

# Estudo de viabilidade do uso de Modelos Grandes de Linguagem em um teclado Android

Makley Tibola Trichez<sup>1</sup>, Márcio Nicolau<sup>2</sup>

<sup>1</sup>ATITUS Educação  
Passo Fundo - RS - Brasil

<sup>2</sup>Curso de Ciência da Computação  
tibolamakley1@gmail.com, marcio.nicolau@atitus.edu.br

**Abstract.** *This study's main objective was to analyze the use of mobile devices, the relevance of artificial intelligence (AI) after ChatGPT and to evaluate the feasibility of large language models on virtual keyboards in the Android operating system. Artificial Intelligence contributes to the efficiency of word correction and prediction, improving typing when applied to spelling correction, using natural language processing algorithms. The proposal was to develop a virtual keyboard application for Android. The proposed application look for alternatives based on generative AI for word correction and suggestion, while similar applications, such as Microsoft SwiftKey and Gboard, use machine learning algorithms to predict words. The careful choice of technologies such as Python, TensorFlow, PyTorch and Kotlin were fundamental to developing this proposal. The methodology has used Kotlin to develop the application, and the integration of LLMs was presented as essential for text generation, exploring the contextualization capacity on Android devices without the need for an internet connection. Given the results presented, the application is successful, but the viability of large language models uses on virtual keyboards in the Android operating system is not fully operational at this moment.*

**Resumo.** *Este estudo tem como principal objetivo analisar o uso de dispositivos móveis, a relevância da inteligência artificial (IA) após o ChatGPT e avaliar a viabilidade de modelos grandes de linguagem em teclados virtuais no sistema operacional Android. A inteligência artificial contribui para a eficiência da correção e da predição de palavras, melhorando a digitação quando aplicada na correção ortográfica, utilizando algoritmos de processamento de linguagem natural. Frente a isso, firmou-se a proposta do desenvolvimento de um aplicativo de teclado virtual para Android. O aplicativo proposto busca alternativas baseadas em IA generativa para correção e sugestão de palavras, enquanto os similares, como Microsoft SwiftKey e o Gboard, utilizam algoritmos de aprendizado de máquina para prever palavras. A escolha cuidadosa de tecnologias como Python, TensorFlow, PyTorch e Kotlin foram fundamentais para desenvolver a proposta. A metodologia utilizou Kotlin para o desenvolvimento do aplicativo, e a integração de LLMs foi apresentado como essencial para a geração de texto, explorando a capacidade de contextualização em dispositivos Android sem a necessidade de conexão com a internet. Diante*

*dos resultados apresentados, verifica-se o sucesso do funcionamento da aplicação, mas falha da viabilidade do uso de modelos grandes de linguagem em teclados virtuais no sistema operacional Android.*

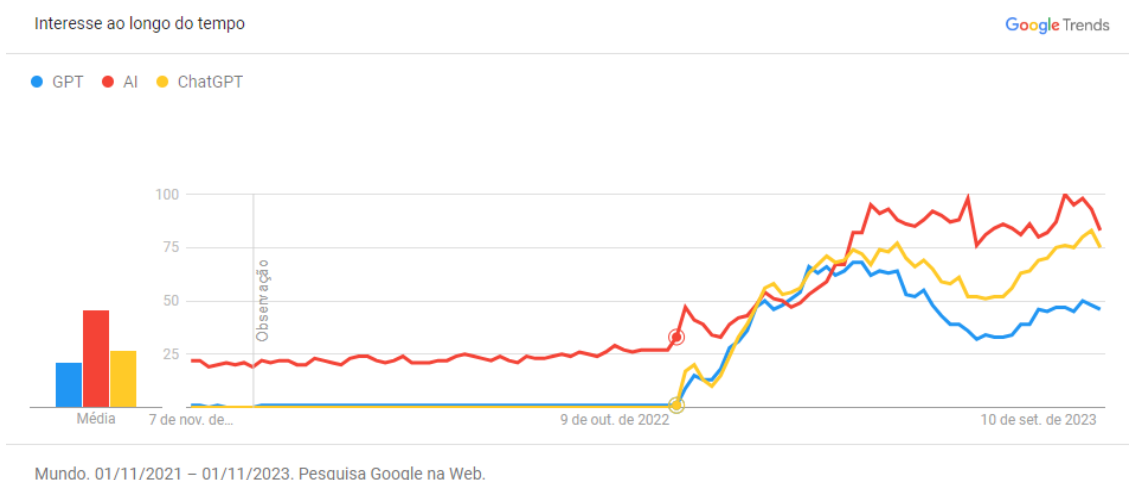
## 1. Introdução

Houve mudança na quantidade de pessoas que utilizam celular para uso pessoal, quando comparado antes e depois dos lançamentos do iPhone e do Android no período de 2007 e 2008. Como forma de comparação, segundo dados da PNAD - Pesquisa Nacional por Amostra de Domicílios (AGÊNCIA IBGE NOTÍCIAS, 2013), em 2005, 55,7 milhões de habitantes de 10 anos ou mais de idade (36,6% do total nessa faixa etária) tinham telefone móvel pessoal, em 2011 esse número aumentou para 115,4 milhões de cidadãos (69,1% referente ao grupo etário), o que reforça a transformação no cenário da posse de celulares pessoais, no Brasil.

O uso dos dispositivos móveis, especialmente dos *smartphones*, tem revolucionado a forma como as pessoas interagem socialmente. Com a ampla gama de recursos e métodos oferecidos por esses dispositivos, a comunicação se tornou mais acessível e instantânea. As pessoas podem visualizar suas correspondências a qualquer momento, o que impulsionou uma transformação significativa na maneira como indivíduos interagem.

Segundo a Agência IBGE Notícias (2018), dos 116 milhões de brasileiros que acessaram a Internet em 2016, 92,4% utilizaram aplicativos de troca de mensagens com o objetivo de se comunicar. Além disso, o acesso via celular foi o meio mais utilizado para acessar a rede, com 94,6% dos internautas utilizando o aparelho portátil visando navegar na internet.

Da mesma forma que os *smartphones* revolucionaram o mundo, a inteligência artificial (IA) conquistou significativa notoriedade com o anúncio do ChatGPT, em 30 de novembro de 2022<sup>1</sup>. Conforme ilustrado na figura 1, observa-se um aumento nas pesquisas relacionadas ao assunto na *web*, após o lançamento deste grande modelo de linguagem.



