Avaliação de OCR Embarcado para Dispositivos Móveis: Desempenho, Privacidade e Usabilidade

João Carlos Nepomuceno Fernandes*, Francisco Erick Souza Gomes*, Douglas de Araújo Rodrigues†,
Vinícius Lagrota Rodrigues da Costa‡, Paulo Antônio Leal Rego†,
José Antonio Fernandes de Macedo† and Pedro Pedrosa Rebouças Filho*
*Instituto Federal de Educação, Ciência e Tecnologia do Ceará
†Universidade Federal do Ceará
‡Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (CEPESC)
Email: {joaocarlos, ericksouza, douglas, jose.macedo}@insightlab.ufc.br,
vinicius.1232@dte.gov.br, pedrosarf@ifce.edu.br

***Abstract*—This study presents a comparative analysis between three OCR technologies for mobile devices: Vision OCR, MLKit Mobile, and Tesseract Mobile. A total of 848 real examples of words extracted from urban environments were used, and the tools were evaluated with and without the application of preprocessing techniques. The metrics used enabled a complete evaluation of the performance of each solution. The results show that Vision OCR performs best, followed by MLKit and Tesseract with optimized preprocessing. Security aspects (processing on the device), usability (ease of operation, installation, and documentation, considering file size in embedded environments), and ease of integration were also analyzed. Native solutions offer superior performance with less need for adjustments, while Tesseract, despite requiring a more sophisticated pipeline, ensures maximum privacy because it is open source and controlled by the developer.**

***Resumo*—Este estudo apresenta uma análise comparativa entre três tecnologias de OCR para dispositivos móveis: *Vision OCR, MLKit Mobile e Tesseract Mobile*. Foram utilizados 848 exemplos reais de palavras extraídas de ambientes urbanos, e as ferramentas foram avaliadas com e sem a aplicação de técnicas de pré-processamento. As métricas utilizadas possibilitaram uma avaliação completa do desempenho de cada solução. Os resultados mostram que o *Vision OCR* apresenta o melhor desempenho, seguido pelo *MLKit* e pelo *Tesseract* com pré-processamento otimizado. Também foram analisados aspectos de segurança (processamento no dispositivo), usabilidade (facilidade de operação, instalação e documentação, considerando o tamanho dos arquivos em ambientes embarcados) e facilidade de integração. As soluções nativas oferecem desempenho superior com menor necessidade de ajustes, enquanto o *Tesseract*, apesar de exigir um pipeline mais sofisticado, garante privacidade máxima por ser de código aberto e controlado pelo desenvolvedor.**

I. Introdução

A popularização dos dispositivos móveis transformou a forma como interagimos com o mundo, conectando o ambiente físico ao digital. Nesse contexto, o Reconhecimento Óptico de Caracteres (*Optical Character Recognition*, OCR) consolidou-se como ferramenta essencial, aplicada desde a automação de dados até a tradução instantânea e o suporte a pessoas com deficiência visual [1]–[3].

Apesar de sua relevância, implementar OCR em dispositivos móveis apresenta desafios, sobretudo quando se busca execução totalmente *on-device*, evitando a exposição de dados a serviços externos. Soluções como o *Vision*, da Apple [4], e o *ML Kit*, do Google [5], integram modelos de *deep learning* para detecção e reconhecimento de texto. Já o *Tesseract Mobile* [6], [7], embora eficaz no reconhecimento, depende de pipelines adicionais para detecção, exigindo pré-processamento ou integração com outros modelos.

A literatura frequentemente compara motores de OCR em ambientes de desktop ou via APIs, mas poucos estudos exploram soluções móveis com processamento local [8], [9]. São ainda mais raras as avaliações comparativas sob condições uniformes em dispositivos reais. Trabalhos como o de Silva *et al.* [10] e Correa *et al.* [11] propõem estratégias complementares, mas não contemplam execução inteiramente embarcada.

As tecnologias analisadas neste estudo — *Vision*, *ML Kit* e *Tesseract* — foram escolhidas pela viabilidade em aplicações móveis: o primeiro otimizado para iOS, o segundo integrado ao Android e o terceiro, por ser de código aberto, permitindo maior controle e privacidade. Modelos recentes baseados em *PyTorch* ou *TensorFlow Lite* não foram considerados devido ao custo de deployment, maior complexidade de integração e aumento no tamanho do aplicativo, demandando otimizações adicionais [12].

