que explora a coerência temporal para executar algoritmos em paralelo. De forma dinâmica, o sistema avalia se o algoritmo mais rápido atinge um nível aceitável de acurácia e, com base nessa avaliação, decide se continuará utilizando esse algoritmo. Os autores demonstram que a computação de borda é claramente mais vantajosa do que a infraestrutura de nuvem tradicional em aplicações MAR, principalmente devido à redução significativa da latência. No entanto, em [Chen et al. 2017], os protótipos utilizados não representam integralmente aplicações MAR, uma vez que não fornecem um fluxo de vídeo enriquecido ao usuário, limitando-se a apresentar indicações visuais ou auditivas. Além disso, o estudo não realiza uma caracterização da aplicação, concentrando-se apenas na latência fim-a-fim e desconsiderando métricas relevantes como consumo de CPU, memória e energia.

Uma arquitetura distribuída denominada ARENA foi apresentada em [Pereira et al. 2021] com o objetivo de simplificar a construção e hospedagem de aplicaçõesAR e VR baseadas em navegador. O ARENA oferece diversos componentes, incluindo: um serviço de diretório geoespacial hierárquico que conecta usuários a servidores e conteúdos próximos, um sistema de autenticação baseado em *tokens* para controle de acesso ao conteúdo, e um barramento PubSub responsável por transmitir, em tempo real, todas as interações dos usuários. Embora seja uma arquitetura abrangente, o ARENA não foi avaliado com foco na descarga de tarefas MAR para a borda, mas sim quanto ao posicionamento do *broker* de mensagens na borda da rede ou na nuvem. O objetivo central dos autores é oferecer uma plataforma escalável, segura e de baixa latência que facilite o desenvolvimento e a execução de aplicações XR em ambientes com múltiplos usuários e dispositivos heterogêneos, concentrando-se, portanto, em aspectos arquiteturais. Apesar de incluir avaliações quantitativas relacionadas ao uso de CPU, memória e largura de banda, o ARENA não se propõe a ser um estudo de caracterização de recursos, direcionando sua análise principalmente à proposta e aplicabilidade da arquitetura desenvolvida.

Com o objetivo de reduzir a latência na interação do usuário, os autores de [Toczé et al. 2019] propõem o MR-Leo, um protótipo voltado à criação de um fluxo de vídeo enriquecido com elementos de realidade mista. Construído sobre uma arquitetura cliente-servidor, o MR-Leo utiliza técnicas de SLAM e renderização gráfica para integrar conteúdos virtuais ao ambiente real de forma dinâmica. O protótipo adota uma organização modular que permite testar diferentes configurações, tornando-o uma ferramenta

versátil para experimentação. Diferentemente dos demais trabalhos discutidos até aqui, este estudo apresenta uma análise detalhada da latência fim-a-fim associada ao algoritmo de SLAM utilizado. No entanto, os autores não integram outras funcionalidades típicas de aplicações de realidade mista, como detecção de objetos, nem abordam questões relacionadas à aceleração por hardware para otimização da latência. Assim, o trabalho não se propõe a realizar uma caracterização completa da aplicação em termos de uso de recursos computacionais.

Os autores de [Zhang et al. 2022] desenvolveram o EdgeXAR, um arcabouço para aplicações MAR que integra computação de borda com o objetivo de superar as limitações de processamento dos dispositivos móveis e reduzir a latência na interação do usuário. A principal inovação do EdgeXAR está em seu sistema híbrido de rastreamento com seis graus de liberdade (*Six Degrees of Freedom* (6-DoF)), que permite a compensação de latência de forma perceptível ao usuário. Para validar a arquitetura, os autores implementaram uma aplicação baseada em um algoritmo de detecção de objetos treinado especificamente para reconhecer cartazes de filmes. Ao apontar o dispositivo para um cartaz, o sistema realiza o reconhecimento e redireciona o usuário para um vídeo com o trailer correspondente. A avaliação de desempenho considera a latência nas comunicações entre o dispositivo do usuário e a borda, bem como entre o usuário e a nuvem. Além disso, são analisados aspectos relacionados à acurácia do algoritmo de detecção de objetos empregado. No entanto, o foco do trabalho não está na caracterização da aplicação nem na execução de múltiplas tarefas MAR, limitando-se à detecção de objetos como funcionalidade principal.

2.2.2 Caracterização de carga de trabalho de realidade mista

Com o objetivo de realizar uma caracterização mais aprofundada do MR-Leo, o trabalho apresentado em [Toczé et al. 2020] detalha o comportamento da aplicação quando descarregada para a borda. A partir do protótipo original, os autores expandem a análise além da latência fim-a-fim — já discutida na seção anterior — para incluir métricas como consumo de CPU e memória. Também foram coletados dados sobre o número de instruções necessárias para a execução da aplicação, permitindo a construção de um modelo estatístico e de um modelo de cargas. Esses recursos viabilizam testes com aplicações que compartilhem perfil e demandas similares às do MR-Leo, especialmente aplicações de realidade aumentada ou mista. Apesar da caracterização detalhada, o estudo concentra-se exclusivamente na tarefa de SLAM e considera apenas cenários de execução baseados em CPU, não explorando como a aceleração por GPU poderia influenciar o desempenho ou contribuir para uma caracterização mais completa da aplicação.

Em [Wang et al. 2022], o foco principal é a criação de um sistema MAR que

adapta dinamicamente suas configurações (como frequência de CPU, tamanho do modelo de detecção e taxa de amostragem da câmera) com o objetivo de minimizar o consumo de energia por quadro sem degradar métricas importantes como latência de serviço e acurácia na detecção de objetos. O trabalho introduz dois componentes principais: o LEAF, um algoritmo de otimização que adapta dinamicamente configurações como frequência da CPU, taxa de amostragem da câmera e tamanho do modelo de detecção para equilibrar consumo de energia, latência e acurácia; e o AIO, um orquestrador de frequência de *offloading* que decide quando utilizar detecção remota ou rastreamento local de objetos, evitando execuções desnecessárias. Apesar de não ser puramente um trabalho de caracterização, os autores trazem uma análise experimental que mede a frequência de CPU, consumo energético e largura de banda a partir da execução da tarefa de detecção de objetos, mas faz essa caracterização somente para a tarefa de detecção de objetos usando apenas CPU, sem considerar uma possível aceleração por GPU.

Trabalhos	Usa Computação de Borda	Tarefa MAR utilizada	Realiza caracterização de perfil de consumo	Recursos caracterizados
[Chen et al. 2017]	✓	Detecção de objetos	✗	-
ARENA [Pereira et al. 2021]	✓	-	✗	-
MR-Leo [Toczé et al. 2019]	✓	SLAM	✗	-
EdgeXAR [Zhang et al. 2022]	✓	Detecção de objetos	✗	-
MR-Leo [Toczé et al. 2020]	✓	SLAM	✓	Rede, CPU e Memória
LEAF + AIO [Wang et al. 2022]	✓	Detecção de objetos	✓	Rede, CPU e Energia
ORB-YOLO [Wu, Miao e Sun 2023]	✗	SLAM e detecção de objetos	✓	CPU e GPU
Este Trabalho	✓	SLAM e detecção de objetos	✓	Rede, CPU, GPU, Memória e Energia

Tabela 2.1: Tabela comparativa dos trabalhos relacionados que realizam descarregamento de realidade mista para a borda.

Procurando integrar as duas principais tarefas MAR, os autores em [Wu, Miao e Sun 2023] desenvolveram o ORB-YOLO, um sistema unificado que combina as capacidades de mapeamento e localização do ORB-SLAM3 com a detecção de objetos do YOLOv8. O YOLOv8 é utilizado para identificar objetos dinâmicos no ambiente. Isso é feito através de um módulo de pré-processamento que detecta objetos e remove pontos dinâmicos da cena. O sistema proposto é composto por cinco módulos principais: pré-processamento, rastreamento, mapeamento local, detecção de laços e fusão de mapas, e ajuste de *bundle* completo. A maior parte do sistema permanece

semelhante ao ORB-SLAM3, mas com a adição do módulo de detecção de objetos do YOLOv8. O trabalho não esta dentro de um escopo de computação de borda e busca focar sua análise somente nas tarefas de SLAM e detecção de objetos sem levar em consideração os possíveis locais de alocação dessas tarefas ou considerar questões relacionadas a comunicação. Sendo assim, uma caracterização é feita a partir da implementação das tarefas em infraestruturas com CPU e GPU, mas sem analisar questões voltadas a rede, memória ou energia.

Diferentemente dos trabalhos relacionados discutidos até aqui — que, em sua maioria, não realizam uma caracterização extensiva ou, quando o fazem, limitam-se a apenas uma das principais tarefas de aplicações MAR ou não consideram o contexto de computação de borda e descarregamento, esta dissertação se destaca por conduzir uma caracterização experimental abrangente de duas tarefas fundamentais: SLAM e detecção de objetos. A análise é realizada em duas infraestruturas computacionais distintas (CPU e GPU), permitindo uma comparação mais completa dos impactos de diferentes ambientes de execução para a infraestrutura de borda. Além disso, enquanto grande parte da literatura foca apenas em métricas como consumo de energia ou latência, este trabalho avança ao caracterizar detalhadamente múltiplos recursos do sistema, incluindo uso de rede, CPU, GPU, memória e energia. Com isso, oferece uma visão mais precisa e abrangente do comportamento e das demandas de tarefas MAR em diferentes cenários. A Tabela 2.1 apresenta uma comparação entre os trabalhos analisados, destacando as características discutidas ao longo desta seção.

2.3 Considerações finais

Este capítulo apresentou os conceitos essenciais e estudos que embasam o uso da computação de borda em aplicações MAR. Inicialmente, discutimos como a borda complementa a nuvem ao oferecer menor latência e maior eficiência energética, atendendo às limitações de dispositivos móveis. A arquitetura do 5G foi destacada por seu suporte a serviços na borda, com recursos como roteamento otimizado e continuidade de sessões. Também foram abordados os esforços de padronização de entidades como ETSI, 3GPP e GSMA, que promovem a interoperabilidade e facilitam a implementação de aplicações na borda. O texto explorou ainda o continuum realidade-virtualidade, com foco nas aplicações MAR, detalhando as tarefas de SLAM e detecção de objetos, que demandam alto poder de processamento e motivam o uso de servidores de borda. Por fim, foram discutidos trabalhos relacionados ao descarregamento e à caracterização de tarefas MAR que foram comparados quanto às tarefas abordadas e aos recursos avaliados (como uso de CPU, GPU, memória, rede e energia), evidenciando lacunas na integração e avaliação conjunta dessas tarefas, o que fundamenta a proposta deste trabalho.

CAPÍTULO 3

enhanced MR-Leo

Este capítulo apresenta uma descrição da arquitetura e funcionamento do protótipo MR-Leo, além do detalhamento da versão modificada e desenvolvida nessa dissertação, chamada de **enhanced MR-Leo**, ou eMR-Leo. O objetivo da Seção 3.1 é oferecer uma visão abrangente dos principais componentes do protótipo — incluindo o serviço de borda, o aplicativo cliente, a rede de comunicação entre os componentes, o serviço de transmissão de vídeo, e os algoritmo de SLAM e renderização utilizados. Já a Seção 3.2 é dedicada para destacar as modificações realizadas no protótipo original, especialmente o serviço de detecção de objetos implementado, destacando também as escolhas de implementação, ferramentas utilizadas e os desafios enfrentados durante o desenvolvimento.

3.1 O protótipo MR-Leo

Nesta seção são apresentados a arquitetura e o funcionamento do MR-Leo, um protótipo de realidade mista investigado neste trabalho.

3.1.1 Visão geral

Mixed Reality Linköping Edge Offloading (MR-Leo) [Lindqvist 2019] é um protótipo desenvolvido na Universidade de Linköping, na Suécia, como parte de uma dissertação de mestrado. O objetivo do protótipo é habilitar aplicações de realidade mista, particularmente AR, por meio da integração com a computação de borda. A arquitetura do MR-Leo é dividida em dois componentes principais: o serviço que executa no servidor de borda e o aplicativo que executa no dispositivo do usuário. Esses dois componentes e o fluxo de dados entre eles são ilustrados na Figura 3.1.

Como mostrado na figura, uma grande parte do protótipo envolve a transferência de vídeo. A partir do dispositivo do usuário, as imagens da câmera são convertidas em um fluxo de vídeo pelo aplicativo e enviadas ao servidor de borda. O serviço de borda recebe os quadros e os guarda numa fila. O serviço de borda implementa diversas tarefas além da recepção e transmissão de vídeo. A principal tarefa é a de realidade

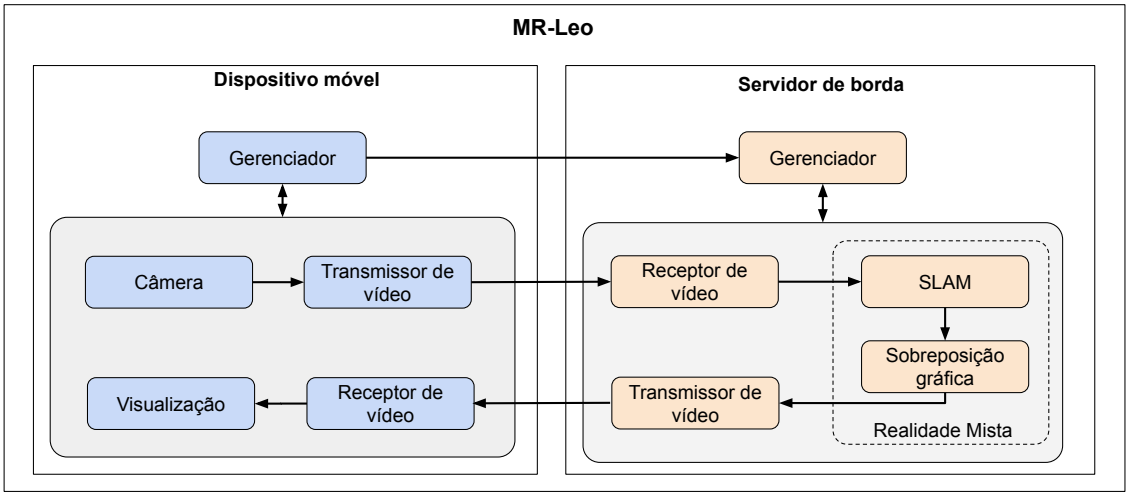


Figura 3.1: Principais componentes do MR-Leo.

mista, composta por duas sub-tarefas: SLAM e sobreposição gráfica. A tarefa de SLAM retira quadros da fila e os processa para criar um mapa 3D do ambiente. Esse mapa é representado por uma nuvem de pontos 3D. O SLAM determina também a posição e rotação do dispositivo do usuário no modelo (mapa), conhecida como pose. À medida que os quadros são processados, o mapeamento do ambiente é atualizado, bem como a pose. Quando uma nuvem de pontos é efetivamente criada e a pose do dispositivo final dentro dela é conhecida, objetos gráficos tridimensionais podem ser adicionados a ela. A decisão de quando adicionar um objeto à nuvem de ponto é feita pelo usuário, com um comando enviado ao servidor de borda. Quando esse botão é acionado, a nuvem de pontos criada e a pose são usadas como entrada pela tarefa de sobreposição gráfica. Essa tarefa cria os objetos virtuais e posiciona cada elemento em um local apropriado no ambiente mapeado, criando assim uma sobreposição à imagem origina do quadro. Por fim, o quadro resultante, que compreende o quadro original com os elementos virtuais adicionados, é renderizado e enviado novamente para o dispositivo final para ser exibido para o usuário.

No *pipeline* descrito acima, o aplicativo que executa no dispositivo final não realiza nenhuma análise do conteúdo das imagens capturadas pela câmera; ele apenas as codifica como vídeo e as envia para o servidor de borda. Todas as tarefas computacionalmente intensivas são implementadas pelo serviço de borda. Portanto, o protótipo segue a arquitetura *high offload* descrito no Capítulo 2.