<sup>1</sup> Disponível em: <<https://openai.com/blog/chatgpt>>. Acesso em: 12 nov. 2023.

**Figura 1. Representação do aumento das pesquisas na *web* por assuntos envolvendo inteligência artificial no mundo. Fonte: Google Trends<sup>2</sup>.**

Diante da pesquisa de Chui et al. (2023), é visível que a IA Generativa tornou-se mais acessível para o público em geral, trazendo possibilidades maiores para desenvolvedores e pesquisadores, os quais implementam diferentes metodologias, que influenciam na aplicabilidade de sistemas de inteligência artificial em diferentes cenários. *Ibidem* ainda traz dados mostrando que as empresas passaram a priorizar o investimento nessa tecnologia devido ao seu potencial retorno financeiro<sup>3</sup>, reconhecendo uma oportunidade transformadora para os seus produtos.

Em virtude do exposto, este artigo visa abranger definições e conceitos relacionados ao seguinte problema de pesquisa: É viável utilizar modelos grandes de linguagem (LLM) - para sugestão de palavras, por exemplo - em teclados virtuais no sistema operacional Android de maneira funcional e eficaz? Para isso, torna-se necessário desenvolver um aplicativo de teclado para dispositivos móveis, e avaliar as implementações de LLMs disponíveis, para que se possa checar a viabilidade da aplicação proposta.

## **2. Referencial Teórico**

A utilização de dispositivos móveis, como *smartphones*, tornou-se ubíqua diante das informações anteriores, transformando a forma como a comunicação é realizada. Com a crescente dependência desses dispositivos para realizar tarefas diárias, como enviar mensagens, navegar na internet ou produzir conteúdo, algumas funções tornaram-se essenciais nos teclados virtuais. Nesse contexto, a correção e predição de palavras são recursos essenciais para melhorar a velocidade e a precisão da digitação em dispositivos móveis.

### **2.1. Autocorreção dos teclados virtuais**

Atualmente, os corretores de gramática são uma das ferramentas mais comuns nos teclados. Eles ajudam a sinalizar palavras que podem estar erradas e, às vezes, substituí-las automaticamente. De acordo com Beaufays e Riley (2017), as pessoas têm uma velocidade de digitação 35% mais lenta em dispositivos móveis do que em computadores. Isso se deve principalmente à limitação do teclado virtual, que muitas vezes apresenta dificuldades na digitação devido ao tamanho reduzido das teclas e à correção automática inadequada, resultando em erros frequentes e retrabalho na revisão.

No entanto, com o avanço da tecnologia da inteligência artificial, os corretores de gramática em teclados virtuais têm se mostrado cada vez mais eficientes. Agora, eles são capazes de auxiliar na correção instantânea de erros e tornar a digitação em

---

<sup>2</sup> Disponível

em: <<https://trends.google.com.br/trends/explore?date=2021-11-01%202023-11-01&q=GPT,AI,ChatGPT&hl=pt>>. Acesso em: 12 nov. 2023

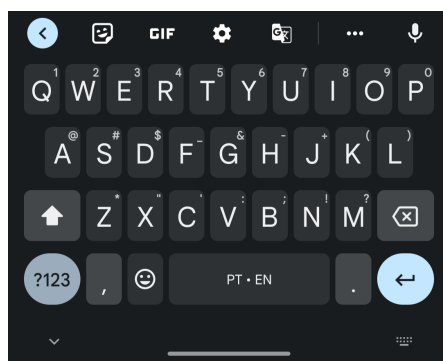
<sup>3</sup> “[...] inteligência artificial gerou aumento de receita em todas as funções de negócio que a utilizam. Além disso, mais de dois terços esperam que suas organizações intensifiquem os investimentos em IA nos próximos três anos.”(CHUI et al., 2023)

dispositivos móveis mais rápida e precisa. O Gboard<sup>4</sup>, por exemplo, utiliza a aprendizagem de máquina para prever e corrigir erros de digitação, além de apresentar sugestões de palavras de acordo com o contexto da frase.

Além disso, com a crescente demanda por dispositivos móveis, a expectativa é que os teclados virtuais se tornem ainda mais sofisticados, oferecendo uma velocidade de digitação cada vez mais próxima àquela dos computadores convencionais. Isso pode incluir recursos como digitação por voz, reconhecimento de gestos e até mesmo a integração com assistentes virtuais, como o Google Assistente e a Siri.

## 2.2. Teclados com predição de palavras

No mercado de aplicativos para celulares, o Gboard e o Microsoft SwiftKey<sup>5</sup> são dois dos teclados virtuais mais populares. Em relação às suas funcionalidades, ambas possuem funções muito similares, sendo as mais famosas a predição de palavras, correção ortográfica e a escrita por voz. A principal diferença entre eles é a aparência, o Gboard (Figura 2) possui uma interface mais simples, enquanto o Microsoft SwiftKey (Figura 3) valoriza a personalização do teclado. Ambos os teclados estão disponíveis para dispositivos Android e iOS<sup>6</sup>.



**Figura 2. Captura de tela do teclado Gboard. Fonte: Elaboração Própria.**

<sup>4</sup>

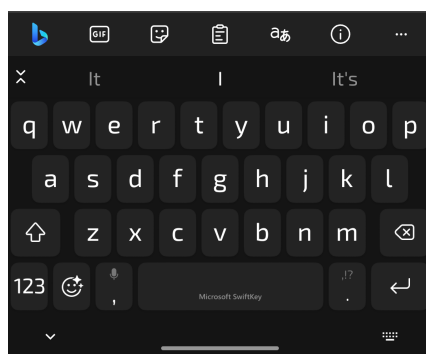
Disponível

em:

<[https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=pt\\_BR&gl=US&pli=1](https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=pt_BR&gl=US&pli=1)>. Acesso em: 13 nov. 2023

<sup>5</sup> Disponível em: <[https://www.microsoft.com/pt-br/swiftkey?activetab=pivot\\_1%3aprimariy2](https://www.microsoft.com/pt-br/swiftkey?activetab=pivot_1%3aprimariy2)>. Acesso em: 13 nov. 2023

<sup>6</sup> De acordo com Nocera et al. (2018), o Android é o sistema operacional mais amplamente utilizado globalmente. Inicialmente desenvolvido pela Android Inc. e adquirido pela Google em julho de 2005, o Android é notável por sua linguagem operacional Java e sua natureza de plataforma de código aberto. Por outro lado, o iOS, o segundo sistema operacional mais conhecido atualmente, foi concebido pela Apple em 2007 e é exclusivo para dispositivos da empresa, como o iPhone. Nocera et al. (2018) destacam que o iOS é distintivo pela qualidade da experiência do usuário em sua interface e pela sua interdependência com produtos específicos da Apple.