Neste trabalho, conduzimos uma análise comparativa de três soluções de OCR *on-device*, avaliando sua precisão em um dataset de imagens reais com regiões de texto previamente delimitadas. Também investigamos o impacto de técnicas de pré-processamento local, com todas as etapas executadas no próprio dispositivo. Além dos resultados quantitativos, discutimos aspectos práticos de integração, ressaltando a preservação da privacidade como benefício da execução local. Esse aspecto é particularmente relevante em aplicações críticas, como o processamento de prontuários médicos, comprovantes bancários ou documentos oficiais, onde a confidencialidade dos dados deve ser preservada.

II. Trabalhos relacionados

A literatura apresenta diversas análises comparativas de tecnologias de OCR. Estudos como os de Malkadi *et al.* [13] e Tafti *et al.* [14] demonstram consistentemente que soluções comerciais e baseadas em nuvem, como as oferecidas pelo

Google, tendem a superar o *Tesseract* em acurácia para tarefas genéricas de extração de texto. Essa tendência é reforçada em estudos de caso específicos, como o de Thammarak *et al.* [15], que ao analisar certificados de veículos em tailandês, obteve 84.43% de acurácia com a *API Google Cloud Vision* contra 47.02% do *Tesseract*. Da mesma forma, Kaur e Sharma [16] observaram um desempenho superior da *API do Google* para o reconhecimento de texto em Punjabi.

Apesar do desempenho inferior em comparações diretas, a eficácia do *Tesseract* pode ser melhorada através da adição de um *pipeline* de pré-processamento nas imagens. O impacto dessa etapa é um tema central na pesquisa sobre OCR, com diversos estudos demonstrando que a aplicação dessas técnicas pode otimizar o reconhecimento em diferentes contextos, desde textos manuscritos até documentos em idiomas específicos [7], [17], [18].

Além disso, trabalhos como o de Sporici *et al.* [19] mostram que técnicas, como o uso de filtros convolucionais otimizados por aprendizagem por reforço, podem aumentar a acurácia do *Tesseract* 4.0 em mais de 300% em datasets desafiadores. Outros estudos, como o de Michalak e Okarma [20], exploram métodos específicos como a filtragem por entropia local para aprimorar a binarização de imagens com iluminação irregular, uma condição particularmente comum e problemática em capturas realizadas por dispositivos móveis.

No entanto, a literatura carece de estudos que comparem diretamente as soluções nativas dos sistemas operacionais móveis dominantes com o *Tesseract* em um cenário de uso realista e com processamento *on-device*. O *Vision*, *framework* oficial da Apple — isto é, um conjunto de ferramentas e bibliotecas de alto nível disponibilizado pelo sistema operacional para tarefas específicas de visão computacional — é amplamente utilizado por aplicativos iOS para OCR, realizando o processamento localmente no próprio dispositivo [4].

O *ML Kit*, disponibilizado pelo Google, é compatível tanto com Android quanto com iOS, mas apenas no Android oferece suporte a reconhecimento de texto *on-device*; no iOS, essa funcionalidade depende de processamento em nuvem [5]. Ambos os *frameworks* incluem, além do reconhecimento de caracteres, mecanismos de detecção automática de regiões de texto na imagem — uma etapa que o *Tesseract* não executa de forma autônoma, exigindo segmentação prévia ou o uso de pipelines externos.

Neste trabalho, no entanto, o foco está na etapa de classificação dos caracteres reconhecidos (*text recognition*), e não na detecção das regiões textuais, conforme refletido no dataset utilizado. Muitas comparações existentes se concentram em versões de desktop ou APIs comerciais em nuvem, negligenciando as particularidades do ambiente *mobile* e os desafios específicos de desenvolvimento.

Além disso, poucos trabalhos combinam a análise de acurácia com fatores práticos cruciais para desenvolvedores, como a facilidade de integração e as implicações de segurança e privacidade associadas ao processamento local. Este artigo visa preencher essa lacuna, oferecendo uma análise multifacetada que não apenas mede o desempenho bruto, mas também avalia a adequação de cada solução ao ecossistema de desenvolvimento móvel.

Recentemente, surgiram modelos leves de OCR baseados em *deep learning*, implementados em *frameworks* como *TensorFlow Lite* e em arquiteturas como *MobileNet*. Os trabalhos de Franco *et al.* [21] e Sari *et al.* [22] abordam a utilização desses modelos para o referente contexto. Apesar de demonstrarem bom desempenho em *benchmarks* acadêmicos, sua adoção em cenários de produção ainda é limitada por fatores práticos: falta de integração nativa em iOS e Android, necessidade de manutenção manual dos pesos de rede e impacto considerável no tamanho do aplicativo final. Diferentemente disso, *frameworks* como *Vision* e *ML Kit* oferecem APIs já integradas ao sistema operacional, simplificando o desenvolvimento e garantindo suporte contínuo.

