UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS CRATEÚS

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MIGUEL BARBOSA FARIAS

AVALIAÇÃO DO IMPACTO COMPUTACIONAL E ENERGÉTICO DO *STREAMING* BIDIRECIONAL VIA *GRPC* NO *OFFLOADING* MULTI-*LINGUAGEM* EM AMBIENTE DE *MOBILE CLOUD COMPUTING*

CRATEÚS

2026

MIGUEL BARBOSA FARIAS

AVALIAÇÃO DO IMPACTO COMPUTACIONAL E ENERGÉTICO DO *STREAMING* BIDIRECIONAL VIA *GRPC* NO *OFFLOADING* MULTI-*LINGUAGEM* EM AMBIENTE DE *MOBILE CLOUD COMPUTING*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em CIÊNCIA DA COMPUTAÇÃO do Campus Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em CIÊNCIA DA COMPUTAÇÃO.

Orientador: Prof. Dr. Filipe Fernandes dos Santos Brasil de Matos

CRATEÚS

2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F238a Farias, Miguel Barbosa.
Avaliação do impacto computacional e energético do streaming bidirecional via gRPC no offloading multi-linguagem em ambiente de mobile cloud computing / Miguel Barbosa Farias. – 2026.
69 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Crateús, Curso de Ciência da Computação, Crateús, 2026.
Orientação: Prof. Dr. Filipe Fernandes dos Santos Brasil de Matos.

1. Offloading. 2. Mobile Cloud Computing. 3. Cloud Computing. 4. gRPC. 5. Video Processing. I. Título.

CDD 004

MIGUEL BARBOSA FARIAS

AVALIAÇÃO DO IMPACTO COMPUTACIONAL E ENERGÉTICO DO *STREAMING* BIDIRECIONAL VIA *GRPC* NO *OFFLOADING* MULTI-*LINGUAGEM* EM AMBIENTE DE *MOBILE CLOUD COMPUTING*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em CIÊNCIA DA COMPUTAÇÃO do Campus Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em CIÊNCIA DA COMPUTAÇÃO.

Aprovada em: 27/01/2026

BANCA EXAMINADORA

Prof. Dr. Filipe Fernandes dos Santos Brasil de Matos (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Francisco Anderson de Almada Gomes
Universidade Federal do Ceará (UFC)

Profa. Me. Vitória Regina Nicolau Silvestre de Almada
Fiserv Brasil

Dedico este trabalho aos meus pais, Eufrásio Pereira Farias e Maria Alves Barbosa, cujos sacrifícios silenciosos e sonhos renunciados edificaram os alicerces que me permitiram trilhar meu próprio caminho.

AGRADECIMENTOS

Primeiramente, agradeço a Deus por todas as bênçãos concedidas ao longo da minha vida, pela força diante das dificuldades e pela coragem que me sustentou nesta jornada.

Aos meus pais, Eufrásio (in memoriam) e Maria, registro meu mais profundo amor e eterna gratidão. A meu pai, cuja ausência física jamais apagou sua presença em minha vida, deixo o reconhecimento por seu legado e por continuar sendo uma fonte diária de inspiração. A minha mãe, por sua força incansável, coragem e dedicação, por ter enfrentado sozinha todos os desafios, fazendo o possível e, muitas vezes, o impossível para que eu pudesse chegar até aqui.

Ao meu irmão Gabriel, por estar sempre ao meu lado, me incentivando e oferecendo apoio, não apenas nos estudos, mas também em momentos pessoais importantes.

Ao meu orientador, professor Filipe, agradeço profundamente por todo o acompanhamento, paciência, incentivo e conselhos que foram fundamentais para a concretização deste trabalho e para meu crescimento acadêmico e pessoal.

Agradeço também aos professores Almada e Vitória, membros da banca examinadora, pela disposição em contribuir com este trabalho e enriquecer sua qualidade com suas observações e sugestões.

À Universidade Federal do Ceará, por ter sido minha casa durante este período de formação e por me proporcionar a oportunidade de realizar minha graduação com excelência.

Aos amigos e colegas que caminharam comigo ao longo dessa trajetória, que me ajudaram tanto em momentos acadêmicos quanto momentos pessoais. Em especial, agradeço a Jorge Roniel, Lucas Eduardo Motta e Francisco Lucas Farias, por todo o apoio acadêmico, quanto em momentos pessoais.

E a todos que, de forma direta ou indireta, contribuíram para minha formação: o meu mais sincero muito obrigado.

“O conforto é o pior dos vícios.”

(Marco Aurélio)

RESUMO

O aumento da complexidade das aplicações móveis impõe desafios significativos aos dispositivos móveis, limitados em recursos computacionais e energéticos. Nesse cenário, o *offloading* computacional tem sido utilizado para transferir tarefas intensivas a servidores remotos, reduzindo o consumo de energia e melhorando o desempenho dos dispositivos móveis. Paralelamente, os avanços recentes nas formas de comunicação possibilitaram o surgimento de modelos baseados em *streaming*, capazes de viabilizar processamento contínuo quase em tempo real. Contudo, grande parte das soluções ainda se baseia no modelo tradicional de requisição e resposta, sem explorar os potenciais benefícios do *streaming* bidirecional e das estratégias *multi-linguagem*. Apesar do potencial do *gRPC*, sua aplicação prática em ambientes móveis e multi-linguagem permanece pouco explorada, especialmente no que se refere ao impacto sobre o desempenho e a eficiência energética. Este trabalho propôs, implementou e avaliou uma solução de *offloading* computacional multi-linguagem baseada em *gRPC* com *streaming* bidirecional, integrando um cliente *Android* em *Java* a servidores implementados em *Go* e *Java*. A metodologia envolve a aplicação de filtros de processamento de vídeo, comparando a execução local e remota por meio das métricas de tempo de execução, uso de CPU e consumo de energia. Os resultados demonstram que a eficácia do *offloading* depende diretamente da complexidade da tarefa: para o filtro simples (*Grayscale*), a execução local mostrou-se mais eficiente, enquanto para o filtro de maior complexidade (*Pencil*) o *offloading* proporcionou ganhos significativos. No melhor cenário avaliado, utilizando o servidor em *Go* e a rede Wi-Fi de 5 GHz, o filtro *Pencil* em resolução 1080p apresentou uma redução de aproximadamente 69% no tempo de execução, acompanhada por menor uso de CPU e redução no consumo de bateria. Conclui-se que o *offloading* via *gRPC* com *streaming* bidirecional é viável e eficaz para aplicações móveis computacionalmente intensivas, desde que aplicado a tarefas complexas e sob condições favoráveis de conectividade. A decisão entre execução local e remota deve ser adaptativa, considerando a complexidade da tarefa, a resolução do conteúdo e as condições de rede, de modo a maximizar o desempenho e a eficiência energética em ambientes de Computação em Nuvem Móvel.

Palavras-chave: *offloading* computacional; computação em nuvem móvel; gRPC; *offloading* multi-linguagem; processamento de vídeo.

ABSTRACT

The increasing complexity of mobile applications imposes significant challenges on mobile devices, which are limited in computational and energy resources. In this context, computational offloading has been employed to transfer intensive tasks to remote servers, thereby reducing energy consumption and improving the performance of mobile devices. In parallel, recent advances in communication have enabled the emergence of streaming-based models capable of supporting continuous processing almost in real time. However, most solutions still rely on the traditional request–response model, without exploring the potential benefits of bidirectional streaming and multi-language strategies. Despite the potential of gRPC, its practical application in mobile and multi-language environments remains underexplored, particularly regarding its impact on performance and energy efficiency. This work proposed, implemented, and evaluated a multi-language computational offloading solution based on gRPC with bidirectional streaming, integrating an Android client in Java with servers implemented in Go and Java. The methodology involved applying video processing filters and comparing local and remote execution through metrics such as execution time, CPU usage, and energy consumption. The results demonstrated that the effectiveness of offloading directly depends on task complexity: for the simple filter (Grayscale), local execution proved more efficient, whereas for the more complex filter (Pencil), offloading provided significant gains. In the best evaluated scenario, using the Go server and the 5 GHz Wi-Fi network, the Pencil filter at 1080p resolution achieved a reduction of approximately 69% in execution time, accompanied by lower CPU usage and reduced battery consumption. It was concluded that offloading via gRPC with bidirectional streaming is both feasible and effective for computationally intensive mobile applications, provided it is applied to complex tasks under favorable connectivity conditions. The decision between local and remote execution should be adaptive, considering task complexity, content resolution, and network conditions, in order to maximize performance and energy efficiency in Mobile Cloud Computing environments.

Keywords: computational offloading; mobile cloud computing; gRPC; multi-language offloading; video processing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

gRPC	*Google Remote Procedure Call*
HTTP/2	*Hypertext Transfer Protocol Version 2*
IoT	*Internet of Things*
MCC	*Mobile Cloud Computing*
MEC	*Mobile Edge Computing*
REST	*Representational State Transfer*

SUMÁRIO

1 INTRODUÇÃO

Apesar da evolução significativa no desempenho de *hardware* dos dispositivos móveis, nas capacidades de conectividade e na eficiência dos sistemas operacionais, tais dispositivos ainda enfrentam limitações críticas, como restrições de desempenho computacional, baixa autonomia energética e espaço de armazenamento limitado. (ZAKIZADEH, 2024; MORA, 2024). Essas restrições tornam-se ainda mais desafiadoras diante da complexidade crescente das aplicações móveis modernas, como processamento de imagens, reconhecimento facial, reconhecimento de fala e execução de modelos de inteligência artificial (GOGGIN, 2025; CÓRDOVA-CÁRDENAS *et al.*, 2025). Esse cenário impulsiona a busca por soluções que permitam a execução eficiente dessas tarefas sem comprometer a fluidez da interação ou esgotar os recursos do dispositivo.

A *Mobile Cloud Computing* (MCC) surgiu como resposta a esse desafio, ao integrar os recursos da nuvem ao ambiente móvel para mitigar tais limitações. A MCC permite que dispositivos móveis executem tarefas complexas fora do ambiente local, migrando dados e processamento para servidores com maior capacidade computacional, maior confiabilidade e fonte de energia praticamente ilimitada (DINH *et al.*, 2013; KUSHWAHA; RAI, 2024). Entre os benefícios desse paradigma destacam-se a extensão da vida útil da bateria, o aumento da capacidade de armazenamento e a melhoria no desempenho das aplicações (AHUJA, 2025). Aplicações como *Google Fotos*, *Flickr* e *Google Drive* exemplificam o uso prático dessa abordagem ao estender virtualmente o armazenamento do dispositivo móvel para a nuvem, otimizando recursos locais e entregando uma melhor usabilidade (DINH *et al.*, 2013).

Neste contexto, o *offloading* computacional ganha destaque como estratégia fundamental para viabilizar a MCC, cujo conceito será aprofundado na Seção 3.3. Na área da saúde, aplicações móveis apoiadas por MCC possibilitam o processamento em tempo real de dados médicos, como diagnósticos assistidos por inteligência artificial, viabilizando atendimentos em regiões com infraestrutura limitada (DINH *et al.*, 2013). Na educação, plataformas de ensino remoto exploram a escalabilidade da nuvem para disponibilizar conteúdos interativos em dispositivos acessíveis, promovendo a inclusão digital (KUSHWAHA; RAI, 2024). No entretenimento, jogos em realidade aumentada e transmissões de mídia demandam elevado desempenho computacional, o qual a MCC, especialmente quando integrada a arquiteturas de computação em borda, é capaz de fornecer mesmo em dispositivos com recursos restritos (SIRIWARDHANA *et al.*, 2021).

Trabalhos recentes, como Matos *et al.* (2021), demonstraram que a escolha da

linguagem de programação no lado servidor impacta significativamente o desempenho de sistemas baseados em *offloading* computacional. Em seus experimentos, os autores utilizaram o *framework* gRPC com comunicação unária para realizar o *offloading* de tarefas computacionais a partir de um aplicativo *Android* para servidores implementados em diferentes linguagens. Essa abordagem, conhecida como *offloading* multi-linguagem, permite que cliente e servidor sejam desenvolvidos de forma independente, cada um na linguagem mais adequada ao seu contexto de execução. Como benefício, essa estratégia explora vantagens específicas de cada linguagem, como o alto desempenho de linguagens compiladas no lado servidor. Os resultados de Matos *et al.* (2021) evidenciam ganhos expressivos de desempenho e economia de energia ao utilizar linguagens mais eficientes no servidor, ressaltando o potencial do *offloading* multi-linguagem para superar as limitações dos dispositivos móveis.

Este trabalho expandiu o estudo de Matos *et al.* (2021), adotando uma abordagem de *offloading* multi-linguagem com suporte a *streaming* bidirecional utilizando gRPC, explorando novas possibilidades de comunicação contínua entre cliente *Android* e servidor. Essa ampliação teve como objetivo avaliar o impacto do uso de protocolos modernos e arquiteturas multi-linguagem na eficiência do *offloading*, considerando fatores como consumo energético e tempo de processamento.

Além disso, este documento está estruturado da seguinte forma: O capítulo 1, de Introdução, apresenta o contexto geral da pesquisa, destacando a motivação, a relevância do tema e os principais desafios que justificam o estudo. O capítulo 2, de Objetivos Gerais, estabelece o objetivo da pesquisa, define o problema investigado, apresenta os objetivos propostos e descreve as contribuições esperadas. O capítulo 3, de Fundamentação Teórica, consolida os principais conceitos que sustentam o estudo, incluindo Computação em Nuvem, MCC, *offloading* computacional, comunicação via gRPC, *streaming* bidirecional e arquiteturas multi-linguagem. O capítulo 4, de Trabalhos Relacionados, analisa a literatura onde esses temas já foram explorados, destacando abordagens correlatas e evidenciando lacunas que motivam o presente trabalho. O capítulo 5, de Arquitetura da Solução e Ambiente Experimental, descreve a arquitetura proposta, o ambiente de execução, os cenários experimentais, as métricas e os procedimentos adotados para a avaliação de desempenho e consumo energético. O capítulo 6, de Resultados, apresenta e analisa os dados obtidos nos experimentos. Por fim, o capítulo 7, de Conclusões, sintetiza as principais contribuições do estudo, discute suas limitações e aponta diretrizes para trabalhos futuros.

1.1 Motivação

Com a rápida expansão do uso de dispositivos móveis, os estilos de vida da sociedade foram completamente transformados, resultando em uma crescente dependência desses aparelhos. O número de assinaturas de redes móveis para *smartphones* em todo o mundo atingiu quase sete bilhões em 2023, com previsão de superar 7,7 bilhões até 2028 (STATISTA, 2023b). Essa tendência tem impulsionado uma mudança nos padrões de consumo digital: no quarto trimestre de 2024, cerca de 62,54% de todo o tráfego global de *internet* foi gerado a partir de dispositivos móveis (STATISTA, 2024).

Os *smartphones*, equipados com Wi-Fi, câmeras de alta qualidade, armazenamento interno, *GPS* e processadores cada vez mais rápidos, tornaram-se indispensáveis no cotidiano moderno. Essa evolução permitiu o surgimento de aplicativos cada vez mais sofisticados, como tradutores de linguagem natural, sistemas de reconhecimento de fala e recursos de realidade aumentada (SIRIWARDHANA *et al.*, 2021). Entre as atividades mais comuns nesse ecossistema, destacam-se o consumo de conteúdos via *streaming*, o uso de redes sociais, mensagens instantâneas e jogos eletrônicos (STATISTA, 2024). Aplicativos como *TikTok*, *Netflix*, *HBO Max* e *Amazon Prime Vídeo*, que figuram entre os mais baixados nas lojas digitais (STORE, 2024), operam com base em tecnologias de *streaming* em tempo real, processamento de imagem e algoritmos de recomendação baseados em aprendizado de máquina.

Esse avanço, no entanto, traz consigo um aumento proporcional na demanda por poder computacional, o que torna os dispositivos móveis cada vez mais exigidos em termos de processamento, armazenamento e energia (SIRIWARDHANA *et al.*, 2021). Uma pesquisa da *Statista* (STATISTA, 2023a) indicou que 71% dos entrevistados consideram a duração da bateria o fator mais importante na escolha de um novo aparelho, superando durabilidade (61%) e qualidade da câmera (50%). Outro estudo, da *Morning Consult* (CONSULT, 2018), revelou que 95% dos proprietários de *smartphones* valorizam mais a vida útil da bateria do que funcionalidades avançadas, como realidade aumentada ou reconhecimento facial.

Diante desse cenário, torna-se evidente a necessidade de soluções que enfrentem as limitações dos dispositivos móveis frente à crescente demanda por desempenho, eficiência energética e autonomia. Os dados apresentados reforçam a importância desta pesquisa, que busca mitigar tais desafios por meio da adoção de técnicas de *offloading* computacional em conjunto com o uso de tecnologias modernas, como o gRPC.

2 OBJETIVOS GERAIS

Analisar comparativamente o impacto do uso do gRPC com comunicação via *streaming* bidirecional sobre a eficiência computacional e energética de aplicações móveis no contexto da Computação em Nuvem Móvel. Neste estudo, entende-se por eficiência a redução do tempo de processamento de tarefas computacionalmente intensivas quando executadas por meio de *offloading* e a diminuição do consumo de bateria. Ao final, espera-se que o uso do *offloading* multi-linguagem aliado ao gRPC com *streaming* bidirecional contribua para a otimização desses indicadores, tornando tarefas anteriormente custosas em termos de tempo de execução, uso de CPU e consumo energético em tarefas executadas de forma mais rápida, eficiente e com menor impacto sobre os recursos do dispositivo móvel.

2.1 Objetivos Específicos

1. **Definir e preparar a aplicação de estudo**: selecionar uma aplicação prática de alta demanda computacional e implementá-la para o contexto experimental, de modo a torná-la compatível com o modelo de *offloading* utilizado.
2. **Projetar o ambiente de testes**: configurar um ambiente que suporte redes locais e o gRPC com *streaming* bidirecional. O cliente, implementado em *Java*, será conectado a dois servidores *backend*: um em *Go* (proposta multi-linguagem) e outro em *Java* (linha de base para comparação).
3. **Executar testes em diferentes cenários**: avaliar o desempenho da aplicação com e sem *offloading*, variando níveis de complexidade computacional e condições de conectividade.
4. **Analisar os dados e comparar abordagens**: comparar os resultados obtidos para os cenários de processamento local, remoto com comunicação via *streaming* bidirecional.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo principal apresentar os conceitos e informações fundamentais para a compreensão deste trabalho, com ênfase nas áreas de Computação Móvel (Seção 3.1), Computação em Nuvem (Seção 3.2), Computação em Nuvem Móvel (Seção 3.2.1), *Offloading* Computacional (Seção 3.3), *Offloading* multi-linguagem (Seção 3.4), o *framework* gRPC e suas formas de comunicação (Seção 3.5).

3.1 Computação Móvel

A Computação Móvel representa um dos pilares da evolução tecnológica contemporânea, sendo definida como a combinação entre dispositivos portáteis e redes sem fio, que permite aos usuários acessarem informações, executarem aplicações e colaborarem remotamente, mesmo em movimento (BHARGAVA; GUPTA, 2008). Esse paradigma surgiu da crescente demanda por comunicação instantânea e acesso a serviços digitais em tempo real, favorecido pelos avanços em conectividade e miniaturização de *hardware* (SATYANARAYANAN, 2001; DINH *et al.*, 2013).