**Figura 3. Captura de tela do teclado Microsoft SwiftKey. Fonte: Elaboração Própria.**

A predição de palavras é uma técnica que na maioria das vezes utiliza algoritmos de aprendizado de máquina a fim de avaliar os padrões de digitação do usuário e fornecer algumas sugestões de palavras que seriam mais prováveis de serem digitadas em seguida (HUNNICUTT et al., 1997). Tal técnica ajuda a acelerar a entrada de texto, principalmente para pessoas que possuem dificuldades ou baixa velocidade de digitação, reduzindo a quantidade de tempo necessário ao usuário para digitar cada palavra. Em vez de digitar todas as letras de um termo, o usuário pode simplesmente selecionar uma sugestão que corresponda à expressão desejada.

Além disso, essa funcionalidade tem sido uma ferramenta valiosa para dispositivos móveis, onde a digitação em teclados virtuais é geralmente mais lenta e menos precisa do que em teclados físicos - como os disponíveis em computadores. Isso é especialmente verdadeiro para pessoas com restrições motoras, as quais podem apresentar dificuldades para digitar em teclados menores. A predição de palavras ajuda a reduzir erros ortográficos comuns e permitir uma digitação mais rápida e precisa em dispositivos móveis (HUNNICUTT et al., 1997).

A precisão de ambos teclados é altamente eficaz para prever palavras, mas as preferências podem variar dependendo das necessidades do usuário. Por exemplo, usuários que preferem uma interface mais integrada ao sistema do dispositivo móvel podem optar pelo Gboard, enquanto os que preferem personalizar e ter o controle total sobre a aparência do teclado podem escolher o Microsoft SwiftKey.

### **2.3. Algoritmos e Inteligência Artificial**

O aumento no uso da Inteligência Artificial surge da demanda por soluções mais eficientes e rápidas para as tarefas complexas enfrentadas no dia a dia da sociedade. Esta tecnologia pode oferecer uma variedade de benefícios em áreas como: medicina, engenharia, previsão do tempo e sistemas de segurança avançados (VERMA, 2018). Além disso, a IA pode auxiliar na análise de grandes volumes de dados e na previsão de tendências, permitindo que as empresas tomem decisões mais assertivas. De acordo com Hinton (2016), o aprendizado profundo é uma técnica que pode funcionar melhor do que as técnicas existentes em problemas antigos que demandam previsão e que possuem grandes quantidades de dados disponíveis.

Ao utilizar a predição de palavras, é possível obter ganhos de precisão e velocidade para escrever no celular, mas somente com boas métricas de previsão é possível chegar a um bom resultado. Dentre os algoritmos que preveem palavras com inteligência artificial, podemos citar algoritmos baseados em Cadeias de Markov, que

utiliza um modelo probabilístico para prever o próximo termo com base no estado atual, que pode ser influenciado pelo histórico de palavras anteriores (MALTBY et al., [s.d.]). Outro algoritmo comum é o N-gram, que é baseado na probabilidade de ocorrência de uma sequência de itens. Além disso, a Google utiliza redes neurais artificiais para aprimorar a predição de palavras em seus teclados, como visto no Gboard, que é capaz de prever termos por meio da utilização de modelos de aprendizado de máquina treinados com extensos conjuntos de dados (BEAUFAYS e RILEY, 2017). Esses modelos são continuamente ajustados para melhorar a precisão da previsão do texto e garantir uma experiência de digitação mais rápida e eficiente para os usuários.

Além da predição de palavras, a IA também pode ser aplicada na correção ortográfica, sendo um recurso muito útil para a produção de textos gramaticalmente aceitáveis. O corretor de texto é capaz de identificar e substituir automaticamente erros de digitação, ortografia e gramática, por meio da análise do texto e comparação com um dicionário de palavras ou regras gramaticais pré-definidas.

Existem diversos algoritmos de correção ortográfica como o algoritmo de distância de Levenshtein, que calcula a distância entre os termos e sugere correções com base nas palavras mais próximas no dicionário (LEVENSCHTEIN, 1966), e existem algoritmos baseados em modelos de redes neurais recorrentes (RNNs), que são utilizados com dados sequenciais, como texto e áudio. Levando em conta as palavras de Shashank Singh e Shailendra Singh (2020), existem três etapas para fazer uma possível correção ortográfica, pré-processamento preciso, a verificação ortográfica e a geração de uma lista com sugestões de correções.

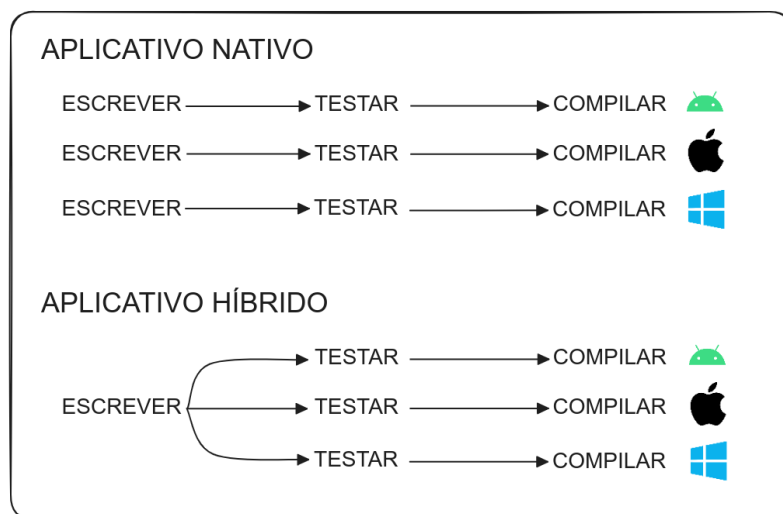
Dessa forma, tanto a predição de palavras quanto à correção ortográfica com inteligência artificial são ferramentas úteis e cada vez mais presentes no uso de celulares, permitindo uma comunicação mais eficiente e com menos erros.