III. Metodologia

Esta seção descreve detalhadamente o conjunto de dados utilizado, os métodos empregados para o reconhecimento de texto e as métricas adotadas para avaliação dos resultados. A Figura 1 ilustra o fluxo do processo experimental, incluindo as etapas de pré-processamento e avaliação dos modelos de OCR. A Figura 2, por sua vez, apresenta exemplos de amostras contidas no conjunto de dados utilizado.

A. Dataset

O *dataset* utilizado é uma versão adaptada do *ICDAR 2003 Robust Reading Dataset* [23], composta por 848 imagens de palavras isoladas com caracteres do alfabeto latino e algarismos decimais, coletadas em ambientes urbanos reais. Diferente do original, que contém cenas completas, esta versão foca em regiões de texto já segmentadas, simulando a saída de um detector de textos e permitindo avaliar especificamente o reconhecimento de caracteres.

Datasets mais recentes não foram utilizados por apresentarem textos em múltiplos idiomas, exigindo uma pré-seleção manual, e porque muitas imagens não estavam recortadas, o que demandaria ferramentas de *cropping* da ROI, tarefa fora do escopo deste estudo. O *ICDAR 2003*, embora antigo, já foi amplamente validado em trabalhos de OCR, mas não em cenários embarcados, justificando sua escolha para avaliação prática *on-device*. Esta escolha implica uma limitação conhecida, que será abordada em trabalhos futuros com a inclusão de datasets contemporâneos e mais complexos.

B. Metodologia de avaliação

Este estudo adotou uma abordagem sistemática para avaliar o desempenho das soluções utilizadas, dividida em duas fases:

1) Pré-processamento: Foram implementadas cinco técnicas de pré-processamento em imagens, cada uma com características específicas:

- Original: Imagem base sem modificações, servindo como referência para comparação dos demais métodos.
- Conversão para Escala de Cinza com equalização de histograma: Transformação não-linear do espaço de cores RGB para tons de cinza utilizando coeficientes

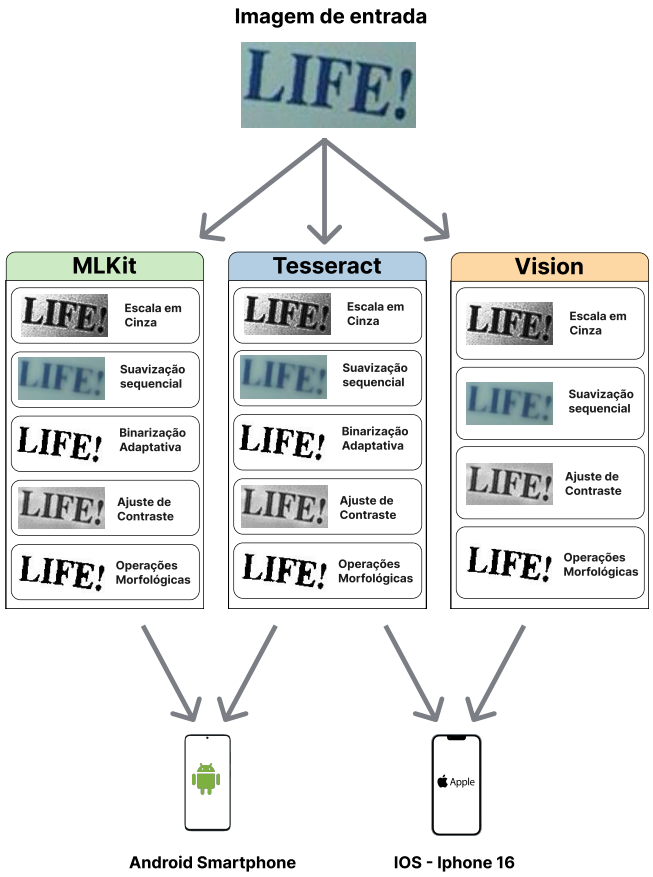

Figura 1. Fluxograma da metodologia adotada para avaliação dos motores de OCR (*MLKit, Tesseract e Vision*) em dispositivos Android e iOS. A imagem de entrada é submetida a diferentes técnicas de pré-processamento. Cada técnica é aplicada de forma independente antes da execução do OCR, permitindo verificar o impacto individual de cada etapa no desempenho de reconhecimento em cada motor e sistema operacional.