Ainda na Figura 3.1, observamos que tanto o dispositivo do usuário quanto o servidor de borda executam uma instância de um gerenciador geral. Esse gerenciador é responsável por configurar tanto o aplicativo quanto o serviço de borda, além de gerenciar a comunicação com o outro dispositivo. Isso torna o protótipo modular, facilitando diferentes configurações de uso, como diferentes soluções de codificação de vídeo (H.264

ou MJPEG) e de protocolos de transporte – *Transmission Control Protocol* (TCP) ou *User Datagram Protocol* (UDP). De fato, antes de qualquer vídeo ser transmitido ou o dado ser processado, os dois gerenciadores negociam qual configuração utilizar para a sessão, de forma que os dois componentes (aplicativo e serviço de borda) sejam fracamente acoplados.

3.1.2 Componente de borda

Como mostrado na parte direita da Figura 3.1, o serviço de borda do MR-Leo é o componente responsável por executar as tarefas computacionalmente mais exigentes relacionadas à realidade mista, incluindo a tarefa de SLAM, a sobreposição dos objetos virtuais e a renderização do quadro resultante. O serviço processa dois tipos principais de entrada enviados pelo aplicativo cliente: quadros de vídeo codificados e mensagens que indicam interações específicas com a interface, como o acionamento de botões. A saída gerada por este componente é um quadro de vídeo enriquecido, ou seja, com objetos virtuais sobrepostos à imagem original, que é enviado novamente para o dispositivo do usuário.

A implementação do serviço de borda foi realizada em C++ utilizando o *framework* Qt[1]. Segundo os autores [Lindqvist 2019], a linguagem C++ foi escolhida pela facilidade de extensão via paradigma de programação orientada a objetos e pela interoperabilidade com bibliotecas de código aberto. As tarefas de transmissão e recepção de vídeo, comunicação com o aplicativo cliente, SLAM e sobreposição gráfica são iniciadas pelo serviço de borda, sendo cada tarefa executada em uma *thread* distinta, cabendo ao modelo de eventos do Qt a responsabilidade de sincronizá-las.

Atualmente, o algoritmos de SLAM usado pelo serviço de borda para criar a nuvem de pontos (descrito na Seção 3.1.6) não têm um tempo de processamento limitado. Isso significa que a análise de um quadro específico pode levar um tempo variável. Durante esse tempo, os próximos quadros que chegam no servidor não podem ser analisados e são enfileirados. Para evitar o acúmulo de latência, o serviço de borda implementa um mecanismo de descarte de pacote para garantir que quadros desatualizados não sejam manipulados. No entanto, para manter a QoS, isso não deve acontecer com muita frequência, sendo desejável que um quadro seja processado antes que o próximo chegue.

A Figura 3.2 mostra o serviço de borda do MR-Leo iniciado via terminal, operando sem interface gráfica. Sua configuração é realizada por meio de parâmetros fornecidos na linha de comando, os quais permitem, por exemplo, acessar a câmera do

[1] https://www.qt.io/

dispositivo do usuário, utilizar um arquivo de vídeo estático ou acessar a câmera da máquina onde o serviço está sendo executado, entre outras funcionalidades adicionais.

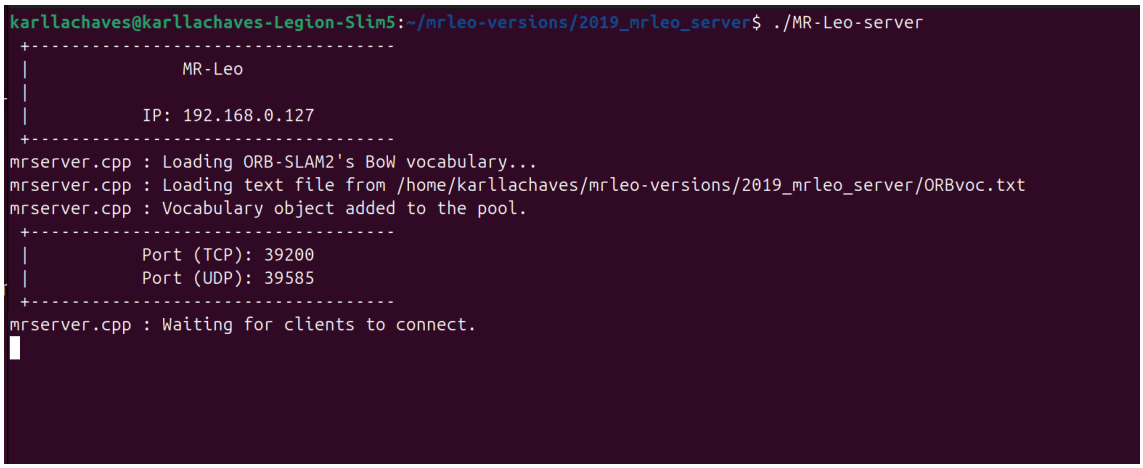


Figura 3.2: Serviço de borda do MR-Leo iniciado via terminal.

3.1.3 Componente do dispositivo de usuário

A parte cliente do sistema é um aplicativo executado no dispositivo do usuário. O aplicativo recebe como entrada os quadros de vídeo capturados pela câmera do dispositivo, bem como os quadros de vídeo enriquecidos enviados pelo servidor de borda, que devem ser exibidos ao usuário. Além disso, o cliente também processa interações do usuário, como, por exemplo, o acionamento de um botão para adicionar um objeto virtual à cena.

O aplicativo foi implementado para ser executado em sistemas Android e possui partes escritas em Java e em C. O código em C foi necessário para a integração do GStreamer, utilizado para o processamento de vídeo e que não possui bibliotecas nativas para um integração com Java.

A Figura 3.3 apresenta a tela inicial exibida ao usuário ao iniciar o aplicativo cliente do MR-Leo. Nessa interface, o usuário pode escolher entre utilizar a câmera do dispositivo móvel, reproduzir um vídeo pré-gravado armazenado no próprio aparelho ou iniciar um protótipo de realidade aumentada baseado em ARCore. Além disso, o usuário pode definir as configurações de execução do aplicativo, selecionando os protocolos de comunicação e os esquemas de codificação de vídeo para os fluxos de *uplink* e *downlink*. As configurações escolhidas são enviadas ao serviço de borda, que então inicia o processamento de acordo com os parâmetros definidos.

O retorno visual do enriquecimento, que é apresentado ao usuário, está ilustrado na Figura 3.4. Essa interface corresponde à segunda etapa da aplicação cliente, onde são

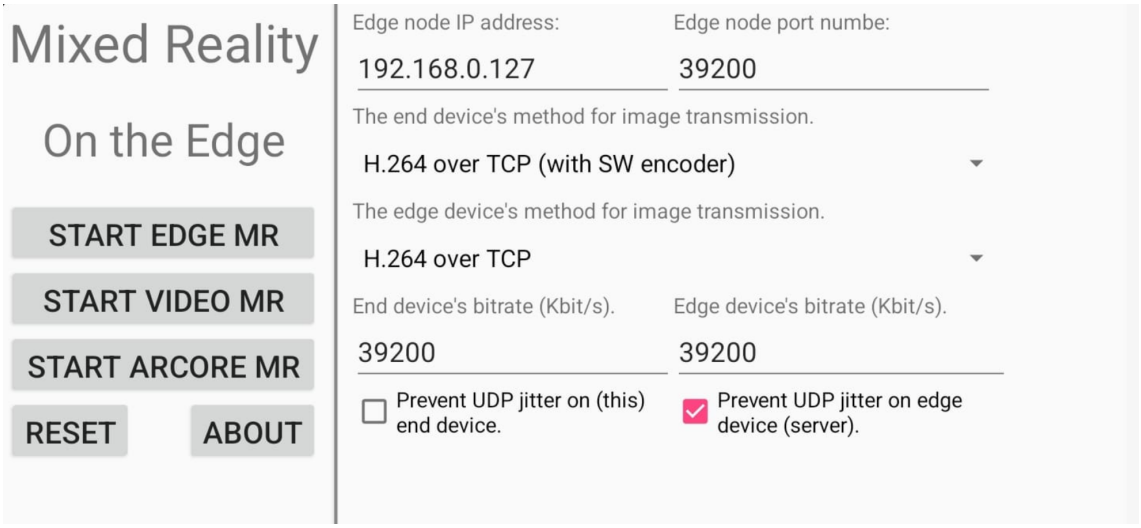


Figura 3.3: Aplicativo cliente do MR-Leo.

exibidos os botões interativos. O usuário pode optar por adicionar ou remover o objeto virtual da cena, bem como encerrar a aplicação. A primeira fase do enriquecimento consiste na geração da nuvem de pontos (Figura 3.4(a)), e a segunda ocorre quando o usuário pressiona o botão verde para inserir um objeto virtual na cena (Figura 3.4(b)).

3.1.4 Rede de comunicação

O MR-Leo precisa de uma rede de comunicação sem fio que possibilite a comunicação entre o aplicativo cliente e o serviço de borda. O componente de rede suporta a transferência de comandos, configurações e grandes arquivos de dados nas duas direções (*uplink* e *downlink*), sendo construído sobre *sockets* UDP e TCP. Por padrão, apenas o TCP é usado para comandos e configuração, enquanto tanto o UDP quanto o TCP podem ser usados para transferir arquivos maiores, como imagens. Ao usar o UDP, os dados do arquivo são divididos em datagramas com o tamanho máximo do pacote definido pelo usuário. Comandos e configurações são enviados no formato JSON.

3.1.5 Transmissão de vídeo

O componente de transmissão de vídeo é responsável por codificar e enviar os quadros, bem como por receber e decodificá-los, tanto no aplicativo cliente quanto no serviço de borda. Como mencionado anteriormente, no serviço de borda, o transmissor e o receptor de vídeo são executados em *threads* separadas e o mecanismo de sinalização do Qt foi utilizado para garantir que apenas o último quadro de imagem de vídeo disponível seja processado, codificado ou transmitido. Atualmente, o MR-Leo pode operar com dois formatos de codificação de vídeo: H.264 e MJPEG. No envio de quadros do dispositivo do

(a) *Criação da nuvem de pontos.*

(b) *Sobreposição do objeto virtual.*

Figura 3.4: Cenas enriquecidas na aplicação do MR-Leo. Em 3.4(a) é gerada uma nuvem de pontos que serve de ancoragem para que em 3.4(b) o objeto virtual seja colocado no quadro de vídeo

usuário para o servidor de borda, é possível utilizar apenas o H.264. Já no sentido oposto — da borda para o dispositivo do usuário —, é possível optar entre H.264 e MJPEG.

O componente de transmissão de vídeo usa o GStreamer, tanto no aplicativo cliente quanto no serviço de borda. O GStreamer é um *framework* de código aberto usado para processar vídeo e áudio, incluindo codificação e decodificação, e com suporte oficial para diversas plataformas, incluindo Linux e Android.

3.1.6 SLAM

A solução de SLAM usado no MR-Leo é o ORB-SLAM2 [Mur-Artal e Tardós 2017], uma biblioteca de código aberto desenvolvida na Universidade de Zaragoza, na Espanha. O ORB-SLAM2 inclui suporte para câmeras monoculares, estéreo e de detecção de

profundidade e tem sido utilizado em diversos projetos de visão computacional.

No MR-Leo, o ORB-SLAM2 é utilizado apensa com câmeras monoculares. A biblioteca implementa três *threads* principais que trabalham em paralelo e que são descritas a seguir.

- A *thread* de rastreamento recebe um quadro da câmera e extrai as características importantes da imagem usando o algoritmo *Oriented FAST and Rotated BRIEF* (ORB) do OpenCV[2]. Os descritores de características extraídos são então usados para calcular a posição da câmera em relação à posição detectada em quadros anteriores. Essa *thread* também pode salvar o grupo de descritores de imagem como um quadro-chave, quando ela determina que há características únicas suficientes presentes em uma imagem.
- A *thread* de mapeamento local atualiza e expande o mapa 3D do ambiente. Quando um novo quadro-chave é inserido pela *thread* de rastreamento, esta *thread* triangula novos pontos 3D com base na correspondência entre quadros-chave, refina o mapa com uma otimização local e remove pontos ruins ou redundantes.
- A *thread* de fechamento de *loop* recebe dados da *thread* de mapeamento local e os utiliza para detectar *loops*. Estes ocorrem quando a câmera retorna a uma posição previamente conhecida. A não eliminação de *loops* pode causar erros cumulativos no mapa.

De fato, o ORB-SLAM2 padrão não suporta a transferência de cálculos matriciais para aceleradores e tudo é processado pela CPU. Ele utiliza também extensivamente o OpenCV para filtragem e análise de imagens. A versão do OpenCV utilizada para viabilizar o funcionamento adequado do ORB-SLAM2 foi a 3.2.0.

3.1.7 Sobreposição gráfica

O componente de sobreposição gráfica é o responsável por desenhar um objeto tridimensional sobre o quadro de vídeo que é processado no MR-Leo. Todos os gráficos virtuais são desenhados sobre o quadro original que chega do dispositivo de usuário. O quadro original não é modificado de nenhuma outra forma. Existem dois objetos virtuais que podem ser desenhados sobre o quadro: os pontos que representam a nuvem de ponto e um objeto tridimensional.

No primeiro caso, o componente de sobreposição gráfica recebe como entrada as coordenadas da nuvem de pontos gerada pelo ORB-SLAM2 e as marca como pontos verdes sobre o quadro, como pode ser visto na Figura 3.5. Este desenho 2D usa funções internas do ORB-SLAM2 que, por sua vez, chamam a biblioteca de imagens do OpenCV.

[2]https://opencv.org/

Figura 3.5: Imagem detalhada que mostra os pontos verdes da nuvem de pontos e o objeto tridimensional renderizado.

No segundo caso, após a criação da nuvem de pontos, é possível adicionar um objeto virtual a ela. O algoritmo usado para escolher quais posições e rotações usar ao inserir os objetos em uma nuvem de pontos foi extraído de uma aplicação de código aberto desenvolvido pela equipe do ORB-SLAM2. Esse algoritmo analisa a nuvem de pontos e tenta encontrar pontos cujas posições compartilham uma superfície plana.

Para renderizar os objetos virtuais, o componente de sobreposição gráfica utiliza a API OpenGL3[3]. A renderização de gráficos 3D OpenGL é feita com a biblioteca de código aberto Pangolin[4].

3.2 eMR-Leo

Tendo como base o protótipo do MR-Leo, modificamos seu código para que o ORB-SLAM2 pudesse executar em GPU. Modificamos também o protótipo para adicionar a tarefa de detecção de objetos, a qual pode ser executada em CPU e GPU. Chamamos o protótipo resultante de eMR-Leo. A seguir, descrevemo as principais alterações realizadas.

[3]https://www.opengl.org/
[4]https://github.com/stevenlovegrove/Pangolin

3.2.1 SLAM com aceleração

Como parte da proposta deste trabalho, a arquitetura original do MR-Leo foi modificada com o objetivo de permitir o uso tanto de CPU quanto de GPU por parte da tarefa MAR de SLAM. A primeira alteração consistiu na substituição do ORB-SLAM2 originalmente utilizado por uma versão adaptada do ORB-SLAM2 que se beneficia do uso de GPU. Após uma investigação na literatura, a ferramenta escolhida foi a *CUDA accelerated ORB-SLAM* [Muzzini et al. 2023] que oferece uma versão modificada do ORB-SLAM2 capaz de paralelizar o processo de rastreamento.