## **2.4. Desenvolvimento Android**

Atualmente, é expressiva a utilização de *smartphones*, como aponta a Agência IBGE Notícias (2018), em que 94,6% dos acessos à internet ocorreram através de dispositivos móveis, estando estes totalmente integrados no cotidiano da maior parte populacional brasileira. “Ademais, com a crescente redução de preço desses aparelhos, é possível que pessoas com menor poder aquisitivo também tenham acesso a essa tecnologia.” (SANTOS, 2018, p. 12).

O Android é um Sistema Operacional (SO) para dispositivos móveis e foi desenvolvido pela Android Inc, e posteriormente adquirido pela Google em 2005 (KRAJCI e CUMMINGS, 2013). Desde então, o sistema é desenvolvido continuamente pela empresa e também pela comunidade de desenvolvedores Android. Por se tratar de um sistema operacional que possui a maior parte de seu código fonte aberto, significa que está disponível para ser modificado e distribuído livremente por qualquer pessoa interessada.

O desenvolvimento Android pode ser realizado em duas principais linguagens de programação de forma nativa, Java e Kotlin (BOSE; ADITI; MUKHERJEE, 2018), também é possível através de alguns *frameworks* para desenvolvimento híbrido como Flutter, que integra a linguagem Dart, e React Native, que faz uso do Javascript.



**Figura 4. Diferenças entre as etapas de desenvolvimento nativo e híbrido. Fonte: Elaboração Própria.**

Quando comparado o desenvolvimento nativo e híbrido, a vantagem tende ao uso de *frameworks*, por sua possibilidade de desenvolver para diversas plataformas, como demonstrado na figura 4, reduzindo o custo de desenvolvimento, porém, quando se aborda a criação de um aplicativo que necessite de métodos de entrada mais complexos e/ou especializados, é mais simples utilizar a Interface de Programação de Aplicativos (API) nativa do Android com Kotlin, por tornar a construção mais rápida.

### 3. Trabalhos Relacionados

Para entender melhor o contexto ao qual este artigo se insere e, identificar lacunas que possam ser exploradas na pesquisa, é importante analisar trabalhos relacionados ao tema de teclados virtuais em *smartphones*. Nesta seção, serão apresentados três trabalhos que abordam objetivos específicos semelhantes a este tema, que visam implementar grandes modelos de linguagem em um teclado virtual para celulares. Embora os trabalhos a seguir tenham explorado algumas soluções que possuam uso de inteligência artificial no teclado virtual, o trabalho proposto busca alternativas baseadas em IA generativa para implementar as funcionalidades de correção e sugestão de palavras.

#### 3.1. Algoritmo de predição de palavras

O Microsoft SwiftKey é um aplicativo de teclado virtual que utiliza um sistema de aprendizado de máquina (ML) para prever as próximas palavras que o usuário irá digitar, levando em consideração o contexto da mensagem e o histórico de digitação. Embora seja preciso em prever palavras, o aplicativo ainda apresenta alguns quesitos que poderiam ser melhorados. Por exemplo, ao abrir o teclado, são apresentadas três sugestões de palavras padrões, porém sugestões relevantes somente são apresentadas após começar a digitar, levando em consideração o contexto do texto.

Em seu artigo, Hard et al. (2018) apresenta o algoritmo de aprendizado federado utilizado no Gboard, o teclado padrão de fábrica em diversos celulares Android. O aprendizado federado permite que um grupo possa treinar e aprimorar um modelo de ML de forma colaborativa (GOOGLE, 2022). Diferentemente do concorrente Microsoft SwiftKey, o Gboard não apresenta sugestões iniciais.

Com o uso de LLM, a predição de palavras em aplicativos de teclado seria eficiente e contextualizada, devido à compreensão abrangente de grandes volumes de texto. Esses modelos capturam relações complexas entre palavras e geram predições mais precisas e relevantes. Além disso, podem prever frases completas e personalizar sugestões com base no contexto.

### 3.2. Correções de palavras

Diversos aplicativos de teclado virtual visam auxiliar o usuário a evitar erros de digitação e a escrever corretamente as palavras. Esta correção é realizada por meio de algoritmos que analisam o texto digitado e sugerem correções para as palavras que estão escritas incorretamente.

Existem diversos aplicativos de teclado virtual que utilizam esta funcionalidade de correção de palavras e ortografia, cada um com seus próprios algoritmos. A seguir, é apresentado uma lista de algumas aplicações que utilizam correção de palavras:

- **Gboard:** teclado virtual desenvolvido pela Google, que utiliza inteligência artificial para prever e corrigir palavras. A correção é sugerida, como uma substituição da palavra incorreta enquanto ocorre a digitação, contudo, caso o usuário não faça o ajuste oferecido no momento, ele deverá selecionar a palavra para poder ver novamente a correção e poder substituí-la;
- **Grammarly Keyboard:** teclado que oferece correções avançadas de gramática e ortografia, além de integração com o aplicativo de mesmo nome (Grammarly) para Windows, Mac e como extensão para o navegador, que oferece sugestões de melhoria para os erros gramaticais. No formato de correção de palavras oferecido pelo Grammarly Keyboard, é apresentado uma notificação de erros de escrita a serem corrigidas no canto superior esquerdo do teclado, permitindo que o usuário, a qualquer momento, possa fazer a correção ou adicionar ao dicionário as palavras que forem reconhecidas com erro ortográfico. Entretanto, não foi possível encontrar referências de modelos ou algoritmos que sejam utilizados para previsão ou correção de palavras.

A implementação de um LLM no ajuste de palavras em teclados virtuais apesar de trazer benefícios similares, como a capacidade de contextualizar o texto digitado, fornecendo sugestões de correção. Ao avaliar a viabilidade e eficácia do uso de LLM na correção de palavras, é possível determinar se essa abordagem pode melhorar a precisão e a qualidade das correções sugeridas.

## 4. Materiais e Métodos

A presente seção expõe os propósitos, a tecnologia e a metodologia que serão utilizados para a construção deste artigo, como também o desenvolvimento do aplicativo que viabiliza a realização deste estudo. Visando fornecer uma visão abrangente sobre o processo de criação e implementação das funcionalidades, a escolha cuidadosa dos materiais e métodos é fundamental para garantir a eficácia e a viabilidade para a elaboração do aplicativo.