Figura 2. Exemplos de amostras contidas no dataset utilizado.

de luminância otimizados (Y = 0.299R + 0.587G + 0.114B), seguida de equalização de histograma para melhor distribuição tonal.

- Suavização sequencial: Aplicação sequencial de filtro Gaussiano (kernel 3×3, σ=0.5) para suavização de alta frequência, seguido por filtro bilateral (d=9, σ_{color}=75, σ_{space}=75) para preservação de bordas textuais.
- Ajuste de Contraste : Equalização adaptativa de histograma (*CLAHE*), com tamanho de *tile* de 8×8 pixels e limite de contraste 2.0, especialmente eficaz para imagens com iluminação irregular (Zimmerman *et al.* [24]).
- Binarização Adaptativa: *Thresholding* automático pelo método de Otsu, que determina iterativamente o limiar ótimo para separação *foreground/background* baseado na distribuição de intensidades dos pixels.
- Operações Morfológicas: Abertura morfológica (erosão seguida de dilatação) com elemento estruturante circular de raio 2 pixels, visando remover artefatos pequenos enquanto mantém a estrutura geral dos caracteres.

A implementação considerou as restrições de cada plataforma: no Android, utilizamos *OpenCV Mobile* combinado com *ML Kit* [5], permitindo a aplicação de todas as técnicas. No iOS, entretanto, o framework *Vision* [4] não oferece suporte nativo à binarização adaptativa, restringindo sua aplicação nesta plataforma e limitando os experimentos a outros métodos de pré-processamento disponíveis. Cada técnica foi aplicada isoladamente para permitir uma análise individual. É importante notar que o *Tesseract* [6] foi aplicado em ambas as plataformas.

2) Inferência dos Modelos OCR: Foram avaliadas três soluções distintas de OCR, com abordagens e arquiteturas variadas:

- *Vision OCR* [4]: API nativa da Apple com processamento local, utilizando o *framework Vision* para reconhecimento de texto em imagens. A inferência ocorre em tempo real, e os resultados são retornados como blocos, linhas e palavras, exigindo *parsing* para extração do texto contínuo.
- *MLKit Mobile* [5]: Biblioteca do Google para dispositivos Android, operando no modo *on-device*(executado localmente). O texto é retornado segmentado por blocos e linhas, de forma similar ao *Vision*, com suporte à múltiplos idiomas. Vale destacar que este *framework* também pode ser utilizado em plataformas iOS; entretanto, nessa configuração, o processamento ocorre via API (remota), e não localmente.
- *Tesseract* [6]: Modelo *open-source* integrado via *wrappers* personalizados para Android e iOS. Foram testadas as configurações de tratamento de uma única linha de texto e *layout* baseado em blocos, esta última foi selecionada por apresentar melhor resultado. A inferência foi realizada em lote, com o OCR aplicado sobre cada imagem pré-processada individualmente.

Todas as soluções foram integradas nativamente aos respectivos aplicativos móveis e configuradas para operar exclusivamente no modo *offline*. As imagens processadas por cada técnica de pré-processamento foram submetidas individualmente a cada modelo de OCR, garantindo que os resultados fossem comparáveis e que os efeitos de cada etapa pudessem ser isolados. Os textos retornados por cada inferência foram armazenados em arquivos estruturados para posterior comparação com os padrões de referência, permitindo o cálculo das métricas de desempenho.

C. Métricas de Avaliação

A avaliação do desempenho das soluções de OCR foi conduzida por meio de quatro métricas comumente utilizadas na literatura: Acurácia por Caractere (*Character Accuracy - CharAcc*), Acurácia por Palavra (*Word Accuracy - WordAcc*), Taxa de Erro por Caractere (*Character Error Rate - CER*) e Taxa de Erro por Palavra (*Word Error Rate - WER*). Apesar das abreviações serem comuns em trabalhos de reconhecimento óptico de caracteres, elas são aqui explicitadas para maior clareza.