Enquanto a versão original do MR-Leo executava a tarefa de SLAM exclusivamente na CPU, a nova versão implementada neste trabalho aproveita o paralelismo da GPU. O *CUDA accelerated ORB-SLAM* redesenha a construção da imagem, que é normalmente sequencial, de forma paralela para que diferentes *kernels Compute Unified Device Architecture* (CUDA) possam ser executados simultaneamente. Além de utilizar *kernels* já disponibilizados pelo ambiente CUDA, os autores desenvolveram novos para lidar com a tarefa de extração de pontos de interesse nos quadros de imagem. Dessa forma, a ferramenta reduz o tempo de processamento total ao aproveitar ao máximo a arquitetura paralela da GPU, minimizando também as transferências de dados entre CPU e GPU, o que contribui para uma execução mais rápida e eficiente do sistema.

3.2.2 Detecção de objetos

Para implementar a tarefa de detecção de objetos, foi escolhido o algoritmo *Yolo Only Look Once* (YOLO). O YOLO [Redmon et al. 2016] é um algoritmo de detecção de objetos que pertence à categoria de métodos baseados em aprendizado profundo, conforme discutido na Subseção 2.1.7. Embora o Fast R-CNN e o Faster R-CNN tenham trazido melhorias significativas em termos de eficiência, eles ainda seguem uma abordagem com múltiplos estágios de execução. O YOLO, por sua vez, realiza a detecção em um único estágio, o que o torna substancialmente mais rápido e adequado para aplicações com requisitos de baixa latência, como as de realidade aumentada móvel.

O YOLO utiliza a Darknet[5] como seu framework original para inferência, sendo este escrito em C e otimizado com CUDA para alto desempenho. A Darknet é responsável por carregar a configuração da rede neural e os pesos pré-treinados realizando o processamento da imagem para identificar objetos. O OpenCV[6] é um biblioteca de visão computacional que atua como suporte ao processo de detecção, sendo usado para capturar quadros de vídeo, realizar o pré-processamento das imagens, como redimensionamento e

[5] https://pjreddie.com/darknet/
[6] https://opencv.org/

normalização e também para o pós-processamento, que inclui desenhar caixas delimitadoras e aplicar supressão de não-máximos.

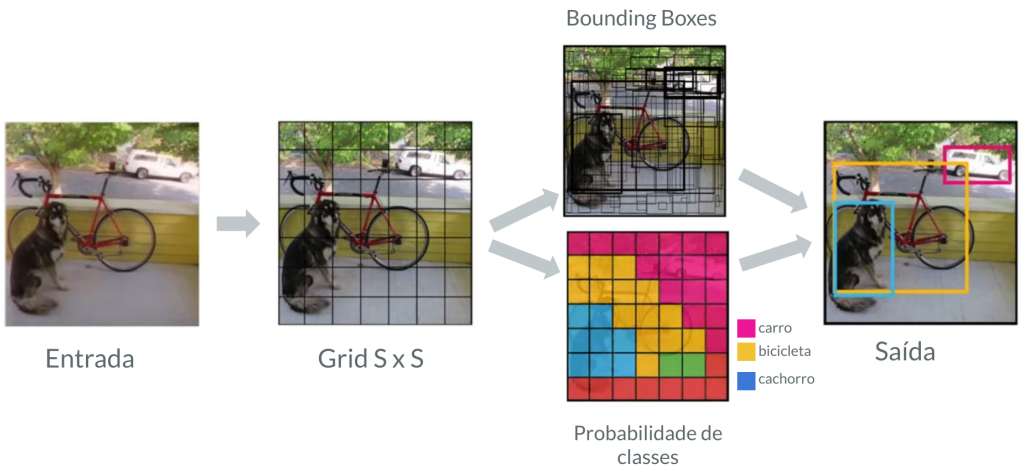


Figura 3.6: Etapas de processamento durante execução do YOLO. Adaptado de [Redmon et al. 2016].

Na Figura 3.6, é possível visualizar todas as etapas pelas quais uma imagem passa durante o processamento na rede neural do YOLO. Inicialmente, um quadro de imagem é capturado com o auxílio da biblioteca OpenCV, que também realiza seu redimensionamento para um tamanho fixo, padronizando a entrada e facilitando o processamento pela rede neural. Em seguida, a imagem é dividida em uma grade de tamanho S×S, que, no trabalho original, possui dimensão 7×7.

Cada célula da grade prediz uma quantidade fixa de caixas delimitadoras, e cada uma dessas caixas possui um nível de confiança associado. Esse nível de confiança indica tanto a probabilidade de que realmente exista um objeto dentro da área delimitada quanto a precisão do posicionamento da caixa.

Após essa etapa, o quadro é carregado na Darknet, que realiza a inferência e retorna as detecções. As caixas resultantes são então interpretadas e desenhadas sobre a imagem. Para eliminar caixas com baixa confiabilidade e reduzir a incidência de falsos positivos, o YOLO utiliza a técnica de supressão não máxima, que avalia o grau de sobreposição entre as caixas delimitadoras e mantém apenas aquelas com maior confiança e melhor posicionamento.

3.2.3 Implementação

Considerando que as tarefas de SLAM e detecção de objetos são fundamentais para aplicações de realidade aumentada e mista, e que o protótipo MR-Leo originalmente implementa apenas a tarefa de SLAM, o primeiro passo deste trabalho foi incorporar a tarefa MAR de detecção de objetos ao protótipo original.

Como já discutido na Seção 3.1. o MR-Leo possui foi implementado de forma modular para facilitar testes e modificações e para alcançar tal característica ele foi codificado em classes C++. A Figura 3.7 ilustra o conjunto de classes específicas do MR-Leo que são utilizadas para processar cada quadro de vídeo recebido pelo servidor de borda. A classe `ImageProcesser` atua como a superclasse, responsável pelo processamento dos quadros de vídeo no sistema. Ela define a interface e as operações básicas que são especializadas por três subclasses distintas, cada uma implementando uma forma específica de processamento. A primeira subclasse, `CannyFilter`, aplica o algoritmo *canny edge detector* [7] para identificar e destacar as bordas nos quadros de vídeo. A segunda subclasse, `EchoImage`, não realiza qualquer processamento nos quadros, simplesmente retornando o vídeo recebido para o cliente. A terceira subclasse, `OrbslamProcesser`, utiliza o ORB-SLAM2 para realizar a estimativa de posição, criando uma nuvem de pontos a partir dos quadros e posicionando objetos virtuais na cena com base no mapeamento gerado.

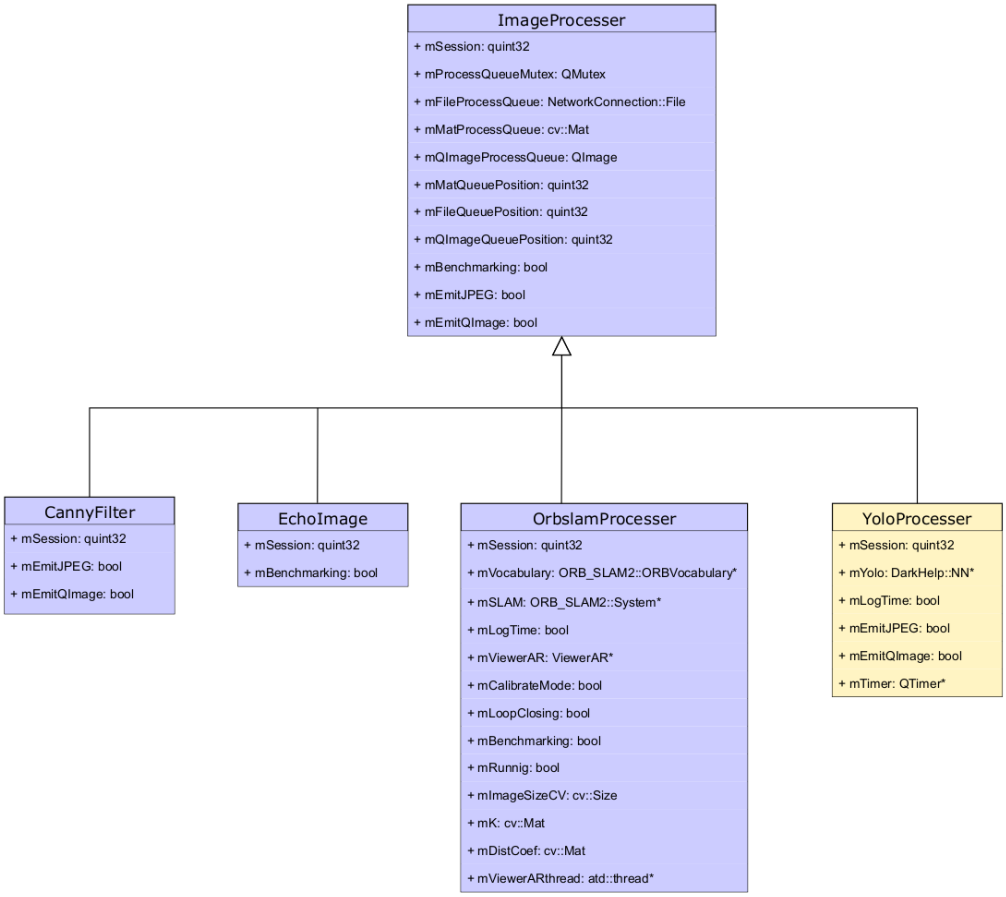


Figura 3.7: Diagrama de classes representando a superclasse e suas subclasses responsáveis pelo processamento de imagem no MR-Leo.

[7] https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

Figura 3.8: Resultado obtido a partir da execução da classe `YoloProcesser`.

Para adicionar um módulo de detecção de objetos ao protótipo, foi criada uma nova classe chamada `YoloProcesser`, destacada em amarelo no diagrama de classes da Figura 3.7. Essa classe utiliza o algoritmo YOLO para realizar as detecções. Para facilitar o acesso às funcionalidades da Darknet, foi adotado neste trabalho o empacotador (*wrapper*) DarkHelp[8]. Com isso, a classe `YoloProcesser` é capaz de receber quadros de vídeo e executar a detecção de objetos. O resultado dessa execução é um quadro de vídeo enriquecido com as informações de detecção, conforme ilustrado na Figura 3.8. Para que a classe implementada pudesse funcionar também foi necessário usar a versão 4.7.0 do OpenCV, por questões de compatibilidade e suporte aos recursos de placa gráfica.

Após viabilizar a detecção de objetos no MR-Leo, o segundo passo de implementação foi incorporar o uso de SLAM com aceleração por GPU. Para isso, foi necessário copiar o código-fonte do *CUDA accelerated ORB-SLAM* para um subdiretório do MR-Leo. A estrutura da versão acelerada do ORB-SLAM2 é semelhante à original, mas inclui arquivos com extensão CUDA, responsáveis por paralelizar partes do processo de rastreamento, como discutido na Seção 3.2.1.

Esses arquivos CUDA foram então compilados e transformados em objetos binários, que posteriormente foram copiados para a raiz do diretório do MR-Leo. O arquivo de configuração que descreve como o projeto do MR-Leo deve ser construído também foi ajustado para incluir instruções que acionam o *NVIDIA CUDA Compiler* (NVCC), possibilitando a integração dos objetos CUDA à implementação do MR-Leo. Com essas modificações, foi possível atingir o objetivo de acelerar a tarefa de SLAM

[8] https://github.com/stephanecharette/DarkHelp

dentro do protótipo.

3.3 Considerações finais

Neste capítulo, foi apresentada a arquitetura do protótipo MR-Leo, bem como as modificações realizadas para o desenvolvimento do enhanced MR-Leo (eMR-Leo). A descrição dos componentes do sistema — incluindo o servidor de borda, o dispositivo de usuário, a rede de comunicação, os mecanismos de transmissão de vídeo, o módulo de SLAM e o processo de renderização — permitiu compreender o funcionamento integrado do protótipo original e suas limitações. Com base nessa estrutura, foi possível propor e implementar duas funcionalidades adicionais: a introdução de uma tarefa de SLAM acelerada por GPU e a integração de um módulo de detecção de objetos.

CAPÍTULO 4

Avaliação de Desempenho e Caracterização da Carga de Trabalho

Neste capítulo, inicialmente apresentamos uma avaliação de desempenho das duas tarefas MARs implementadas no Capítulo 3, ou seja, SLAM e detecção de objetos. A Seção 4.1 descreve o cenário considerado nos experimentos, o ambiente de teste, as métricas avaliadas e os resultados obtidos. Essa seção discute também a demanda de recursos exigida pelas duas tarefas. A segunda parte deste capítulo (Seção 4.2) descreve a abordagem utilizada para caracterizar e modelar a carga de trabalho criada por essas tarefas, com foco na demanda de trabalho que chega para a borda. Essa seção discute também as percepções e modelos obtidos, os quais, como mencionado no Capítulo 1, podem ser usados na avaliação de algoritmos de alocação de recursos na borda da rede. A Seção 4.3 apresenta as considerações finais do capítulo.

4.1 Avaliação de desempenho

O objetivo da avaliação de desempenho é analisar como as duas tarefas, SLAM e detecção de objetos, utilizam os diferentes recursos computacionais e de comunicação e como esse uso impacta a vazão e a latência fim-a-fim da aplicação. Nas subseções seguintes, primeiramente descrevemos o cenário de experimentação (Seção 4.1.1), os ambientes de testes utilizados (Seção 4.1.2) e as métricas avaliadas (Seção 4.1.3). Em seguida discutimos a avaliação de desempenho da tarefa de SLAM (Seção 4.1.4) e de detecção de objetos (Seção 4.1.5).

4.1.1 Cenário de experimentação

Para garantir as mesmas condições em rodadas diferentes de um mesmo experimento, utilizamos um vídeo pré-gravado com 60 segundos de duração, em vez de capturas de imagens em tempo real. O vídeo em questão possui resolução de 640x480 *pixels* e taxa de 30 FPS. Cada quadro do vídeo é (a) reproduzido num *smartphone*, (b) codificado, (c)

transmitido para o servidor de borda, (d) decodificado, (e) processado, ou seja, enriquecido com elementos virtuais, (f) renderizado, (g) codificado, (h) transmitido de volta para o dispositivo (*smartphone*) com os elementos virtuais adicionados, (i) decodificado e (j) exibido para o usuário. Esse fluxo é ilustrado na Figura 4.1.

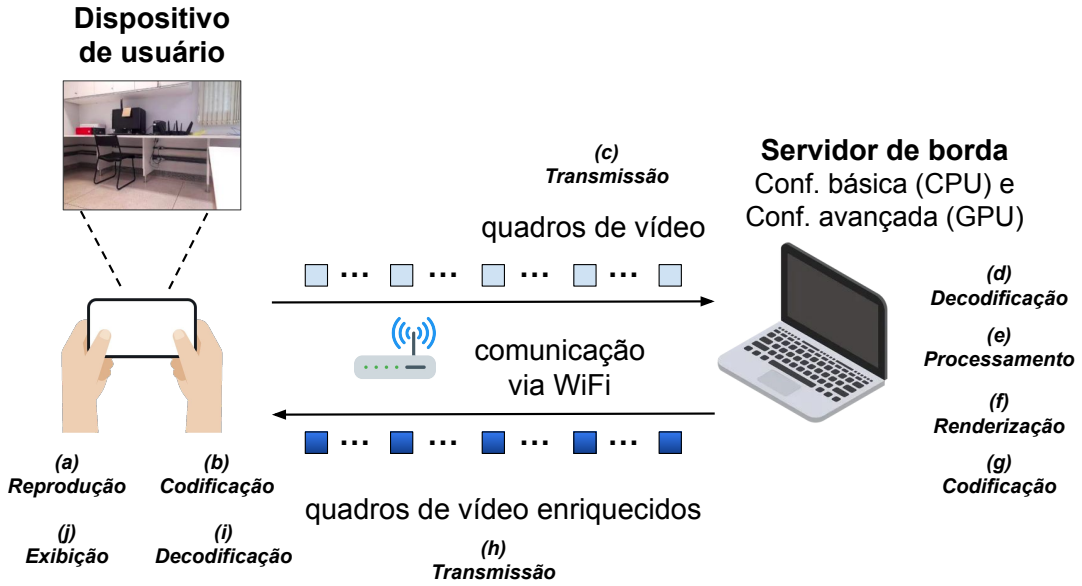


Figura 4.1: Ambientes de experimentação utilizados.

