# HarpIA: a tool for comparative analysis of embedded AI models on Android

**Ricardo Miranda Filho**[1], **Pedro Matias**[1], **Rosiane de Freitas**[1]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)

{ricardo.filho, pvsm, rosiane}@icomp.ufam.edu.br

**Abstract** This work addresses the fragmentation of the Android ecosystem, which hinders the benchmarking of machine learning frameworks, making it a slow process that depends on frequent recompilations. To mitigate this limitation, we present HarpIA, a benchmarking tool whose dynamic architecture enables the evaluation of TensorFlow, PyTorch, and MindSpore models without the need to recompile the APK for each test. The study focused on validating the tool, whose reliability was verified by comparing inference time and energy consumption metrics—collected during tests with the ImageNet-V2 dataset on models converted via ONNX—with performance references from the literature. As a result, HarpIA emerges as a solution that supports developers in performing performance comparisons more efficiently, aiding in the selection of frameworks for the Android environment.

**Resumo** Este trabalho discute a fragmentação do ecossistema Android, que dificulta o benchmarking de frameworks de aprendizado de máquina, tornando-o um processo lento e dependente de recompilações. Para mitigar essa limitação, apresentamos a HarpIA, uma ferramenta de benchmarking cuja arquitetura dinâmica possibilita a avaliação de modelos de TensorFlow, PyTorch e MindSpore sem a necessidade de recompilar o APK a cada teste. O foco do estudo foi a validação da ferramenta, cuja confiabilidade foi verificada ao se comparar as métricas de tempo de inferência e consumo energético — coletadas durante testes com o dataset ImageNet-V2 em modelos convertidos via ONNX — com referências de desempenho da literatura. Como resultado, a HarpIA se apresenta como uma solução que apoia desenvolvedores na realização de comparações de desempenho de forma mais eficiente, auxiliando na escolha de frameworks para o ambiente Android.

## 1. Introdução

A inteligência artificial (IA), e mais especificamente o aprendizado profundo, transcendeu os limites dos data centers para integrar-se profundamente aos dispositivos móveis e embarcados contemporâneos. Equipados com *Systems-on-a-Chip* (SoCs) cada vez mais potentes — que integram CPUs multi-core, unidades de processamento gráfico (GPUs) robustas e aceleradores de hardware dedicados, como as Unidades de Processamento Neural (NPUs) — os smartphones modernos possuem uma capacidade computacional que viabilizou um novo paradigma tecnológico. Aplicações que vão desde assistentes de voz e editores de imagens até o pós-processamento fotográfico demonstram a ubiquidade dessas tecnologias [Xu et al. 2019, Hu et al. 2023].

Historicamente, as tarefas de inferência de modelos de IA eram predominantemente executadas na nuvem. Nessa abordagem, servidores externos eram responsáveis tanto pelo treinamento quanto pela inferência, e o dispositivo móvel atuava meramente como um terminal para envio e recebimento de dados, uma dinâmica ditada pelas

limitações de bateria, processamento e memória dos aparelhos. A Figura 1 ilustra esquematicamente esse modelo.


**Figura 1. Esquema gráfico da abordagem com inferência em servidor externo (Off-device).**

Contudo, a crescente demanda por interações em tempo real, maior privacidade e funcionamento offline impulsionou uma mudança fundamental: a migração da inferência da nuvem para o próprio dispositivo, abordagem conhecida como *on-device*. Ao executar os modelos localmente, os aplicativos ganham em responsividade, reduzem a latência, garantem a privacidade ao não transmitir dados sensíveis e operam independentemente da conectividade com a internet. Embora o treinamento dos modelos ainda possa ocorrer em servidores, a execução local da inferência melhora substancialmente a experiência do usuário e a segurança dos dados, conforme mostrado na Figura 2.


**Figura 2. Esquema gráfico da abordagem com inferência no dispositivo móvel (On-device).**

Apesar de suas vantagens, a implementação de IA *on-device* no ecossistema Android apresenta um desafio notável devido à sua natureza fragmentada. Essa fragmentação manifesta-se em múltiplas camadas:

- **Heterogeneidade de hardware:** existe uma vasta diversidade de SoCs de fabricantes como Qualcomm, Samsung e MediaTek, cada um com arquiteturas distintas de CPU (e.g., ARM big.LITTLE), GPU e aceleradores de IA dedicados;
- **Fragmentação de software:** sobreposta ao hardware, há uma pilha de software complexa, envolvendo diferentes versões do sistema operacional, drivers específicos e múltiplas bibliotecas de inferência concorrentes, como TensorFlow Lite, PyTorch Mobile, ncnn, MNN, entre outras.

Essa combinação de fatores cria um cenário em que o desempenho é, como mostrado em estudos de benchmarking, "severamente fragmentado". Não existe uma solução única (*one-size-fits-all*); a escolha da biblioteca de software pode impactar mais o desempenho do que otimizações no próprio modelo de IA ou diferenças de hardware. Para os desenvolvedores, essa complexidade gera um dilema prático: como tomar decisões eficientes em meio a tantas variáveis?