A *CharAcc* mede a proporção de caracteres corretamente reconhecidos em relação ao total de caracteres do texto de referência, considerando correspondência exata com letras maiúsculas/minúsculas e símbolos especiais. Essa métrica é definida conforme a Equação 1:

$$CharAcc = \left(1 - \frac{\sum \text{Erros de Caractere}}{\sum \text{Total de Caracteres}}\right) \times 100\% \quad (1)$$

A *WordAcc* quantifica o percentual de palavras que foram reconhecidas com 100% de precisão, sendo uma palavra considerada incorreta mesmo com um único caractere errado. A fórmula é apresentada na Equação 2:

$$WordAcc = \frac{\text{Número de Palavras Perfeitas}}{\text{Total de Palavras}} \times 100\% \quad (2)$$

A *CER* representa a distância de edição entre o texto reconhecido e o texto de referência, somando substituições (S), inserções (I) e deleções (D) de caracteres em relação ao número total de caracteres (N), conforme a Equação 3:

$$CER = \frac{S + I + D}{N} \times 100\% \quad (3)$$

De forma análoga, a *WER* avalia as alterações no nível de palavras completas, considerando substituições (S_w), inserções (I_w) e deleções (D_w) de palavras em relação ao total de palavras (N_w), como mostrado na Equação 4:

$$WER = \frac{S_w + I_w + D_w}{N_w} \times 100\% \quad (4)$$

Essas métricas oferecem perspectivas complementares. Enquanto *CharAcc* e *WordAcc* destacam a taxa de acerto, *CER* e *WER* quantificam a magnitude dos erros. Em cenários práticos de OCR, a *CER* tende a ser mais informativa, pois considera todos os erros individualmente. Já a *WordAcc* é mais rigorosa ao penalizar palavras parcialmente incorretas da mesma forma que palavras completamente erradas.

D. Configuração dos Dispositivos

Os experimentos foram conduzidos em dispositivos móveis sob condições reais de aplicações embarcadas, com processamento integralmente *on-device*.

- Android: Dispositivo virtual *Pixel 7* no *Android Studio*, executando Android 16 (API 36). Configuração equivalente a hardware físico intermediário-avançado com chip *Google Tensor G2*, tela de 6.31"e resolução 1080×2400 px.
- **iOS**: Dispositivo físico *iPhone 16* com *iOS 18* e chip *Apple A18 Pro*, tela de 6.3"e resolução 1179×2556 px.

IV. Resultados e Discussões

Nesta seção, os resultados são apresentados e discutidos de forma integrada, considerando a performance dos modelos de OCR em dispositivos Android e iOS, com foco no impacto das técnicas de pré-processamento. A análise é baseada nas métricas *WordAcc*, *CharAcc*, *WER* e *CER*, previamente definidas na Seção de Metodologia.

A. Resultados por Plataforma e Tecnologia

A Tabela I apresenta os resultados obtidos na plataforma Android utilizando os modelos *MLKit* e *Tesseract* com diferentes pré-processamentos aplicados.

Tabela I
Resultados no Android: *MLKit* e *Tesseract*

Modelo	Processamento	WordAcc (%) ↑	CharAcc (%) ↑	WER (%) ↓	CER (%) ↓
MLKit	Tons de Cinza	80.19	89.48	19.81	10.52
	Original	80.07	89.82	19.85	10.18
	Suavização sequencial	76.30	87.78	23.70	12.22
	Ajuste de Contraste	76.06	86.06	23.94	13.94
	Limiarização Adaptativa	54.72	73.02	45.28	26.98
	Morfologia	48.70	69.86	51.30	30.14
Tesseract	Tons de Cinza	63.68	77.33	36.32	22.67
	Original	59.79	74.70	40.21	25.30
	Suavização sequencial	58.45	72.83	41.55	27.17
	Ajuste de Contraste	58.25	72.26	41.75	27.74
	Morfologia	42.37	62.89	57.63	37.11
	Limiarização Adaptativa	31.37	50.96	68.63	49.04

Legenda: ■ Melhor resultado de cada tecnologia. Setas indicam se valores maiores (↑) ou menores (↓) são desejáveis.

No Android, observa-se que o *MLKit* se destaca na configuração original, sendo ligeiramente superado pela versão em tons de cinza. Isso sugere que o modelo se beneficia de pré-processamentos leves, mas é sensível a alterações mais invasivas, como morfologia e binarização adaptativa, que causaram perdas de desempenho. O *Tesseract*, por sua vez, parte de um desempenho inferior na imagem original, mas apresenta melhora notável com a aplicação de pré-processamento em escala de cinza, o que evidencia o potencial de otimização mesmo em soluções *open-source* mais simples.Em seguida, a Tabela II apresenta os resultados da plataforma iOS, comparando o modelo nativo *Vision OCR* com o *Tesseract*.