Para a tarefa de SLAM, o enriquecimento consiste na criação da nuvem de pontos e sua sobreposição no quadro processado. Para a tarefa de detecção de objetos, o enriquecimento da cena ocorre quando uma caixa delimitadora (*bounding box*) é desenhada em volta do objeto detectado. Na avaliação descrita a seguir, cada experimento é executado 30 vezes. Em todos os experimentos, usamos o *encoder/decoder* H.264, o qual é amplamente utilizado em *streaming* de vídeo.

4.1.2 Ambiente de testes

Para entender como as tarefas de SLAM e detecção de objetos utilizam diferentes recursos computacionais e de comunicação, utilizamos dois ambientes de experimentação. O primeiro, denominado configuração básica, utiliza um *laptop* Dell G15, equipado com um processador Intel Core i5-13450HX de 6 núcleos e 24 GB de memória RAM DDR5, como servidor de borda. O dispositivo do usuário é representado por um *smartphone* Samsung Galaxy A21s com processador octa-core Exynos 850 e memória RAM de 4 GB, executando o sistema operacional Android na versão 12. Esse dispositivo se conecta ao servidor de borda usando uma rede Wi-Fi de 2,4GHz (802.11b).

De fato, o grande problema da descarga de tarefas pra um servidor de borda é a latência de comunicação introduzida entre o cliente (dispositivo do usuário) e o servidor de borda. Nesta situação, para que a descarga para a borda da rede seja vantajosa, a redução no tempo médio de processamento de um quadro precisa ser maior que o tempo médio de latência de comunicação introduzido pela descarga de tarefas. Uma das formas de reduzir significativamente o tempo de processamento de tarefas computacionalmente intensivas é acelerá-las com o uso de GPUs. Dessa forma, avaliamos também o desempenho das tarefas de SLAM e detecção de objetos em uma configuração onde o servidor de borda possui uma GPU. Essa configuração, denominada avançada, usa como servidor de borda um *laptop* Lenovo Legion Slim 5, com um processador Intel Core i5-13420H de 8 núcleos e 16 GB de memória RAM DDR5 e uma GPU NVIDIA GeForce RTX 3050, com 6 GB de memória. Como na configuração básica, o dispositivo do usuário é o *smartphone* Samsung Galaxy A21s, o qual se conecta ao servidor de borda usando uma rede Wi-Fi de 2,4GHz (mesma rede da configuração básica). Em ambas as configurações, o servidor de borda executa o Ubuntu 24.04 LTS como sistema operacional. A Figura 4.1 ilustra as duas configurações utilizadas neste trabalho.

4.1.3 Métricas avaliadas

Para medir o desempenho das tarefas de SLAM e detecção de objeto nas duas configurações, investigamos dois aspectos distintos, porém importantes para garantir a QoE dos usuários de aplicações MAR. O primeiro aspecto é a latência fim-a-fim da tarefa, que deve ser baixa para que a aplicação responda de forma suficientemente rápida aos movimentos e interações do usuário. O segundo aspecto é a vazão (*throughput*) da tarefa, ou seja, a quantidade do fluxo de dados recebido e processado pelo servidor de borda. A vazão contribui para a QoE por estar relacionado à forma como os elementos virtuais adicionados (enriquecimento) se integram de maneira fluida com a realidade.

Neste trabalho, a latência fim-a-fim é investigada através do *Frame Round Trip Time* (FRTT), ou seja, o tempo decorrido deste o momento em que um quadro de vídeo é reproduzido no *smartphone* – passo (a) do *pipeline* da Figura 4.1, até o momento em que ele é exibido com o enriquecimento para o usuário – passo (j) do *pipeline* da Figura 4.1. Um componente importante do FRTT é o tempo de processamento do quadro. Como mencionado, a eficácia da descarga de tarefas para a borda depende significativamente de quanto o tempo de processamento de um quadro pode ser reduzido pela introdução de um elemento de borda com maior capacidade de processamento que o dispositivo. Dessa forma, avaliamos também o tempo de processamento de um quadro no servidor de borda.

Por outro lado, a vazão é medida através do FPS. Medimos também a porcentagem de quadros enviados para o servidor de borda que são efetivamente exibidos para o

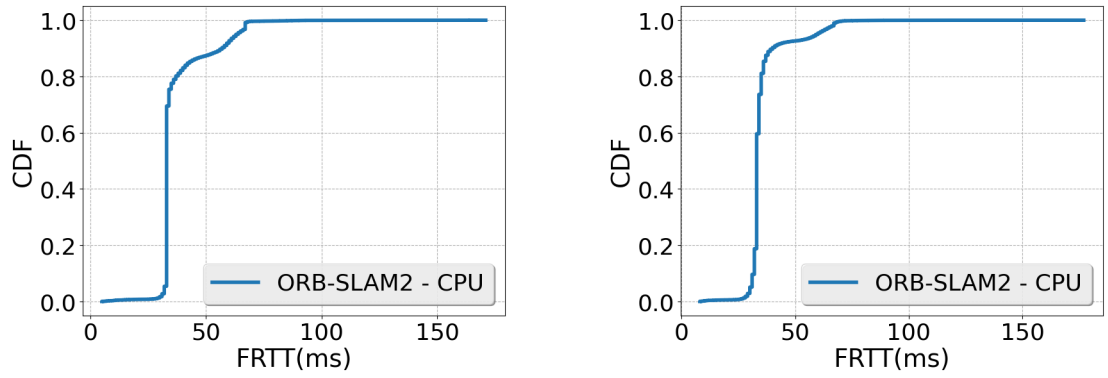
usuário, ou seja, o complemento da porcentagem do descarte de quadros ocasionados por acúmulo de latência.

Por fim, para caracterizar o perfil de consumo de recursos das duas tarefas, investigamos também a demanda de processamento (tanto de CPU quanto de GPU), memória, rede e energia requerida pelas tarefas.

4.1.4 Avaliação da tarefa de SLAM

Nesta seção, discutimos a avaliação de desempenho da tarefa de SLAM. Avaliamos o desempenho da tarefa tanto na configuração básica, como na avançada, começando pelo protocolo de transporte.

Nos trabalhos envolvendo MAR, o protocolo de transporte geralmente utilizado é o TCP devido à garantia que todos os quadros de vídeo chegarão e na ordem correta. Entretanto, isso gera um custo adicional devido aos pacotes de confirmação (*acks*). Para analisar esse compromisso, avaliamos a latência fim-a-fim e a vazão da tarefa tanto com o protocolo TCP quanto o UDP.

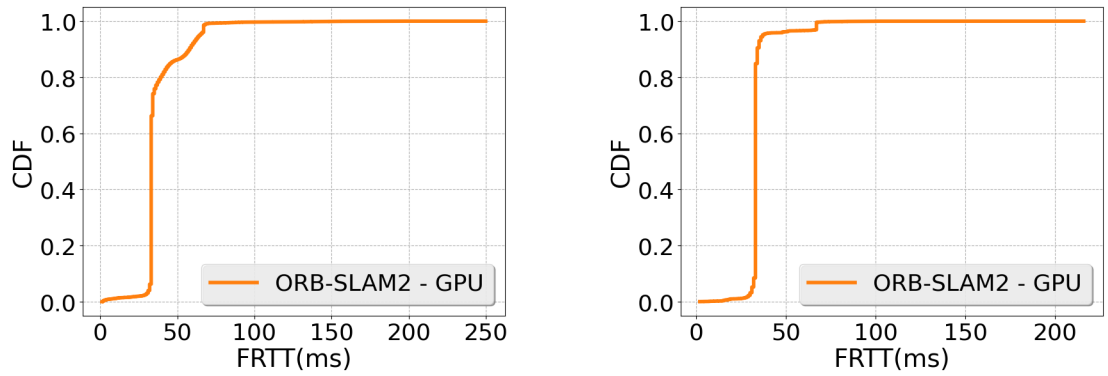


(a) *TCP* (b) *UDP*

Figura 4.2: Comparação entre os valores de FRTT para a tarefa de SLAM utilizando os protocolos de comunicação TCP (a) e UDP (b) para a configuração básica.

As Figuras 4.2(a) e 4.2(b) exibem as funções de distribuição cumulativa, ou *Cumulative Distribution Function* (CDF), para o FRTT considerando os protocolo de comunicação TCP e UDP, respectivamente, para a configuração básica. No protocolo TCP, o FRTT de 80% das amostras ficou abaixo de 38 ms, enquanto no UDP essa estatística alcançou alcançou 42 ms, evidenciando, portanto, que não houve um impacto significativo do protocolo de transporte utilizado na latência fim-a-fim.

O FRTT para a tarefa de SLAM na configuração avançada é mostrado nas Figuras 4.3(a) e 4.3(b). Usando o TCP, o FRTT de 80% das amostras ficou abaixo de 38 ms, enquanto com UDP, essa estatística alcançou 37ms. Novamente, observamos que o protocolo de transporte utilizado não afetou significativamente a latência fim-a-fim. De



(a) *TCP* (b) *UDP*

Figura 4.3: Comparação entre os valores de FRTT para a tarefa de sLAM utilizando os protocolos de comunicação TCP (a) e UDP (b) para a configuração avançada.

maneira análoga, o uso de GPU para a tarefa de SLAM também não trouxe impacto significativo para a latência fim-a-fim. Esse resultado é coerente com outros trabalhos da literatura [Song et al. 2018], que mostram que, mesmo em versões modificadas como o CUDA-accelerated ORB-SLAM, os ganhos de desempenho usando GPU podem ser menores do que o esperado, ou em alguns casos quase nulos. Mesmo quando se acelera partes como a extração de características, a maior parte da lógica do ORB-SLAM2 ainda permanece na CPU, como o rastreamento de pose, o ajuste, a construção e manutenção do grafo de mapas e a detecção de *loops*.

Analisamos também a vazão média, em termos de FPS, ou quadros por segundo, obtida pelos dois protocolos de transporte nas duas configurações. Os resultados são apresentados na Tabela 4.1. Novamente, os resultados confirmam as conclusões anteriores, mostrando que o protocolo e a infraestrutura de computação utilizada não tiveram impacto significativo na latência fim-a-fim e na vazão da tarefa de SLAM.

	TCP		**UDP**	
Configuração	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
Básica	27	1,75	22	8,49
Avançada	27	1,64	29	0,46

Tabela 4.1: Vazão média da tarefa de SLAM medida em FPS

Como mencionado anteriormente, para um *smartphone* a latência máxima tolerável é de 100 ms. Assim, os valores de latência fim-a-fim obtidos durante a execução da tarefa de SLAM no MR-Leo podem ser considerados adequados. Em relação à vazão, a literatura geralmente adota 24 FPS como o limite mínimo aceitável. Considerando esse critério, os valores de vazão obtidos para a tarefa de SLAM também se mostram satisfatórios, exceto na configuração básica com o protocolo UDP, que atingiu apenas 22 FPS.

Como não foi observada diferença significativa entre o uso dos diferentes protocolos de comunicação, o restante da análise realizada leva em conta somente os dados obtidos a partir dos experimentos feitos com o protocolo TCP.

Para analisar a latência-fim-a-fim de um quadro de forma mais detalhada, a métrica de FRTT foi decomposta em dois componentes: tempo de transmissão e o tempo de processamento. O tempo de transmissão corresponde à soma do tempo de transmissão do quadro do UE para o servidor de borda e do tempo de transmissão do quadro do servidor de borda de volta para o UE, depois de enriquecido. Para a configuração básica, foram registrados 38 ms de FRTT médio. Desses 38 ms, 24 ms são dedicados ao tempo de transmissão, ou seja, aproximadamente 63% do FRTT e sua redução depende essencialmente da tecnologia de comunicação sem fio utilizada e do formato de compressão do vídeo, uma vez que esse último tem impacto no tamanho das fases de codificação e decodificação do vídeo. Embora interessantes, a redução do tempo de transmissão não é o foco deste estudo e será tratado em trabalhos futuros, como será descrito no Capítulo 5.

O tempo de processamento (14 ms), por sua vez, representa cerca de 37% do FRTT e inclui o tempo associado ao serviço de comunicação, à criação da nuvem de pontos e à renderização do quadro enriquecido. O tempo do serviço de comunicação refere-se especificamente ao tempo de enfileiramento do quadro logo após sua recepção no servidor de borda, abrangendo apenas os processos internos de comunicação realizados pelo servidor de borda.

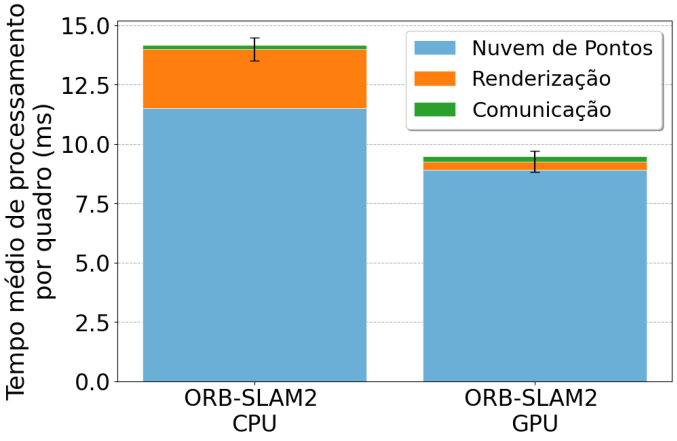


Figura 4.4: Análise detalhada do tempo médio gasto pelo SLAM para processar um quadro de vídeo.

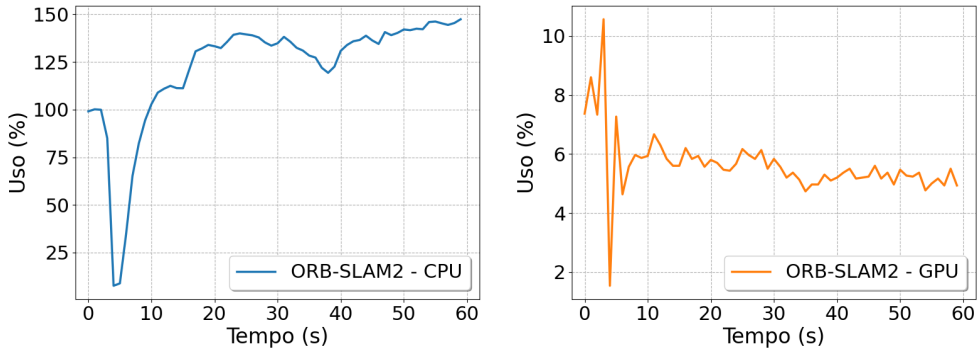
A Figura 4.4 ilustra o detalhamento do tempo de processamento nas duas configurações. Dos 14 ms totais para a configuração básica, foram gastos 11 ms com a criação da nuvem de pontos, 2 ms com a renderização e 1 ms para os serviços de comunicação. A maior redução observada na configuração avançada está relacionada ao tempo de renderização. Com o uso da GPU, esse tempo é reduzido em aproximadamente

quatro vezes, caindo para 0,58 ms. Embora o ORB-SLAM2, de modo geral, não dependa de uma placa gráfica para operar adequadamente, os resultados evidenciam que o uso da GPU proporciona ganhos significativos nos processos de renderização. Em relação ao tempo médio gasto com os serviços de comunicação, não foram observadas diferenças relevantes dentre os experimentos realizados em CPU e GPU, com ambos consumindo aproximadamente 1 ms.