Forman e Zahorjan (FORMAN; ZAHORJAN, 1994) destacam que, com o surgimento de redes sem fio, foi possível equipar dispositivos portáteis com capacidades de comunicação móvel, transformando significativamente a maneira como os indivíduos interagem com os dispositivos. Esse tipo de comunicação, que sustenta a essência da Computação Móvel, pode ser visualizado na Figura 1, onde é apresentada uma ilustração da troca de informações entre dispositivos por meio de redes sem fio.

Figura 1 – Ilustração representando a comunicação entre dispositivos por meio de uma rede sem fio.

Fonte – Elaborado pelo autor.

Contudo, mesmo com os avanços, a computação móvel ainda é marcada por limitações que impactam seu desempenho (DINH *et al.*, 2013). Entre essas restrições, destacam-se a limitação energética, onde processos computacionalmente intensos drenam rapidamente a bateria, e as deficiências nas redes sem fio, que ainda sofrem com latência elevada e desconexões frequentes, especialmente em cenários de alta mobilidade (DE, 2016; BHARGAVA; GUPTA, 2008).

Outras limitações incluem a capacidade de armazenamento, memória e poder de processamento, que, embora tenham evoluído, ainda são limitadas em comparação com computadores de mesa (DE, 2016; KUMAR *et al.*, 2013). Porém, como reforçam Kumar *et al.* (KUMAR *et al.*, 2013), os dispositivos móveis, por uma questão de compromisso entre portabilidade e custo, jamais igualarão os recursos computacionais de *desktops*, o que os torna dependentes de soluções complementares para executar tarefas mais exigentes.

Frente a essas limitações, a Computação Móvel tem evoluído com o apoio de novos paradigmas que são capazes de ampliar seu potencial e contornar suas restrições (DINH *et al.*, 2013; SHI *et al.*, 2016; YOUSEFPOUR *et al.*, 2019). Dentre essas abordagens, destaca-se a

Mobile Cloud Computing (MCC), que se apresenta como uma solução inovadora ao integrar dispositivos móveis a recursos computacionais remotos, como servidores em nuvem ou borda, possibilitando a execução de tarefas de alta demanda, otimizando o consumo energético e superando as limitações de processamento e armazenamento dos dispositivos locais (DINH *et al.*, 2013).

A seguir, esse paradigma será explorado com maior profundidade, apresentando seus conceitos fundamentais no contexto da Computação Móvel.

3.2 Computação em Nuvem

Segundo a Microsoft (2025), a computação em nuvem pode ser descrita como o fornecimento de serviços de computação, incluindo servidores, armazenamento, banco de dados e entre outros serviços através da rede. Do ponto de vista acadêmico, a computação em nuvem é definida pelo *National Institute of Standards and Technology (NIST)* como um modelo que possibilita o acesso sob demanda, via rede, a um conjunto compartilhado de recursos computacionais configuráveis, como redes, servidores, armazenamento, aplicações e serviços, que podem ser rapidamente provisionados e liberados com mínimo esforço de gerenciamento (MELL; GRANCE, 2011).

Buyya *et al.* (2019) complementam essa visão ao caracterizar a computação em nuvem como um modelo orientado a mercado, no qual recursos computacionais são disponibilizados de forma escalável e elástica, permitindo que aplicações sejam desenvolvidas e executadas sem a necessidade de gerenciamento direto da infraestrutura física subjacente. Essa abordagem favorece a redução de custos operacionais e aumenta a flexibilidade no desenvolvimento de sistemas distribuídos de grande escala.

No contexto atual, a computação em nuvem exerce papel central na viabilização de tecnologias emergentes, como a Computação em Nuvem Móvel. Nesse cenário, a nuvem atua como uma extensão computacional dos dispositivos móveis, mitigando suas limitações intrínsecas de processamento, armazenamento e consumo energético, aspecto fundamental para aplicações móveis modernas de alta complexidade. Conforme discutido por Armbrust *et al.* (2010), esse paradigma permite explorar a escalabilidade e a disponibilidade da nuvem para superar restrições locais de *hardware*, viabilizando aplicações móveis mais complexas e energeticamente eficientes.

Dessa forma, a computação em nuvem estabelece a base tecnológica sobre a qual

estratégias de *offloading* computacional são construídas, permitindo a execução eficiente de aplicações móveis modernas e servindo como elemento central para arquiteturas de Computação em Nuvem Móvel.

3.2.1 Computação em Nuvem Móvel

A Computação em Nuvem Móvel é um paradigma computacional que integra dispositivos móveis, redes de internet e recursos de computação em nuvem para viabilizar a execução de aplicações de alta demanda e o processamento de dados intensivos (CHEN *et al.*, 2018). Esse modelo visa superar as limitações estruturais dos dispositivos, como capacidade de processamento, armazenamento e bateria, ao transferir parte ou a totalidade da carga de trabalho para servidores remotos altamente escaláveis (DINH *et al.*, 2013).

Segundo a Amazon Web Services (2025a), a MCC permite que aplicações complexas sejam executadas com alto desempenho ao explorar o poder computacional e a elasticidade da nuvem, resultando em menor consumo de energia e melhor tempo de resposta no cliente.

Dinh *et al.* (2013) afirmam que essa tecnologia viabiliza aplicações modernas, como realidade aumentada, jogos e serviços baseados em localização. Sua arquitetura normalmente envolve três camadas: (i) o dispositivo cliente; (ii) a rede de comunicação móvel, que conecta os dispositivos ao servidor remoto; e (iii) os servidores em nuvem ou em borda, onde ocorrem o armazenamento e a execução computacional (DINH *et al.*, 2013). A execução remota de tarefas reduz significativamente a carga local, possibilitando maior autonomia de bateria e fluidez em aplicações exigentes (RAHIMI *et al.*, 2014).

Rahimi *et al.* (2014) identificam o *offloading* computacional como o núcleo operacional deste paradigma. Além dos benefícios técnicos, essa abordagem oferece manutenção simplificada e escalabilidade elástica, pois, ao deslocar para a nuvem as partes críticas do processamento, permite que o ciclo de vida das aplicações seja gerido com maior flexibilidade, viabilizando atualizações contínuas de forma transparente ao usuário (DINH *et al.*, 2013).

Apesar das vantagens oferecidas, a dependência de conectividade estável e de baixa latência com a nuvem pode representar um gargalo para sistemas sensíveis ao tempo de resposta, como monitoramento em tempo real ou processamento de vídeo (CHEN *et al.*, 2018).

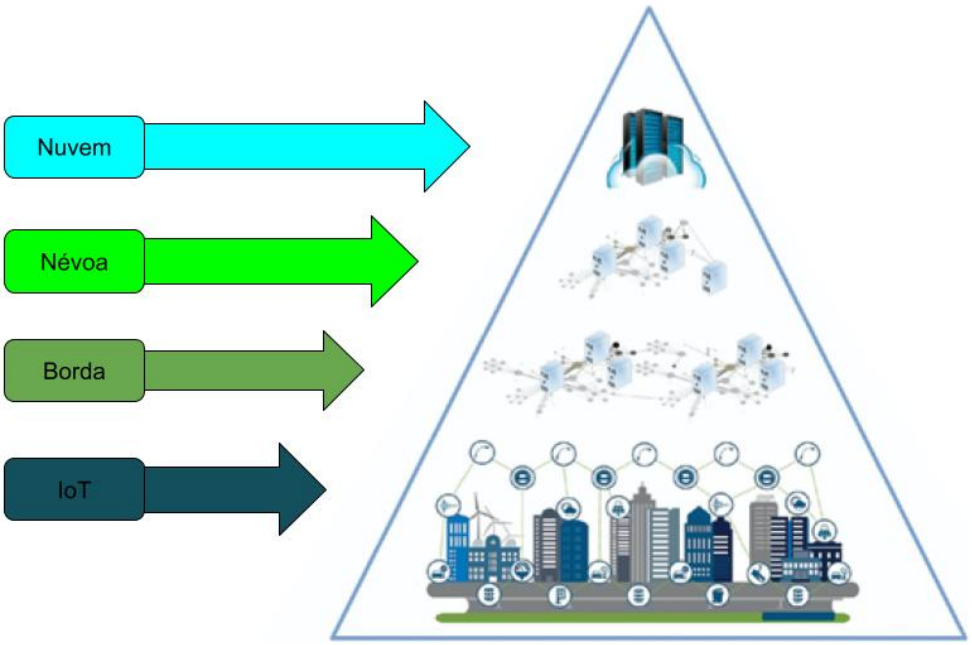
Diante dessas limitações, surgem arquiteturas complementares, como a *Edge Computing* e a *Fog Computing*, que visam aproximar os recursos computacionais dos dispositivos finais, otimizando a disponibilidade do serviço (SHI *et al.*, 2016; YOUSEFPOUR *et al.*, 2019).

A seguir, apresenta-se uma discussão detalhada sobre a computação em borda e sua relevância no cenário móvel distribuído.

3.2.2 Computação em Borda (Edge Computing)

A computação em borda (*Edge Computing*) consiste em deslocar parte significativa das operações de processamento, armazenamento e gestão de dados para pontos periféricos da rede, mais próximos dos dispositivos que geram e consomem essas informações (SHI *et al.*, 2016). Segundo a *Amazon Web Services (2025b)*, essa abordagem visa aproximar os recursos computacionais dos terminais e dos usuários finais, mitigando gargalos associados à latência, à largura de banda e ao tráfego elevado em *datacenters* centralizados. Como resultado, obtêm-se ganhos substanciais em responsividade, eficiência operacional e confiabilidade do sistema.

Figura 2 – Visão hierárquica da computação distribuída em múltiplas camadas: IoT, borda, névoa e nuvem.



Fonte: Adaptado de (AL-DULAIMY *et al.*, 2020).

A Figura 2 ilustra a arquitetura hierárquica da computação distribuída contemporânea,

composta por quatro camadas funcionais: dispositivos IoT (geradores de dados em tempo real), camada de borda (*edge*), camada de névoa (*fog*) e nuvem centralizada. No contexto da borda, destacam-se os *cloudlets*[1], os quais, em conjunto com *gateways*, roteadores e outros dispositivos intermediários, possibilitam a execução de tarefas computacionalmente intensivas de forma distribuída. Essa descentralização reduz o tempo de resposta, melhora a escalabilidade e eleva o desempenho de aplicações sensíveis ao tempo de resposta, como monitoramento em tempo real, *streaming* de vídeo, reconhecimento facial e controle de tráfego em ambientes urbanos inteligentes.

Ao contrário da MCC, que depende majoritariamente da conectividade com servidores remotos localizados em grandes centros de dados, a *Edge Computing* viabiliza o processamento local dos dados, reduzindo significativamente o tempo de ida e volta entre o dispositivo final e o ponto de execução da tarefa (YOUSEFPOUR *et al.*, 2019). A proximidade física entre a origem dos dados e o ponto de processamento eleva a responsividade e a confiabilidade de aplicações críticas, mas também introduz desafios relacionados às capacidades computacionais limitadas dos nós de borda e à complexidade de administrar uma infraestrutura distribuída em larga escala (SHI *et al.*, 2016; YOUSEFPOUR *et al.*, 2019).

Nesse cenário, a *Fog Computing* emerge como uma extensão da *Edge Computing*, operando como uma camada intermediária entre os dispositivos finais e a nuvem. Essa camada introduz maior elasticidade, distribuindo as funções de computação, armazenamento e rede de forma mais abrangente e hierárquica, contribuindo para reduzir a sobrecarga nos nós de borda. A estrutura e os benefícios desse modelo arquitetural serão discutidos em maior profundidade na subseção seguinte.

3.2.3 Computação em Névoa (Fog Computing)

A computação em névoa (*Fog Computing*) é um conceito que a *OpenFog Consortium* a define como “uma arquitetura horizontal em nível de sistema que distribui funções de computação, armazenamento, controle e rede mais perto dos usuários” (YOUSEFPOUR *et al.*, 2019; OpenFog Consortium Architecture Working Group, 2017).

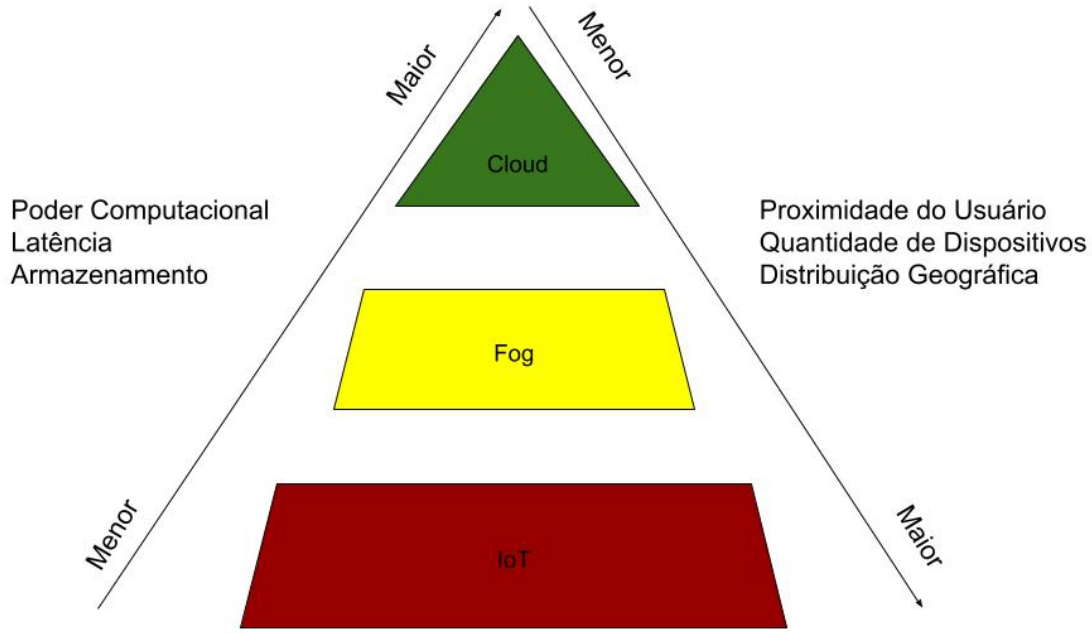
A arquitetura da névoa é hierárquica e geograficamente distribuída, permitindo que a

[1] *Cloudlets* são micro–*datacenters* implantados estrategicamente na borda da rede, projetados para oferecer serviços de computação e armazenamento com baixa latência. Propostos originalmente por SATYANARAYANAN *et al.*, atuam como extensões locais da nuvem, processando aplicações sensíveis à latência o mais próximo possível do usuário (SATYANARAYANAN *et al.*, 2009).

computação, o armazenamento e a tomada de decisões ocorram em qualquer ponto otimizado entre o dispositivo e a nuvem (YOUSEFPOUR *et al.*, 2019). Na prática, a computação em névoa atua em pontos intermediários da rede, como roteadores, *gateways* ou *switches*, realizando tarefas como agregação, filtragem, compressão e análise preliminar de dados antes de enviá-los para a nuvem.

A Figura 3 ilustra a arquitetura em três camadas proposta por Yousefpour *et al.* (2019) para o ecossistema de computação distribuída envolvendo dispositivos *Internet of Things* (IoT), computação em névoa e computação em nuvem. Na base, encontram-se os dispositivos finais (*smart devices*), responsáveis pela geração de dados em tempo real. No nível intermediário, a camada de névoa opera em dispositivos como *gateways*, *switches* e roteadores. No topo, a camada de nuvem concentra alto poder computacional, grande capacidade de armazenamento e recursos de conectividade, porém com maior latência. Observa-se que, à medida que se avança da base da pirâmide para o topo, há aumento do poder de processamento, mas também aumento do tempo de resposta. Por outro lado, quanto mais próximo dos dispositivos IoT e *Fog*, maior é a capacidade de resposta em tempo real (YOUSEFPOUR *et al.*, 2019).

Figura 3 – Arquitetura hierárquica envolvendo dispositivos finais, computação em névoa e nuvem.



Fonte: Adaptado de Yousefpour *et al.* (YOUSEFPOUR *et al.*, 2019).

Essa arquitetura reforça a importância da computação em névoa como camada intermediária, sendo responsável por equilibrar os requisitos de desempenho e disponibilidade entre os dispositivos de borda e a nuvem.

Na próxima seção, será aprofundado o conceito de *offloading* multi-linguagem, destacando como a integração de diferentes linguagens de programação amplia ainda mais a eficiência e a flexibilidade dessa solução.

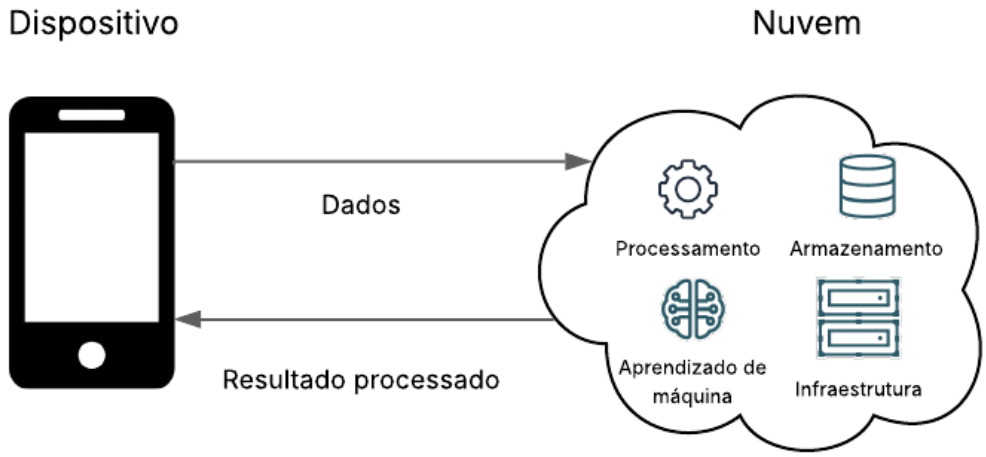
3.3 *Offloading* Computacional

O *offloading* computacional é um paradigma fundamental na computação em nuvem móvel que consiste na transferência de tarefas computacionalmente intensivas de dispositivos locais, frequentemente com recursos limitados, como *smartphones*, para servidores remotos (KUSHWAHA; RAI, 2024). Essa técnica tem como objetivo principal melhorar o desempenho de aplicações, reduzir o consumo de energia no dispositivo local e possibilitar a execução

de tarefas que seriam inviáveis devido às limitações de *hardware*, como bateria, memória ou capacidade de processamento (KUMAR *et al.*, 2013). Segundo Kumar *et al.* (2018), o *offloading* computacional é particularmente relevante no contexto da MCC, pois permite que dispositivos móveis aproveitem os recursos abundantes da nuvem, resultando em conservação de recursos e aumento de desempenho.

A esquematização do *offloading* computacional é ilustrada na Figura 4. Neste modelo, o dispositivo móvel, atuando como cliente, envia os dados brutos ou a própria tarefa a ser executada para a nuvem. A nuvem, por sua vez, aproveita sua infraestrutura para executar a computação intensiva. Uma vez concluída a tarefa, apenas o resultado processado é retornado ao dispositivo. Esse fluxo demonstra o princípio do *offloading*: a carga computacional é delegada ao servidor remoto, aliviando o dispositivo de tarefas que consumiriam excessivamente seus recursos de *CPU*, memória e bateria, e permitindo que o usuário final receba uma resposta com maior rapidez.

Figura 4 – Esquematização do funcionamento do *offloading* computacional.