Diversas tecnologias foram minuciosamente avaliadas e selecionadas com o objetivo de garantir um desenvolvimento eficaz e eficiente do aplicativo. As tecnologias escolhidas desempenham papéis cruciais em diferentes aspectos do processo de construção e funcionamento do aplicativo.



#### 4.1. Modelos Grandes de Linguagem (LLM)

Modelos Grandes de Linguagem, são modelos de inteligência artificial treinados em um vasto conjunto de dados, podendo ser generalistas ou ajustados para se adequar a casos de uso específicos, tornando-o especializado. Esses modelos utilizam arquiteturas baseadas em Transformers (VASWANI, 2017) para realizar o processamento de linguagem natural (NLP), o que lhes permite realizar funções como tradução, compreensão e complementação de texto, entre outras tarefas de linguagem (ELASTIC, 2023; GOOGLE, 2023).

Uma funcionalidade essencial dos LLMs é a sua capacidade de geração de texto, o que pode ser explorado para aplicação em predição e correção de palavras. Esses modelos têm a capacidade de gerar texto com qualidade, a partir da escolha de palavras que seriam mais indicadas para completar o contexto atual da escrita, com base em seleção probabilística, o que é especialmente útil na correção de erros gramaticais. Essa capacidade de geração de conteúdo textual confere uma importância significativa aos LLMs no contexto do desenvolvimento do aplicativo.

#### 4.2. Python

O Python é uma linguagem de programação de propósito geral, que disponibiliza uma ampla gama de recursos e funcionalidades para o desenvolvimento de aplicações. Para o seu ambiente, o gerenciador de pacotes pip desempenha um papel fundamental na instalação, gerenciamento e versionamento de dependências.

O pip simplifica o processo de instalação de pacotes, fornecendo um único comando para adicionar novas dependências ao ambiente de desenvolvimento. O Python desenvolve um papel importante na área de aprendizado de máquina, sendo uma ferramenta chave e amplamente adotado pela comunidade de desenvolvedores (SULTONOV, 2023, p. 30) para ajustes finos em LLMs, juntamente com a biblioteca Transformers e *frameworks* TensorFlow e PyTorch, as quais suas utilidades são denotadas a seguir:

- **Biblioteca Transformers:** Esta biblioteca foi criada pela Hugging Face e provê o estado da arte em relação ao aprendizado de máquina. Amplamente utilizada para processamento de linguagem natural, pois fornece suporte a interoperabilidade entre TensorFlow e PyTorch (TRANSFORMERS, 2023).
- **Framework TensorFlow:** Pacote para aprendizado de máquina e aprendizado profundo desenvolvido pelo Google. É amplamente utilizado em uma variedade de aplicações de inteligência artificial, incluindo visão computacional, processamento de linguagem natural e análise de dados. O TensorFlow permite a criação, treinamento e implantação eficaz de modelos de aprendizado de máquina e redes neurais. Ele oferece escalabilidade e suporte a hardware acelerado o que o torna uma ferramenta essencial para projetos de IA e pesquisa em aprendizado de máquina. Sua versão menor chamada de TensorFlow Lite é otimizada para a execução de modelos em dispositivos com recursos limitados de processamento e memória. Essa biblioteca oferece eficiência computacional e latência reduzida, permitindo que os modelos sejam executados sem acesso à internet. Ela suporta uma ampla variedade de tipos de modelos, incluindo redes neurais, e é amplamente adotado na comunidade de desenvolvedores para implementar IA em dispositivos móveis (TENSORFLOW, 2023).

- **Framework PyTorch:** É uma biblioteca de código aberto amplamente usada para desenvolver modelos de aprendizado profundo. Ele é flexível e popular entre pesquisadores e cientistas, sendo usado em pesquisas de ponta. Esta biblioteca é comumente empregada em tarefas de processamento de linguagem natural e visão computacional, com suporte a modelos pré-treinados. Além disso, é uma escolha frequente para experimentação rápida e desenvolvimento de algoritmos de aprendizado de máquina.

### 4.3. Kotlin

O aplicativo em questão procura englobar soluções de última geração para torná-lo inteligente e eficaz, desta forma, é instaurado o princípio de desenvolvimento nativo, capaz de proporcionar acesso a API do Android. Para tal ação, é usado a linguagem de programação Kotlin, criada pela empresa JetBrains, possibilitando o desenvolvimento de forma produtiva e segura para o sistema operacional Android (ANDROID, 2023).

Esta linguagem de programação supre a necessidade no quesito envolvido pela interface apresentada ao usuário, viabilizando a entrada de texto no dispositivo móvel e integração de bibliotecas de terceiros através do Gradle, o qual permite a instalação, gerenciamento e versionamento de dependências (GRADLE, 2023).

Como meio de tornar possível o uso e testes em dispositivos Android, o Gradle também desempenha um papel fundamental na construção do aplicativo Android. Ele automatiza o processo de compilação e geração de APK<sup>7</sup>. Isso significa que o Gradle é responsável por traduzir o código-fonte, recursos e configurações do projeto em um aplicativo Android executável, pronto para ser instalado e executado nos dispositivos (GRADLE, 2023).

Como abordagem metodológica, utiliza-se uma análise de viabilidade para avaliar a incorporação de um teclado virtual com LLMs em dispositivos Android, sem a necessidade de conexão com a internet, que possibilite a aplicação dos conceitos de aprendizado de máquina em uma solução prática para o usuário. O estudo incluiu a escolha de um LLM otimizado para uso em dispositivos móveis e a criação de um aplicativo em Kotlin. Isso permite a exploração, compreensão das complexidades, oportunidades associadas à integração desses componentes e a viabilidade do projeto.

## 5. Análise dos Resultados

Mediante os métodos apresentados, o presente capítulo expõe os meios empregados para a pesquisa poder ser aplicada, abrangendo todas as funcionalidades que a ferramenta proposta neste artigo detém.

Os testes foram conduzidos em dispositivos virtuais e físicos. O dispositivo virtual possuía uma configuração com heapSize<sup>8</sup> e RAM, respectivamente de 4 e 8 gigabytes, Android versão 13 e API 33, proporcionando um ambiente representativo para avaliação. Essa configuração se assemelha às características do Motorola Edge 30

---

<sup>7</sup>APK (Android Package Kit) é um arquivo que contém todos os elementos necessários para instalar e executar um aplicativo Android em dispositivos móveis. Ele inclui o código-fonte, recursos e outras informações necessárias para o seu funcionamento.