No iOS, o *Vision OCR* apresentou o melhor desempenho. Pré-processamentos adicionais não trouxeram melhorias, e em alguns casos, como ajuste de contraste, que reduziu a acurácia. O *Tesseract* teve desempenho inferior, no original, e reduções

Tabela II
RESULTADOS NO IOS: VISION E TESSERACT

Modelo	Processamento	WordAcc (%) ↑	CharAcc (%) ↑	WER (%) ↓	CER (%) ↓
Vision OCR	Original	86.79	91.38	13.21	08.62
	Tons de Cinza	81.13	89.53	20.17	09.15
	Suavização sequencial	80.90	87.11	19.04	12.89
	Ajuste de Contraste	61.67	68.95	38.27	31.05
	Morfologia	48.26	57.72	51.74	42.28
Tesseract	Original	56.81	65.47	43.19	34.53
	Suavização sequencial	44.10	53.57	55.90	46.43
	Ajuste de Contraste	44.78	51.29	55.22	48.71
	Tons de Cinza	34.11	56.00	73.64	38.64
	Morfologia	32.41	36.75	69.48	60.89

Legenda: ■ Melhor resultado de cada tecnologia. Setas indicam se valores maiores (↑) ou menores (↓) são desejáveis.

significativas na maioria dos pré-processamentos, indicando baixa adaptação à plataforma.

De forma geral, o *Vision OCR* se confirma como a solução mais eficaz no iOS, enquanto o *MLKit* mantém boa performance no Android, especialmente com ajustes simples como tons de cinza. O *Tesseract*, embora mais limitado, obtém ganhos pontuais com pré-processamentos leves, podendo ser viável em cenários locais e *offline*.

B. Impacto dos Modelos Embarcados no Aplicativo

O tamanho dos modelos utilizados é um ponto relevante em aplicações móveis, especialmente porque, neste estudo, eles são embarcados no próprio aplicativo. Essa abordagem garante que a funcionalidade opere sem conexão com a internet, mas exige atenção devido ao impacto no tamanho final do app e na usabilidade — entendida como a facilidade de instalação, operação e manutenção da solução, sobretudo em ambientes totalmente embarcados.

No iOS, o *Vision*, solução nativa da Apple para OCR, utiliza modelos de *machine learning* já integrados ao sistema operacional. Ao usar funcionalidades como `VNDocumentCameraViewController` ou `VNRecognizeTextRequest`, o aplicativo acessa esses modelos diretamente, sem aumentar o tamanho do binário, garantindo funcionalidade *offline* imediata, sem *downloads* adicionais, e mantendo o app leve e simples de instalar e usar.

Já no Android, o *ML Kit* adota uma estratégia diferente. Embora também permita uso *offline*, o desenvolvedor pode optar por embarcar os modelos diretamente no aplicativo, o que aumenta o tamanho do APK ou AAB inicial, mas elimina a latência de *download* e assegura a disponibilidade imediata da funcionalidade após a instalação, adicionando cerca de 300 *kilobytes* ao pacote [5]. Por fim, o *Tesseract* requer uma inclusão mais explícita dos componentes, como o motor principal e os arquivos de idioma, que podem adicionar de 5 a 15 *megabytes* ou mais por idioma, além do tamanho do motor compilado [25]. Essa abordagem oferece controle total ao desenvolvedor, mas demanda gerenciamento manual dos componentes e equilíbrio entre tamanho e funcionalidade, o que pode afetar a usabilidade.

Em síntese, o *Vision Framework* proporciona uma solução nativa sem impacto no tamanho do app e com usabilidade favorecida, o *ML Kit* oferece embarque opcional de modelos com impacto moderado e instalação simplificada, enquanto o *Tesseract* garante maior controle, porém com maior esforço de gestão para preservar a experiência do usuário.

C. Segurança e Privacidade no Processamento On-Device

Do ponto de vista da segurança dos dados, todas as soluções avaliadas oferecem suporte ao processamento *on-device*, ou seja, sem envio das imagens para servidores externos — característica essencial para aplicações sensíveis, como leitura de documentos ou dados pessoais.