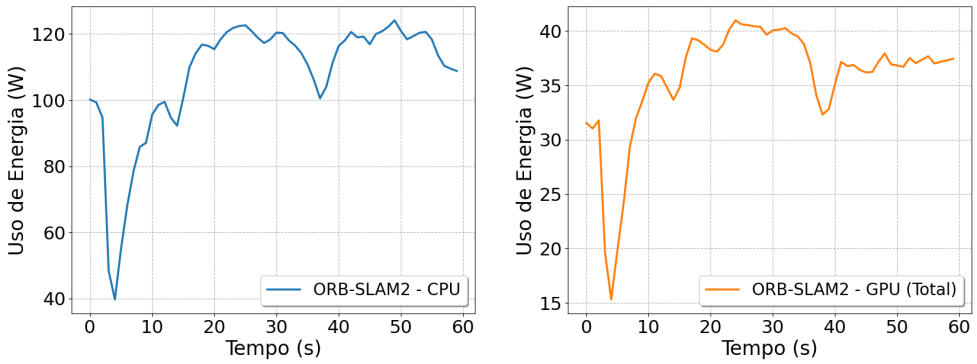
A Figura 4.5 ilustra o uso de recursos e o consumo energético do servidor de borda em decorrência da execução da tarefa de SLAM durante a transmissão de todo o vídeo. Na Figura 4.5(a) é apresentada a utilização de recursos (CPU e GPU). Na configuração básica, o uso máximo de CPU foi de 147%, indicando que a tarefa utilizou 100% de um núcleo e quase metade de outro, ou seja, o processamento demandou efetivamente dois núcleos dos oito disponíveis no servidor de borda. Na configuração avançada, o consumo de GPU é baixo em comparação à CPU, alcançando até 10%. O uso da GPU pelo ORB-SLAM2 foi modesto, reforçando que a maior parte das operações computacionais foi realizada pela CPU. Mesmo com acelerações específicas, o algoritmo não delega grande parte do processamento à GPU.

O consumo energético é ilustrado na Figura 4.5(b). Ele acompanha o padrão das curvas de uso de recursos, evidenciando que quando o percentual de uso aumenta, o consumo energético também tende a subir. Observamos que no momento de maior utilização de CPU, o consumo energético na configuração básica atinge aproximadamente 120 W. Por outro lado, no momento de maior utilização na configuração avançada, o consumo energético atinge 40 W. Na configuração avançada, o consumo energético foi obtido a partir da soma das medições fornecidas pelas ferramentas *nvidia-smi* e *powerstat*. Enquanto o *nvidia-smi* registra o consumo de energia diretamente da GPU, o *powerstat* coleta o consumo da CPU. Portanto, o valor final reflete o consumo energético geral do servidor de borda na configuração avançada e não apenas da GPU isoladamente. A diferença entre o consumo energético das duas configurações é explicado pelos modelos dos servidores de borda, sendo o servidor da configuração básica mais antigo e, portanto, menos eficiente energeticamente.

Avaliamos também o consumo de memória nos servidores de borda ao longo da transmissão do video. A Figura 4.6(a) mostra o consumo de RAM na configuração básica, enquanto a Figura 4.6(b) ilustra o consumo de RAM e *Video Random Access Memory* (VRAM) para a configuração avançada. Na configuração básica, observamos que o consumo de RAM cresce rapidamente nos 10 primeiros segundos devido ao carregamento do vocabulário no inicio da execução e à extração de características e estimação da pose nos primeiros quadros. Após esse instante, o ORB-SLAM2 entra em um estado de rastreamento contínuo, onde os dados são processados geralmente um quadro por vez. Neste estado, o número de pontos no mapa espacial pode crescer gradualmente, mas o



(a) *Consumo de CPU e GPU para o SLAM.*



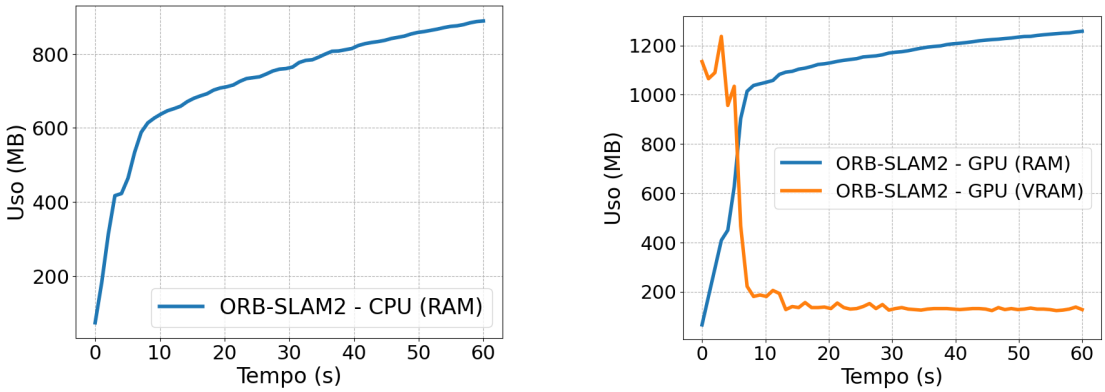
(b) *Consumo energético de CPU e GPU para o SLAM.*

Figura 4.5: Comparação entre o consumo de recursos (a) e o consumo energético (b) da tarefa de SLAM.

uso de memória é incremental e muito menor que no início. Na configuração avançada, observamos que o uso de RAM segue esse mesmo comportamento. O uso de VRAM também é maior nos primeiros segundos da transmissão, caindo drasticamente na fase de rastreamento, onde a extração de características (executada na GPU) é menos frequente.

Com o objetivo de compreender melhor a demanda computacional gerada pelo ORB-SLAM2, realizamos um novo conjunto de experimentos para coletar o número de instruções executadas por quadro de vídeo. Para isso, utilizamos o programa *perf*[1], que permite a contagem do número de instruções executadas por método durante a execução de um programa. Isolamos a contagem de instruções exclusivamente para a função responsável pela execução do ORB-SLAM2, permitindo medir a quantidade de instruções associadas ao algoritmo. Para esta coleta, o experimento com o vídeo de referência foi executado cinco vezes, utilizando apenas a configuração básica do servidor de borda, ou seja, com o servidor equipado exclusivamente com CPU. A Figura 4.7 mostra o histograma gerado para as cinco execuções do experimento. O número de instruções por quadro é majoritariamente concentrado em valores entre 300 e 600 milhões, indicando que

[1] https://man7.org/linux/man-pages/man1/perf.1.html



(a) *Consumo de RAM na configuração básica.*

(b) *Consumo de RAM e VRAM na configuração avançada.*

Figura 4.6: Comparação entre os consumos de RAM e VRAM para os experimentos executados na configuração básica 4.6(a) e na configuração avançada 4.6(b) do SLAM.

o comportamento computacional do ORB-SLAM2 é relativamente estável para a maioria dos quadros. O formato assimétrico com uma cauda para a direita indica que valores mais alto são alcançados, mas são mais raros em comparação com os picos observados. Apesar da concentração principal, o histograma revela a presença de quadros mais exigentes computacionalmente, que podem ser atribuídos a eventos como mudanças abruptas na cena dinâmica do vídeo de referência ou a necessidade de regerar a nuvem de pontos.

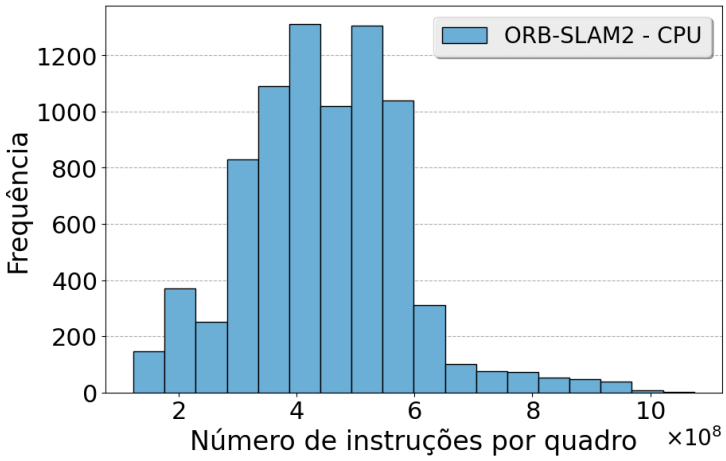
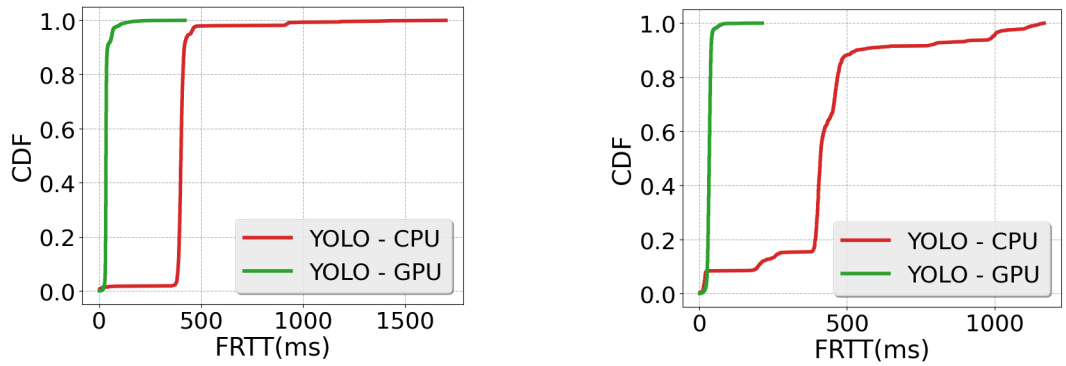


Figura 4.7: Histograma do número de instruções executadas por quadro de vídeo.

4.1.5 Avaliação da tarefa de detecção de objetos

Descrevemos, nesta seção, os resultados de desempenho obtidos com a tarefa de detecção de objetos.

Iniciamos discutindo a latência fim-a-fim, medida através do FRTT. Como na tarefa de SLAM, avaliamos o FRTT usando o protocolo TCP e UDP nas configurações básica e avançada. A Figura 4.8(a) exibe as CDFs para o FRTT para a configuração básica (CPU) e a avançada (GPU) usando o TCP. Na configuração avançada, o FRTT de 90% das amostras está abaixo de 41 ms, enquanto na configuração básica essa estatística está abaixo de 470 ms. Esse resultado demonstra que, diferentemente do SLAM, a detecção de objetos é altamente impactada pela arquitetura de processamento da borda, ou seja, a latência fim-a-fim da tarefa reduz significativamente quando executada em um hardware com aceleração. Isso ocorre porque o YOLO é uma rede convolucional profunda, e as operações de convolução são extremamente paralelizáveis. As CDF para o FRTT para a configuração básica e a avançada usando o UDP é mostrada na Figura 4.8(b). Novamente observamos uma grande redução na latência-fim-a-fim da tarefa ao utilizar uma arquitetura com GPU. O protocolo de comunicação utilizado, por sua vez, não afeta significativamente os resultados, como também observado no SLAM.



(a) *CDF para a latência com o protocolo TCP.* (b) *CDF para a latência com o protocolo UDP.*

Figura 4.8: Comparação entre os valores de FRTT para a tarefa de detecção de objetos utilizando diferentes protocolos de comunicação (TCP (a) e UDP (b))

Analisamos também a vazão média, em termos de FPS, referentes à tarefa de detecção de objetos, como apresentados na Tabela 4.8. Considerando os valores aceitáveis de latência fim-a-fim e taxa de quadros por segundo discutidos anteriormente — 100 ms e 24 FPS respectivamente — apenas a configuração avançada pode proporcionar uma experiência adequada para o usuário final quando ele faz uso de detecção de objetos. Na configuração avançada, a latência é reduzida em cerca de 90%, alcançando valores próximos de 40 ms. Os resultados de vazão também evidenciam a relevância do uso de GPU para essa tarefa: enquanto o uso de CPU não permite atingir os 24 FPS, com GPU é possível superar esse valor, alcançando até 29 FPS. Nas análises seguintes, consideramos apenas o uso do protocolo TCP.

Como na análise do desempenho do SLAM, também decompomos o FRTT em

	TCP		UDP	
Configuração	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
Básica	2	0,02	2	0,05
Avançada	29	1,21	29	0,34

Tabela 4.2: Vazão média da tarefa de detecção de objetos medida em FPS.

tempo de transmissão e tempo de processamento. O tempo de processamento, por sua vez, foi decomposto em tempo de detecção — que inclui a identificação do objeto e a criação da caixa delimitadora — e o tempo de comunicação. A Figura 4.9 evidencia novamente a diferença significativa entre o uso de CPU e GPU para essa tarefa. Na configuração básica, o tempo médio de processamento por quadro é de aproximadamente 430 ms, sendo 404 ms destinados à detecção e os 26 ms restantes referentes aos serviços de comunicação internos do servidor de borda. Já na configuração avançada, o tempo médio por quadro é reduzido em cerca de 98%, caindo para apenas 7,4 ms. Desse total, 6,6 ms são dedicados à detecção, enquanto os 0,8 ms restantes correspondem aos serviços de comunicação do servidor.

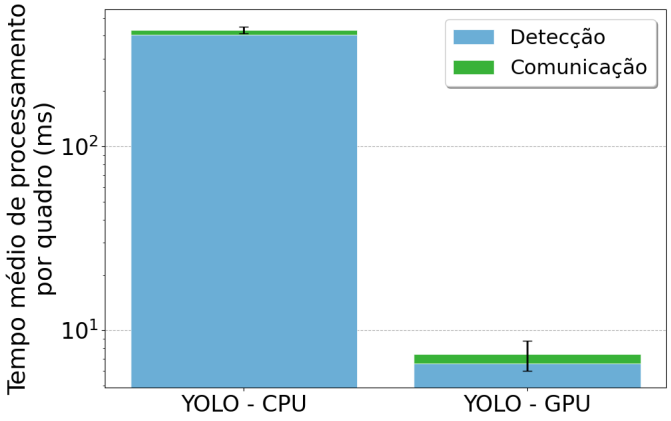
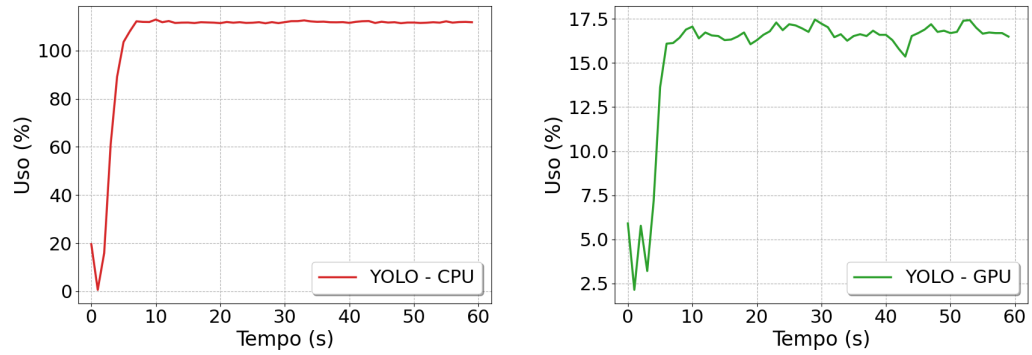


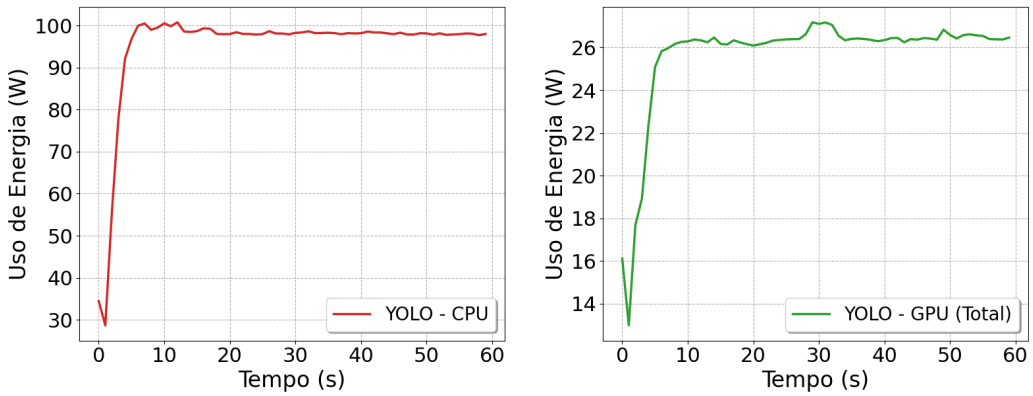
Figura 4.9: Análise detalhada do tempo médio gasto pelo YOLO para processar um quadro de vídeo.