Fonte – Elaborado pelo autor.

O *offloading* computacional demonstra sua versatilidade ao ser aplicado em diversas áreas, possibilitando funcionalidades avançadas em dispositivos móveis com recursos limitados. Em jogos móveis, por exemplo, (JIANG *et al.*, 2018) desenvolveram a arquitetura *Mirror*, que permite a renderização gráfica complexa em servidores remotos, viabilizando experiências de alta qualidade em dispositivos de baixa potência. Em aplicações de realidade aumentada, LIU *et al.* (2018) propuseram um método de *offloading* de dados que minimiza a latência, permitindo

interações em tempo real, como reconhecimento de objetos, em dispositivos móveis. Além disso, uma pesquisa recente destaca a diversidade de aplicações beneficiadas pelo *offloading*, incluindo processamento multimídia, visão computacional, renderização gráfica, IoT e veículos conectados (ZHANG, 2023).

A integração do *offloading* com arquiteturas de *Mobile Edge Computing* (MEC) tem ampliado suas possibilidades, especialmente em cenários que exigem baixa latência e alta eficiência energética (MAO *et al.*, 2017; GU *et al.*, 2019; CHEN *et al.*, 2018). Por exemplo, em aplicações de IoT, o *offloading* permite que dispositivos com recursos limitados, como sensores, deleguem tarefas complexas a servidores de borda, reduzindo o consumo de energia e o tempo de resposta (YOUSEFPOUR *et al.*, 2019). Em veículos conectados, o *offloading* suporta aplicações como reconhecimento facial e navegação em tempo real, onde a latência é crítica para evitar falhas ou acidentes (ZHANG, 2023).

Essas aplicações são frequentemente otimizadas por modelos de decisão que utilizam algoritmos que consideram múltiplos objetivos, como minimização de latência, consumo de energia e maximização da utilização de recursos (CHEN *et al.*, 2018; QIN *et al.*, 2025). Por exemplo, o *offloading* é vantajoso quando o tempo de transferência de dados e recebimento de resultados é menor que o tempo de computação local, e quando a energia economizada supera o gasto com comunicação (KUMAR *et al.*, 2013).

Na próxima seção, será aprofundado o conceito de *offloading* multi-linguagem, destacando como a integração de diferentes linguagens de programação amplia ainda mais a eficiência e a flexibilidade dessa solução.

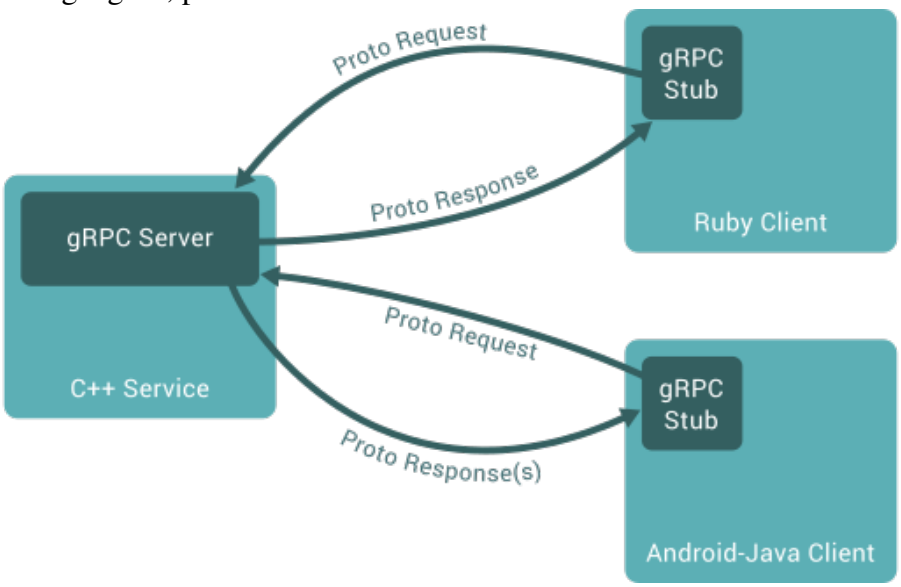
3.4 *Offloading* Multi-Linguagem

O *offloading* multi-linguagem refere-se à prática de transferir tarefas computacionais de um dispositivo local, geralmente com recursos limitados, para servidores remotos ou infraestruturas de nuvem, utilizando componentes de *software* implementados em diferentes linguagens de programação para otimizar desempenho, interoperabilidade e eficiência (MATOS *et al.*, 2021). Essa abordagem estende o conceito de *offloading* computacional, permitindo que sistemas heterogêneos, compostos por clientes e servidores escritos em linguagens distintas, como *Java*, *Python*, *C++* ou *Go*, colaborem juntos de forma eficiente. Segundo Matos *et al.* (2021), o *offloading* multi-linguagem é particularmente vantajoso em cenários de MCC, onde a diversidade de linguagens pode explorar as vantagens específicas de cada uma.

A motivação para o *offloading* multi-linguagem surge da crescente complexidade das aplicações móveis e da necessidade de integrar sistemas heterogêneos em ambientes de computação distribuída (MATOS *et al.*, 2021). No início dos anos 2010, com a popularização de dispositivos móveis, a computação em nuvem começou a ser explorada para aliviar as limitações de *hardware*, como bateria e capacidade de processamento, conforme descrito por DINH *et al.*. No entanto, a maioria dos sistemas de *offloading* iniciais era restrita a uma única linguagem, limitando a interoperabilidade entre plataformas (TANENBAUM; STEEN, 2023).

O advento de frameworks como gRPC, que suporta múltiplas linguagens por meio de *Protocol Buffers*, permitiu a implementação de sistemas de *offloading* mais flexíveis, nos quais clientes e servidores podem ser desenvolvidos em linguagens diferentes sem comprometer a eficiência da comunicação (CHAMAS *et al.*, 2017; Google, 2025; INDRASIRI; KURUPPU, 2020). Essa evolução reflete a necessidade de adaptar o *offloading* às demandas de sistemas heterogêneos, onde diferentes linguagens são escolhidas com base nas características específicas da tarefa, como eficiência computacional, facilidade de manutenção ou suporte a bibliotecas especializadas (MATOS *et al.*, 2021).

Figura 5 – *gRPC* viabilizando comunicação entre serviços cliente e servidor escritos em diferentes linguagens, por meio de chamadas remotas serializadas em *Protocol Buffers*.



Fonte – Documentação oficial do *gRPC*.

Além disso, frameworks como o *COCAME*, descrito por Kumar *et al.* 2018, demonstram que o *offloading* multi-linguagem pode alcançar até 95% de eficiência em tarefas como multiplicação de matrizes, ao combinar clientes em linguagens de alto nível com servidores em linguagens de baixo nível, como *C++*.

Em dispositivos IoT, por exemplo, o *offloading* multi-linguagem permite que sensores com clientes em linguagens como *Python* transfiram tarefas complexas, como análise de dados, para servidores em *C++* ou *Java*, reduzindo a latência e o consumo energético (LIU *et al.*, 2019).

Em aplicações de realidade aumentada, o uso de técnicas de *offloading* computacional é fundamental para garantir a responsividade e a eficiência energética em dispositivos móveis (SIRIWARDHANA *et al.*, 2021). Segundo Siriwardhana *et al.* (2021) , a integração da realidade aumentada com a computação em borda permite o descarregamento de tarefas computacionalmente intensivas, como o rastreamento de objetos e o processamento de gráficos *3D*, para servidores de computação em borda com alta capacidade de processamento. Esse modelo possibilita a execução de aplicações imersivas e interativas mesmo em dispositivos com recursos limitados, ao mesmo tempo em que reduz significativamente a latência e o consumo energético (SIRIWARDHANA *et al.*, 2021).

Além disso, em jogos móveis, o *offloading* multi-linguagem permite a renderização gráfica em servidores *Go*, enquanto o cliente em *Java* gerencia a interface do usuário, proporcionando uma melhor usabilidade em dispositivos de baixo poder computacional (CAI *et al.*, 2016). Esses exemplos destacam a versatilidade do *offloading* multi-linguagem, que combina a eficiência de diferentes linguagens para atender às demandas específicas de cada aplicação, consolidando sua relevância em sistemas distribuídos modernos.

Na seção seguinte, será abordado o gRPC, responsável por viabilizar o *offloading* multi-linguagem discutido nesta seção.

3.5 Comunicação via gRPC

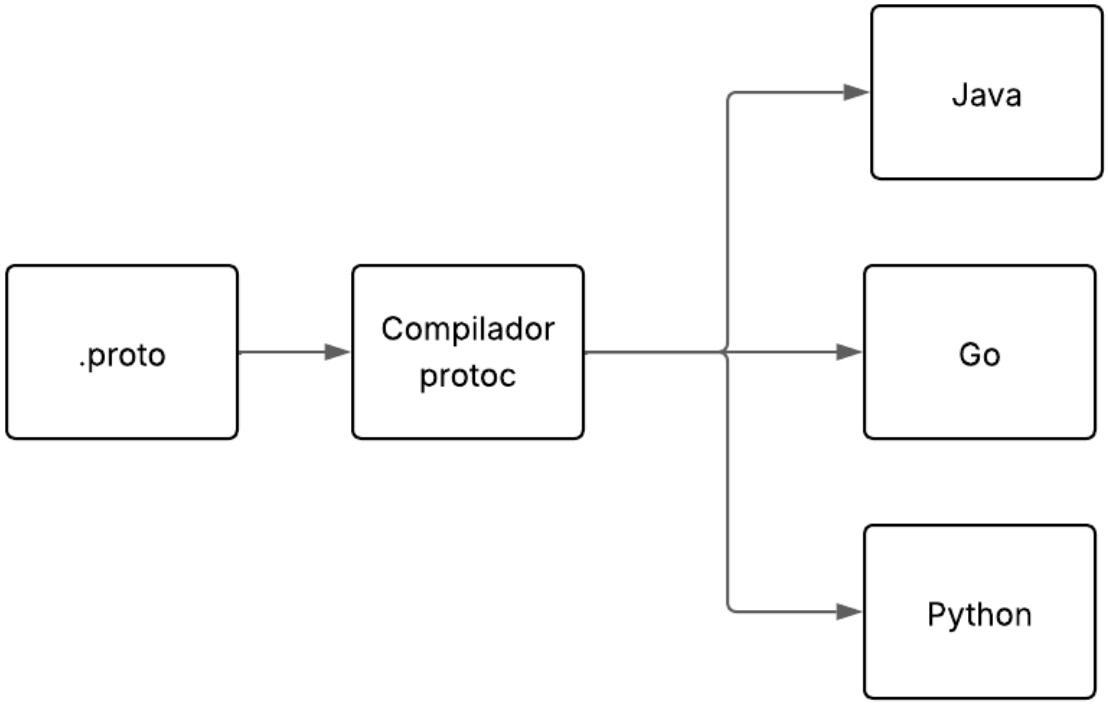
O gRPC (*Google Remote Procedure Call*) é um *framework* de chamada de procedimento remoto (*RPC*) moderno e de alto desempenho que permite que aplicações cliente e servidor comuniquem-se diretamente através da invocação de métodos como se estivessem executando localmente. O principal diferencial do gRPC reside no uso de *Protocol Buffers* (*Protobuf*) como linguagem de definição de interface (*IDL – Interface Definition Language*) e mecanismo de serialização binária, o que garante baixo *overhead* e interoperabilidade entre múltiplas linguagens de programação (Google, 2025).

Segundo Chamas *et al.* (2017), o gRPC é particularmente vantajoso em aplicações que exigem alta performance, como sistemas distribuídos em tempo real e aplicações de IoT, devido à sua capacidade de reduzir a latência em até 60% em comparação com *Representational*

State Transfer (REST) tradicionais.

Conforme ilustrado na Figura 6, o arquivo `.proto` é compilado pelo compilador `protoc`, que gera os *stubs* (códigos de cliente e servidor que encapsulam todas as chamadas gRPC) em diferentes linguagens, como *Java*, *Go* e *Python*. Esse mecanismo assegura interoperabilidade em sistemas heterogêneos e viabiliza abordagens de *offloading* computacional entre linguagens e, até mesmo, plataformas distintas (ARAUJO *et al.*, 2020).

Figura 6 – Fluxo de geração de código em diferentes linguagens: através da linguagem de definição de interface *proto*, o compilador *protoc* consegue transpilar o código equivalente para outras linguagens.



Fonte – Elaborado pelo autor.

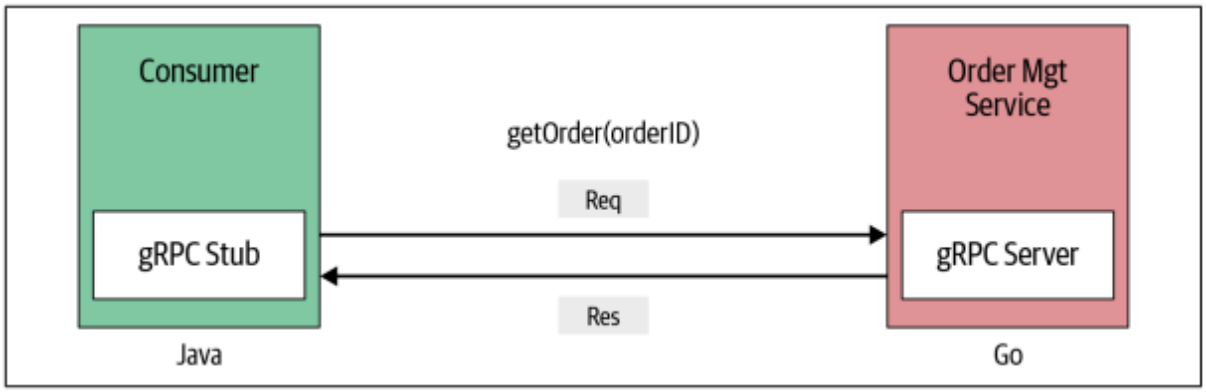
No gRPC, existem dois modelos principais de interação: comunicação unária e comunicação via *streaming* bidirecional. A escolha entre eles deve considerar os requisitos da aplicação: tarefas pontuais que exigem respostas rápidas favorecem chamadas unárias, enquanto cenários que demandam troca contínua de dados, como o envio de vídeo em tempo real, beneficiam-se do *streaming* bidirecional.

3.5.1 Comunicação Unária

A comunicação unária no gRPC é o modelo mais simples e amplamente utilizado, no qual o cliente envia uma única requisição ao servidor, que processa a solicitação e retorna uma única resposta. NISWAR *et al.* (2024) avaliaram o desempenho de diferentes abordagens de comunicação entre microserviços e constataram que o gRPC, ao empregar comunicação unária baseada em *Hypertext Transfer Protocol Version 2* (HTTP/2)/2 e *Protocol Buffers*, obteve os menores tempos de resposta, especialmente em cenários com alta carga de leitura, apresentando até 50% de melhoria em relação ao estilo REST tradicional.

A Figura 7, apresentada por Indrasiri e Kuruppu (INDRASIRI; KURUPPU, 2020, p. 27), ilustra o fluxo de uma chamada *gRPC*. No exemplo ilustrado, um cliente desenvolvido em Java realiza a invocação do método `getOrder(orderID)` por meio de um *stub*, responsável por encapsular a lógica de serialização e a comunicação em rede, de modo que a chamada se assemelha a uma execução local. A requisição é transmitida ao servidor, implementado em Go, onde é desserializada e encaminhada à lógica de negócio para processamento. Em seguida, uma resposta única é enviada de volta ao *stub*, que a repassa à aplicação cliente como retorno da operação.

Figura 7 – Comunicação unária entre cliente *gRPC* (*Java*) e servidor (*Go*) utilizando chamadas remotas.



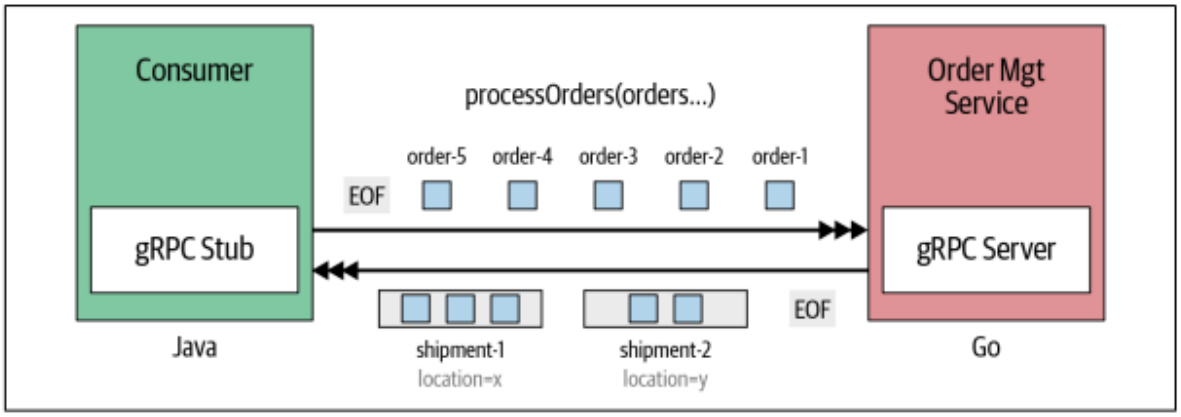
Fonte: Elaborado por Indrasiri e Kuruppu (INDRASIRI; KURUPPU, 2020, p. 44).

No entanto, a comunicação unária é limitada em cenários que exigem troca contínua de dados, como monitoramento em tempo real, onde múltiplas mensagens precisam ser enviadas e recebidas de forma assíncrona, destacando a necessidade de modelos alternativos como o *streaming* bidirecional (CHAMAS *et al.*, 2017).

3.5.2 Comunicação via Streaming Bidirecional

Segundo INDRASIRI e KURUPPU (2020), o *streaming* bidirecional no gRPC permite que cliente e servidor mantenham uma conexão persistente, na qual ambos podem enviar e receber fluxos de mensagens de forma assíncrona, sem a necessidade de estabelecer novas conexões para cada troca. Esse modelo é particularmente adequado para aplicações que exigem comunicação contínua, como chats em tempo real, *streaming* de dados de sensores IoT ou sistemas de realidade aumentada, onde dados brutos e resultados processados são trocados constantemente. (INDRASIRI; KURUPPU, 2020).

Figura 8 – Comunicação via *streaming* bidirecional no *gRPC*, onde o cliente (*Java*) envia uma sequência de pedidos e o servidor (*Go*) responde com múltiplos envios em paralelo, utilizando um único canal persistente.



Fonte: Elaborado por Indrasiri e Kuruppu (INDRASIRI; KURUPPU, 2020, p. 53).

A Figura 8, extraída de Indrasiri e Kuruppu (INDRASIRI; KURUPPU, p. 56), apresenta uma representação do padrão de comunicação com *streaming* bidirecional no contexto do gRPC. No diagrama, um consumidor implementado em *Java* estabelece comunicação com um serviço de gerenciamento de ordens desenvolvido em *Go*, enviando uma sequência de pedidos (*order-3*, *order-2*, *order-1*) por meio de um *stub* gRPC. O servidor, por sua vez, processa esses pedidos de forma contínua e retorna múltiplas remessas (como *shipment-1* e *shipment-2*) destinadas a diferentes locais (como *location-x* e *location-y*), sem necessidade de aguardar o término completo da transmissão do cliente. Essa interação *full-duplex* é suportada pelo protocolo HTTP/2, que possibilita a *multiplexação* de diversos fluxos de mensagens sobre uma única conexão persistente, conforme detalhado pelos autores (INDRASIRI; KURUPPU, 2020).