Diante desse cenário, este trabalho propõe a ferramenta *HarpIA*, projetada para simplificar e acelerar a avaliação comparativa de modelos e frameworks no Android. Seu

diferencial está na capacidade de realizar testes de forma dinâmica, sem recompilar e reinstalar o APK a cada experimento, reduzindo tempo e esforço. Dessa forma, desenvolvedores e pesquisadores podem avaliar rapidamente múltiplas configurações e identificar soluções mais adequadas para seus casos de uso.

Este trabalho está organizado da seguinte forma: a Seção 2 discute os trabalhos relacionados; a Seção 3 detalha o desenvolvimento da ferramenta HarpIA; a Seção 4 descreve o design dos experimentos e a avaliação; a Seção 5 exibe e discute os resultados obtidos.

## 2. Trabalhos relacionados

Esta seção apresenta os principais trabalhos da literatura relacionados ao tema, destacando suas características e contribuições. A Tabela 1 consolida a comparação entre diferentes benchmarks, evidenciando objetivos, bibliotecas de aprendizado de máquina suportadas, arquiteturas de redes neurais avaliadas e ferramentas de medição de desempenho. Essa visão facilita a compreensão das abordagens existentes e posiciona o presente trabalho no contexto atual da pesquisa.

**Tabela 1. Comparação consolidada e corrigida entre benchmarks relacionados, seus focos, bibliotecas suportadas, arquiteturas exploradas e ferramentas de avaliação.**

| Benchmark | Referência | Foco Principal | Foco Secundário | Bibliotecas de ML Suportadas | Tipos de RNs exploradas | Ferramenta (Tipo e Nome) |
|---|---|---|---|---|---|---|
| MLPerf Mobile | [Reddi et al. 2020] | Desempenho de Sistema | Hardware, Acurácia | Backend-agnóstico (Ref. em TFLite) | CNNs, Transformers | Aplicação (MLPerf Mobile app) |
| AIoTBench | [Luo et al. 2020] | Bibliotecas de ML e Arquiteturas | Hardware | Tensorflow, Pytorch, Caffe2 | CNNs | Framework de Pesquisa (AIoTBench) |
| AI-Benchmark | [Ignatov et al. 2019] | Aceleração de Hardware (SoC) | Bibliotecas de ML (TFLite) | Tensorflow (via TFLite/NNAPI) | CNNs | Aplicação (AI Benchmark app) |
| gaugeNN | [Almeida et al. 2021] | Arquiteturas de GNNs | Princípios de GNNs | Pytorch, DGL | GNNs | Framework de Código Aberto |
| MDLBench | [Zhang et al. 2022] | Bibliotecas de ML (Inferência) | Hardware, Modelos | TFLite, PyTorch, ncnn, MACE, MNN, SNPE | CNNs, Transformers | Suíte de Benchmark (MDLBench) |
| **Este trabalho** | | **Frameworks de ML (Ecossistema)** | Arquiteturas de NNs | **Tensorflow, PyTorch, MindSpore** | CNNs | **Aplicação (HarpIA)** |

O *AI Benchmark*, de Ignatov et al. [Ignatov et al. 2019], endereça a ausência de ferramentas para avaliar aceleradores de hardware (NPUs e DSPs) cada vez mais comuns em SoCs móveis. Utilizando TensorFlow Lite e a API NNAPI, compara execuções em hardware especializado e fallback em CPU, medindo o desempenho de chipsets de fabricantes como Qualcomm e Samsung, consolidando-se como referência para avaliação de hardware no Android.

Luo et al. [Luo et al. 2020] propõem o *AIoTBench*, uma suíte de benchmarks que compara frameworks e arquiteturas de redes neurais em dispositivos móveis e embarcados, incluindo ResNet50, InceptionV3 e MobileNetV2. Além de latência e acurácia, introduzem métricas como VIPS (imagens válidas por segundo) e VOPS (FLOPs válidos por segundo) para análises mais detalhadas.

Reddi et al. [Reddi et al. 2020] apresentam o *MLPerf Mobile*, benchmark open source apoiado pela indústria. Seu objetivo é avaliar sistemas completos em tarefas de Visão Computacional e NLP, exigindo uma acurácia mínima para garantir comparabilidade justa entre implementações. Essa padronização impulsiona otimizações em todo o ecossistema de IA móvel.

O *MDLBench*, proposto por Zhang et al. [Zhang et al. 2022], foca na análise de bibliotecas de inferência como TFLite, PyTorch Mobile e SNPE, evidenciando a fragmentação de desempenho e a inexistência de uma solução universalmente ótima. A escolha da biblioteca depende fortemente de modelo, hardware e versão de software.

Por fim, este trabalho complementa essas abordagens ao comparar frameworks completos de ML no ecossistema Android, considerando desde a criação e treinamento até a inferência, com destaque para a inclusão do MindSpore — framework emergente ainda pouco avaliado na plataforma móvel. O objetivo principal é fornecer uma ferramenta projetada para simplificar e acelerar a avaliação comparativa de modelos e frameworks no Android. Seu diferencial está na capacidade de realizar testes de forma dinâmica, sem recompilar e reinstalar o APK a cada experimento, reduzindo tempo e esforço. Dessa forma, desenvolvedores e pesquisadores podem avaliar rapidamente múltiplas configurações e identificar soluções mais adequadas para seus casos de uso.