Entretanto, o nível de controle sobre a privacidade varia. O *Tesseract Mobile* é o mais seguro nesse quesito: por ser uma biblioteca *open-source* e executada integralmente no dispositivo, nenhum dado sai do ambiente local, e não há dependência de políticas de terceiros. O *Vision OCR OCR* também opera de forma totalmente local, dentro do ecossistema Apple, com fortes garantias de privacidade, mas com menor flexibilidade sobre o pipeline interno. Já o *MLKit Mobile*, embora também suporte modo *on-device*, é parte do ecossistema Google, o que requer atenção redobrada às configurações de nuvem e políticas de privacidade para evitar o envio involuntário de dados.

Embora existam propostas como a de Correa *et al.* [11], que visam aprimorar o reconhecimento em documentos complexos, essas abordagens normalmente dependem de processamento externo ou combinação de serviços na nuvem. O presente estudo, ao se concentrar na execução local e autônoma dos modelos, elimina esses riscos e reforça a importância da computação de borda para a preservação da privacidade.

V. CONCLUSÃO

Este trabalho avaliou comparativamente o desempenho de diferentes modelos de OCR em dispositivos móveis de plataformas Android e iOS, considerando o impacto de diversas técnicas de pré-processamento de imagem. Os resultados demonstraram que soluções nativas como o *Vision OCR OCR e o MLKit Mobile* apresentam desempenho superior em suas configurações originais ou com ajustes mínimos na entrada.

Destaca-se que o *Vision OCR* obteve o melhor resultado geral sem necessidade de pré-processamento, evidenciando a qualidade do *framework* da Apple. O *MLKit* também mostrou-se eficaz, com ganhos discretos por meio de pré-processamento. Por outro lado, o *Tesseract Mobile*, apesar de sua performance inferior, demonstrou evolução com técnicas simples de aprimoramento, o que reforça seu valor em cenários com restrições de privacidade e custo, devido à sua natureza open-source e totalmente local.

Do ponto de vista da segurança dos dados, todas as tecnologias avaliadas oferecem, em geral, suporte ao processamento *on-device*, importante para aplicações que lidam com

informações sensíveis, como documentos e dados pessoais. Contudo, é importante destacar uma exceção: na plataforma iOS, o *MLKit* não realiza OCR localmente, encaminhando as imagens para os servidores do Google para processamento. Essa particularidade impõe limitação na questão de segurança, em cenários que demandam total controle e sigilo dos dados.

Nesse contexto, o *Tesseract* se destaca como a solução mais segura e auditável, por ser completamente *open-source* e executado de forma inteiramente local. Já o Vision OCR, apesar de também operar *on-device*, possui um *pipeline* fechado e menos flexível, ainda que com fortes garantias de privacidade dentro do ecossistema Apple.

A. Trabalhos Futuros

Como próximos passos, propõe-se a exploração de modelos de OCR multilíngue, ampliando a aplicabilidade da solução em cenários reais que envolvem textos em diferentes idiomas, em datasets mais recentes e desafiadores. Outra linha promissora é a avaliação de modelos neurais leves, como implementações em *TensorFlow Lite* e *PyTorch Mobile* presentes no estado da arte, que podem oferecer melhor compromisso entre acurácia e eficiência em dispositivos móveis. Estratégias de otimização de tempo de execução, incluindo técnicas de quantização e *pruning*, também serão consideradas para reduzir o consumo de recursos sem perda significativa de desempenho.

Além disso, vislumbra-se a investigação de modelos de OCR baseados em LLMs, aproveitando a popularização de arquiteturas multimodais para explorar abordagens com *transformers* ou modelos de linguagem visual. Outra direção relevante é a integração com sistemas de validação semântica, especialmente em aplicações críticas como a leitura de documentos oficiais, onde a verificação do conteúdo pode aumentar a confiabilidade dos resultados.

Referências

[1] B. Chatpaitoon and R. Itthipanichpong, "Effect of optical text recognition (ocr) and text-to-speech (tts) smartphone app on vision-related quality of life in visually impaired people." *Investigative Ophthalmology & Visual Science*, vol. 64, no. 8, pp. 2808–2808, 2023.

[2] S. Kompalli, S. Nayak, S. Setlur, and V. Govindaraju, "Challenges in ocr of devanagari documents," in *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE, 2005, pp. 327–331.

[3] M. Laine and O. S. Nevalainen, "A standalone ocr system for mobile cameraphones," in *2006 IEEE 17th international symposium on personal, indoor and mobile radio communications*. IEEE, 2006, pp. 1–5.