Analisando a demanda computacional da tarefa de detecção de objetos, a Figura 4.10(a) ilustra o uso de CPU para o servidor de borda na configuração básica e de GPU para o servidor na configuração avançada. Na configuração básica, o uso de CPU atinge 113%, ou seja, são utilizados apenas 2 núcleos de processamento dos 8 disponíveis. No entanto, esse uso baixo de CPU está relacionado ao grande número de quadros descartados durante o experimento. Em outras palavras, como os quadros levaram muito tempo para serem processados, a grande maioria deles foram descartados para não comprometer a fluidez do vídeo. De fato, nossos testes mostram que 89% dos quadros recebidos pelo servidor de borda nesta configuração foram descartados pela aplicação porque quadros

mais recentes foram recebidos antes dos quadros que estavam na fila serem processados. Por outro lado, na configuração avançada, a tarefa de detecção de objeto, que é altamente paralelizável, utilizou 17,5% da GPU. Neste caso, o baixo uso do recursos de processamento está relacionado com a eficiência da GPU utilizada, uma vez que menos de 1% dos quadros foram descartados neste experimento. O consumo energético, ilustrado na Figura 4.10(b), acompanha, como esperado, o uso de CPU e GPU, atingindo um máximo de 100 W para a configuração básica e 27 W para a configuração avançada.



(a) *Uso de CPU e GPU.*

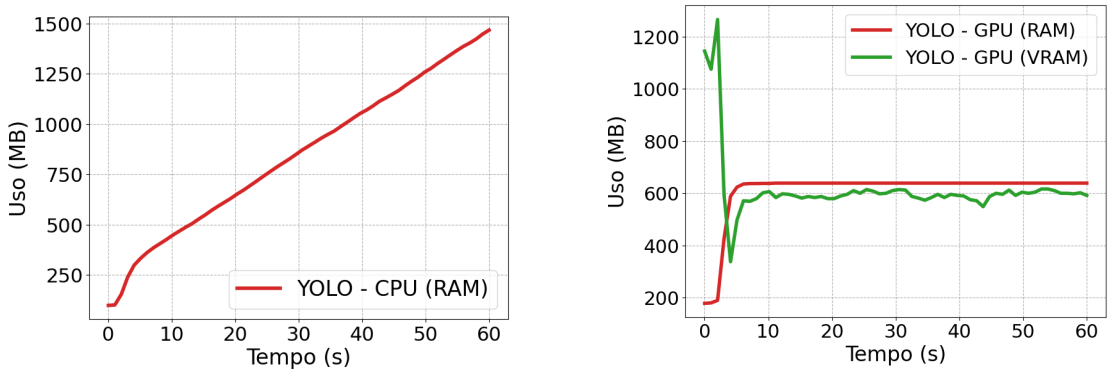


(b) *Consumo energético de CPU e GPU.*

Figura 4.10: Comparação entre o consumo de recursos (a) e o consumo energético (b) da tarefa de detecção de objetos.

O consumo de RAM e VRAM para a tarefa de detecção de objetos é mostrada na Figura 4.11. Na configuração básica, ilustrada na Figura 4.11(a), o consumo de RAM foi crescente. Como muitos quadros não estão sendo processados nesta configuração, acreditamos que a memória alocada por eles não está sendo adequadamente liberada e muitos dados estão sendo mantidos na memória de forma desnecessária. Por outro lado, na configuração avançada, como mostrado na Figura 4.11(b), o consumo de RAM é reduzido para cerca de 639 MB, caindo para menos da metade. Isso evidencia como o uso da GPU desloca e divide a carga de trabalho entre a memória RAM e a VRAM.

Durante o início do processamento, observa-se um pico de uso da VRAM, alcançando aproximadamente 1200 MB. Esse comportamento ocorre porque, nessa fase, o modelo e seus tensores são carregados na memória da GPU. Após a inicialização, o consumo de memória cai e se estabiliza, refletindo a fase de inferência, na qual o volume de dados processados permanece quase constante.



(a) *Consumo de RAM na configuração básica.*

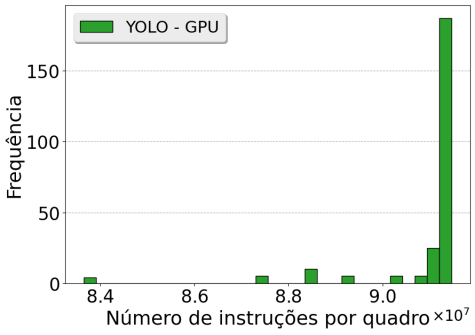
(b) *Consumo de RAM e VRAM na configuração avançada.*

Figura 4.11: Comparação entre os consumos de RAM e VRAM para a tarefa de detecção de objetos na configuração básica 4.11(a) e na configuração avançada 4.11(b).

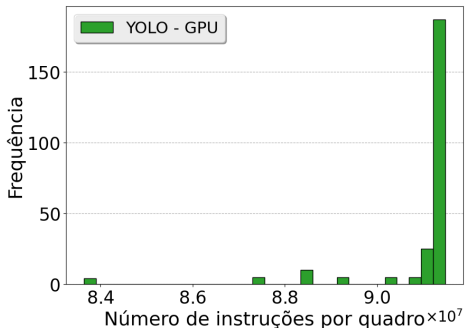
Para coletar o número de instruções executadas pela YOLO, usamos uma ferramenta específica da NVIDIA, chamada Nsight Compute CLI[2], que permite coletar dados diretamente da placa de vídeo, diferentemente do *perf* que não possui métricas relacionadas à GPU. O Nsight Compute CLI permite coletar especificamente o número de instruções executadas por cada kernel CUDA durante a execução do programa. Um kernel CUDA é uma função escrita em CUDA que é executada diretamente na placa gráfica. Para viabilizar a coleta, definimos um ponto de início e um ponto de finalização dentro da função `YoloProcessor`, e os resultados foram armazenados em um arquivo para facilitar a leitura e a posterior manipulação dos dados. Cada teste foi repetido cinco vezes, por 60 segundos. Com uma taxa de quadros igual a 30 era de se esperar uma quantidade próxima de 1800 quadros processados, mas a coleta detalhada de métricas com o Nsight introduziu uma sobrecarga significativa no sistema, reduzindo drasticamente a taxa de processamento de quadros para aproximadamente 250 quadros processados (cerca de 50 quadros por teste).

Para verificar o quanto o resultado obtido dependia do conteúdo do vídeo, refizemos o mesmo experimento para um vídeo de um campus universitário. Ambos os vídeos utilizados, que foram disponibilizados por [Toczé et al. 2020], possuem uma taxa de 30 FPS. Observamos na Figura 4.12 que a quantidade de instruções executadas por

[2]https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html



(a) *Número de instruções para o vídeo de referência.*



(b) *Número de instruções para o vídeo do campus.*

Figura 4.12: Comparação entre as quantidade de instruções executadas pelo YOLO a partir dos experimentos realizado com o vídeo de referência e com o vídeo do campus.

quadro para cada vídeo é muito similar. A maioria dos quadros apresenta cerca de 91 milhões de instruções executadas, tanto para o vídeo de referência (4.12(a)) quanto para o vídeo do campus universitário (4.12(b)). A constância no número de instruções entre vídeos distintos indica que o comportamento computacional do YOLO é determinado primariamente pela estrutura do modelo e pela quantidade de quadros processados, e não pelo conteúdo visual. Como os mesmos kernels CUDA são chamados para cada quadro, a quantidade de instruções executadas permanece praticamente constante, resultando em uma alta similaridade entre os vídeos analisados.

4.2 Caracterização da carga de trabalho

4.2.1 Caracterização da tarefa de SLAM

A partir dos dados experimentais coletados, realizamos uma análise estatística do ORB-SLAM2 com o objetivo de derivar um modelo estatístico que possa ser posteriormente implementado em um gerador de carga.

Para investigar qual distribuição melhor se ajusta aos dados coletados neste trabalho, utilizamos a biblioteca Python chamada *fitter*[3]. Essa biblioteca é capaz de analisar os dados e ranquear os melhores ajustes de acordo com um conjunto de distribuições teóricas pré-configuradas. Usamos também a análise visual da distribuição, permitindo observar de forma gráfica a adequação entre o histograma dos dados e a curva teórica estimada.

[3]https://fitter.readthedocs.io/en/latest/

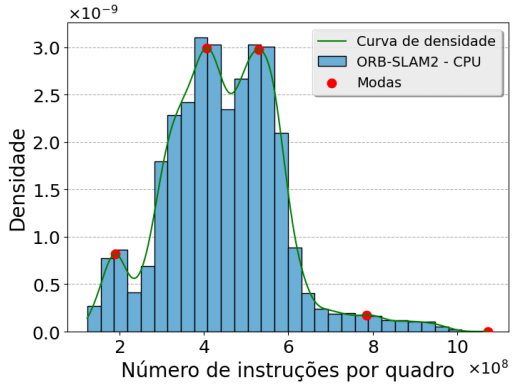


Figura 4.13: Pico dos valores mais observados de número de instruções.

Valores das modas	190.620.726
	407.493.566
	529.246.739
	786.069.839
	1.073.331.230

Tabela 4.3: Valores das modas atingidas pelo ORB-SLAM2.

A Figura 4.13 apresenta a curva de densidade e as modas observadas nas medições do número de instruções executadas por quadro pelo ORB-SLAM2, considerando apenas a aquitetura básica. A curva de densidade corresponde à função densidade de probabilidade — *Probability Density Function* (PDF) — e descreve como os valores do número de instruções estão distribuídos. Neste caso, a distribuição apresenta cinco modas, sendo duas delas os maiores picos de frequência. A primeira em 407,49 milhões de instruções (MI) e a segunda em 529,24 MI. Essas duas modas principais podem estar associadas aos momentos em que o algoritmo do ORB-SLAM2 realiza o rastreamento de um mapa já existente. Já a moda inicial, localizada no primeiro pico, em 190,62 MI, pode estar relacionada à fase de inicialização, quando o mapeamento ainda está sendo construído. Os últimos dois picos menores, como visto Tabela 4.3, mas que também apresentam valores elevados de número de instruções, podem estar relacionados a momentos em que o algoritmo perde o rastreamento e precisa se relocalizar no ambiente, aumentando, assim, o volume de instruções executadas.

Para os dados do ORB-SLAM2, a biblioteca *fitter* indicou que a distribuição que mais se ajustou aos dados foi a normal. A Figura 4.14 também confirma esse ajuste visual para a distribuição normal. A linha tracejada verde representa a distribuição real dos dados e evidencia as modas observadas anteriormente. O primeiro pico, localizado em aproximadamente 407 milhões de instruções, juntamente com o pico mais a esquerda, em 190 milhões, correspondem a uma primeira distribuição normal. Já o próximo pico em 529 milhões, com os outros picos menores e menos significativos em termos de frequência, são associados a uma segunda distribuição normal. A linha vermelha, por sua vez, ilustra a mistura Gaussiana dessas duas distribuições. A técnica de mistura Gaussiana é aplicada quando diferentes comportamentos são identificados em uma única amostra [McLachlan e Rathnayake 2014], como é o caso dos picos observados no ORB-SLAM2, que sugerem regimes de operação distintos para o mesmo algoritmo. A aplicação da mistura permite uma visualização mais suavizada dos dados.

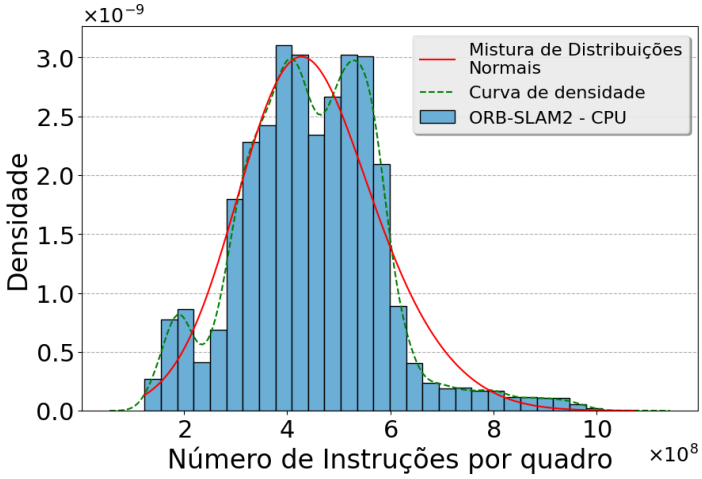


Figura 4.14: Mistura Gaussiana das duas curvas normais apresentadas pelos dados relativos ao número de instruções executadas por quadro pelo ORB-SLAM2.

Parâmetros das distribuições normais do ORB-SLAM2		
Distribuição Normal 1	μ	494.033.461,55
	σ	147.943.827,02
	%	47
Distribuição Normal 2	μ	404.543.944,05
	σ	110.994.236,45
	%	53

Tabela 4.4: Parâmetros das curvas normais ajustadas ao ORB-SLAM2.

Os parâmetros das distribuições normais são apresentados na Tabela 4.4. A primeira distribuição apresenta uma média de aproximadamente 494 MI, com desvio padrão de 148 milhões para 47% dos valores, enquanto a segunda possui média em torno de 405 MI e desvio padrão de 111 milhões para 53% dos valores.

Para verificar a qualidade dos ajustes (*fittering*), utilizamos um gráfico Q-Q (*Quantile-Quantile*), que compara os quantis da amostra coletada com os quantis da distribuição normal teórica. Caso os dados experimentais se aproximem da linha reta que representa a distribuição teórica, consideramos que há um ajuste aceitável. Como mostra a Figura 4.15, a proximidade dos pontos experimentais em relação à linha teórica ajustada indica que a modelagem foi satisfatória, com pequenas discrepâncias observadas apenas nas extremidades da distribuição.

4.2.2 Caraterização da tarefa de detecção de objetos

A partir dos dados experimentais coletados, realizamos também uma análise estatística do comportamento da YOLO para derivar um modelo estatístico que possa ser posteriormente implementado em um gerador de carga.

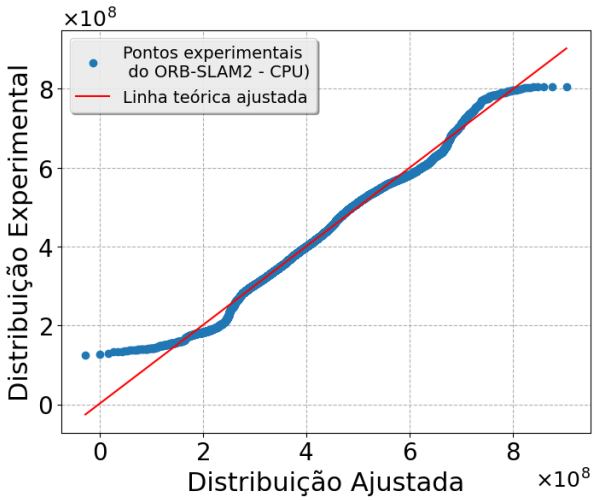


Figura 4.15: Gráfico Q-Q entre os dados experimentais e a distribuição ajustada para o ORB-SLAM2.

A biblioteca *fitter* indicou que a distribuição que melhor se ajusta à PDF do número de instruções executadas pela YOLO é a distribuição normal generalizada. Como visto na Figura 4.16, a distribuição apresenta duas modas, a primeira em 88,59 MI e a segunda em 91,38 MI. As modas evidenciam o comportamento regular do algoritmo do YOLO que aplica as mesmas operações para todos os quadros (leitura da imagem, convoluções, detecção, etc).