## 3. HarpIA

### 3.1. Seleção das Bibliotecas de Inferência

Trabalhos como Xu et al. [Xu et al. 2019] e Hu et al. [Hu et al. 2023] analisaram bibliotecas de inteligência artificial considerando criação, treinamento e inferência de modelos em diferentes plataformas. Esses estudos serviram de base para identificar frameworks relevantes sob critérios como desempenho, suporte a dispositivos móveis e manutenção ativa. A Tabela 2 resume as principais características desses frameworks, incluindo suporte a APIs móveis, capacidade de criação e treinamento, além do status de atualização.

**Tabela 2. Comparativo entre os Frameworks citados na literatura e relatórios técnicos.**

| Framework | Proprietário | Mobile API | Criação e Treino | Status |
|---|---|---|---|---|
| **ADLink** | LF AI | - | Não (Otimização e implantação) | Ativo (Incubação) |
| **Caffe** | Berkeley AI Research (BAIR) | Sim (Implantação) | Sim | Manutenção / Legado |
| **Caffe2** | Facebook (Meta) | Sim | Sim | Obsoleto (Fundido com PyTorch) |
| **CNNDroid** | Oskouei et al. (Acadêmico) | Sim (Android via RenderScript) | Não (Motor de inferência) | Inativo / Arquivado |
| **Deeplearning4j** | Eclipse Foundation (Originalmente Skymind) | Sim (JVM/Android) | Sim | Ativo |
| **FeatherCNN** | Tencent | Sim (Foco em ARM) | Não (Motor de inferência) | Inativo / Manutenção |
| **MACE** | Xiaomi | Sim (Heterogêneos móveis) | Não (Motor de inferência) | Ativo |
| **MindSpore** | Huawei | Sim (MindSpore Lite) | Sim | Ativo |
| **MNN** | Alibaba | Sim (Multiplataforma) | Sim (MNN-Train) | Ativo |
| **MxNet** | Apache Software Foundation | Sim | Sim | Inativo / Abandonado |
| **NeoML** | ABBYY | Sim (Multiplataforma) | Sim | Ativo |
| **NNABLA** | Sony | Sim (C/JS Runtimes) | Sim | Manutenção |
| **ONNX** | LF AI & Data Foundation | Não (Formato padrão) | Não | Ativo (Padrão da Indústria) |
| **PaddlePaddle** | Baidu | Sim (Paddle-Lite) | Sim | Ativo |
| **PyTorch** | Linux Foundation (Originalmente Facebook) | Sim (ExecuTorch) | Sim | Ativo |
| **ShaderNN** | LF AI & Data Foundation (OPPO) | Não (Motor de inferência) | Não | Ativo (Sandbox) |
| **SNPE** | Qualcomm | Sim (Snapdragon) | Não (SDK inferência) | Ativo |
| **Tengine** | OPEN AI LAB | Sim (Embarcados / AIoT) | Não (Motor de inferência) | Ativo |
| **TensorFlow** | Google | Sim (TensorFlow Lite / LiteRT) | Sim | Ativo |

Para este estudo, foram priorizados TensorFlow, PyTorch e MindSpore, segundo três critérios: **(i)** suporte completo à criação e treinamento de redes neurais; **(ii)** presença de APIs Java/Kotlin para integração com aplicações Android; **(iii)** manutenção ativa e atualizações recentes.

O TensorFlow combina grafos de fluxo de dados flexíveis com uma versão otimizada (TensorFlow Lite), voltada para baixa latência e eficiência energética em dispositivos móveis [Abadi et al. 2016]. Já o PyTorch, conhecido pela abordagem de grafos dinâmicos e facilidade de prototipagem, conta com o PyTorch Mobile para implantação on-device

[Paszke et al. 2019]. Por fim, o MindSpore, desenvolvido pela Huawei, adota arquitetura AI-native para cenários em nuvem, borda e dispositivos móveis, disponibilizando o MindSpore Lite como solução otimizada para inferência embarcada [Huawei 2022].

## 3.2. Desenvolvimento da Ferramenta

### 3.2.1. Integração dos Interpretadores

A integração dos interpretadores de machine learning no aplicativo HarpIA permite a execução local de modelos otimizados para dispositivos Android. Cada interpretador adota uma abordagem específica para conectar o ambiente Android (Java/Kotlin) ao núcleo de execução em C++, utilizando a Java Native Interface (JNI) e bibliotecas nativas. A Figura 3 mostra o fluxo geral de dados no aplicativo.

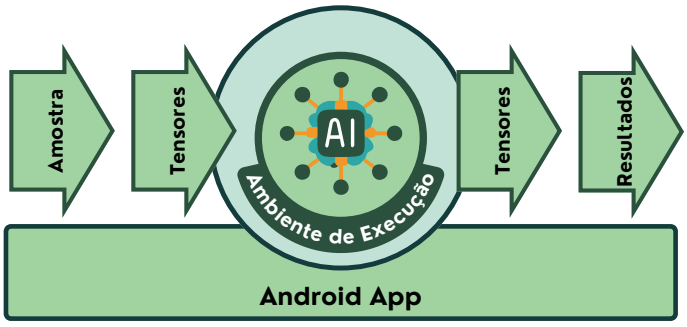


**Figura 3. Fluxo dos dados em um aplicativo Android que executa modelos de IA.**

O LiteRT utiliza bibliotecas nativas em C++ e comunicação via JNI. Os modelos são serializados em FlatBuffers, no formato .tflite ou .litert, que armazena arquitetura, pesos e metadados do modelo em um único arquivo binário.