Essas aplicações demonstram a versatilidade do gRPC em suportar diferentes modelos de comunicação, tornando-o uma ferramenta essencial para sistemas distribuídos modernos,

especialmente em contextos de *offloading* computacional e multi-linguagem.

3.6 Considerações finais

Este capítulo apresentou os fundamentos teóricos que sustentam a pesquisa desenvolvida neste trabalho, abordando os conceitos de Computação em Nuvem, MCC, *offloading* computacional, *offloading* multi-linguagem e o *framework* gRPC, com ênfase em seus modos de comunicação unária e *streaming* bidirecional. Esses elementos formam a base conceitual necessária para compreender a proposta prática do trabalho, que consiste no desenvolvimento e avaliação de uma aplicação móvel voltada para o processamento de vídeos por meio de *offloading* computacional.

A MCC foi discutida como uma abordagem que integra dispositivos móveis a servidores remotos, sejam em nuvem ou borda, utilizando redes como 4G, 5G ou *Wi-Fi*. Essa integração permite a execução de aplicações de alta exigência computacional, superando as limitações de *hardware* dos dispositivos móveis, como capacidade de processamento e consumo energético. No contexto deste trabalho, a MCC é aplicada diretamente na arquitetura proposta, que conecta um cliente *Android* a um servidor remoto para realizar o processamento intensivo de filtros visuais em vídeos.

O *offloading* computacional foi explorado em sua capacidade de transferir carga de trabalho de dispositivos com recursos limitados para servidores remotos na nuvem ou em borda. Enquanto o conceito de *offloading* multi-linguagem foi destacado como uma evolução do *offloading* tradicional, permitindo que cliente e servidor sejam implementados em linguagens de programação distintas, aproveitando as vantagens específicas de cada uma. Na proposta deste trabalho, a escolha de *Java* para o cliente, integrado ao ecossistema *Android*, e *Go* para o servidor, exemplifica o *offloading* multi-linguagem.

Por fim, o protocolo gRPC foi abordado como uma solução para comunicação cliente-servidor, oferecendo dois modos principais: unário, ideal para requisições simples, e *streaming* bidirecional, adequado para fluxos contínuos de dados em tempo real.

No próximo capítulo, serão apresentados os Trabalhos Relacionados, nos quais esses temas foram explorados na literatura.

4 TRABALHOS RELACIONADOS

Este capítulo apresenta os principais trabalhos relacionados ao tema desta pesquisa, com foco em técnicas que exploram o *offloading* computacional como estratégia para mitigar limitações de desempenho, consumo energético e capacidade de armazenamento em dispositivos móveis. O levantamento bibliográfico foi conduzido por meio de buscas em repositórios e bases de dados de artigos científicos amplamente reconhecidos, tais como *Google Scholar*, *IEEE Xplore*, *ACM Digital Library* e periódicos da área. As pesquisas foram orientadas por palavras-chave relacionadas a *computational offloading*, *mobile cloud computing*, *multi-language offloading* e *gRPC*.

A seleção dos trabalhos seguiu como critério principal a similaridade com a proposta deste estudo, considerando aspectos como o uso de *offloading* computacional em aplicações móveis, a adoção de arquiteturas cliente-servidor, a avaliação experimental de desempenho e eficiência energética, bem como o emprego de múltiplas linguagens de programação ou diferentes modelos de comunicação. A partir desse processo, foram selecionados quatro estudos diretamente relacionados, por apresentarem maior similaridade com os objetivos, a abordagem experimental e o escopo desta pesquisa.

4.1 *An Empirical Study about the Adoption of Multi-language Technique in Computation Offloading in a Mobile Cloud Computing Scenario*

Matos *et al*. (2021) apresentaram um estudo empírico com o objetivo de avaliar como a escolha da linguagem de programação no lado servidor afeta o desempenho do *offloading* computacional em contextos de MCC. Os autores propuseram uma abordagem multi-linguagem, integrando o uso do *framework* gRPC e explorando o impacto de linguagens compiladas, parcialmente compiladas e interpretadas, como *C++*, *Java*, *Go* e *Python*, no tempo de execução e no consumo de energia durante a execução de tarefas computacionalmente intensivas. A proposta visou ainda analisar a viabilidade prática dessa abordagem e os ganhos associados ao uso de linguagens mais eficientes no contexto da MCC.

Para a avaliação, foram desenvolvidos dois aplicativos *Android*: *MatrixGRPC*, que realiza multiplicação de matrizes, e *BenchImageGRPC*, que aplica filtros em imagens com resoluções variadas. Essas aplicações foram executadas localmente e via *offloading*, conectando-se a servidores implementados nas quatro linguagens. Cada servidor foi construído com bibliotecas

específicas da linguagem utilizada, garantindo comparabilidade e isonomia entre as versões. A troca de dados utilizou chamadas gRPC com serialização em *Protocol Buffers* e comunicação do tipo unária.

Os testes foram conduzidos com base em métricas como tempo total de execução, tempo de rede e consumo de energia do dispositivo cliente. Os resultados revelaram que servidores em *C++* e *Go* superaram significativamente os demais em desempenho e eficiência energética. A linguagem *Go*, em especial, destacou-se como alternativa viável por oferecer bom equilíbrio entre complexidade de implementação e desempenho. Os resultados obtidos foram promissores para a técnica proposta: demonstraram que a abordagem multi-linguagem para *offloading* pode reduzir o tempo de processamento em até 39 vezes (no caso de multiplicação de matrizes 1000×1000 com servidor *Go*) e o consumo de energia do dispositivo móvel em aproximadamente até 96%. Embora houvesse uma exceção para imagens pequenas (0,3 MP) no *BenchImageGRPC*, onde o processamento local foi ligeiramente mais vantajoso em tempo e consumo de energia, o estudo geral confirmou os ganhos significativos associados ao uso de linguagens de programação mais eficientes no lado do servidor para tarefas computacionalmente intensivas.

Embora o trabalho tenha contribuído para destacar a relevância da linguagem de programação no contexto do *offloading*, algumas limitações podem ser apontadas. O estudo não contemplou diferentes tipos de tarefas com variados padrões de comunicação, restringindo-se a chamadas unárias. O presente trabalho expandiu esse aspecto, investigando o uso e os potenciais benefícios do gRPC com *streaming* bidirecional.

4.2 *Performance Analysis of Computational Offloading on Embedded Platforms Using the gRPC Framework*

Araújo *et al.* (2020) desenvolveram um estudo que analisou o *offloading* computacional em plataformas embarcadas, empregando o *framework* gRPC em cenários de IoT. O principal objetivo foi avaliar como diferentes linguagens de programação, sistemas operacionais e arquiteturas impactam o desempenho de tarefas computacionalmente intensas, como multiplicação de matrizes e processamento de imagens. A motivação baseou-se nas limitações típicas de dispositivos embarcados, como memória reduzida, baixo poder de processamento e autonomia energética restrita.

Os testes foram estruturados com dois casos de uso: multiplicação de matrizes

1000×1000 e aplicação de filtros de imagem (*cartoon*, *bilateral* e escala de cinza). Os clientes foram desenvolvidos em *Kotlin* e executados em *smartphones Android*. Os servidores, por sua vez, foram implementados em *C++*, *Python*, *Java*, *Go* e *Ruby* e executaram em um *BeagleBone Black* com três sistemas operacionais distintos: *Debian Linux*, *Android KitKat* e *Android Vanilla*. A comunicação entre cliente e servidor foi feita via chamadas remotas usando gRPC.

Os resultados indicaram que o servidor implementado em *C++* sobre *Debian Linux* apresentou o melhor desempenho, seguido pela linguagem *Python*. Embora a linguagem *Go* não tenha superado essas configurações em tempo de execução, demonstrou um desempenho estável e competitivo, aliado a importantes vantagens técnicas que justificam sua adoção neste trabalho.

Entre seus diferenciais, destacam-se a integração direta com o *framework* gRPC, utilizando *Protocol Buffers* como formato de serialização binária de alto desempenho. Portanto, a escolha da linguagem *Go* fundamenta-se não apenas em métricas de desempenho, mas principalmente em seu equilíbrio entre eficiência e simplicidade de implementação, consolidando-se como uma solução robusta para a implementação do servidor neste cenário.

Embora o estudo de Araújo *et al.* (2020) traga contribuições relevantes ao explorar o *offloading* em múltiplas linguagens e sistemas operacionais embarcados, apresenta limitações que diferem do escopo da presente pesquisa. Uma dessas limitações foi a ausência de análise sobre o consumo de energia, aspecto crítico em sistemas embarcados.

4.3 *On Computation Offloading and Energy Efficiency on Android Devices*

Carvalho *et al.* (2023) realizaram uma pesquisa experimental voltada à avaliação da eficiência energética do *offloading* computacional em dispositivos *Android*. O foco central foi investigar sob quais condições a transferência de carga computacional para servidores externos, em *edge* e *cloud*, representa de fato uma vantagem em termos de economia de bateria. Os autores adaptaram o *benchmark EdgeBench* para incluir métricas de energia e processaram três tipos de tarefas: *AudioTranscriber* para transcrição de áudio, *ImageScaler* para redimensionamento de imagem e *SensorReader* para leitura de dados simulados.

Os experimentos foram realizados em três dispositivos *Android* distintos (*Xiaomi Redmi 7*, *Samsung Galaxy A5* e *Samsung Galaxy Note 3*), simulando diferentes faixas de capacidade computacional. A comunicação cliente–servidor foi implementada via *sockets*, e o consumo de energia foi monitorado com o uso do *Battery Historian*. Os testes revelaram que, para tarefas com baixo volume de dados, como leitura de sensores, o processamento local

apresentou melhor desempenho energético.

Nas tarefas mais intensivas, como a transcrição de áudio, o *offloading* foi responsável por economias expressivas de energia nos dispositivos móveis analisados. Especificamente, observou-se uma redução de aproximadamente 99% no consumo de energia nos dispositivos *Samsung* (*Galaxy A5 e Galaxy Note 3*), tanto para *offloading* para o *Edge* quanto para a *Cloud*. Para o *Xiaomi Redmi* 7 (XR7), a economia foi de cerca de 80%. Esses resultados evidenciam o impacto positivo do *offloading* computacional na autonomia energética de dispositivos com diferentes perfis de *hardware*, especialmente quando aplicado a tarefas de alta demanda computacional.

Além disso, os resultados mostraram que a economia de energia foi acompanhada por uma redução significativa no tempo de execução para as tarefas computacionalmente mais exigentes. Por exemplo, na aplicação de redimensionamento de imagens, o tempo de uso da *CPU* no processamento local variou entre 63,0 e 79,0 segundos, enquanto no *offloading*, esse tempo foi reduzido para valores entre 14,0 e 28,0 segundos, a depender do dispositivo. A análise também identificou padrões consistentes de uso da *CPU*, os quais explicam parte dos ganhos energéticos, especialmente na tarefa de transcrição de áudio, que apresentou os maiores picos de consumo em execução local, como no caso do *Galaxy Note* 3, que registrou 238,0 segundos de uso da *CPU* e descarga de 32,9 mAh.

O presente trabalho amplia o escopo de pesquisa ao explorar o uso combinado de gRPC com *streaming* bidirecional e linguagens compiladas, como *Go*, analisando não apenas o desempenho, mas também o consumo de energia.

4.4 *A comparative study about the performance of multi-language tools in computation offloading scenarios*

Matos *et al*. (2025) analisam, de forma empírica e comparativa, o impacto de diferentes ferramentas de comunicação e linguagens de programação no desempenho do *offloading* computacional em cenários de MCC. O estudo enfatiza o papel de mecanismos de serialização como *Protocol Buffers* e *FlatBuffers* na interoperabilidade entre linguagens e adota o paradigma de *Remote Procedure Call* (RPC) para viabilizar a comunicação entre processos distribuídos implementados em linguagens distintas.

A avaliação experimental considerou um cliente *Android* desenvolvido em *Java*, que realizou o *offloading* de tarefas para servidores implementados em *Java*, *Go* e *C++*, utilizando três frameworks de comunicação: *Apache Thrift*, *gRPC* com *Protocol Buffers* e *gRPC* com

FlatBuffers. Foram utilizadas três aplicações de *benchmark*: *SorterMLApp*, *MatrixMLApp* e *BenchImageMLApp*, contemplando ordenação de vetores, operações com matrizes e processamento de imagens. Ao todo, foram avaliados 81 cenários distintos, cada um executado 33 vezes, totalizando 2673 execuções. As métricas analisadas incluíram tempo total de execução, tempo de rede, consumo de energia do dispositivo móvel e volume de dados trafegados, com medições realizadas por meio do *Monsoon PowerMonitor* e do *Wireshark*.

Os resultados indicaram que o *Apache Thrift* e o *gRPC* com *FlatBuffers* apresentaram desempenho superior ao *gRPC* com *Protocol Buffers*, especialmente em cenários com maior volume de dados. O *gRPC* com *Protocol Buffers* foi o *framework* mais lento em aproximadamente 81% dos cenários e o mais intensivo em consumo de energia em cerca de 78%, consumindo, no mínimo, 19% mais tempo e 20% mais energia em relação à melhor alternativa. Em contraste, o *Apache Thrift* destacou-se como o *framework* mais rápido em aproximadamente 83% dos cenários e o mais eficiente energeticamente em cerca de 66% dos casos.

No que se refere às linguagens de programação no lado servidor, a linguagem *Go* apresentou o melhor desempenho global, enquanto implementações em *Java*, utilizando apenas bibliotecas nativas, chegaram a consumir até cinco vezes mais energia e apresentar tempo de execução até seis vezes superior quando comparadas a *Go* e *C++*. Adicionalmente, o estudo confirmou a forte influência da rede no desempenho do *offloading*, indicando que até 90% do tempo total de execução pode ser atribuído às operações de comunicação. Com base nesses resultados, os autores concluem que a escolha do *framework* de comunicação e da linguagem de programação no servidor é determinante para a eficiência do *offloading* computacional.

Diferentemente do estudo de Matos *et al*. (2025), que analisa vários *frameworks* e linguagens com foco em chamadas RPC unárias, este trabalho investiga o uso do *gRPC* com *streaming* bidirecional. Enquanto o estudo comparado prioriza a avaliação de tecnologias de comunicação como *Apache Thrift* e formatos de serialização como *Protocol Buffers*. Nesse trabalho, o foco recai sobre o impacto do modelo de comunicação e da arquitetura multi-linguagem nas métricas de desempenho e no consumo de energia, ampliando a compreensão sobre a eficiência do *offloading* em aplicações móveis ao avaliar como o *streaming* bidirecional e a escolha da linguagem do servidor afetam resultados práticos.

4.5 Considerações Finais

Este capítulo apresentou e analisou criticamente os principais trabalhos relacionados ao tema do presente estudo. A partir dessa análise comparativa, observou-se que, embora relevantes, os trabalhos de Matos *et al.*(2021, 2025), Araújo *et al.*(2020) e Carvalho *et al.*(2023) apresentam limitações metodológicas e técnicas que ainda não foram plenamente superadas pela literatura.

De forma empírica, Matos *et al.* (2021) concentraram sua avaliação em *offloading* multi-linguagem utilizando exclusivamente chamadas unárias, o que restringe a análise de cenários com comunicação contínua. Em um estudo mais recente, Matos *et al.* (2025) ampliaram essa investigação ao comparar diferentes frameworks RPC multi-linguagem, incluindo *Apache Thrift* e *gRPC* com *FlatBuffers*. Apesar dos avanços, o trabalho ainda se limita a padrões de comunicação baseados em chamadas unárias, não explorando mecanismos de *streaming* contínuo, especialmente relevantes para aplicações que manipulam grandes volumes de dados.

Araújo *et al.* (2020) investigaram o impacto do sistema operacional e da linguagem de programação no desempenho do *offloading*, porém não contemplaram métricas de energia, fundamentais no contexto de dispositivos móveis. Já o estudo de Carvalho *et al.* (2023) baseou-se em um modelo de comunicação por *sockets*, tecnologicamente defasado frente às soluções modernas baseadas em RPC, além de não realizar uma comparação entre diferentes linguagens de programação no lado servidor.

Com o objetivo de mitigar as limitações observadas nos trabalhos relacionados, o presente estudo adotou uma abordagem baseada no uso da linguagem *Go* no lado servidor, tendo a linguagem *Java* como linha de base para comparação, integrada ao *framework* gRPC com suporte a *streaming* bidirecional. No lado cliente, utilizou-se a linguagem *Java* em ambiente *Android*, permitindo a avaliação de um cenário de *offloading* computacional multi-linguagem.

A Tabela 1 apresenta uma visão consolidada desses trabalhos, permitindo uma comparação direta quanto às ferramentas adotadas, métricas avaliadas e principais limitações identificadas.

Tabela 1 – Comparação entre os trabalhos relacionados e o presente estudo

Trabalho	**Comunicação**	**Linguagens Servidor**	**Métricas Avaliadas**	**Aplicativos Desenvolvidos**
(MATOS *et al.*, 2021)	*gRPC* (unário)	*Go*, *C++*, *Java*	Tempo de execução, energia	*MatrixGRPC*, *BenchImageGRPC*
(MATOS *et al.*, 2025)	*gRPC* (unário), *Thrift*	*Go*, *C++*, *Java*	Tempo de execução, tempo de rede, energia, dados trafegados	*SorterMLApp*, *MatrixMLApp*, *BenchImageMLApp*
(ARAUJO *et al.*, 2020)	*gRPC* (unário)	*Go*, *C++*, *Java*, *Python*	Tempo de processamento	Multiplicação de matrizes 1000×1000, Processamento de imagens (filtros: *cartoon*, *bilateral*, *escala de cinza*)
(CARVALHO *et al.*, 2023)	*Sockets TCP*	*Java*	Tempo de execução	*AudioTranscriber*, *ImageScaler*, *SensorReader*
Este trabalho	***gRPC* (streaming bidirecional)**	***Go*, *Java***	**Tempo total, Consumo de Energia, Uso de *CPU***	***BenchVideoGRPC***

Fonte: Elaborado pelo autor.

5 ARQUITETURA DA SOLUÇÃO E AMBIENTE EXPERIMENTAL

Os capítulos anteriores deste trabalho apresentaram conceitos fundamentais para a compreensão da proposta desenvolvida, incluindo tópicos como Computação em Nuvem, Computação em Nuvem Móvel, gRPC e conceitos ligados à técnica de *offloading* computacional. Além disso, foram discutidos trabalhos relacionados com foco em abordagens de *offloading* que exploram diferentes linguagens e arquiteturas.

Considerando esse contexto, este trabalho avalia o desempenho do *offloading* computacional multi-linguagem em um cenário de comunicação baseado em *streaming* bidirecional via *gRPC*, com foco nos impactos sobre o tempo de execução e o consumo energético em aplicações móveis. Para viabilizar essa avaliação, foi necessário o desenvolvimento de uma aplicação móvel experimental, utilizada como base para a execução de tarefas computacionalmente intensivas tanto de forma local quanto remota. A Seção 5.2 apresenta a configuração geral dos experimentos, onde a seção 5.2.1 apresenta os dispositivos utilizados na condução dos experimentos, bem como suas principais características técnicas. Em seguida, a Seção 5.2.2 descreve os fatores considerados no experimento e porque foram selecionados. A Seção 5.2.3 detalha o número de iterações realizadas para cada cenário experimental e os critérios adotados para garantir a confiabilidade dos resultados. Por fim, a Seção 5.2.4 apresenta as métricas coletadas para a avaliação do desempenho e da eficiência energética da solução proposta.