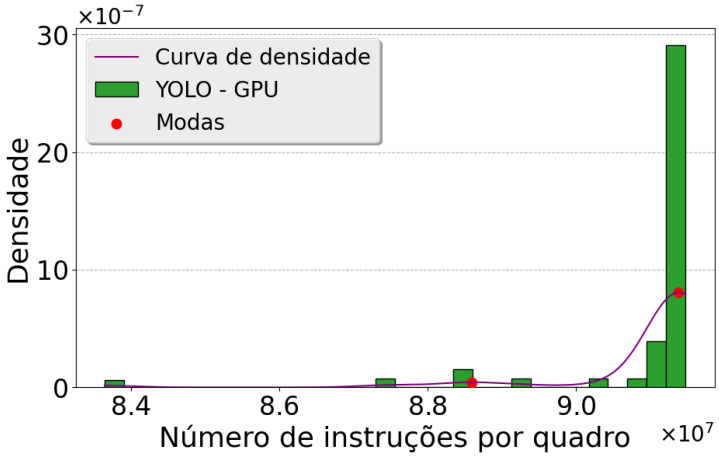


Figura 4.16: Modas obtidas no número de instruções do YOLO.

A distribuição experimental do YOLO indica a presença de duas curvas normais generalizadas, também chamadas de GenNorm. Embora uma das curvas — aquela gerada pela moda em 88 milhões de instruções — seja pouco representativa dentro da amostra, é necessário criar uma mistura das duas, que pode ser visualizada na Figura 4.17. A curva gerada pela GenNorm próxima da moda de 91 milhões de instruções é mais significativa e representa a maior parte da amostra de dados referentes ao YOLO.

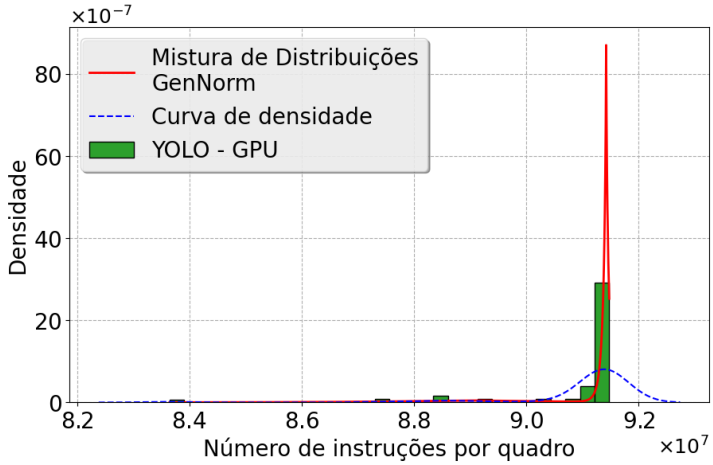


Figura 4.17: Mistura Gaussiana das duas distribuições normais generalizadas apresentadas pelos dados relativos ao número de instruções executadas por quadro pelo YOLO.

Parâmetros das distribuições normais generalizadas do YOLO		
Distribuição GenNorm 1	μ	91.419.712
	α	40.427,32
	β	0,80
	%	82
Distribuição GenNorm 2	μ	89.062.240,77
	α	2.052.161,90
	β	1,25
	%	18

Tabela 4.5: Parâmetros das curvas normais generalizadas ajustadas ao YOLO.

As distribuições normais generalizadas possuem três parâmetros: o *shape* (β), que determina o formato da curva (valores de $\beta > 2$ indicam um formato de "sino"mais concentrado, enquanto $\beta < 2$ indica caudas mais pesadas); o *location* (μ), que representa o centro da distribuição, ou a posição média; e o *scale* (α), que controla a dispersão dos dados. Todos esses parâmetros, relacionados às duas distribuições GenNorm do YOLO, podem ser visualizados na Tabela 4.5.

A primeira distribuição GenNorm representa 82% dos dados e possui uma média em torno de 91 milhões de instruções. O valor de α, em torno de 40 mil, indica que a maioria dos quadros está próxima do valor médio, e o β menor que 2 revela a presença de uma cauda pesada à esquerda da moda. A segunda distribuição GenNorm é menos representativa, correspondendo a apenas 18% do total da amostra. O valor de β, também menor que 2, indica uma cauda, embora menos acentuada que a da primeira distribuição. Sua média está em torno de 89 milhões de instruções, e o alto valor de α sugere uma dispersão de valores bem maior.

O gráfico Q-Q da Figura 4.18 mostra que a distribuição ajustada apresenta alguma aderência aos dados experimentais, principalmente na região central. Os desvios observados nas extremidades indicam limitações na captura dos comportamentos de cauda, que no caso do YOLO apresenta uma cauda muito longa e pesada antes de iniciar o pico dos valores mais concentrados, dificultando ajustes perfeitos nas extremidades da distribuição.

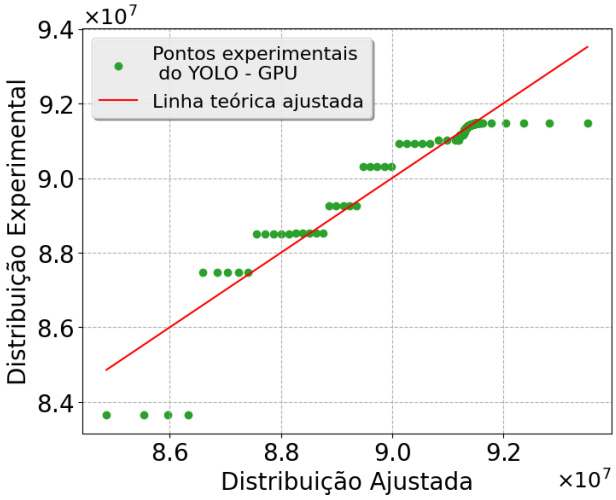


Figura 4.18: Gráfico Q-Q entre os dados experimentais e a distribuição ajustada para o YOLO.

4.2.3 Modelos derivados da caracterização

A partir de todos os dados obtidos na etapa de caracterização, criamos dois modelos: um para a aplicação e outro estatístico que descreve a carga de trabalho das tarefas da aplicação. O modelo da aplicação é um grafo direcionado que representa a aplicação MAR, onde os nós são as tarefas existentes e as arestas são as formas de comunicações possíveis entre os nós. A Figura 4.19 mostra o modelo de aplicação elaborado.

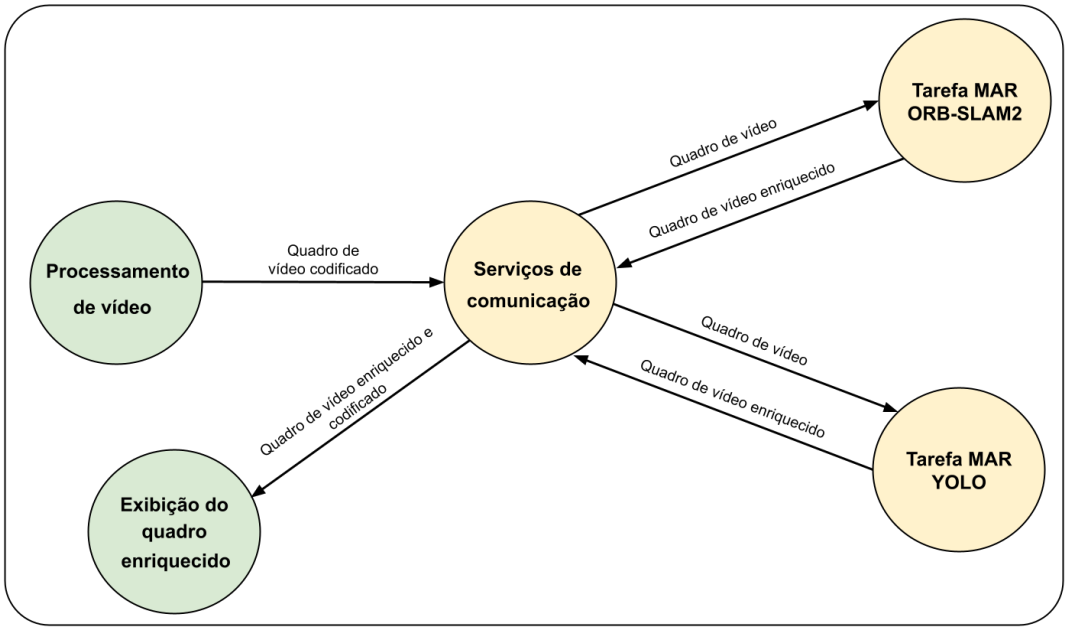


Figura 4.19: Modelo da aplicação

O processamento de vídeo representa a captura da cena realizada pelo UE, que constitui a entrada do sistema. Após passar pelos serviços de comunicação, o quadro pode ser processado pela tarefa MAR (ORB-SLAM2 ou YOLO). Ambas fornecem um quadro de vídeo enriquecido, que é posteriormente exibido no dispositivo do usuário. As funções ilustradas em verde são aquelas processadas no próprio dispositivo do usuário, enquanto as ilustradas em amarelo correspondem às tarefas descarregadas para a borda.

O modelo estatístico da carga de trabalho para um servidor de borda que executa as tarefas de SLAM e detecção de objeto é apresentado na Tabela 4.6. Cada uma das tarefas foi caracterizada para o hardware onde a tarefa obteve o melhor desempenho. Dessa forma, o modelo de carga para o SLAM é focado em CPU, o da YOLO foi centrado em GPU.

A tarefa de processamento de vídeo é periódica. Como o video é reproduzido numa taxa de 30 FPS, um quadro de vídeo chega ao servidor de borda para ser processado a cada 33 ms. Essa tarefa dura 33 ms e transfere aproximadamente 39 KB de dados (o tamanho de um quadro de vídeo). A quantidade de instruções executada para processar cada quadro, ou seja, a demanda por processamento, depende da tarefa.

Como discutido na Seção 4.2.1, para o SLAM, essa é uma demanda de CPU e é representada pelas distribuições descritas na Tabela 4.4 e reproduzida na Tabela 4.6. Com relação à demanda de memória RAM consumida no servidor de borda pela tarefas de SLAM observamos um valor médio de 408 MB quando o serviço de borda executa sem nenhum cliente conectado. Por outro lado, cada cliente (UE) que se conecta ao servidor, gera um consumo adicional médio de RAM de 288 MB.

Hardware		Atividade	Modelo
Tarefa ORB-SLAM2	CPU	Chegada da tarefa	Periodicamente, a cada 33 ms
		Computação	Para 43%: $\mathcal{N}(\mu = 494\,\text{MI}, \sigma = 147{,}97\text{MI})$ Para 53%: $\mathcal{N}(\mu = 404\,\text{MI}, \sigma = 110{,}99\text{MI})$
		Comunicação	39 kB por quadro
		Memória usada pelo servidor de borda somente	408 MB
		Memória adicional para cada dispositivo de usuário	288 MB
Tarefa YOLO	GPU	Chegada da tarefa	Periodicamente, a cada 33 ms
		Computação	Para 82%: $\mathcal{GN}(\mu = 91\,\text{MI}, \alpha = 0{,}40\text{MI}, \beta = 0{,}80)$ Para 18%: $\mathcal{GN}(\mu = 89\,\text{MI}, \alpha = 2{,}05\text{MI}, \beta = 1{,}25)$
		Comunicação	39 kB por quadro
		Memória usada pelo servidor de borda somente	208 MB
		Memória (RAM e VRAM) adicional para cada dispositivo de usuário	416 MB para cada memória

Tabela 4.6: Visão geral do modelo de geração de cargas para as tarefas ORB-SLAM2 e YOLO.

	TCP		UDP	
Configuração	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
Básica	0,12%	x	0,13%	x
Avançada	0,60%	x	0,28%	x

Tabela 4.7: Quadros perdidos pra tarefa de SLAM.

Como discutido na Seção 4.2.2, para a detecção de objetos, essa é uma demanda de GPU e é representada pelas distribuições descritas na Tabela 4.5 e reproduzida na Tabela 4.6. Com relação à demanda de memória RAM consumida no servidor de borda pela tarefas de SLAM observamos um valor médio de 208 MB quando o serviço de borda executa sem nenhum cliente conectado. Por outro lado, cada cliente (UE) que se conecta ao servidor, gera um consumo adicional médio de RAM e de VRAM de aproximadamente 416 MB

4.3 Considerações finais

As análises realizadas neste capítulo permitiram uma compreensão aprofundada do comportamento das duas tarefas MAR — SLAM e detecção de objetos — implemen-

	TCP		UDP	
Configuração	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
Básica	90%	x	91%	x
Avançada	0,17%	x	0,17%	x

Tabela 4.8: Quadros perdidos pra tarefa de detecção.

tadas no protótipo MR-Leo e sua versão estendida, o eMR-Leo. Através de experimentos foi possível quantificar a latência fim-a-fim, a vazão, e o consumo de recursos computacionais e energéticos em diferentes configurações de hardware. Os resultados revelaram que, enquanto a tarefa de SLAM apresenta desempenho relativamente estável e satisfatório mesmo sem grande aproveitamento da GPU, o uso da aceleração gráfica com YOLO para a tarefa de detecção de objetos proporcionou ganhos expressivos em eficiência e redução de latência. A caracterização estatística da carga de trabalho, baseada na contagem de instruções e no ajuste de distribuições teóricas, permitiu derivar modelos para futuras avaliações e simulações de alocação de recursos na borda. Com isso, o capítulo discutiu uma avaliação quantitativa e contribuições para a modelagem de sistemas MAR em contextos de computação de borda.

CAPÍTULO 5

Conclusão e Trabalhos Futuros

Esta dissertação teve como objetivo investigar como a computação de borda pode beneficiar aplicações de realidade aumentada móvel, com foco em duas tarefas computacionalmente intensivas e fundamentais nesse contexto: o mapeamento e localização simultâneos (SLAM) e a detecção de objetos. Para tanto, foram realizadas extensões no protótipo MR-Leo, que passou a incorporar versões otimizadas dessas tarefas tanto para CPU quanto para GPU, permitindo uma avaliação de desempenho e consumo de recursos em diferentes ambiente de borda.

Para a primeira pergunta de pesquisa formulada neste trabalho – "Quais são as principais tarefas que compõem um fluxo típico de aplicações MAR?" – concluímos, mediante pesquisa bibliográfica, que SLAM e detecção de objetos são as tarefas mais referenciadas pelos trabalhos da literatura.

Para a segunda pergunta de pesquisa – "Existem implementações reais de código aberto representativas dessas tarefas?" – encontramos, através de pesquisa bibliográfica e buscas em repositórios de código aberto, duas bibliotecas, ORB-SLAM2 e YOLO amplamente utilizadas para as tarefas de SLAM e para e detecção de objetos, respectivamente.

Para a terceira pergunta – "Em termos de latência fim-a-fim e vazão, como a computação de borda pode impactar no desempenho dessas tarefas, considerando diferentes arquiteturas de hardware (CPU e GPU)?" a extensão do MR-Leo e a implementação do eMR-Leo nos permitiu concluir que a tarefa de SLAM, baseada em ORB-SLAM2 apresenta um desempenho aceitável quando executada em CPU e não apresenta ganhos significativos de desempenho quando executada em GPU. Isso ocorre porque o ORB-SLAM2 favorece tarefas sequenciais, o que dificulta a paralelização massiva típica de GPUs. A tarefa de detecção de objetos, baseada em YOLO, só obteve resultado satisfatório quando acelerada por GPU, evidenciando a necessidade desse tipo de recurso na borda para a execução dessa tarefa.