[4] Apple Inc., "Vision Framework - Apple Developer Documentation," 2024, acessado em: 30 de julho de 2025. [Online]. Available: https://developer.apple.com/documentation/vision?language=objc

[5] Google Developers, "ML Kit — Google for Developers," 2024, acessado em: 30 de julho de 2025. [Online]. Available: https://developers.google.com/ml-kit?hl=pt-br

[6] R. Smith, "An overview of the tesseract ocr engine," in *Ninth international conference on document analysis and recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 629–633.

[7] S. Badla, "Improving the efficiency of tesseract ocr engine," 2014.

[8] Y. Wang, Y. Liu, T. Wu, and I. Duncan, "A cost-effective ocr implementation to prevent phishing on mobile platforms," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 2020, pp. 1–8.

[9] T. Mantoro, A. M. Sobri, and W. Usino, "Optical character recognition (ocr) performance in server-based mobile environment," in *2013 International Conference on Advanced Computer Science Applications and Technologies*. IEEE, 2013, pp. 423–428.

[10] L. V. da Silva, P. L. J. D. Junior, and S. S. da Costa Botelho, "An optical character recognition post-processing method for technical documents," in *Conference on Graphics, Patterns and Images (SIBGRAPI)*. SBC, 2023, pp. 126–131.

[11] I. L. Correa, P. L. J. Drews, and R. N. Rodrigues, "Combination of optical character recognition engines for documents containing sparse text and alphanumeric codes," in *2021 34th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. IEEE, 2021, pp. 299–306.

[12] Mindee, "doctr: Document text recognition," https://github.com/mindee/doctr, 2021.

[13] A. Malkadi, M. Alahmadi, and S. Haiduc, "A study on the accuracy of ocr engines for source code transcription from programming screencasts," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 65–75.

[14] A. P. Tafti, A. Baghaie, M. Assefi, H. R. Arabnia, Z. Yu, and P. Peissig, "Ocr as a service: an experimental evaluation of google docs ocr, tesseract, abbyy finereader, and transym," in *International Symposium on Visual Computing*. Springer, 2016, pp. 735–746.

[15] K. Thammarak, P. Kongkla, Y. Sirisathitkul, and S. Intakosum, "Comparative analysis of tesseract and google cloud vision for thai vehicle registration certificate," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 2, pp. 1849–1858, 2022.

[16] R. Kaur and D. V. Sharma, "Punjabi text recognition system for portable devices: A comparative performance analysis of cloud vision api with tesseract," *Journal of Computer Science and Engineering (JCSE)*, vol. 2, no. 2, pp. 104–111, 2021.

[17] L. R. Mursari and A. Wibowo, "The effectiveness of image preprocessing on digital handwritten scripts recognition with the implementation of ocr tesseract," *Computer Engineering and Applications Journal*, vol. 10, no. 3, pp. 177–186, 2021.

[18] U. Hengaju and B. K. Bal, "Improving the recognition accuracy of tesseract-ocr engine on nepali text images via preprocessing," *Advancement in Image Processing and Pattern Recognition*, vol. 3, no. 2, p. 3, 2023.

[19] D. Sporici, E. Cușnir, and C.-A. Boiangiu, "Improving the accuracy of tesseract 4.0 ocr engine using convolution-based preprocessing," *Symmetry*, vol. 12, no. 5, p. 715, 2020.

[20] H. Michalak and K. Okarma, "Improvement of image binarization methods using image preprocessing with local entropy filtering for alphanumerical character recognition purposes," *entropy*, vol. 21, no. 6, p. 562, 2019.

[21] A. Franco, "Optical recognition of the philippines'ancient text: A deep learning approach," *INTERNATIONAL JOURNAL*, vol. 8, no. 03, 2025.

[22] B. P. Sari, R. W. Sholikah, and R. V. H. Ginardi, "The development of mobile application for object recognition based on deep learning to assist people with visually impaired," in *2024 2nd International Symposium on Information Technology and Digital Innovation (ISITDI)*. IEEE, 2024, pp. 222–227.

[23] S. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young, "Icdar 2003 robust reading competitions," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, 2003, pp. 682–687.

[24] J. B. Zimmerman, S. M. Pizer, E. V. Staab, J. R. Perry, W. McCartney, and B. C. Brenton, "An evaluation of the effectiveness of adaptive histogram equalization for contrast enhancement," *IEEE Transactions on Medical Imaging*, vol. 7, no. 4, pp. 304–312, 1988.

[25] Tesseract OCR, "Data Files — Tesseract OCR," https://tesseract-ocr.github.io/tessdoc/tess3/Data-Files.html, 2025, acessado em 8 de agosto de 2025.