5.1 O aplicativo BenchVideoGRPC

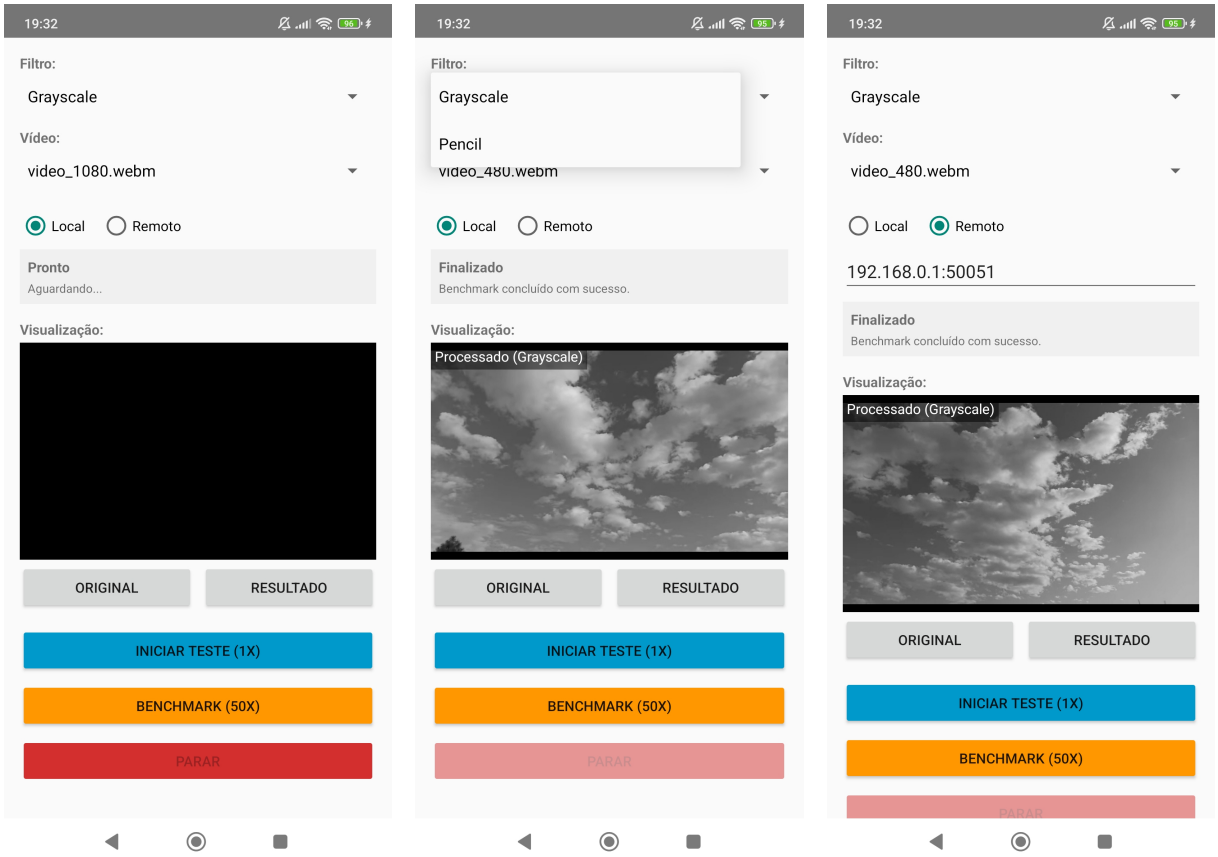
A arquitetura do sistema é composta por uma aplicação cliente, escrita em *Java* e executada em um dispositivo *Android*, e duas aplicações servidoras distintas: uma desenvolvida em *Go* e outra em *Java*. Para fins de experimento, os mesmos algoritmos de processamento de imagem foram implementados de forma equivalente, tanto na aplicação cliente *Android* quanto nas duas aplicações servidoras, em *Go* e *Java*.

A presente pesquisa utilizará uma estrutura de aplicação baseada em conceitos semelhantes aos adotados em (MATOS *et al.*, 2021), que demonstram os benefícios de uma arquitetura multi-linguagem e da comunicação eficiente no processo de *offloading* computacional. Como base experimental, Será desenvolvida a aplicação *BenchVideoGRPC*[1], versão modificada da aplicação *BenchImageGRPC*, originalmente voltada ao processamento de imagens, agora

[1] Repositório disponível em: <https://github.com/MiguelFariasDev/BenchVideoGRPC>.

adaptada especificamente para lidar com vídeos. A Figura 9a apresenta a tela principal da aplicação, enquanto a Figura 9b ilustra a interface de seleção dos filtros disponíveis. A Figura 9c mostra a configuração do modo de execução remoto.

Figura 9 – Capturas de tela da aplicação *BenchVideoGRPC*



(a) Tela principal (b) Seleção de filtros (c) Configuração do modo remoto

Fonte: Elaborado pelo autor.

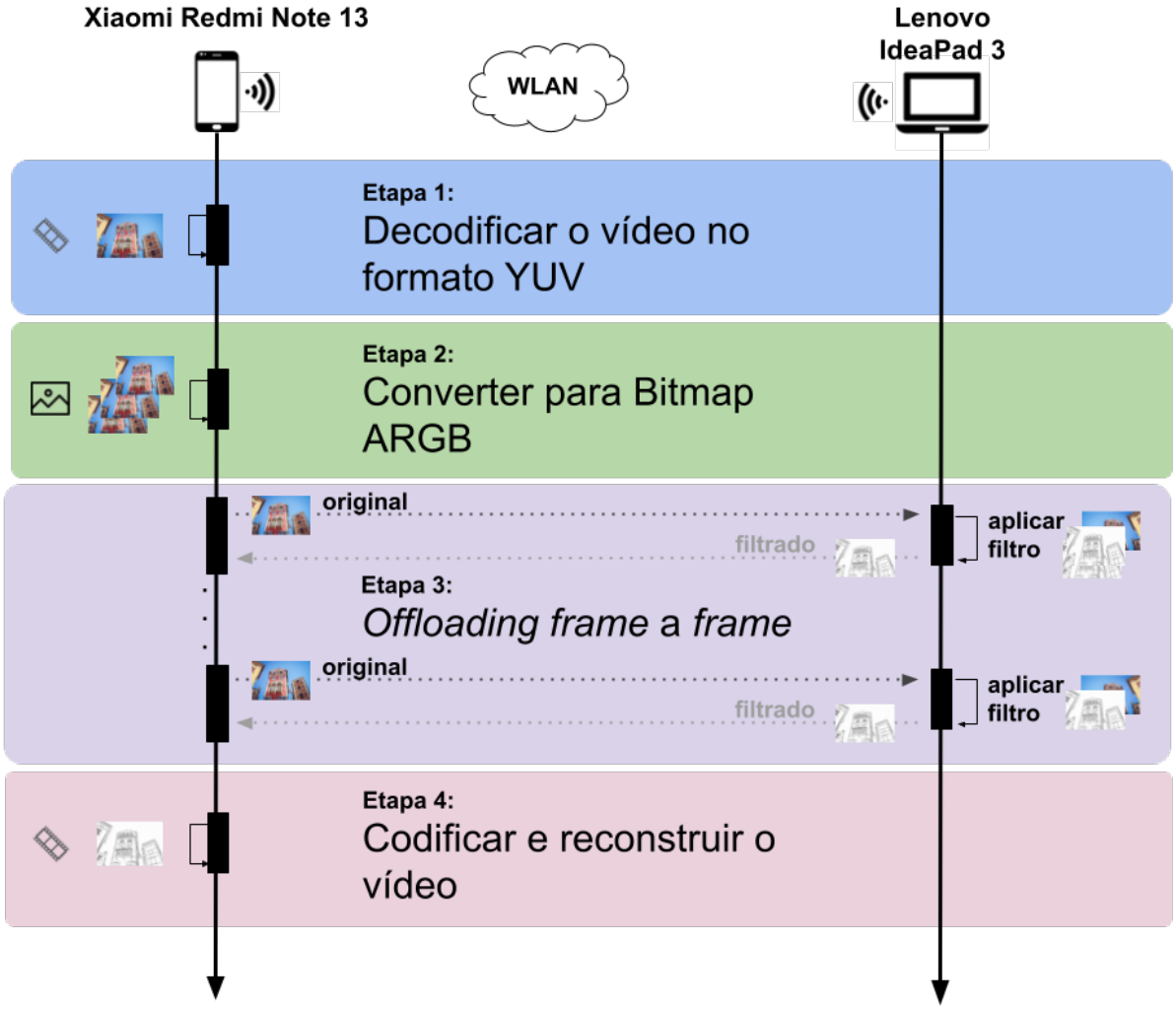
As tarefas realizadas pela *BenchVideoGRPC* concentram-se em duas operações principais de processamento de imagem aplicadas aos *frames* de vídeo: o filtro *Grayscale*, que converte o vídeo colorido em escala de cinza, e o filtro *Pencil*, que gera um efeito de desenho a lápis. A escolha dos filtros *Grayscale* e *Pencil* fundamenta-se na necessidade de avaliar o desempenho da solução sob diferentes níveis de complexidade de processamento. O filtro *Grayscale* representa uma operação de baixa complexidade computacional, enquanto o filtro *Pencil* envolve operações mais complexas, caracterizando uma carga computacional mais intensiva. Dessa forma, a utilização desses dois filtros permite analisar o comportamento da solução tanto em cenários de baixo custo computacional quanto em situações mais exigentes, possibilitando uma comparação mais abrangente dos impactos do *offloading* sobre o tempo de

execução e o consumo energético do dispositivo móvel.

Para os experimentos também foram definidos três níveis de qualidade de vídeo, correspondentes as seguintes resoluções: 480p, 720p e 1080p. A escolha dessas resoluções permite avaliar o comportamento do *offloading* computacional sob diferentes volumes de dados e níveis de processamento. A resolução 480p representa um cenário de baixa exigência computacional, enquanto 720p e 1080p caracterizam, respectivamente, níveis intermediário e elevado de complexidade. Além disso, embora tenham sido realizados testes preliminares em resoluções mais elevadas, como 1280p (HD) e 4K, por conta de limitações de *hardware* do *smartphone* utilizado, não foram consideradas essas resoluções nesse estudo.

5.1.1 Pipeline de processamento

Figura 10 – Fluxo de processamento de vídeo da solução proposta.



Fonte: Elaborado pelo autor.

O *pipeline* de processamento inicia-se no dispositivo móvel, onde os *frames* do vídeo são extraídos por meio das APIs nativas de mídia da plataforma *Android*. A decodificação é realizada pelo *MediaCodec*, que disponibiliza cada quadro inicialmente em formato YUV, que é um modelo de representação de cores que separa a informação de luminância (Y) dos componentes de crominância (U e V), amplamente utilizado em sistemas de vídeo por permitir maior eficiência de processamento. Em seguida, os *frames* são convertidos para objetos do tipo *Bitmap*, internamente representados na configuração *Alpha Red Green Blue* (ARGB), etapa necessária para viabilizar a manipulação da imagem em memória e a aplicação dos filtros.

No cenário de execução remota, cada *frame* convertido é serializado em um vetor de *bytes* e enviado ao servidor por meio de *streaming* bidirecional via gRPC. No lado do servidor, os dados recebidos são desserializados e reconstruídos em estruturas de imagem manipuláveis em memória, sobre as quais o algoritmo de filtragem selecionado é aplicado diretamente. Após a conclusão do processamento, a imagem resultante é novamente serializada e transmitida de volta ao cliente utilizando o mesmo canal de comunicação. A aplicação *Android* recebe os dados processados, reconstrói o *Bitmap* correspondente ao *frame* filtrado e o insere no codificador de vídeo local, respeitando a ordem sequencial dos quadros para garantir a correta reconstrução do vídeo de saída.

No cenário de execução local, o *frame* percorre um fluxo equivalente de manipulação em memória, no entanto, o algoritmo de filtragem é executado diretamente na CPU do dispositivo móvel, sem a etapa de comunicação em rede. O funcionamento básico desse *pipeline*, em ambos os cenários de execução, é ilustrado na Figura 10.

5.1.1.1 Controle de Fluxo e Tratamento de Falhas

Para garantir a ordem correta dos quadros processados e preservar a estabilidade do dispositivo móvel durante a execução dos experimentos, a aplicação adota um controle de fluxo síncrono. Nesse modelo, cada quadro do vídeo é enviado ao servidor individualmente, e o cliente aguarda o retorno do processamento antes de prosseguir com o envio do próximo quadro. Essa estratégia foi adotada principalmente para evitar problemas relacionados ao uso excessivo de memória e à organização dos dados processados. Em testes iniciais, observou-se que a velocidade de extração dos quadros no dispositivo móvel podia ser superior à capacidade de transmissão da rede, ocasionando o acúmulo de imagens em memória, ocasionando erros do tipo *Out of Memory Error*.

Além disso, a transmissão sequencial simplifica substancialmente a lógica do cliente, pois garante, por definição, que a ordem de recebimento seja idêntica à de envio. Caso fosse adotado um modelo assíncrono, os quadros poderiam retornar fora de ordem devido a variações na latência da rede ou no tempo de processamento do servidor. Isso exigiria a implementação de um *buffer* de reordenação no dispositivo móvel, aumentando o consumo de memória e a complexidade algorítmica para gerenciar a remontagem correta da sequência de *frames* antes da codificação final.

No que se refere ao tratamento de falhas, foi definido um tempo limite de 8 segundos para a resposta do servidor a cada requisição. Caso o processamento remoto não seja concluído dentro desse intervalo ou ocorra uma falha de comunicação, a execução da iteração é abortada e os dados coletados são registrados como falha de execução. Essa abordagem foi adotada em virtude das observações realizadas durante a fase de testes, nas quais se verificou que tentativas de recuperação de erro resultavam em inconsistências temporais no vídeo final, como a repetição indevida de quadros anteriores, gerando um efeito de congelamento do vídeo, e divergências na duração total do vídeo em relação ao original.

5.2 Configuração dos Experimentos

Esta seção descreve a configuração experimental adotada neste estudo, apresentando uma visão geral dos dispositivos utilizados, do ambiente de execução e das condições sob as quais os experimentos foram conduzidos, de modo a garantir a clareza dos resultados apresentados. Esses elementos são descritos de forma detalhada nas subseções que se seguem, abrangendo a caracterização dos dispositivos (5.2.1), os fatores experimentais considerados (5.2.2), o número de iterações realizadas (5.2.3) e as métricas adotadas para avaliação (5.2.4).

5.2.1 Dispositivos utilizados

A seleção dos dispositivos empregados nos experimentos segue uma abordagem metodológica alinhada à literatura, em especial ao trabalho de (MATOS *et al.*, 2021), no qual o dispositivo móvel representa um perfil intermediário, sujeito a limitações inerentes de energia e capacidade de processamento. Em contrapartida, o servidor dispõe de poder computacional significativamente superior, aliado a uma fonte de energia praticamente ilimitada.

O *Xiaomi Redmi Note 13* foi adotado como representante de um dispositivo móvel

recente e de perfil intermediário, oferecendo maior capacidade de processamento, memória e autonomia energética, comparado ao *smartphone* utilizado pelos experimentos realizados em (MATOS *et al.*, 2021).

Como servidor de processamento, foi utilizado um *notebook Lenovo IdeaPad 3*, configurado para atuar como servidor no ambiente experimental. O dispositivo apresenta desempenho computacional superior ao do *smartphone* utilizado nos testes, sendo um candidato natural a receber e processar tarefas submetidas pelo *smartphone* cliente.

A infraestrutura de rede utilizada nos experimentos contou com dois roteadores: um *D-Link DSL-2740*, operando na frequência de 2,4 GHz, e um *TP-Link XC220*, que opera exclusivamente na frequência 5 GHz. Tanto o dispositivo móvel quanto o servidor conectaram-se à rede exclusivamente por meio de comunicação sem fio, configurando uma topologia em estrela. Ressalta-se que a rede foi dedicada exclusivamente à execução dos experimentos, sem a presença de tráfego concorrente de outros dispositivos.

5.2.2 Fatores do experimento

A escolha dos fatores experimentais foi orientada pelo objetivo de analisar o impacto do *offloading* computacional sob diferentes condições operacionais, contemplando variações no modo de processamento, na implementação do servidor, na qualidade do vídeo e na conectividade de rede.

Os experimentos foram conduzidos a partir da combinação dos fatores definidos neste estudo, incluindo o tipo de processamento adotado (local e remoto), o filtro de vídeo aplicado (*Grayscale* e *Pencil*), a linguagem de programação empregada no servidor (*Go* e *Java*), as resoluções de vídeo consideradas (480p, 720p e 1080p) e o tipo de rede utilizada (Wi-Fi de 2,4 GHz e Wi-Fi 5 GHz). A combinação desses fatores resulta em um total de 48 cenários experimentais distintos, possibilitando uma avaliação comparativa abrangente do comportamento do sistema.

Os filtros de vídeo e as resoluções adotadas foram previamente discutidos na Seção 5.1, na qual são apresentadas as motivações técnicas para sua escolha e suas complexidades computacionais inerentes.

O tipo de processamento foi incluído como fator experimental para permitir uma comparação direta entre a execução local no dispositivo móvel e a execução remota via *offloading*, de modo a quantificar ganhos ou perdas ao delegar o processamento ao servidor. Também

considerou-se a linguagem de programação do servidor para avaliar como diferentes ambientes de execução afetam o tempo de processamento e o consumo de recursos em uma arquitetura multi-linguagem. Foram definidos três cenários: (I) Execução local, em que todo o processamento ocorre na *CPU* do dispositivo móvel e serve como referência para medir o custo computacional e o consumo de energia do cliente; (II) Execução remota com servidor em *Java*, escolhida por sua similaridade com o ambiente Android e por sua ampla adoção em aplicações de servidor; (III) Execução remota com servidor em *Go*, selecionada com base em evidências que apontam bom desempenho e eficiência quando utilizada no lado do servidor.

Esses cenários permitem analisar não o *offloading*, mas também como a escolha da linguagem e do ambiente de execução no servidor influencia o tempo total de resposta e o consumo energético do cliente. Estudos comparativos na literatura sustentam essa diferenciação e justificam a avaliação de ambos os ambientes (SOARES, 2024; ANDERSSON; BRENDEN, 2018; CAVALCANTE, 2025).

O tipo de rede foi considerado como variável experimental com o objetivo de capturar a influência das condições de conectividade sobre o desempenho do *offloading*, uma vez que variações na largura de banda e na latência impactam diretamente o tempo de comunicação entre cliente e servidor e, por conseguinte, o tempo total de resposta e o consumo de recursos. Destaca-se que a operação na rede Wi-Fi de 5 GHz tende a apresentar menor interferência proveniente de dispositivos que costumam operar na faixa de 2,4 GHz, o que diminui a latência e promove uma maior estabilidade de conexão.

Essa abordagem permite identificar tanto a influência isolada de cada fator quanto os efeitos combinados entre eles, fornecendo elementos para uma análise mais ampla dos resultados obtidos.