A extensão do MR-Leo e a implementação do eMR-Leo também nos permitiu responder à quarta pergunta de pesquisa – "Qual é o perfil de consumo de recursos dessas tarefas em diferentes arquiteturas de hardware?". Através dessas implementações, conseguimos quantificar e caracterizar o perfil das tarefas de SLAM e detecção de

objetos em termos de recursos de processamento, armazenamento e comunicação na borda da rede, bem como o tempo de chegada das tarefas no servidor de borda e sua duração. Em termos de processamento, a tarefa de SLAM possui um comportamento mais heterogêneo. Apesar da concentração principal em torno da média, a distribuição empírica revela a presença de quadros mais exigentes computacionalmente, que podem ser atribuídos a eventos como mudanças abruptas na cena dinâmica do vídeo de referência ou a necessidade de regerar a nuvem de pontos. A tarefa de detecção de objetos, por sua vez, possui um comportamento mais homogêneo em relação ao processamento de quadros, exigindo aproximadamente a mesma quantidade de instruções por quadro. Isso ocorre porque a YOLO aplica as mesmas operações para todos os quadros.

Por fim, utilizando a caracterização do perfil das tarefas de SLAM e detecção de objetos, derivamos um modelo de aplicação que pode imitar fielmente o comportamento das duas tarefas, respondendo portanto à quinta e ultima pergunta de pesquisa – "Como derivar modelos de cargas de trabalho para aplicações MAR levando em consideração a infraestrutura de borda que se mostrou mais adequada para a execução de cada tarefa?". Com relação a essa última pergunta, é importante destacar a importância do modelo de carga de trabalho derivado neste trabalho. De fato, grande parte do tempo consumido nesta dissertação foi gasto com a integração de diferentes bibliotecas e resolução de problemas relacionados com compatibilidade de dependências e versões de software. Estes problemas dificultam a investigação de estratégias de alocação de recursos e orquestração de serviços em computação de borda, essenciais para melhorar o QoE dos usuários MAR. Ao derivar um modelo de carga abrangente em termos de tarefas e de arquiteturas de hardware, este trabalho contribui para mitigar esse problema.

Em resumo, como contribuição, este trabalho oferece uma caracterização detalhada do comportamento computacional de tarefas MAR em um ambiente realista com suporte a descarregamento na borda, integrando diferentes componentes de software e hardware e viabilizando decisões mais informadas sobre a alocação dessas tarefas. Além disso, a criação de um modelo estatístico baseado nos dados coletados representa uma contribuição original, com potencial para subsidiar a construção de geradores de carga e o desenvolvimento de mecanismos automáticos de orquestração em arquiteturas distribuídas para aplicações XR.

Uma direção futura de pesquisa consiste em estender simuladores de computação de borda, como por exemplo o EdgeCloudSim, com o modelo de carga derivado neste trabalho e comparar o desempenho de diferentes políticas de orquestração usando o modelo de carga proposto. Também é interessante comparar o quanto um modelo de carga de trabalho teórico se difere do nosso modelo, o qual foi obtido a partir de protótipos reais, bem como o impacto dessas diferenças sobre as estratégias de alocação de recursos.

Por fim, a tecnologia de rede de acesso tem um impacto importante na compu-

tação de borda que, devido ao tempo, não foi possível investigar neste trabalho. Nesse sentido, uma direção futura de pesquisa consiste em avaliar os protótipos aqui desenvolvidos em tecnologias de comunicação sem fio com maior capacidade como 5G e Wi-Fi 6E e Wi-Fi 7. Essas tecnologias possuem menor latência e maior estabilidade para aplicações críticas.

Referências Bibliográficas

[3GPP 2020]3GPP. *5G; System architecture for the 5G System (5GS) (3GPP TS 23.501 version 16.6.0 Release 16)*. [S.l.], 2020.

[3GPP 2022]3GPP. *Technical Specification Group Services and System Aspects; Architecture for enabling Edge Applications; (Release 18)*. [S.l.], 2022.

[Ahvar, Orgerie e Lebre 2022]AHVAR, E.; ORGERIE, A.-C.; LEBRE, A. Estimating energy consumption of cloud, fog, and edge computing infrastructures. *IEEE Transactions on Sustainable Computing*, v. 7, n. 2, p. 277–288, 2022.

[Alriksson et al. 2021]ALRIKSSON, F. et al. Xr and 5g: Extended reality at scale with time-critical communication. *Ericsson Technology Review*, v. 2021, n. 8, p. 2–13, 2021.

[Alsadik e Karam 2021]ALSADIK, B.; KARAM, S. The simultaneous localization and mapping (slam)-an overview. *Journal of Applied Science and Technology Trends*, v. 2, n. 02, p. 147–158, 2021.

[Aulinas et al. 2008]AULINAS, J. et al. The slam problem: a survey. *Artificial Intelligence Research and Development*, IOS Press, p. 363–371, 2008.

[Barros et al. 2022]BARROS, A. M. et al. A comprehensive survey of visual slam algorithms. *Robotics*, v. 11, n. 1, 2022. ISSN 2218-6581. Disponível em: <https://www.mdpi.com/2218-6581/11/1/24>.

[Bekele et al. 2018]BEKELE, M. K. et al. A survey of augmented, virtual, and mixed reality for cultural heritage. *J. Comput. Cult. Herit.*, Association for Computing Machinery, New York, NY, USA, v. 11, n. 2, 2018.

[Cao et al. 2023]CAO, J. et al. Mobile augmented reality: User interfaces, frameworks, and intelligence. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 55, n. 9, 2023.

[Carmigniani et al. 2011]CARMIGNIANI, J. et al. Augmented reality technologies, systems and applications. *Multimed Tools Appl*, v. 51, p. 341–377, 2011.

[Chatzopoulos et al. 2017]CHATZOPOULOS, D. et al. Mobile augmented reality survey: From where we are to where we go. *IEEE Access*, v. 5, p. 6917–6950, 2017.

[Chen et al. 2017]CHEN, Z. et al. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In: *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. [S.l.: s.n.], 2017.

[Cruz, Achir e Viana 2022]CRUZ, P.; ACHIR, N.; VIANA, A. C. On the edge of the deployment: A survey on multi-access edge computing. *ACM Comput. Surv.*, v. 55, n. 5, 2022.

[Dai et al. 2017]DAI, A. et al. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Trans. Graph.*, Association for Computing Machinery, New York, NY, USA, v. 36, n. 4, 2017.

[Dalal e Triggs 2005]DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. [S.l.: s.n.], 2005. v. 1, p. 886–893 vol. 1.

[Davison et al. 2007]DAVISON, A. J. et al. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 29, n. 6, p. 1052–1067, 2007.

[ETSI ISG MEC 2016]ETSI ISG MEC. *Mobile Edge Computing (MEC); Framework and Reference Architecture*. [S.l.], March 2016. Disponível em: <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf>.

[Felzenszwalb, McAllester e Ramanan 2008]FELZENSZWALB, P.; MCALLESTER, D.; RAMANAN, D. A discriminatively trained, multiscale, deformable part model. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2008. p. 1–8.

[Forster, Pizzoli e Scaramuzza 2014]FORSTER, C.; PIZZOLI, M.; SCARAMUZZA, D. Svo: Fast semi-direct monocular visual odometry. In: IEEE. *2014 IEEE international conference on robotics and automation (ICRA)*. [S.l.], 2014. p. 15–22.

[Girshick et al. 2014]GIRSHICK, R. et al. *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*. 2014. 580-587 p.

[Global System for Mobile Communications Association (GSMA) 2019]Global System for Mobile Communications Association (GSMA). *Energy Efficiency: An Overview*. 2019. https://www.gsma.com/solutions-and-impact/technologies/networks/gsma_resources/energy-efficiency-an-overview/.

[GSM Association 2021]GSM Association. *Operator Platform Telco Edge Requirements*. [S.l.], June 2021. Disponível em: <https://www.gsma.com/futurenetworks/resources/operator-platform-telco-edge-requirements/>.

[Gubbi et al. 2013]GUBBI, J. et al. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, v. 29, n. 7, p. 1645–1660, 2013. ISSN 0167-739X. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X13000241>.

[Hassan, Yau e Wu 2019]HASSAN, N.; YAU, K.-L. A.; WU, C. Edge computing in 5g: A review. *IEEE Access*, v. 7, p. 127276–127289, 2019.

[Hu et al. 2020]HU, M. et al. Object detecting augmented reality system. In: IEEE. *2020 IEEE 20th International Conference on Communication Technology (ICCT)*. [S.l.], 2020. p. 1432–1438.

[Khan et al. 2019]KHAN, W. Z. et al. Edge computing: A survey. *Future Generation Computer Systems*, v. 97, p. 219–235, 2019.

[Kim et al. 2023]KIM, H. et al. Mobile edge computing enabler layer: Edge-native application architecture for mobile networks. *IEEE Communications Standards Magazine*, IEEE, v. 7, n. 4, p. 50–59, 2023.

[Larsen, Checko e Christiansen 2019]LARSEN, L. M. P.; CHECKO, A.; CHRISTIANSEN, H. L. A Survey of the Functional Splits Proposed for 5G Mobile Crosshaul Networks. *IEEE Communications Surveys Tutorials*, v. 21, n. 1, p. 146–172, 2019.

[Lindqvist 2019]LINDQVIST, J. *Edge Computing for Mixed Reality*. Dissertação (Mestrado) — Linköping University, Suécia, 2019.

[Liu, Li e Gruteser 2019]LIU, L.; LI, H.; GRUTESER, M. Edge assisted real-time object detection for mobile augmented reality. In: *The 25th annual international conference on mobile computing and networking*. [S.l.: s.n.], 2019. p. 1–16.

[Malta, Farinha e Mendes 2023]MALTA, A.; FARINHA, T.; MENDES, M. Augmented reality in maintenance—history and perspectives. *Journal of Imaging*, v. 9, n. 7, 2023. ISSN 2313-433X. Disponível em: <https://www.mdpi.com/2313-433X/9/7/142>.

[Mansoub, Abri e Yarıcı 2019]MANSOUB, S. K.; ABRI, R.; YARICI, A. Concurrent real-time object detection on multiple live streams using optimization cpu and gpu resources in yolov3. *SIGNAL*, p. 23–28, 2019.

[Masood e Egger 2019]MASOOD, T.; EGGER, J. Augmented reality in support of industry 4.0: Implementation challenges and success factors. *Robotics and Computer-Integrated Manufacturing*, v. 58, p. 181–195, 2019.

[McLachlan e Rathnayake 2014]MCLACHLAN, G. J.; RATHNAYAKE, S. On the number of components in a gaussian mixture model. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Wiley Online Library, v. 4, n. 5, p. 341–355, 2014.

[Moro et al. 2021]MORO, C. et al. Hololens and mobile augmented reality in medical and health science education: A randomised controlled trial. *British Journal of Educational Technology*, v. 52, n. 2, p. 680–694, 2021.

[Morín, Pérez e Armada 2022]MORíN, D. G.; PéREZ, P.; ARMADA, A. G. Toward the distributed implementation of immersive augmented reality architectures on 5g networks. *IEEE Communications Magazine*, v. 60, n. 2, p. 46–52, 2022.

[Mur-Artal et al. 2015]MUR-ARTAL et al. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, v. 31, n. 5, p. 1147–1163, 2015.

[Mur-Artal e Tardós 2017]MUR-ARTAL, R.; TARDóS, J. D. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, v. 33, n. 5, p. 1255–1262, 2017.

[Muzzini et al. 2023]MUZZINI, F. et al. Brief announcement: Optimized gpu-accelerated feature extraction for orb-slam systems. In: *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*. New York, NY, USA: Association for Computing Machinery, 2023. (SPAA '23). ISBN 978-1-4503-9545-8/23/06. Disponível em: <https://doi.org/10.1145/3558481.3591310>.

[Pereira et al. 2021]PEREIRA, N. et al. Arena: The augmented reality edge networking architecture. In: *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. [S.l.: s.n.], 2021. p. 479–488.

[Pires et al. 2024]PIRES, M. et al. Uma avaliação empírica sobre o uso de webassembly para descarga de funções de realidade aumentada. In: *Anais do XLII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.]: SBC, 2024. p. 854–867.

[Redmon et al. 2016]REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 779–788.

[Ren et al. 2019]REN, J. et al. A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. *ACM Comput. Surv.*, v. 52, n. 6, 2019.

[Rosenberg 1992]ROSENBERG, L. B. The use of virtual fixtures as perceptual overlays to enhance operator performance in remote environments. *Air force material command*, p. 1–42, 1992.

[Satyanarayanan 2017]SATYANARAYANAN, M. The emergence of edge computing. *Computer*, v. 50, n. 1, p. 30–39, 2017.

[Shin e Kim 2022]SHIN, D.-J.; KIM, J.-J. A deep learning framework performance evaluation to use yolo in nvidia jetson platform. *Applied Sciences*, v. 12, n. 8, 2022. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/12/8/3734>.

[Siriwardhana et al. 2021]SIRIWARDHANA, Y. et al. A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications, and technical aspects. *IEEE Communications Surveys & Tutorials*, v. 23, n. 2, p. 1160–1192, 2021.

[Song et al. 2018]SONG, J. et al. Mis-slam: Real-time large-scale dense deformable slam system in minimal invasive surgery based on heterogeneous computing. *IEEE Robotics and Automation Letters*, v. 3, n. 4, p. 4068–4075, 2018.

[Tateno et al. 2017]TATENO, K. et al. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2017. p. 6565–6574.

[Thomas e David 1992]THOMAS, P. C.; DAVID, W. Augmented reality: An application of heads-up display technology to manual manufacturing processes. In: ACM SIGCHI BULLETIN. *Hawaii international conference on system sciences*. [S.l.], 1992. v. 2, p. 659–669.

[Toczé et al. 2019]TOCZé, K. et al. Performance study of mixed reality for edge computing. In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. [S.l.: s.n.], 2019. p. 285–294.

[Toczé et al. 2020]TOCZé, K. et al. Characterization and modeling of an edge computing mixed reality workload. *Journal of Cloud Computing*, v. 9, n. 46, 2020.

[Viola e Jones 2001]VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. [S.l.: s.n.], 2001. v. 1, p. I–I.

[Wang et al. 2022]WANG, H. et al. Leaf+ aio: Edge-assisted energy-aware object detection for mobile augmented reality. *IEEE Transactions on Mobile Computing*, IEEE, v. 22, n. 10, p. 5933–5948, 2022.

[Wang et al. 2023]WANG, X. et al. Small object detection based on deep learning for remote sensing: A comprehensive review. *Remote Sensing*, v. 15, n. 13, 2023. ISSN 2072-4292. Disponível em: <https://www.mdpi.com/2072-4292/15/13/3265>.

[Westphal 2017]WESTPHAL, C. Challenges in networking to support augmented reality and virtual reality. In: *IEEE ICNC*. [S.l.: s.n.], 2017.

[Wu, Miao e Sun 2023]WU, X.; MIAO, Y.; SUN, Z. Orb-yolo: An indoor imu-aided visual-inertial slam system for dynamic environment. In: IEEE. *2023 International Conference on Artificial Intelligence of Things and Systems (AIoTSys)*. [S.l.], 2023. p. 71–78.

[Yadhav et al. 2024]YADHAV, V. et al. Dynamic Computational Offloading for Mobile Devices. In: *Proceedings of the 14th International Conference on Cloud Computing and Services Science - CLOSER*. [S.l.]: SciTePress, 2024. (CLOSER'24), p. 265—-276.

[Zaidi et al. 2022]ZAIDI, S. S. A. et al. A survey of modern deep learning based object detection models. *Digital Signal Processing*, Elsevier, v. 126, p. 103514, 2022.

[Zhang et al. 2022]ZHANG, W. et al. Edgexar: A 6-dof camera multi-target interaction framework for mar with user-friendly latency compensation. *Proc. ACM Hum.-Comput. Interact.*, Association for Computing Machinery, New York, NY, USA, v. 6, n. EICS, jun 2022. Disponível em: <https://doi.org/10.1145/3532202>.

[Zou et al. 2023]ZOU, Z. et al. Object detection in 20 years: A survey. *Proceedings of the IEEE*, v. 111, n. 3, p. 257–276, 2023.