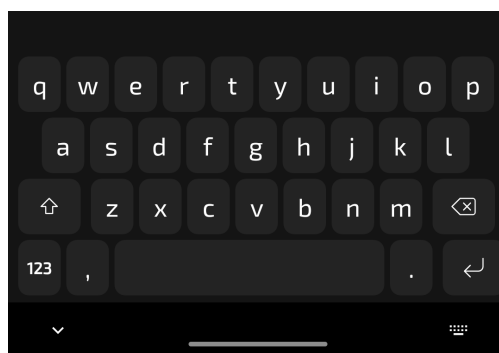
<sup>8</sup>No Android, *heapSize* refere-se à quantidade máxima de memória que pode ser alocada para o *heap* da Máquina Virtual Java (JVM) durante a execução de uma aplicação. O *heap* é a área de dados em tempo de execução na JVM onde os objetos são armazenados.

Fusion, um dispositivo físico utilizado nos testes, que opera com a mesma quantidade de RAM e versão do Android.

Durante a maior parte do desenvolvimento, as configurações mencionadas foram mantidas, com notificação imediata em caso de ressalvas durante a leitura. Para uma compreensão mais aprofundada sobre os utilitários e componentes contidos, o detalhamento do aplicativo será dividido em subitens.

### 5.1. Desenvolvimento do teclado

Inicialmente, para que fosse possível ter uma direção de desenvolvimento, foi necessário escolher uma base de design da interface do teclado, escolhendo como referência o Microsoft SwiftKey em sua versão padrão escura, direcionando a interface do usuário, para ter as ações de escrita e um espaçamento superior para as sugestões de palavras como apresentado na figura 5.



**Figura 5. Design do aplicativo proposto. Fonte: Elaboração Própria.**

Durante o estudo da documentação do *framework* Android, foi possível encontrar uma classe essencial para a continuidade do desenvolvimento. O *InputMethodService*<sup>9</sup>, classe que fornece funcionalidades essenciais para a criação de um serviço de método de entrada (IME) personalizado. Ele é responsável pela interação entre o usuário e o teclado virtual, gerenciando a entrada de texto, a exibição do teclado na tela e o processamento dos eventos de digitação.

Para tornar possível a utilização da aplicação em qualquer campo do sistema operacional, foi necessário a implementação de um componente *Service*<sup>10</sup>. Os serviços desempenham um papel crucial na execução de tarefas em segundo plano, sem interferir na interação direta com a interface do usuário principal. Devido o teclado ser um recurso do SO frequentemente utilizado, é essencial que a aplicação esteja sempre de prontidão para uso.

Ao ser integrado o componente de serviço na aplicação, é assegurada uma maior flexibilidade e eficiência na manipulação de entradas de texto e outras operações em segundo plano. Isso resulta em um uso mais fluido e responsivo da aplicação, independentemente do campo de entrada de texto.

<sup>9</sup>

Disponível

em:

<<https://developer.android.com/reference/kotlin/android/inputmethodservice/InputMethodService>>.

Acesso em: 11 nov. 2023

<sup>10</sup> Disponível em: <<https://developer.android.com/reference/android/app/Service>>. Acesso em: 11 nov. 2023

## 5.2. Escolha do LLM

Inicialmente, foi realizada a busca em repositórios de IA para escolher alguns modelos, para avaliar a viabilidade de execução em um ambiente Android. Os modelos selecionados incluem o BERT, LLama2 e GPT-2, os quais todos são pré-treinados com conteúdo em inglês. A descrição dos resultados obtidos por cada modelo estará destacada por tópicos de seus mesmos nomes.

### 5.2.1. Bidirectional Encoder Representations from Transformers (BERT)

Inicialmente, foram avaliados três modelos BERT para testes: bert-base-uncased, google/mobilebert-uncased e nlpueb/legal-bert-small-uncased, todos obtidos no site da *Hugging Face*. A escolha do modelo BERT foi motivada pelo seu pré-treinamento como um Modelo de Linguagem Mascarada (MLM), destacando-se por sua capacidade de preenchimento de lacunas em um texto marcado com tokens específicos. O processo de execução desses modelos foi uniforme, conforme descrito a seguir.

Os testes foram conduzidos com sucesso em um ambiente Python, utilizando a biblioteca Transformers. No entanto, visando a execução em dispositivos Android, foi imperativo converter os modelos para o formato compatível com o TensorFlow Lite. Essa conversão foi realizada através da biblioteca Optimum<sup>11</sup>. Após a conclusão bem-sucedida do processo de conversão, os modelos foram incorporados ao diretório *assets* do código-fonte. Contudo, na fase de utilização dos modelos, foram observados resultados inesperados, exigindo uma análise mais aprofundada (Figura 6):

**Figura 6. Erro apresentado ao utilizar modelos BERT em ambiente Android. Fonte: Elaboração Própria.**

Durante o estudo mais aprofundado, sobre a arquitetura de Transformer, Alammar (2019), apresentou algumas informações sobre dois principais blocos, *Encoder* e *Decoder*, os quais permitiu uma análise mais detalhada do uso de modelos de linguagem generativa. Para uma compreensão mais detalhada sobre suas características e importância, será abordado individualmente cada um dos blocos:

- **Encoder:** é um componente da estrutura Transformer encarregada por aprender representações contextuais de palavras em uma sequência de texto. O *Encoder* utiliza a camada de auto atenção (*self attention*) para capturar as relações entre todas as palavras da sequência, permitindo uma visão bidirecional do contexto. Essa abordagem permite que seja compreendido melhor o significado das palavras em um contexto mais amplo, facilitando tarefas de processamento de linguagem natural, como classificação de texto e reconhecimento de entidades.