5.2.3 Iterações

Conforme recomendado por Jain (1991), a repetição de experimentos tem como objetivo mitigar a influência de variações pontuais e interferências externas, especialmente relacionadas à rede, assegurando maior robustez estatística aos resultados obtidos. Os resultados reportados correspondem a médias calculadas a partir das múltiplas execuções realizadas.

Com esse propósito, cada cenário experimental foi executado 50 vezes, totalizando 2 400 iterações ao longo de toda o conjunto de experimentos (2 tipos de processamento $\times$ 2 filtros $\times$ 2 linguagens de programação $\times$ 3 resoluções de vídeo $\times$ 2 tipos de rede $\times$ 50 repetições).

5.2.4 Métricas Coletadas

As métricas foram definidas com o objetivo de fornecer uma visão quantitativa e comparável do impacto do *offloading* computacional nos diferentes cenários experimentais, permitindo analisar tanto os custos associados à execução local quanto os benefícios potenciais da execução remota. A definição dessas métricas está alinhada às práticas consolidadas na literatura de avaliação de desempenho em *Mobile Cloud Computing*, priorizando indicadores capazes de capturar o tempo de execução e o impacto energético da aplicação.

5.2.4.1 Tempo médio

O tempo médio corresponde à média aritmética dos tempos de execução individuais de cada vídeo em um determinado cenário experimental. Assim, com essa métrica, é possível caracterizar o comportamento médio do sistema em cada cenário avaliado, viabilizando comparações diretas entre diferentes combinações de fatores, tais como linguagens de programação, modo de execução, resoluções de vídeo e tipos de rede.

5.2.4.2 Consumo de Bateria

A avaliação do consumo de energia foi realizada por meio do registro de *logs* da corrente elétrica do dispositivo durante a execução das tarefas experimentais. Para a obtenção dos dados, utilizou-se a API `BatteryManager` do sistema *Android*, acessando a propriedade `BATTERY_PROPERTY_CURRENT_NOW`, que retorna a corrente elétrica consumida pela bateria, expressa em microamperes (μA), considerando o consumo do dispositivo como um todo, e não apenas da aplicação em questão.

Durante cada iteração experimental, os valores de corrente instantânea fornecidos pela API foram amostrados ao longo do tempo. A partir dessa série de amostras, foi calculada a corrente média, inicialmente expressa em microamperes (μA). Em seguida, essa grandeza foi convertida para miliampères (mA), de modo a padronizar a unidade utilizada na análise e apresentação dos resultados.

Com base no tempo total de duração da iteração, denotado por t e medido em segundos, foi estimada a carga elétrica consumida durante a execução, expressa em miliampère-hora (mAh). Essa estimativa foi obtida a partir da relação entre a corrente média e o tempo de execução, conforme a Equação 5.1:

$$C_{iter} = \bar{I}_{\text{mA}} \times \frac{t}{3600} \tag{5.1}$$

onde C_{iter} representa a carga elétrica estimada consumida por iteração, expressa em miliampère-hora (mAh), $\bar{I}_{\text{mA}}$ corresponde à corrente média ao longo da iteração, em miliampères (mA), e t é a duração média das iterações em segundos. O fator 3600 é utilizado para a conversão de segundos para horas. Conforme descrito por (HALLIDAY *et al.*, 2016), essa formulação decorre diretamente da definição de carga elétrica.

A adoção dessa abordagem para a estimativa do consumo energético foi motivada pela necessidade de analisar, o comportamento de cada cenário experimental e de cada iteração individualmente. Cabe destacar que os valores obtidos por meio dessa metodologia representam estimativas aproximadas do consumo energético, uma vez que a API utilizada fornece medições globais do dispositivo e não permite a separação exata do consumo atribuído exclusivamente à aplicação.

5.2.4.3 Utilização de CPU

A utilização de CPU do dispositivo móvel foi monitorada com o objetivo de mensurar o impacto do *offloading* computacional no uso da CPU. Para essa medição, foi utilizada a API `android.os.Process.getElapsedCpuTime()`, que retorna o tempo total, em milissegundos, durante o qual o processo da aplicação esteve em execução no processador.

O cálculo da utilização da CPU foi realizado a partir da diferença entre o tempo de CPU registrado ao final e ao início da execução do vídeo (`endCpuTime - startCpuTime`), dividido pela diferença do tempo de execução (`endTime - startTime`). O resultado é convertido em porcentagem, representando a fração de uso do processador pelo aplicativo durante o intervalo monitorado.

5.2.5 Resumo Geral dos Experimentos

A Tabela 2 apresenta uma visão consolidada dos parâmetros e das configurações adotadas nos experimentos realizados, sintetizando os principais aspectos do ambiente de execução, dos fatores avaliados e das métricas consideradas ao longo da análise experimental.

Tabela 2 – Resumo geral dos experimentos

Objetivo geral	Avaliar o desempenho do *offloading* computacional utilizando *streaming* bidirecional via *gRPC* entre um cliente *Android* (*Java*) e um servidor *Remoto*.
Sistema	*BenchVideoGRPC*; *smartphone* com processador *Qualcomm Snapdragon 685* (2,8 GHz, *octa-core*), 8 GB de RAM e *Android 15*; *notebook* com processador *AMD Ryzen 5 5500U* (2,10 GHz, 6 cores), 20 GB de RAM e *Fedora 43* (64 bits). As redes utilizadas foram uma rede Wi-Fi de 2,4 GHz com roteador *D-Link DSL-2740* e uma rede Wi-Fi de 5 GHz com *TP-Link EC230*.
Fatores	Tipo de processamento (local ou remoto); filtro (*Pencil* ou *Grayscale*); linguagens de programação servidoras (*Java* e *Go*); resoluções de vídeo (480p, 720p e 1080p); tipo de rede (2,4 GHz e Wi-Fi 5 GHz).
Iterações	Cada experimento foi executado 50 vezes para cada combinação de fatores, totalizando 2 400 iterações (2 tipos de processamento $\times$ 2 filtros $\times$ 2 linguagens de programação $\times$ 3 resoluções de vídeo $\times$ 2 tipos de rede $\times$ 50 iterações).
Métricas	Tempo Médio; Utilização de CPU; Consumo de Energia.

Fonte: Elaborado pelo autor.

6 RESULTADOS

Este capítulo apresenta os resultados experimentais organizados por filtro. Para assegurar consistência estatística e reduzir a influência de ruídos nas métricas coletadas, adotou-se um procedimento de pré-processamento dos dados em duas etapas. Primeiro, identificaram-se e removeram-se valores discrepantes por meio do método do Intervalo Interquartil. Em seguida, selecionaram-se aleatoriamente 35 iterações, e com base nesses valores aleatórios, foram gerados os gráficos comparativos, os quais incluem barras de erro representando o Intervalo de Confiança de 95%.

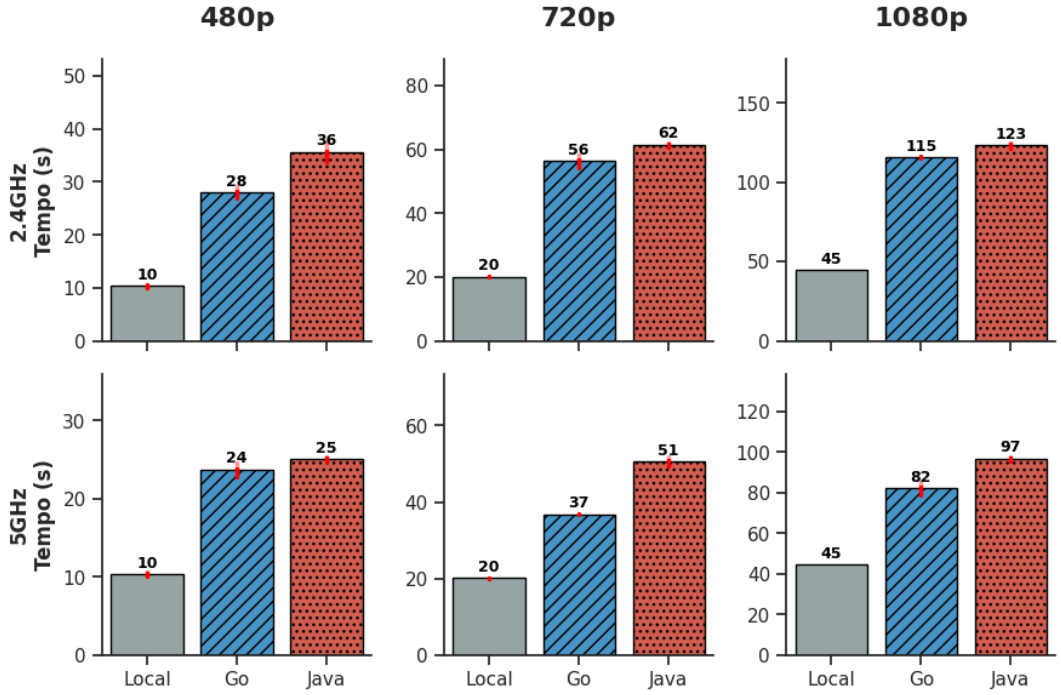
6.1 Filtro *Grayscale*

6.1.1 Tempo médio por iteração

A análise dos tempos médios de execução do filtro *Grayscale* demonstra que a execução local apresenta desempenho superior em todas as resoluções avaliadas. Esse resultado está diretamente relacionado à baixa complexidade computacional da operação, que demanda pouco processamento e, portanto, não se beneficia de forma significativa da execução remota. No cenário de execução local, os tempos médios por iteração apresentaram-se significativamente inferiores aos observados para cenários de processamento remoto, embora também tenham demonstrado crescimento progressivo conforme o aumento da resolução do vídeo. Em 480p, o tempo médio foi de aproximadamente 10s por iteração, elevando-se para cerca de 20s em 720p e atingindo aproximadamente 45s em 1080p.

No que se refere à rede, observa-se que a utilização da rede Wi-Fi de 5 GHz apresenta tempos médios inferiores quando comparada à rede Wi-Fi de 2,4 GHz, em todas as resoluções analisadas. Especificamente, em 480p a redução foi de 14% para o servidor em *Go* e 30% para o servidor em *Java*; em 720p, de 34% (*Go*) e 17% (*Java*); e em 1080p, de 28% (*Go*) e 21% (*Java*). Esse comportamento evidencia que a rede Wi-Fi de 5 GHz contribui positivamente para a redução do tempo médio total no *offloading*. Ainda assim, mesmo com os ganhos obtidos pela rede Wi-Fi de 5 GHz, os tempos médios permanecem superiores aos observados no cenário de execução local. A comparação entre as linguagens mostra que o servidor escrito em *Go* apresentou tempos médios consistentemente menores do que o servidor escrito em *Java*, indicando maior eficiência na execução remota. Entretanto, essa vantagem entre linguagens não foi suficiente para tornar o

Figura 11 – Tempo médio de execução do filtro *Grayscale* nos cenários local e remoto.



Fonte: Elaborado pelo autor.

offloading competitivo frente ao cenário local.

6.1.2 Bateria

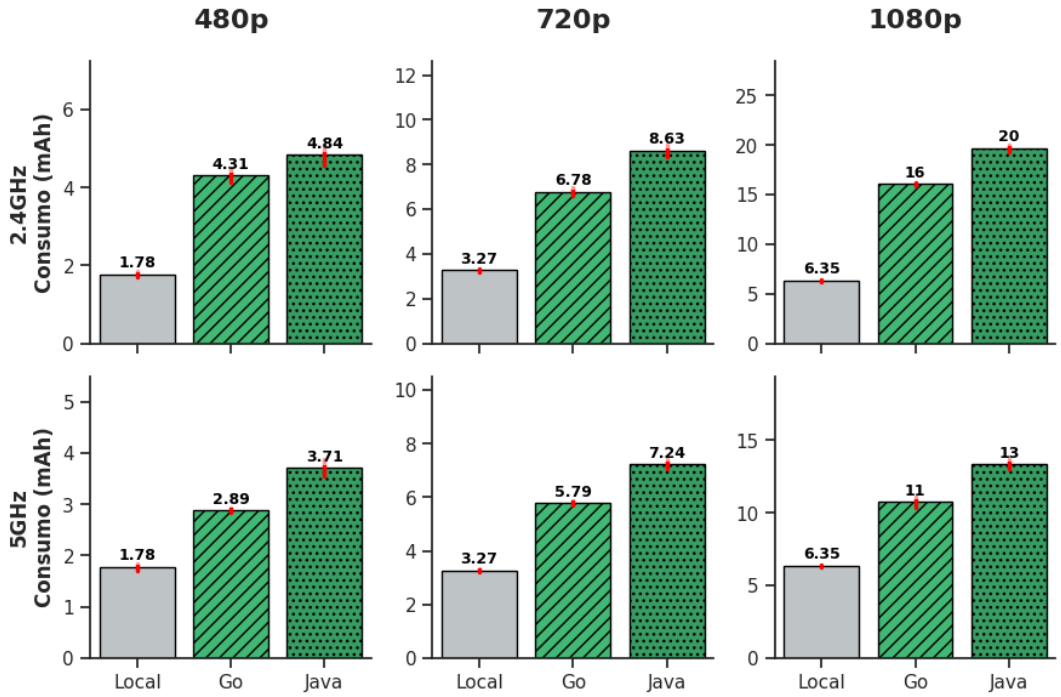
A análise do consumo de bateria indica que a execução local apresenta comportamento mais eficiente em todos os cenários avaliados, como a tarefa é concluída mais rapidamente no cenário local, o aplicativo permanece ativo por um intervalo de tempo reduzido, refletindo em menor consumo energético por iteração.

De forma geral, os cenários com *offloading* apresentaram consumo médio de bateria mais elevado quando comparados à execução local. Esse comportamento ocorre porque os atrasos associados à comunicação remota fazem com que o aplicativo permaneça em execução por mais tempo, mantendo componentes como CPU ativos por um período prolongado. Como consequência, o consumo de bateria por iteração é ampliado.

Os dados apresentados na Figura 12 evidenciam esse comportamento. Enquanto no cenário local o consumo médio por iteração permaneceu em valores mais baixos, nos cenários com *offloading* observam-se valores mais elevados.

No cenário em que foi utilizada a rede Wi-Fi de 5 GHz, foi registrado um consumo médio de bateria inferior ao observado na rede Wi-Fi de 2,4 GHz, conforme também indicado

Figura 12 – Consumo médio de bateria por iteração em mAh do filtro *Grayscale* nos cenários local e remoto.



Fonte: Elaborado pelo autor.

na Figura 12. Especificamente, em 480p a redução foi de 33% para o servidor em *Go* e 23% para o servidor em *Java*; em 720p, de 14% (*Go*) e 16% (*Java*); e em 1080p, de 31% (*Go*) e 35% (*Java*). Esse resultado sugere que a rede Wi-Fi de 5 GHz reduz parcialmente o consumo de energia quando comparado aos resultados da rede Wi-Fi de 2,4 GHz. Ainda assim, mesmo com esse ganho, os valores observados no cenário de *offloading* permanecem superiores aos da execução local.

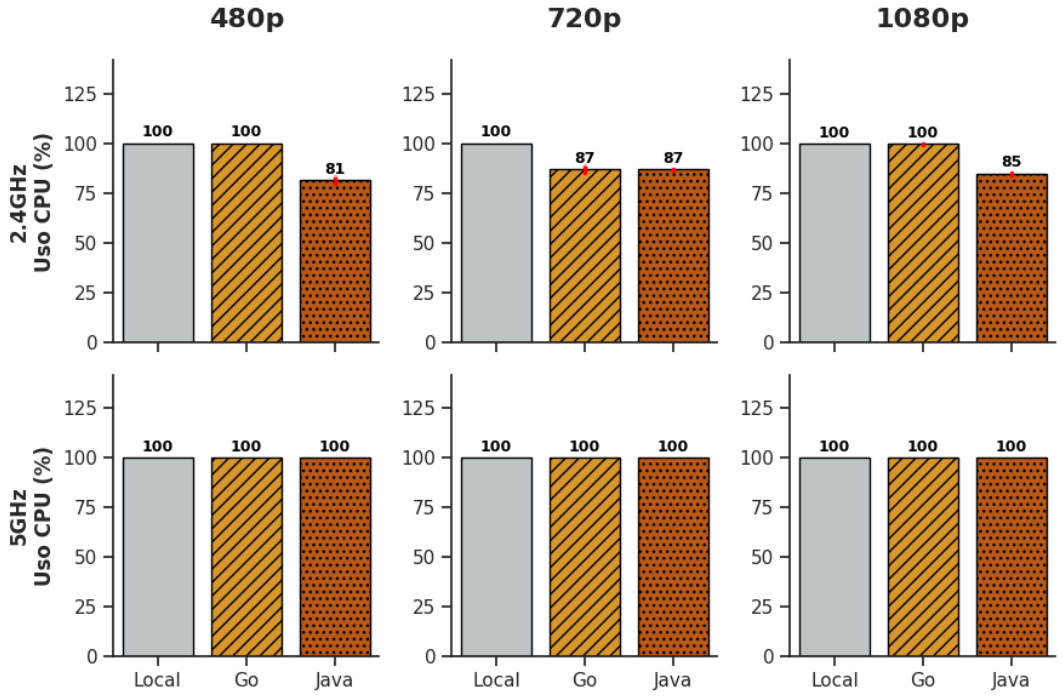
A comparação entre as linguagens também revela diferenças relevantes. O servidor implementado em *Go* apresentou consumo médio de bateria inferior ao servidor em *Java*, indicando maior eficiência no processamento remoto. Entretanto, como evidenciado pelos valores da Figura 12, essa vantagem não foi suficiente para compensar o aumento do tempo total de execução imposto pelo *offloading*.

6.1.3 CPU

No cenário de execução local, observa-se que o uso de CPU manteve-se consistentemente em 100% para todas as resoluções avaliadas. Esse comportamento era esperado, uma vez que todo o processamento do filtro é realizado diretamente no dispositivo móvel. A saturação do

processador indica que a aplicação explora plenamente os recursos disponíveis para concluir a tarefa, onde se notou um moderado aquecimento do dispositivo ao longo dos testes.

Figura 13 – Uso médio de CPU do filtro *Grayscale* nos cenários local e remoto, considerando diferentes condições de rede e linguagens de servidor.



Fonte: Elaborado pelo autor.

Nos cenários de *offloading* utilizando rede Wi-Fi de 2,4 GHz, observa-se uma redução no uso médio de CPU em alguns casos, especialmente no servidor escrito em *Java*. Esse comportamento sugere que, em função da menor largura de banda da rede Wi-Fi de 2,4 GHz, o dispositivo móvel permanece parte do tempo em estado de espera, aguardando a transmissão dos dados ou respostas do servidor, o que reduz a média de utilização do processador, pois o mesmo fica ocioso enquanto aguarda o processamento do *frame* no servidor.