O MindSpore Lite conecta o ambiente Android ao runtime C++ por meio da biblioteca libmindspore-lite.so, com APIs Java no pacote org.mindspore. Os modelos precisam ser convertidos para o formato .ms e são carregados diretamente do armazenamento do dispositivo, com suporte opcional para aceleração via GPU, DSP ou NPU.

O PyTorch Mobile conecta o runtime em C++ ao Android via JNI, permitindo carregar modelos .pt a partir do armazenamento local durante a execução. Essa funcionalidade foi descontinuada no ExecuTorch, que exige a integração do modelo diretamente no APK no momento da compilação, inviabilizando o carregamento dinâmico.

## 3.3. Tratamento de Tipos

Para que a *HarpIA* funcione como ferramenta de benchmarking geral, capaz de comparar modelos heterogêneos, o tratamento de tipos de dados é aspecto técnico essencial. Após otimizações, modelos podem exigir tensores de entrada em formatos variados (por exemplo, `float32`, `float16`, `int8`), incompatíveis com estruturas-padrão do Android como `Bitmap`.

A *HarpIA* resolve essa incompatibilidade com um pipeline automatizado de pré-processamento, dispensando código manual para cada modelo. O fluxo ocorre em três etapas: (i) inspeção do modelo via interpretador (por exemplo TensorFlow Lite ou MindSpore Lite) para obter tipo e forma do tensor de entrada; (ii) extração de pixels do `Bitmap`

para um `IntArray` de alta performance, no qual cada inteiro armazena valores ARGB; (iii) conversão e normalização desses valores em um `java.nio.ByteBuffer`, diretamente acessível pelo interpretador nativo, sem cópias adicionais.

A conversão no `ByteBuffer` adapta-se ao tipo de dado: - `float32`: normaliza cada canal RGB no intervalo [0,1]; - `float16`: mesmo processo, mas convertendo para meia precisão; - `int8`/`uint8`: aplica a quantização (com *scale* e *zero-point*) para 8 bits.

### 3.3.1. Coleta das Métricas

Para avaliar o desempenho dos modelos e frameworks de IA em dispositivos Android, a HarpIA implementa métodos específicos para a coleta de métricas de tempo de execução e consumo energético. O tempo de execução ($T_{\text{exec}}$) foi calculado utilizando o método `elapsedRealtime` da classe `SystemClock`, que retorna o tempo em milissegundos desde o boot do dispositivo. O tempo inicial ($T_{\text{start}}$) é registrado imediatamente antes da execução do algoritmo de inferência e o tempo final ($T_{\text{end}}$) é registrado após sua conclusão. A diferença entre esses dois valores representa o tempo total de execução da inferência, conforme a Equação 1, sendo que a implementação desse cálculo é ilustrada no Código 1.

$$\mathrm{T_{exec}} = \mathrm{T_{end}} - \mathrm{T_{start}} \tag{1}$$

**Listing 1. Código para calcular o tempo de execução.**

```
long timeStart = SystemClock.elapsedRealtime();
// Execucao do Modelo para Realizar Inferencia
long timeEnd = SystemClock.elapsedRealtime();
long timeElapsed = timeEnd - timeStart;
```

Para estimar o consumo de energia, foram utilizados os métodos da API `BatteryManager`, que permitem obter a voltagem (`EXTRA_VOLTAGE`) e a corrente instantânea (`BATTERY_PROPERTY_CURRENT_NOW`) da bateria. As medições de corrente e voltagem são realizadas em intervalos discretos de tempo ($\Delta t$) durante a execução da inferência. A energia total consumida ($E$) é aproximada pela soma do produto da voltagem, corrente e o intervalo de tempo de cada amostra, como descrito na Equação 2, sendo que a coleta de dados utilizada para essa estimativa é demonstrada no Código 2.

$$E = \sum_{i=1}^{n-1} (V_i \cdot I_i \cdot \Delta t_i) \tag{2}$$

Onde $V_i$ é a voltagem (V), $I_i$ a corrente (A) e $\Delta t_i$ o intervalo de tempo entre as amostras. Para obter resultados precisos e evitar interferências, é crucial que os testes sejam realizados com o dispositivo desconectado de fontes externas de energia (carregadores).

**Listing 2. Código para coletar dados de consumo de energia.**

```
long time = getTime();
```

```
Intent intent = context.registerReceiver(null,
   INTENT_BATTERY_FILTER);
double voltage =
   intent.getIntExtra(BatteryManager.EXTRA_VOLTAGE, -1);
double current = batteryManager.getIntProperty(
   BatteryManager.BATTERY_PROPERTY_CURRENT_NOW);
```

## 4. Design de Experimentos

### 4.1. Seleção dos Modelos de Redes Neurais

Para os experimentos, foram selecionadas arquiteturas convolucionais representativas de diferentes estratégias de otimização, visando desempenho e eficiência em dispositivos com recursos limitados. Incluem-se Inception [Szegedy et al. 2014], com módulos multi-escala; SqueezeNet [Iandola et al. 2016], de baixa quantidade de parâmetros; MnasNet [Tan et al. 2018], otimizada via busca neural automatizada; e MobileNetV3 [Howard et al. 2019], que incorpora módulos *squeeze-and-excitation* para melhor desempenho em dispositivos móveis. Essa seleção cobre variadas abordagens de eficiência, permitindo uma análise comparativa ampla.