---

<sup>11</sup> Optimum é uma extensão da biblioteca Transformers, que tem como objetivo fornecer um conjunto de ferramentas de otimização de desempenho. Essas funcionalidades são projetadas para treinar e executar modelos de forma eficiente em hardware específico, buscando extrair o máximo de desempenho possível. Mais informações disponíveis em: <<https://huggingface.co/docs/optimum/index>>. Acesso em: 16 nov. 2023

- **Decoder:** é uma parte da arquitetura Transformer que é responsável pela geração autorregressiva de texto, palavra por palavra, levando em consideração o contexto à esquerda. O *Decoder* utiliza a camada de auto atenção com máscara para impedir que a posição atual acesse informações futuras durante a geração de texto. Essa abordagem permite que seja gerado texto coerente e fluente, baseado no contexto anterior.

Com base nas informações analisadas do trabalho de Alammr (2019), podemos concluir que o modelo abordado neste tópico, embora possua a arquitetura Transformer, difere dos LLM utilizados para fins generativos. O BERT utiliza apenas o bloco Encoder para aprender representações contextuais de palavras em uma sequência de texto. No entanto, devido à ausência do bloco *Decoder* responsável pela geração autorregressiva de texto, ele não tem a capacidade de gerar palavras subsequentes. Portanto, esse aspecto foi decisivo para a exclusão dos modelos BERT do escopo de trabalho.

### 5.2.2. LLama2

O LLama2 é uma IA Generativa de código aberto, publicado pela Meta, sua versão mais compacta possui 12GB. A sua implantação bem-sucedida, demonstrou a viabilidade de integrar este modelo a computadores domésticos. Por outro lado, a tentativa de implementação do LLM em smartphones foi caracterizada por diversos obstáculos.

Mesmo após o encontro de projetos que supostamente permitiam o seu uso em dispositivos móveis, foram encontrados erros de *OutOfMemory*. O exposto sugere que a memória disponível do smartphone não é suficiente para suportar o tamanho substancial do modelo. Uma possível alternativa seria aumentar o tamanho da memória do dispositivo simulado no Android Studio, entretanto, mesmo com o aumento da memória RAM para 16 GB e elevação do *heapSize* do dispositivo ao máximo, não foi possível executar o modelo.

Para atenuar essas limitações, a quantização do LLama2 para 8 bits foi considerada. A quantização é um processo no qual os parâmetros do modelo são representados com menos bits, resultando em um modelo menor, com um pequeno comprometimento na precisão. Entretanto, durante a busca por estratégias de quantização, surgiu o modelo GPT-2. Apesar de não ser um LLM tão grande quanto o LLama2, apresentou requisitos de processamento e tamanho mais adequados para a implementação em smartphones. Assim, o GPT-2 tornou-se a preferência para o estudo.

### 5.2.3. GPT-2

O GPT-2, desenvolvido pela OpenAI, encontra-se publicamente disponível em seu repositório. Uma ferramenta que incorporou o uso desse modelo foi o KerasNLP, uma biblioteca dedicada ao processamento de linguagem natural que opera em conjunto com o TensorFlow. Os testes iniciais executados em um computador doméstico transcorreram sem contratempos. Na busca por informações sobre o modelo, deparou-se com a documentação do TensorFlow, a qual inclui um exemplo<sup>12</sup> de integração do LLM em um ambiente Android, visando a autocompletar textos. A aplicação prática do

---

<sup>12</sup> Disponível em: <[https://www.tensorflow.org/lite/examples/auto\\_complete/overview?hl=pt-br](https://www.tensorflow.org/lite/examples/auto_complete/overview?hl=pt-br)>. Acesso em: 16 out. 2023

exemplo descrito na documentação permitiu avançar para a próxima etapa do projeto, que envolvia ajustar o tamanho e formato da saída de dados do modelo.

Para viabilizar a exibição de sugestões no cabeçalho do teclado, é imperativo obter com precisão as três palavras mais relevantes. Com esse objetivo em mente, tornou-se essencial examinar a documentação do KerasNLP em busca de maneiras de ajustar a quantidade de sequências de palavras a serem geradas. Além disso, buscou-se modificar o parâmetro *top\_k*, que é responsável por determinar a quantidade de resultados com as melhores acurácias.

Durante essa análise, identificou-se um parâmetro chamado *max\_length*, o qual poderia ser empregado para definir o tamanho do retorno do texto gerado. Para realizar essa definição, é necessário fazer a soma da quantidade de palavras do texto de entrada e somar com a quantidade de palavras a serem retornadas na saída. Entretanto, não foi possível encontrar uma forma de obter o tamanho do texto de entrada de forma dinâmica, impedindo o progresso nesse objetivo. Além da impossibilidade de especificar o *max\_length* de forma dinâmica, não foi encontrada na documentação do KerasNLP uma maneira de ajustar a geração dos três textos com maior precisão.

Diante da impossibilidade de prosseguir com o uso do KerasNLP, foi abordado o uso da biblioteca Transformers, o qual também possui o modelo GPT-2 disponibilizado pela Hugging Face. Apesar de existirem modelos convertidos para TensorFlow Lite no próprio repositório, ainda assim foi necessário fazer os ajustes que foram propostos anteriormente.

Apesar do tamanho do modelo não se aproximar do LLama2, como meio de evitar o erro de *OutOfMemory*, é carregado somente a parte do modelo que é essencial para sua execução como é demonstrado na figura 7.

```
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')
```

**Figura 7. Carregamento do modelo gpt2. Fonte: Elaboração Própria.**

```
outputs = model(
    # Entrada de dados
)

logits = outputs.logits
logits = logits[0,-1]

best_logits, best_indices = logits.topk(3)

best_words = [tokenizer.decode([idx.item()]) for idx in best_indices]
```

**Figura 8. Obtendo os melhores índices dos tokens gerados pelo LLM. Fonte: Elaboração Própria.**

Com o uso da biblioteca Transformers, se tornou possível indicar o *top\_k*, permitindo obter o retorno de três índices dos *tokens* gerados com a maior precisão, para isso foi necessário obter o *logits* que possui todos os *tokens* gerados pelo modelo, e então utilizar o *tokenizer* para mapear as palavras representadas pelos *token*. O algoritmo é demonstrado na figura 8. Em sequência o modelo é convertido para TensorFlow Lite, possibilitando executá-lo em um dispositivo Android.

### 5.3. Integração do LLM com o teclado virtual

Considerado como núcleo da aplicação, a integração permite o uso do LLM durante a digitação, sendo realizado a apresentação das sugestões de palavras. A partir dos resultados é possível escolher um dos termos o qual pode realizar a ação de adição ou substituição de uma palavra, dependendo da posição do cursor no campo de texto.