No cenário de execução do filtro *Grayscale* utilizando a rede Wi-Fi de 5 GHz, observa-se que o uso de CPU permaneceu em 100% em todos os casos analisados. Esse comportamento está associado, principalmente, à maior largura de banda disponibilizada pela rede de 5 GHz, que permite o tráfego mais rápido e contínuo dos dados entre cliente e servidor. Como a operação de conversão para escala de cinza apresenta baixa complexidade computacional, o dispositivo móvel permanece processando quadros de forma praticamente ininterrupta, sem períodos significativos de ociosidade. Nesse contexto, o processador permanece ocupado com a extração e envio de *frames*, não havendo períodos relevantes de inatividade capazes de reduzir

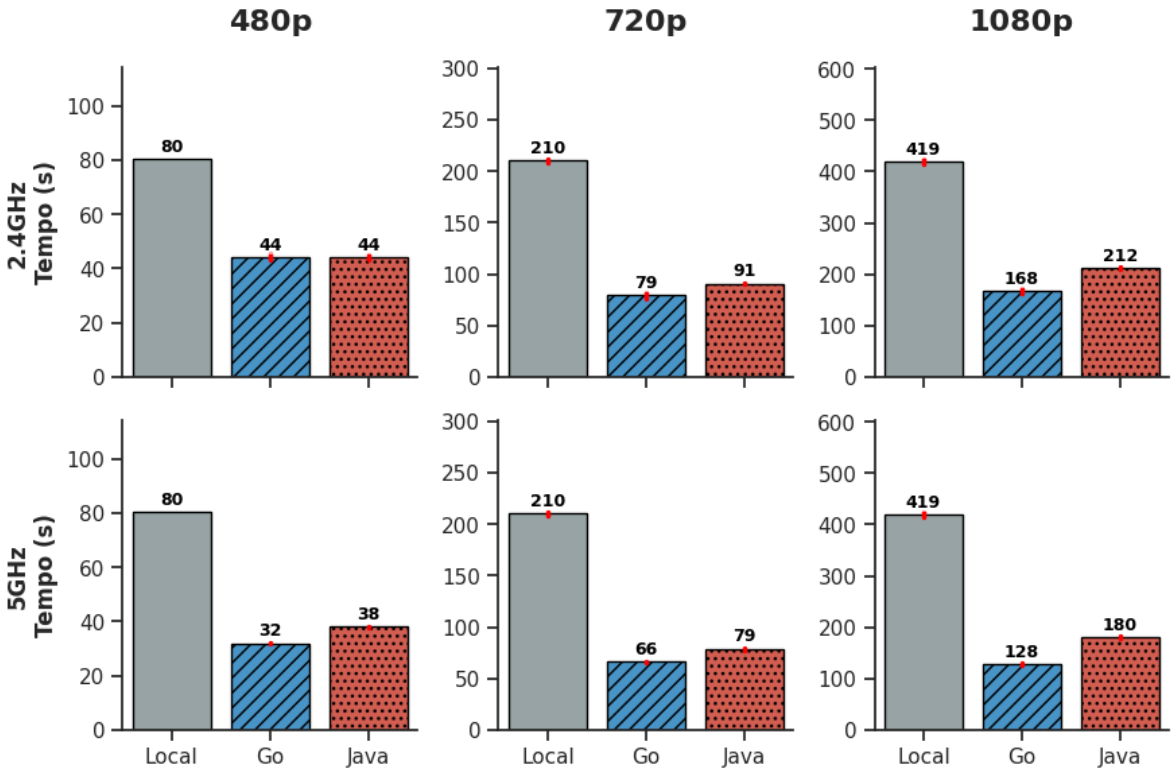
a média de uso de CPU. Assim, o *offloading* em 5 GHz não diminui a carga computacional imediata do dispositivo, mas altera a natureza do trabalho executado, deslocando o esforço de processamento de imagem para operações de extração e transmissão de *frames*.

6.2 Filtro *Pencil*

6.2.1 Tempo médio por iteração

A análise do tempo médio de processamento para o filtro *Pencil* evidencia um comportamento substancialmente distinto daquele observado para o filtro *Grayscale*, refletindo a elevada complexidade computacional desse algoritmo. A Figura 14 apresenta os tempos médios por iteração obtidos nos diferentes cenários avaliados.

Figura 14 – Tempo médio por iteração para o filtro *Pencil* nos cenários local e de *offloading*.



Fonte: Elaborado pelo autor.

No cenário de execução local, os tempos médios por iteração mostraram-se excessivamente elevados, aumentando de forma acentuada com a resolução do vídeo. Em 480p, o tempo médio foi de aproximadamente 80 s, elevando-se para cerca de 210 s em 720p e atingindo

aproximadamente 419 s em 1080p. Esse crescimento indica que o custo computacional do filtro *Pencil* escala de forma acentuada conforme aumenta a resolução do vídeo, tornando a execução local pouco viável sob a ótica temporal.

Nos cenários com *offloading*, observa-se uma redução expressiva do tempo médio por iteração em todas as resoluções analisadas. Em 480p, o tempo médio reduziu significativamente, com reduções superiores a 45% em relação ao cenário local. Em 720p, o tempo médio local foi reduzido em aproximadamente 62% com *Go* na rede Wi-Fi de 2,4 GHz e em 69% na rede Wi-Fi de 5 GHz, enquanto o servidor em *Java* apresentou reduções de cerca de 57% na rede Wi-Fi de 2,4 GHz e 62% na rede Wi-Fi de 5 GHz. Os ganhos tornam-se mais expressivos nessa resolução, com destaque para a redução observada no melhor cenário remoto, em que a utilização da rede Wi-Fi de 5 GHz proporcionou uma diminuição de aproximadamente 48% no tempo médio em relação à rede Wi-Fi de 2,4 GHz para *Go*.

Para 1080p, o cenário mais crítico, o tempo médio local de 419 s foi reduzido em aproximadamente 60% com *Go* na rede Wi-Fi de 2,4 GHz e em 69% na rede Wi-Fi de 5 GHz, enquanto o servidor em *Java* apresentou reduções de cerca de 49% na rede Wi-Fi de 2,4 GHz e 57% na rede Wi-Fi de 5 GHz. Nessa resolução, a vantagem do servidor em *Go* torna-se mais evidente, indicando maior eficiência na execução remota de tarefas computacionalmente intensivas. Além disso, a influência da rede Wi-Fi de 5 GHz é notável, reduzindo o tempo médio em cerca de 23% para *Go* e 15% para *Java* em comparação com a rede de 2,4 GHz.

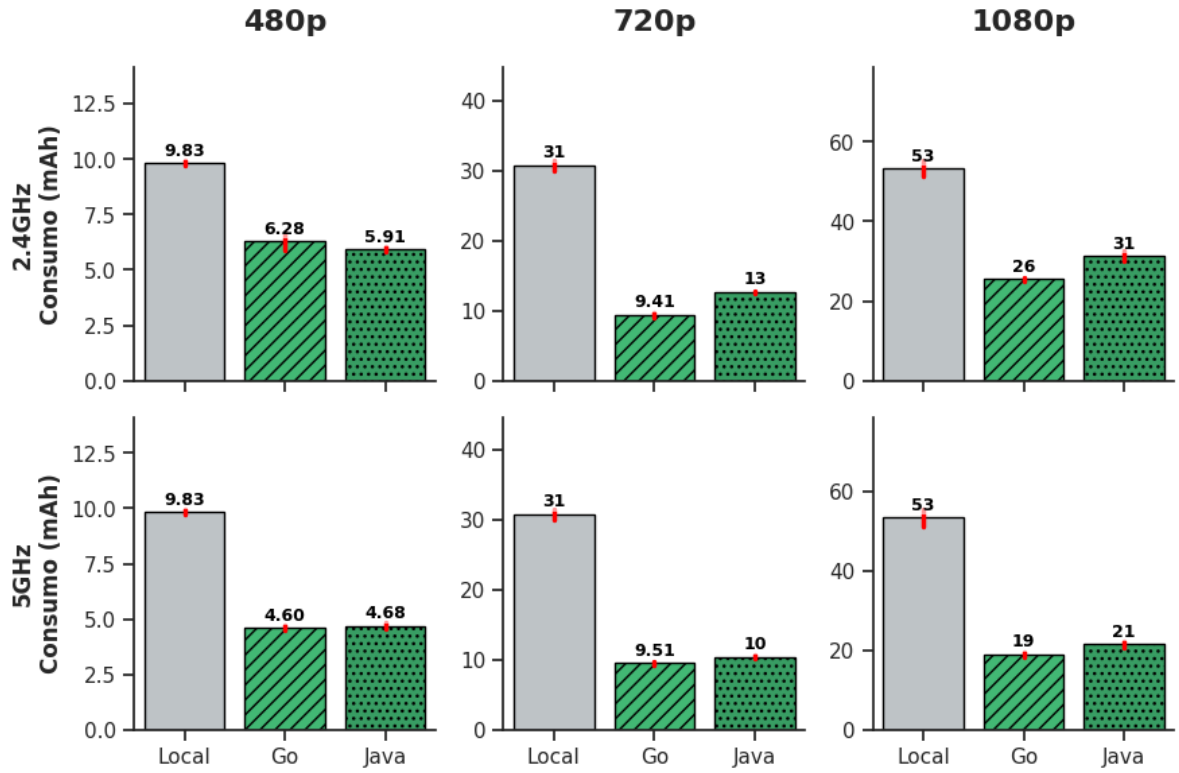
6.2.2 Bateria

A análise do consumo de bateria para o filtro *Pencil* evidencia diferenças expressivas entre a execução local e os cenários com *offloading*, conforme apresentado na Figura 15. Diferentemente do observado para a tarefa de baixa complexidade, o elevado custo computacional do filtro *Pencil* faz com que o tempo prolongado de execução local resulte em maior consumo energético por iteração.

No cenário de execução local, o consumo médio de bateria cresce de forma acentuada com o aumento da resolução. Em 480p, o consumo foi de aproximadamente 9,83 mAh, aumentando para cerca de 31 mAh em 720p e atingindo aproximadamente 53 mAh em 1080p. Esse consumo está diretamente associado ao elevado tempo médio de processamento local, que mantém a CPU do dispositivo ativo por longos períodos.

Nos cenários com *offloading*, observa-se uma redução significativa no consumo

Figura 15 – Consumo médio de bateria para o filtro *Pencil* nos cenários local e de *offloading*.



Fonte: Elaborado pelo autor.

de bateria em todas as resoluções analisadas. Em 480p, considerando o cenário local como referência, observa-se uma redução aproximada de 36% com o servidor em *Go* e de 40% com *Java* na rede Wi-Fi de 2,4 GHz. Na rede Wi-Fi de 5 GHz, as reduções foram ainda mais significativas, atingindo cerca de 53% com *Go* e 52% com *Java*, evidenciando o impacto positivo da maior capacidade da rede na diminuição do consumo energético do dispositivo móvel.

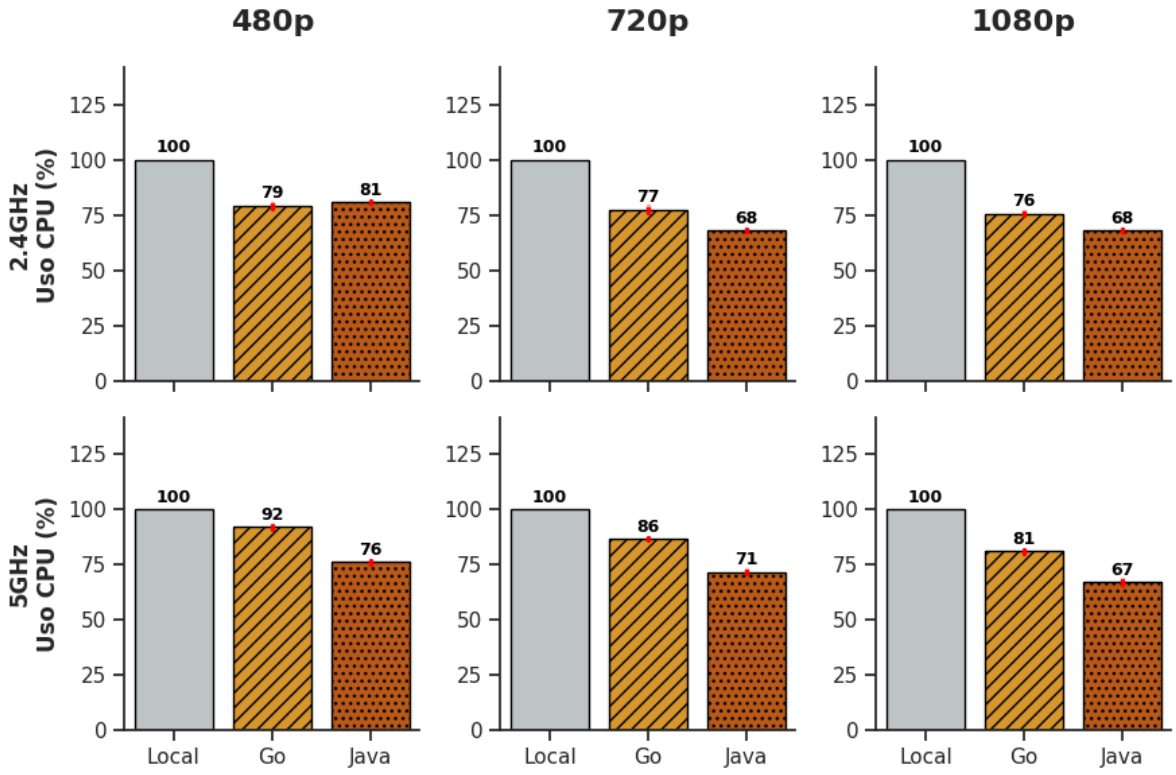
Considerando o cenário local como referência, em 720p, o consumo médio local foi reduzido em aproximadamente 69% com o servidor em *Go* na rede Wi-Fi de 2,4 GHz e em 58% com o servidor em *Java*. Na rede Wi-Fi de 5 GHz, as reduções foram de cerca de 69% para *Go* e 67% para *Java*.

Em 1080p, tomando 53 mAh como referência local, observou-se redução de aproximadamente 51% com *Go* na rede Wi-Fi de 2,4 GHz e 41% com *Java* na rede Wi-Fi de 2,4 GHz. Na rede Wi-Fi de 5 GHz, as reduções foram de cerca de 64% para *Go* e 60% para *Java*.

6.2.3 CPU

A análise do uso da CPU durante a execução do filtro *Pencil* revela um comportamento distinto em relação ao observado no filtro *Grayscale*. Nos cenários de execução local, a CPU operou consistentemente em 100% de utilização em todas as resoluções testadas, condição que, aliada aos longos tempos de processamento, resultou em sobreaquecimento perceptível do dispositivo. Os dados podem ser visualizados na Figura 16

Figura 16 – Uso médio da CPU para o filtro *Pencil* em diferentes resoluções e configurações.



Fonte: Elaborado pelo autor.

Nos cenários de *offloading*, observa-se uma redução consistente na utilização da CPU em relação à execução local. Essa redução torna-se mais evidente à medida que a resolução do vídeo aumenta e está diretamente associada à delegação do processamento principal ao servidor remoto.

Para a resolução de 480p, tomando o cenário local como referência, o *offloading* resultou em reduções aproximadas de 21% com o servidor em *Go* e 19% com *Java* na rede Wi-Fi de 2,4 GHz. Na rede Wi-Fi de 5 GHz, as reduções observadas foram mais limitadas, cerca de 8% com *Go* e 24% com *Java*.

Em 720p, na rede Wi-Fi de 2,4 GHz, as reduções no uso médio de CPU foram de aproximadamente 23% com *Go* e 32% com *Java*. Já na rede Wi-Fi de 5 GHz, observaram-se reduções em torno de 14% para *Go* e 29% para *Java*, evidenciando que a diminuição do uso do processador depende diretamente da capacidade da rede e do ambiente de execução do servidor. Nota-se que a rede Wi-Fi de 2,4 GHz apresentou maior redução, resultado associado a períodos mais longos de ociosidade devido à menor largura de banda, enquanto na rede Wi-Fi de 5 GHz a maior capacidade de transmissão manteve o processador menos ocioso.

Para a resolução de 1080p, observa-se o menor uso médio de CPU entre todos os cenários de *offloading*. Na rede Wi-Fi de 2,4 GHz, os valores foram de aproximadamente 76% com *Go* e 68% com *Java*, enquanto na rede Wi-Fi de 5 GHz reduziram-se para cerca de 81% com *Go* e 67% com *Java*. Esse comportamento reforça que, em tarefas mais pesadas, a CPU do dispositivo móvel permanece por períodos mais longos em estado de espera, aguardando o retorno dos resultados do servidor. Na rede Wi-Fi de 2,4 GHz, os intervalos de ociosidade são prolongados pela menor largura de banda, ao passo que na rede Wi-Fi de 5 GHz a maior capacidade de transmissão reduz esses períodos, resultando em valores ligeiramente superiores de utilização em alguns casos, especialmente com o servidor em *Go*.

De forma consolidada, os resultados indicam que o filtro *Pencil* exige o máximo do poder computacional do *smartphone*, sendo a execução local o cenário mais crítico em termos de desempenho e consumo energético. O *offloading* mostrou-se eficaz para mitigar esse impacto, sobretudo ao reduzir o percentual de uso da CPU e, consequentemente, o consumo de energia da bateria. Esses resultados evidenciam o potencial do *offloading* como alternativa viável para tarefas de alta complexidade, contribuindo para acelerar a execução e ampliar a autonomia energética dos dispositivos móveis.

6.3 Análise consolidada dos resultados

Os resultados experimentais apresentados ao longo deste capítulo demonstram que a decisão entre execução local e *offloading* não é trivial, devendo ser tomada com base nas características da tarefa, na resolução do vídeo e nas condições de conectividade disponíveis. Para o filtro *Grayscale*, a execução local mostrou-se mais eficiente do ponto de vista temporal e energético. O *offloading*, nesse caso, introduziu sobrecarga adicional que não foi compensada pelos ganhos no processamento remoto, resultando em maiores tempos médios e maior consumo de bateria. Embora a utilização de redes *Wi-Fi* de 5 GHz e a adoção do servidor escrito em *Go*

tenham reduzido parcialmente esses custos, tais melhorias não foram suficientes para tornar o *offloading* competitivo em relação à execução local para esse tipo de tarefa.

Em contrapartida, para o filtro *Pencil*, a execução local mostrou-se extremamente custosa, especialmente em resoluções mais altas. Os tempos médios excessivos, o uso contínuo de *CPU* em níveis elevados e o consumo significativo de bateria evidenciam que esse cenário impõe forte limitação ao uso do filtro em dispositivos móveis. A análise das métricas indica que o *offloading* foi capaz de reduzir significativamente o tempo médio por iteração e o consumo de bateria, ao custo de uma menor utilização da *CPU* local, que passou a permanecer parte do tempo em estado de espera. Esse comportamento contribui para mitigar o sobreaquecimento do dispositivo e assim reduzir também o consumo de energia. Entre as soluções avaliadas, o servidor implementado em *Go* apresentou o melhor desempenho temporal e menor consumo de energia, sobretudo quando combinado com a rede Wi-Fi de 5 GHz.

De forma geral, os resultados reforçam que o *offloading* é mais indicado para tarefas computacionalmente intensivas e em cenários com boa conectividade, enquanto a execução local permanece mais adequada para tarefas simples e de curta duração. Assim, este estudo evidencia a importância de estratégias adaptativas de decisão, capazes de considerar dinamicamente o custo computacional da tarefa, a resolução do conteúdo e as condições de rede, de modo a maximizar o desempenho e a eficiência energética em aplicações de processamento de vídeo em dispositivos móveis.

6.4 Ameaças à validade

Esta seção discute os fatores que podem limitar a validade dos resultados obtidos e a generalização das conclusões deste estudo, combinando considerações relativas à validade interna, externa, construção e de conclusão.