Os modelos foram previamente treinados e validados no conjunto de dados ImageNet [Deng et al. 2009], que contém mais de 1,2 milhões de imagens em mil classes, referência consagrada para avaliação de CNNs. Assim, os experimentos deste trabalho focaram na validação da ferramenta HarpIA, comparando os resultados obtidos com aqueles reportados na literatura.

### 4.2. Conversão dos Modelos

Para os experimentos realizados com a *HarpIA*, adotou-se um pipeline de preparação de modelos que assegura consistência e comparabilidade entre os diferentes frameworks. O processo, ilustrado na Figura 4, emprega o formato Open Neural Network Exchange (ONNX) como padrão intermediário para conversão, garantindo uniformidade entre as versões.

O fluxo de trabalho foi conduzido em três etapas principais. Primeiramente, realizaram-se a **obtenção dos modelos base**, a partir de arquiteturas definidas na Subseção 4.1, obtidas do PyTorch Model Zoo. Esses modelos, no formato nativo (`.pt`), serviram como base para os testes no ambiente PyTorch.

Em seguida, procedeu-se à **conversão para o formato intermediário ONNX**, garantindo que todas as versões destinadas aos outros frameworks partissem da mesma fonte. O uso do ONNX como formato padronizado é fundamental para minimizar discrepâncias decorrentes de processos de conversão diretos e potencialmente inconsistentes para cada plataforma.

Por fim, realizaram-se as **conversões para os formatos finais**. A partir dos arquivos `.onnx`, os modelos foram convertidos para os formatos compatíveis com cada interpretador alvo: para o TensorFlow Lite, geraram-se arquivos `.tflite`; para o MindSpore, arquivos `.ms`, utilizando as ferramentas oficiais disponibilizadas por cada ecossistema.

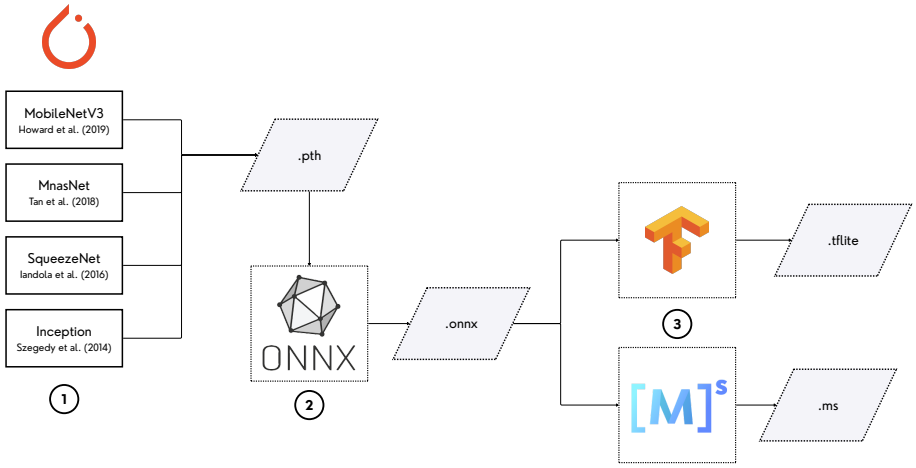


**Figura 4. Fluxo de conversão de modelos a partir do PyTorch Zoo, utilizando o formato ONNX como intermediário para gerar versões compatíveis com TensorFlow Lite e MindSpore.**

## 4.3. Dispositivos

A Tabela 3 mostra os dispositivos móveis utilizados para a realização dos experimentos envolvendo os modelos implementados e portados para ambiente Android utilizando diferentes frameworks. Esses dispositivos foram selecionados para representar uma variedade de chipsets, arquiteturas de processador e versões do sistema operacional Android, permitindo uma análise mais abrangente do desempenho dos modelos em diferentes configurações de hardware.