6.4.1 Validade interna

Em relação à validade interna, as medições de consumo de bateria, embora realizadas com aplicativos de terceiros encerrados, podem ter sido influenciadas por processos em segundo plano do sistema operacional e por variáveis externas, como o nível de brilho da tela no momento da execução. Soma-se a isso o fato de que o *smartphone* utilizado nos experimentos possui aproximadamente dois anos de uso contínuo, o que implica desgaste natural da bateria ao longo

do tempo. Dessa forma, a capacidade real de armazenamento energético pode estar reduzida em relação ao valor nominal originalmente especificado pelo fabricante. Além disso, as condições da rede Wi-Fi, mesmo em ambiente controlado e dedicado, estão sujeitas a flutuações de sinal e interferências, podendo afetar os tempos de transmissão e, consequentemente, os resultados de desempenho e consumo nos cenários de *offloading*.

6.4.2 Validade externa

No que se refere à validade externa, os experimentos foram conduzidos utilizando um único modelo de *smartphone* e um servidor com configuração específica, o que pode limitar a generalização dos resultados para outros dispositivos móveis, diferentes versões de sistemas operacionais e infraestruturas de rede distintas. Embora o conjunto de vídeos empregado contemple diferentes resoluções, ele pode não abranger todas as características de conteúdo que influenciam o desempenho do processamento. Ademais, não foi possível realizar experimentos com resoluções superiores a 1080p em decorrência de limitações técnicas relacionadas ao *hardware* e ao suporte a *codec* do dispositivo móvel utilizado, uma vez que tentativas em qualidades mais elevadas resultaram em falhas ou instabilidade da aplicação.

Adicionalmente, o ambiente de rede controlado, sem concorrência de tráfego de outros dispositivos, não representa necessariamente condições reais de uma rede doméstica. Dessa forma, o cenário experimental, conduzido sob configurações específicas de *hardware*, *software* e rede, restringe a extrapolação direta dos resultados para contextos mais amplos. Assim, as conclusões deste estudo devem ser interpretadas considerando essas limitações inerentes ao ambiente experimental adotado.

6.4.3 Validade de construção

Quanto à validade de construção, as métricas de tempo total, corrente média e uso de CPU foram coletadas via *APIs* do sistema, o que as torna robustas. A variável de corrente média pode introduzir aproximações em relação ao valor real, embora a medição do consumo de bateria seja fundamentada em conceitos físicos bem estabelecidos. Apesar de ter sido utilizado o mesmo dispositivo e metodologia de medição em todos os cenários, permitindo comparações relativas, os valores absolutos devem ser interpretados com cautela, visto que cada iteração pode apresentar certa discrepância entre valores obtidos anteriormente. Adicionalmente, a arquitetura adotada neste estudo não considerou um modelo de *streaming* tradicional para o envio contínuo

dos dados de vídeo. Essa decisão de projeto pode influenciar as métricas medidas, uma vez que o padrão de comunicação impacta o comportamento do *offloading*.

6.4.4 Validade de conclusão

As conclusões apresentadas neste estudo estão sujeitas a limitações relacionadas ao processo de inferência estatística e à forma como determinadas métricas foram quantificadas. Embora cada cenário experimental tenha sido executado 50 vezes, as análises apresentadas basearam-se em uma amostra aleatória de 35 iterações após a remoção de valores discrepantes por meio do método do Intervalo Interquartil.

Para as métricas de tempo médio por iteração e utilização de CPU, o número de repetições adotado, aliado ao uso de intervalos de confiança, fornece suporte para as comparações realizadas. Ainda que a metodologia tenha sido aplicada de forma consistente em todos os experimentos, diferenças pontuais entre cenários próximos podem não ser estatisticamente significativas.

Dessa forma, as conclusões apresentadas devem ser entendidas como evidências iniciais dentro do escopo experimental definido, reforçando a necessidade de estudos futuros complementares para validação e aprofundamento dos achados.

7 CONCLUSÕES

Os resultados obtidos ao longo desta pesquisa permitiram avaliar, de forma experimental, o impacto do *offloading* computacional no desempenho e no consumo de energia de aplicações móveis. A análise demonstrou que a arquitetura proposta é tecnicamente viável e eficaz na redução da carga computacional do processamento local e da redução do consumo de energia, configurando-se como uma solução robusta para cenários de Computação em Nuvem Móvel.

Observou-se uma correlação direta entre a complexidade das tarefas avaliadas e os benefícios proporcionados pelo *offloading*. À medida que a demanda computacional aumenta, a migração do processamento para servidores remotos torna-se progressivamente mais vantajosa, resultando em uma redução expressiva do uso da CPU no dispositivo móvel e em ganhos significativos de eficiência energética, desde que haja condições de rede favoráveis. Esse comportamento está alinhado com os achados reportados por Carvalho *et al.* (2023), que indicam que o *offloading* computacional tende a ser efetivamente vantajoso apenas em cenários caracterizados por cargas de processamento mais custosas. Tal constatação é particularmente relevante para desenvolvedores de aplicações móveis complexas, uma vez que evidencia a necessidade de decidir entre execução local ou remota. Nesse sentido, torna-se essencial avaliar o uso do *offloading* computacional em determinados cenários, pois essa estratégia pode representar ganhos significativos de desempenho e eficiência energética, especialmente quando as cargas de processamento são elevadas.

Adicionalmente, os resultados evidenciam que a escolha da linguagem de implementação do servidor e da infraestrutura de rede exerce grande influência no desempenho do sistema. No contexto deste estudo, a combinação de um servidor implementado na linguagem *Go* com comunicação via rede Wi-Fi de 5 GHz apresentou os melhores resultados, especialmente no cenário de maior custo computacional, representado pelo filtro *Pencil*. Essa constatação reforça a importância de alinhar linguagens eficientes no lado do servidor e infraestruturas de rede de alto desempenho para maximizar os benefícios do *offloading* computacional em ambientes móveis.

Entretanto, é importante destacar que os resultados devem ser interpretados levando em conta as limitações do cenário experimental adotado. Os experimentos foram conduzidos com um único modelo de *smartphone*, um servidor com configuração fixa e uma rede controlada, sem concorrência de tráfego, o que restringe a generalização direta dos achados para ambientes reais. Além disso, limitações técnicas de *hardware* e suporte a *codec* impediram a avaliação

de resoluções superiores a 1080p, não sendo possível estender os resultados para conteúdos de outras resoluções. Soma-se a isso o fato de o estudo ter se concentrado em um conjunto restrito de filtros de processamento de imagens, representativos, o que delimita o escopo das conclusões às cargas computacionais analisadas.

7.1 Trabalhos Futuros

Como continuidade deste trabalho, propõe-se a ampliação do estudo por meio da utilização de dispositivos móveis mais recentes, bem como de aparelhos com menor capacidade computacional, tanto para o cliente quanto para o servidor, permitindo avaliar o impacto da heterogeneidade do *hardware* sobre a viabilidade e a eficiência do *offloading* computacional. A inclusão de dispositivos com diferentes perfis de desempenho possibilita uma análise mais representativa de cenários reais de uso.

Outra direção relevante consiste na incorporação de algoritmos adicionais com distintos níveis de complexidade computacional, de modo a investigar o comportamento do *offloading* em tarefas com cargas diversas. Ademais, trabalhos futuros podem explorar o uso de técnicas de *multi-threading* tanto no cliente quanto no servidor, com o objetivo de avaliar os ganhos de paralelismo no processamento dos dados e seu impacto sobre o tempo total de execução e o consumo energético do dispositivo móvel.

Adicionalmente, a realização de experimentos em ambientes de rede não dedicadas, com concorrência de tráfego e variações de latência e largura de banda, pode fornecer uma visão mais fiel das condições de uso encontradas em redes domésticas. Também se propõe a variação do tipo e da localização dos servidores, considerando os cenários de *cloud* e *edge computing*, para analisar o impacto da proximidade computacional sobre o desempenho e a eficiência energética do *offloading*.

Por fim, uma linha promissora de investigação consiste no desenvolvimento de um modelo de *Machine Learning* para tomada de decisão adaptativa de *offloading*, capaz de selecionar dinamicamente a melhor estratégia de execução com base em características do dispositivo, da rede e da tarefa, contribuindo para uma abordagem mais inteligente e eficiente de *offloading* computacional em ambientes de Computação em Nuvem Móvel.

REFERÊNCIAS

AHUJA, S. P. Mobile cloud computing: Impacts on battery life, memory usage, and application performance. **Journal of Mobile Systems and Cloud Integration**, v. 7, n. 1, p. 15–30, 2025. Discusses how MCC decreases memory usage and improves battery life and performance of mobile applications. Disponível em: <https://www.sciencedirect.com/org/science/article/pii/S215618342500004X>.

AL-DULAIMY, A.; SHARMA, Y.; KHAN, M. G.; TAHERI, J. Introduction to edge computing. **Edge Computing: Models, Technologies and Applications**, p. 3–25, 2020.

Amazon Web Services. **O que é computação em nuvem móvel?** 2025. Acesso em: 18 jul. 2025. Disponível em: <https://aws.amazon.com/what-is/mobile-cloud-computing/>.

Amazon Web Services. **What is Edge Computing?** 2025. Disponível em: <https://aws.amazon.com/what-is/edge-computing/>.

ANDERSSON, T.; BRENDEN, C. **Parallelism in Go and Java: Performance analysis**. Dissertação (Mestrado) — Blekinge Institute of Technology, 2018.

ARAUJO, M. S.; MAIA, M. E. F.; REGO, P. A. L.; SOUZA, J. N. Performance analysis of computational offloading on embedded platforms using the *gRPC* framework. In: **8th International Workshop on ADVANCEs in ICT Infrastructures and Services**. Cancún, Mexico: [s.n.], 2020. p. 1–8.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. **Communications of the ACM**, v. 53, n. 4, p. 50–58, 2010.

BHARGAVA, B.; GUPTA, I. **Mobile Computing: Principles and Applications**. [S.l.]: Springer, 2008.

BUYYA, R.; VECCHIOLA, C.; SELVI, S. T. **Cloud Computing: A Hands-On Approach**. 2. ed. Boston: Morgan Kaufmann, 2019. ISBN 9780128053308.

CAI, W. *et al.* A survey on cloud gaming: Future of computer games. **IEEE Access**, v. 4, p. 7605–7620, 2016.

CARVALHO, G.; VELASQUEZ, K.; FERNANDES, J. P.; CABRAL, B. On computation offloading and energy efficiency on android devices. In: **IEEE International Conference on Communications Workshops (ICC Workshops)**. [S.l.: s.n.], 2023. p. 1836–1841.

CAVALCANTE, L. d. M. **Comparativo entre Golang e Java, enfatizando a performance de Go**. [S.l.], 2025. Trabalho comparativo sobre desempenho e características entre Go e Java. Disponível em: <http://educapes.capes.gov.br/handle/capes/1057581>.

CHAMAS, C. L.; CORDEIRO, D.; ELER, M. M. Comparing rest, soap, socket and grpc in computation offloading of mobile applications: An energy cost analysis. In: IEEE. **2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)**. [S.l.], 2017. p. 219–224.

CHEN, X.; LI, W.; LU, S.; ZHOU, Z.; FU, X. Efficient resource allocation for on-demand mobile-edge cloud computing. **IEEE Transactions on Vehicular Technology**, v. 67, n. 9, p. 8769–8780, 2018.

CONSULT, M. **Preferências de usuários de smartphones**. 2018. Disponível em: <https://pro.morningconsult.com/instant-intel/smartphone-owners-prefer-simple-features-like-battery-life-durability-camera-quality>.

CÓRDOVA-CÁRDENAS, R.; AMOR, D.; GUTIÉRREZ, Á. Edge ai in practice: A survey and deployment framework for neural networks on embedded systems. **Electronics**, v. 14, n. 24, p. 4877, 2025. Systematic review of constraints and deployment challenges for AI and deep learning on embedded hardware. Disponível em: <https://www.mdpi.com/2079-9292/14/24/4877>.

DE, D. **Mobile Cloud Computing: Architectures, Algorithms and Applications**. [S.l.]: CRC Press, 2016.

DINH, H. T.; LEE, C.; NIYATO, D.; WANG, P. A survey of mobile cloud computing: Architecture, applications, and approaches. **Wireless Communications and Mobile Computing**, v. 13, n. 18, p. 1587–1611, 2013.

FORMAN, G. H.; ZAHORJAN, J. The challenges of mobile computing. **IEEE Computer**, IEEE, v. 27, n. 6, p. 38–47, 1994.

GOGGIN, G. Mobile ai: Communication and mobility after the smartphone. **Communication and Change**, v. 1, n. 1, p. 5, 2025. Discusses the integration of AI into mobile technologies and emergent challenges within mobile AI paradigms. Disponível em: <https://link.springer.com/article/10.1007/s44382-025-00002-3>.

Google. **gRPC: A High Performance, Open Source Universal RPC Framework**. 2025. <https://grpc.io/docs/what-is-grpc/introduction/>. Acesso em: 20 jul. 2025.

GU, X.; JIN, L.; ZHAO, N.; ZHANG, G. Energy-efficient computation offloading and transmit power allocation scheme for mobile edge computing. **Mobile Information Systems**, p. Article ID 3613250, 2019.

HALLIDAY, D.; RESNICK, R.; WALKER, J. **Fundamentos de Física – Volume 3: Eletromagnetismo**. 10. ed. Rio de Janeiro: LTC, 2016.

INDRASIRI, K.; KURUPPU, D. **gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes**. Sebastopol: O'Reilly Media, 2020. ISBN 9781492058335.

JAIN, R. **The Art of Computer Systems Performance Analysis**. [S.l.]: Wiley, 1991.

JIANG, M. H.; VISSER, O. W.; PRASETYA, I. S. W. B.; IOSUP, A. A mirroring architecture for sophisticated mobile games using computation-offloading. **Concurrency and Computation: Practice and Experience**, v. 30, n. 20, p. e4494, 2018.

KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. **Mobile Networks and Applications**, v. 18, n. 1, p. 129–140, 2013.

KUMAR, S.; KHANNA, A.; TYAGI, M.; FORE, V. A survey of mobile computation offloading: Applications, approaches and challenges. In: IEEE. **International Conference on Advanced Computing and Communication Systems**. [S.l.], 2018. p. 1–6.

KUSHWAHA, S.; RAI, A. Mobile cloud computing: The future of cloud. In: IEEE. **IEEE OTCON**. [S.l.], 2024. p. 1–6.

LIU, L.; LI, H.; GRUTESER, M. Edge assisted real-time object detection for mobile augmented reality. In: **Proceedings of the 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19)**. ACM, 2019. Disponível em: <https://dl.acm.org/doi/10.1145/3300061.3300116>.

LIU, W.; LI, D.; LI, X. Data offloading and sharing for latency minimization in augmented reality based on mobile-edge computing. In: IEEE. **2018 IEEE Globecom Workshops (GC Wkshps)**. [S.l.], 2018. p. 1–6.

MAO, Y. *et al.* A survey on mobile edge computing: The communication perspective. **IEEE Communications Surveys & Tutorials**, v. 19, n. 4, p. 2322–2358, 2017.

MATOS, F. d.; REGO, P. A. L.; TRINTA, F.; CASTOR, F. A comparative study about the performance of multi-language tools in computation offloading scenarios. **The Journal of Supercomputing**, Springer, v. 81, p. 1441–1475, 2025. ISSN 0920-8542.

MATOS, F. F. S. B. d.; REGO, P. A. L.; TRINTA, F. A. M. An empirical study about the adoption of multi-language technique in computation offloading in a mobile cloud computing scenario. **11th International Conference on Cloud Computing and Services Science (CLOSER)**, p. 207–214, 2021.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. Gaithersburg, MD, USA, 2011.

Microsoft Azure. **O que é computação em nuvem?** 2025. Acesso em: 16 dez. 2025. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-cloud-computing/>.

MORA, H. Mobile cloud computing paradigm: A survey of operational concerns, challenges and open issues. **Emerging Telecommunication Technologies**, 2024. Critical analysis of operational challenges including energy, connectivity and resource limitations.

NISWAR, M.; SAFRUDDIN, R. A.; BUSTAMIN, A.; ASWAD, I. Performance evaluation of microservices communication with rest, graphql, and grpc. **International Journal of Electronics and Telecommunications**, v. 70, n. 2, p. 429–436, 2024.

OpenFog Consortium Architecture Working Group. **OpenFog Reference Architecture for Fog Computing**. [S.l.], 2017. White paper; acesso em: 21 jul. 2025. Disponível em: <https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf>.

QIN, Y.; CHEN, J.; JIN, L.; YAO, R.; GONG, Z. Task offloading optimization in mobile edge computing based on a deep reinforcement learning algorithm using density clustering and ensemble learning. **Scientific Reports**, 2025.

RAHIMI, M. R.; REN, J.; LIU, C. H.; VASILAKOS, A. V.; VENKATASUBRAMANIAN, N. Mobile cloud computing: A survey, state of the art and future directions. **Mobile Networks and Applications**, v. 19, n. 2, p. 133–143, 2014.

SATYANARAYANAN, M. Pervasive computing: Vision and challenges. **IEEE Personal Communications**, v. 8, n. 4, p. 10–17, 2001.

SATYANARAYANAN, M.; BAHL, P.; CáCERES, R.; DAVIES, N. The case for vm-based cloudlets in mobile computing. **IEEE Pervasive Computing**, v. 8, n. 4, p. 14–23, 2009.

SHI, W. *et al.* Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, v. 3, n. 5, p. 637–646, 2016.

SIRIWARDHANA, Y.; PORAMBAGE, P.; LIYANAGE, M.; YLIANTTILA, M. 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. 1364–1393, 2021.

SOARES, L. O. Go x java e grpc x rest: Um estudo empírico. In: **SBC Latin American Computing Conference**. [s.n.], 2024. Análise comparativa de desempenho entre Java e Go com arquiteturas de comunicação REST e gRPC. Disponível em: <https://sol.sbc.org.br/index.php/sblp/article/view/30258>.

STATISTA. **Fatores mais importantes na compra de smartphones**. 2023. Disponível em: <https://www.statista.com/statistics/1230056/most-important-phone-features-for-new-phone-purchase/>.

STATISTA. **Número de usuários de smartphones mundial**. 2023. Disponível em: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>.

STATISTA. **Participação do tráfego móvel global**. 2024. Disponível em: <https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/>.

STORE, A. A. **Top apps de entretenimento**. 2024. Disponível em: <https://apps.apple.com/us/charts/iphone/entertainment-apps/6016>.

TANENBAUM, A. S.; STEEN, M. van. **Distributed Systems**. 4. ed. [S.l.]: distributed-systems.net, 2023. Versão 4.03 (janeiro de 2025). Disponível em: <https://www.distributed-systems.net/index.php/books/ds4/>.

YOUSEFPOUR, A.; ISHIGAKI, G.; FLAUZAC, R.; CONTI, M.; SADASIVAM, K.; JUE, J. P.; CHEN, X. All one needs to know about fog computing and related edge computing paradigms: A complete survey. **Journal of Systems Architecture**, v. 98, p. 289–330, 2019.

ZAKIZADEH, M. Mobile cloud computing: A complete survey. **Research Contribution — Mobile Cloud Computing**, 2024. Survey highlighting mitigation of energy consumption and overhead by MCC. Disponível em: <https://www.researchgate.net/publication/385732037_Mobile_Cloud_Computing_A_Complete_Survey>.

ZHANG, M. Development of analytical offloading for innovative internet of vehicles based on mobile edge computing. **Journal of Grid Computing**, v. 22, n. 4, 2023.