**Tabela 3. Comparativo dos dados técnicos dos dispositivos usados nos experimentos.**

| Dados Técnicos | Dispositivo 1 | Dispositivo 2 | Dispositivo 3 | Dispositivo 4 |
|---|---|---|---|---|
| Nomes | Motorola Moto G200 | Google Pixel 7 Pro | Xiaomi POCO X5 | Samsung Galaxy S21 FE |
| Chipset | Snapdragon 888 Qualcomm (SM8350) | Google Tensor G2 (5 nm) | Qualcomm Snapdragon 695 (SM6375) | Samsung Exynos 2100 |
| Processador | 1x 2.99 GHz Kryo 680<br>3x 2.42 GHz Kryo 680<br>4x 1.80 GHz Kryo 680 | 2x 2.85 GHz Cortex-X1<br>2x 2.35 GHz Cortex-A78<br>4x 1.80 GHz Cortex-A55 | 2x 2.2 GHz Kryo 660 Gold<br>6x 1.7 GHz Kryo 660 Silver | 1x 2.9 GHz Cortex-X1<br>3x 2.8 GHz Cortex-A78<br>4x 2.2 GHz Cortex-A55 |
| GPU | Qualcomm Adreno 660 | Mali-G710 MP7 | Qualcomm Adreno 619 | Mali-G78 MP14 |
| 64 Bits | Sim | Sim | Sim | Sim |
| Android | 12 | 13 | 13 | 14 |
| RAM | 8 GB | 12 GB LPDDR5X | 6 GB | 6 GB |
| Tipo de Bateria | LiPo | LiPo | LiPo | LiPo |
| Amperagem | 5000 mAh | 5000 mAh | 5000 mAh | 4500 mAh |

## 4.4. Avaliação

O objetivo desta fase experimental é validar a HarpIA, demonstrando sua capacidade de gerar medições de desempenho consistentes e confiáveis em ambiente de teste realista. Para isso, a ferramenta realizou avaliação comparativa de modelos de aprendizado profundo, usando como métricas o tempo de inferência (latência) e estimativa de consumo energético.

Definiu-se um cenário controlado, empregando subconjunto do dataset ImageNet-V2 como entrada padronizada para modelos de classificação. O uso de dataset reconhecido garante tarefas representativas e resultados relevantes.

A execução foi orquestrada pela HarpIA, que gerenciou todo o fluxo experimental em dispositivos Android. Para cada imagem, realizou-se o carregamento do modelo, a preparação do tensor e a inferência, coletando métricas automaticamente.

Para cada combinação de modelo, framework e dispositivo, foram feitas 10 coletas independentes de tempo e consumo, cujas médias estão nas Tabelas.

Durante as medições, adotou-se controle básico de processos: conectividade desativada, nenhum app aberto além da HarpIA e brilho mínimo. Como os dispositivos são comerciais, não houve bloqueio de processos internos.

A validação focou na consistência, sem declarar "vencedor", mas avaliando a coerência e plausibilidade dos dados. A confiabilidade foi aferida ao verificar se latências e trade-offs estavam alinhados com padrões e tendências documentados em benchmarks como MLPerf e MDLBench.

## 5. Resultados e Discussões

A finalidade principal desta etapa experimental é validar a ferramenta HarpIA, demonstrando que suas medições de desempenho são consistentes e confiáveis em comparação com benchmarks e estudos consolidados na literatura. Para isso, a HarpIA foi empregada na avaliação comparativa de diferentes modelos de redes neurais profundas, medindo tempo de inferência e consumo energético sob variados cenários. Os resultados, resumidos na Tabela 4, demonstram consistência com benchmarks reconhecidos, como os padrões observados no MLPerf Mobile, o que valida a confiabilidade da ferramenta para medições ágeis e comparáveis.

**Tabela 4. Comparativo de tempo de inferência (ms) e consumo de energia (J) com base em proporcionalidade ao TensorFlow (Versão Corrigida).**

| Modelo | Dispositivo | TensorFlow | | | | PyTorch | | | | MindSpore | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | GPU | | CPU | | GPU | | CPU | | GPU | | CPU | |
| | | Tempo | Energia | Tempo | Energia | Tempo | Energia | Tempo | Energia | Tempo | Energia | Tempo | Energia |
| InceptionV3 | Motorola Moto G200 | 64.9 | 0.277 | 229.0 | 0.539 | 63.2 | 0.270 | 79.1 | 0.186 | 66.3 | 0.265 | 82.5 | 0.195 |
| | Google Pixel 7 Pro | 58.7 | 0.195 | 130.3 | 0.516 | 61.7 | 0.205 | 91.7 | 0.363 | 59.8 | 0.190 | 92.3 | 0.350 |
| | Xiaomi POCO X5 | 100.3 | 0.225 | 209.1 | 0.462 | 119.5 | 0.268 | 241.2 | 0.533 | 102.4 | 0.230 | 215.3 | 0.475 |
| | Samsung Galaxy S21 FE | 56.2 | 0.235 | 154.1 | 0.563 | 59.8 | 0.250 | 89.7 | 0.328 | 57.5 | 0.230 | 91.2 | 0.332 |
| SqueezeNet | Motorola Moto G200 | 42.7 | 0.167 | 92.6 | 0.222 | 25.2 | 0.098 | 26.5 | 0.064 | 26.1 | 0.092 | 27.3 | 0.070 |
| | Google Pixel 7 Pro | 38.9 | 0.148 | 48.6 | 0.249 | 31.4 | 0.119 | 28.0 | 0.144 | 32.0 | 0.122 | 29.1 | 0.125 |
| | Xiaomi POCO X5 | 56.8 | 0.135 | 77.5 | 0.184 | 37.9 | 0.090 | 76.6 | 0.181 | 38.5 | 0.092 | 78.0 | 0.185 |
| | Samsung Galaxy S21 FE | 35.9 | 0.158 | 72.6 | 0.275 | 43.1 | 0.190 | 40.8 | 0.146 | 42.0 | 0.185 | 41.2 | 0.148 |
| MnasNet | Motorola Moto G200 | 39.8 | 0.141 | 60.1 | 0.162 | 27.8 | 0.098 | 28.1 | 0.077 | 28.5 | 0.095 | 29.0 | 0.080 |
| | Google Pixel 7 Pro | 29.7 | 0.176 | 32.9 | 0.140 | 34.2 | 0.145 | 32.0 | 0.189 | 33.0 | 0.142 | 32.5 | 0.180 |
| | Xiaomi POCO X5 | 36.7 | 0.101 | 49.3 | 0.130 | 36.3 | 0.100 | 71.8 | 0.189 | 37.2 | 0.102 | 50.5 | 0.133 |
| | Samsung Galaxy S21 FE | 34.3 | 0.146 | 48.2 | 0.182 | 32.5 | 0.138 | 34.4 | 0.130 | 34.0 | 0.135 | 48.5 | 0.184 |
| MobileNetV3 | Motorola Moto G200 | 55.7 | 0.156 | 59.7 | 0.189 | 27.3 | 0.086 | 27.6 | 0.077 | 27.5 | 0.085 | 28.0 | 0.080 |
| | Google Pixel 7 Pro | 27.7 | 0.168 | 31.9 | 0.188 | 33.3 | 0.197 | 31.4 | 0.191 | 32.5 | 0.192 | 28.5 | 0.174 |
| | Xiaomi POCO X5 | 45.9 | 0.128 | 53.8 | 0.134 | 40.1 | 0.100 | 66.8 | 0.187 | 54.5 | 0.136 | 46.2 | 0.129 |
| | Samsung Galaxy S21 FE | 45.2 | 0.174 | 55.9 | 0.197 | 32.7 | 0.115 | 32.0 | 0.126 | 33.1 | 0.117 | 32.5 | 0.122 |

A análise dos resultados revela que a arquitetura da rede é um fator determinante para o desempenho. Modelos mais complexos, como o InceptionV3, consistentemente apresentaram maiores tempos de inferência e consumo energético. Por exemplo, no Xiaomi POCO X5 com CPU e TensorFlow, o InceptionV3 levou 209.1 ms para a inferência,

sendo aproximadamente 2.7 vezes mais lento que a SqueezeNet na mesma configuração (77.5 ms). Em contraste, a SqueezeNet evidenciou sua alta eficiência, registrando frequentemente os menores tempos e consumo entre os modelos avaliados, especialmente com aceleração de GPU. Já os modelos MnasNet e MobileNetV3, projetados por busca de arquitetura neural, apresentaram um equilíbrio notável entre desempenho e eficiência. No Pixel 7 Pro (GPU, TensorFlow), a MobileNetV3 (27.7 ms) foi aproximadamente 6,7% mais rápida e 4,5% mais eficiente em energia que a MnasNet (29.7 ms), refletindo o impacto de otimizações de arquitetura.

O desempenho também é fortemente influenciado pelo hardware do dispositivo. Diferenças marcantes foram observadas entre dispositivos de ponta (flagships) e intermediários. No POCO X5, um aparelho intermediário, a execução do InceptionV3 na GPU com TensorFlow (100.3 ms) foi cerca de 1.67 vezes mais lenta que a média dos aparelhos de ponta testados (Moto G200, Pixel 7 Pro, S21 FE), que ficou em 59.9 ms. Isso evidencia as limitações de sua GPU Adreno 619 em comparação com hardwares mais robustos. Dentro do segmento de alto desempenho, o Google Pixel 7 Pro se destacou em eficiência energética, particularmente com o modelo InceptionV3 (GPU, TensorFlow), consumindo apenas 0.195 J, o menor valor entre os flagships. Este resultado reforça que a sinergia entre hardware (como o chip Google Tensor) e software pode levar a uma maior eficiência energética, superando diferenças em especificações brutas.

O framework utilizado impacta diretamente a eficiência da inferência. O TensorFlow demonstrou um desempenho geral sólido, especialmente com aceleração via GPU em modelos mais complexos. No Samsung Galaxy S21 FE, por exemplo, o TensorFlow executou o modelo InceptionV3 em 56.2 ms na GPU, superando o PyTorch (59.8 ms) e o MindSpore (57.5 ms). O PyTorch mostrou-se bastante competitivo na CPU, superando o TensorFlow em diversos cenários, como no caso do InceptionV3 no S21 FE (89.7 ms vs. 154.1 ms). No entanto, seu desempenho relativo tende a diminuir em cenários com GPU. O MindSpore, por sua vez, apresentou resultados promissores e competitivos, posicionando-se frequentemente entre o TensorFlow e o PyTorch, validando seu foco em desempenho para dispositivos móveis.

A execução em GPU, quando comparada à CPU, reduziu significativamente o tempo de inferência, com ganhos de até 3.5 vezes, como observado no modelo InceptionV3 com TensorFlow no Moto G200 (64.9 ms na GPU vs. 229.0 ms na CPU). Modelos com maior grau de paralelismo, como o InceptionV3, foram os que mais se beneficiaram dessa aceleração. Energeticamente, a GPU também se mostrou mais vantajosa. Embora consuma mais potência instantânea, sua maior velocidade de processamento reduz o tempo total de operação e, consequentemente, o consumo total de energia. No S21 FE com InceptionV3 (TensorFlow), a execução na GPU consumiu 0.235 J — aproximadamente 58% menos que os 0.563 J consumidos na CPU. Isso evidencia que a aceleração por GPU é uma estratégia fundamental para otimizar tanto o desempenho quanto a autonomia da bateria.

Os resultados apresentados confirmam expectativas teóricas amplamente discutidas na literatura, ao mesmo tempo em que destacam nuances específicas observadas nos testes. A consistência entre os dados medidos pela HarpIA e os padrões esperados para diferentes modelos, dispositivos e frameworks reforça sua confiabilidade como ferramenta de benchmark. Assim, conclui-se que a HarpIA é capaz de fornecer medições precisas,

reproduzíveis e alinhadas com a realidade prática do desempenho em dispositivos móveis e embarcados, validando seu uso como plataforma de avaliação de inferência em aprendizado profundo.

## 6. Considerações Finais

Este trabalho abordou um desafio da IA em dispositivos móveis: escolher frameworks de forma informada em um ecossistema Android fragmentado. A diversidade de hardware e software torna a otimização de modelos complexa e demorada. Para isso, o estudo trouxe duas contribuições: análise comparativa dos frameworks mais usados e o desenvolvimento da HarpIA, ferramenta que viabiliza essa análise de forma ágil e eficiente.

A avaliação gerou dados sobre desempenho de TensorFlow, PyTorch e, inéditamente, MindSpore no Android, considerando tempo de inferência e consumo energético. O diferencial da HarpIA é desacoplar o ciclo de testes do ciclo de compilação, eliminando a recompilação do APK a cada experimento, reduzindo esforço e tempo e otimizando o fluxo em MLOps.

A ferramenta será disponibilizada publicamente após integrar suporte ao ONNX Runtime. Não há suporte para novos frameworks, pois as máquinas virtuais são compiladas junto ao APK.

Em síntese, o trabalho contribui para IA embarcada ao fornecer dados acionáveis e solução prática para avaliar e otimizar modelos Android. Futuramente, planeja-se expandir para arquiteturas emergentes como Transformers e incluir métricas adicionais, como uso de memória e estratégias de quantização, aprofundando a compreensão dos trade-offs no desenvolvimento de IA móvel.

## Agradecimentos

Este trabalho é parte do Projeto de PD&I SWPERFI (Técnicas de IA para a Análise, Teste e Otimização de Desempenho de Software), parceria entre UFAM e MOTOROLA, com membros do grupo de pesquisa ALGOX (Algoritmos, Otimização e Complexidade Computacional), do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil). Também recebe apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Código de Financiamento 001, sendo parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Amazonas – FAPEAM – por meio do projeto POSGRAD. Esta pesquisa está sendo, ainda, financiada em parte com recursos do Projeto EVAAT-GCN (FINEP 01.24.0661 Ref 3311/24).

## Referências

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: A system for large-scale machine learning.

Almeida, M., Laskaridis, S., Mehrotra, A., Dudziak, L., Leontiadis, I., and Lane, N. D. (2021). Smart at what cost? characterising mobile deep neural networks in the wild.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. (2019). Searching for mobilenetv3.

Hu, H., Huang, Y., Chen, Q., Zhuo, T. Y., and Chen, C. (2023). A first look at on-device models in ios apps. *ACM Trans. Softw. Eng. Methodol.*, 33(1).

Huawei (2022). *Huawei MindSpore AI Development Framework*, page 137–162. Springer Nature Singapore.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size.

Ignatov, A., Timofte, R., Chou, W., Wang, K., Wu, M., Hartley, T., and Van Gool, L. (2019). *AI Benchmark: Running Deep Neural Networks on Android Smartphones*, page 288–314. Springer International Publishing.

Luo, C., He, X., Zhan, J., Wang, L., Gao, W., and Dai, J. (2020). Comparison and benchmarking of ai models and frameworks on mobile devices.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library.

Reddi, V. J., Kanter, D., Mattson, P., Duke, J., Nguyen, T., Chukka, R., Shiring, K., Tan, K.-S., Charlebois, M., Chou, W., El-Khamy, M., Hong, J., John, T. S., Trinh, C., Buch, M., Mazumder, M., Markovic, R., Atta, T., Cakir, F., Charkhabi, M., Chen, X., Chiang, C.-M., Dexter, D., Heo, T., Schmuelling, G., Shabani, M., and Zika, D. (2020). Mlperf mobile inference benchmark.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842.

Tan, M., Chen, B., Pang, R., Vasudevan, V., and Le, Q. V. (2018). Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626.

Xu, M., Liu, J., Liu, Y., Lin, F. X., Liu, Y., and Liu, X. (2019). A first look at deep learning apps on smartphones. In *The World Wide Web Conference*, WWW '19. ACM.

Zhang, Q., Li, X., Che, X., Ma, X., Zhou, A., Xu, M., Wang, S., Ma, Y., and Liu, X. (2022). A comprehensive benchmark of deep learning libraries on mobile devices. In *Proceedings of the ACM Web Conference 2022*, WWW '22. ACM.