Para utilizar o modelo dentro da aplicação Android, é necessário importar o LLM no diretório *assets*, para que assim seja possível carregar o modelo durante a execução do aplicativo. Para gerenciar o processo do modelo, foi criada uma classe chamada *LLMNextWord*, que possui dois métodos principais, um responsável pela inicialização do modelo (Figura 9), e o segundo possui a obrigação de gerar e retornar as sugestões (Figura 10).

```
private suspend fun loadTFModel(): Interpreter = withContext(Dispatchers.IO) { this: CoroutineScope
    val assetFileDescriptor = getApplication<Application>().assets.openFd(MODEL_PATH)
    assetFileDescriptor.use { it: AssetFileDescriptor
        val fileChannel = FileInputStream(assetFileDescriptor.fileDescriptor).channel
        val modelBuffer =
            fileChannel.map(FileChannel.MapMode.READ_ONLY, it.startOffset, it.declaredLength)

        val opts = Interpreter.Options()
        opts.numThreads = NUM_LITE_THREADS
        return@use Interpreter(modelBuffer, opts)
    } ^withContext
}
```

Figura 9. Método de inicialização do LLM. Fonte: Elaboração Própria.

```
fun promptToken(prompt: String): List<String> {
    var nextToken: List<String> = listOf()
    nextTokenJob = viewModelScope.launch { this: CoroutineScope
        initJob.join()
        nextTokenJob?.cancelAndJoin()
        nextToken = getNextWord(prompt)
    }
    return nextToken
}

private fun getNextWord(startPrompt: String, quantityOfTokens: Int = 3): List<String>{
    val tokens = tokenizer.encode(startPrompt)

    // calculate the number of inputted tokens
    val maxTokens = tokens.takeLast(SEQUENCE_LENGTH).toIntArray()
    // calculate the number of tokens needed to fill the input sequence model
    val paddedTokens = maxTokens + IntArray(SEQUENCE_LENGTH - maxTokens.size)
    val inputIds = Array(1) { paddedTokens }

    val predictions: PredictionsType =
        Array(1) { Array(SEQUENCE_LENGTH) { FloatArray(VOCAB_SIZE) } }
    val modelOutputs = mutableMapOf<Int, Any>()

    tflite.runForMultipleInputsOutputs(arrayOf(inputIds), modelOutputs)
    val outputLogits = predictions[0][maxTokens.size - 1]

    val nextToken: List<Int> = getNextTokens(outputLogits, quantityOfTokens)
    val decodedTokens = tokenizer.decode(nextToken)

    return decodedTokens.split(...delimiters: " ")
}
```

**Figura 10. Método responsável por executar o modelo e obter as respostas. Fonte: Elaboração Própria.**

Os resultados são retornados em uma *string* separada por espaço, sendo necessário realizar a divisão, transformando em uma lista. Para utilizar o modelo, é necessário fazer a inicialização juntamente da interface através do *onCreateInputView* (Figura 11.a), que é requisitado pelo sistema quando for necessário. Essa inicialização é importante para que o modelo esteja disponível para uso de forma imediata à escrita no teclado. Além disso, para liberar a memória ocupada pelo modelo durante o uso do celular, é implementado o método *onDestroy* (Figura 11.b), que é executado pelo SO, quando notificado que o serviço não está sendo mais utilizado.

```
override fun onCreateInputView(): View {  
    llm = LLMNextWord(application)  
  
    keyboardView = layoutInflater.inflate(  
        R.layout.keyboard,  
        root: null  
    ) as KeyboardView?  
  
    keyboard = Keyboard( context: this, R.xml.qwerty)  
    keyboardView!!.keyboard = keyboard  
    keyboardView!!.setOnKeyboardActionListener(this)  
    return keyboardView!!  
}  
  
override fun onDestroy() {  
    super.onDestroy()  
    llm.onCleared()  
}
```

**Figura 11. (a) Método de inicialização (b) Método de destruição. Fonte: Elaboração Própria.**

Para gerar palavras, é fundamental que o texto tenha um tamanho mínimo de uma palavra, para que o LLM possa ter um conteúdo base de contexto. O método *updateSuggestions* é então implementado com o objetivo de obter o texto apropriado para a geração das palavras, assegurando que as respostas sugeridas estejam sempre alinhadas com o contexto mais recente.

Para garantir a consistência contextual, o método *updateSuggestions* é desenvolvido para lidar com diferentes posições do cursor (Figura 12). Como o GPT-2 é um modelo *decoder* e não tem a capacidade de sugerir palavras no meio do texto, a lógica é ajustada para oferecer sugestões com base no contexto à esquerda do cursor. Se o caractere imediatamente à esquerda do cursor não for um espaço, a cadeia de caracteres à esquerda é removida até o primeiro espaço em branco. Isso permite sugerir uma palavra para substituir a atual na posição do cursor.



```

fun updateSuggestions() {
    val extractedText = currentInputConnection.getExtractedText(ExtractedTextRequest(), flags: 0);

    val cursorPosition = extractedText.selectionStart

    if (cursorPosition != 0 && extractedText.text[cursorPosition - 1].isLetterOrDigit()) {
        // get the last word
        val lastWord = extractedText.text.subSequence(0, cursorPosition).split(...delimiters: " ").last()
        // remove the last word from extractedText
        extractedText.text = extractedText.text.subSequence(0, cursorPosition - lastWord.length)
    }

    if (extractedText.text.isNotEmpty()) {
        val extractedTokens = llm.promptToken(extractedText.text.toString())

        candidateView?.updatePredictions(extractedTokens)
    }
}

```

**Figura 12. Pré-processamento das informações e geração de palavras. Fonte: Elaboração Própria.**

Se o pré-processamento resultar em um conteúdo não vazio, a função *promptToken* é acionada para gerar palavras por meio do LLM e, posteriormente, atualizar a interface com as sugestões correspondentes. O método *updateSuggestions* é utilizado em dois momentos distintos: primeiro, à medida que as teclas do teclado são pressionadas (Figura 13); e segundo, quando uma sugestão é selecionada no topo da interface da aplicação (Figura 14).

```

override fun onKey(primaryCode: Int, keyCodes: IntArray) {
    when (primaryCode) {
        Keyboard.KEYCODE_ALT -> {
            isAlt = !isAlt
            val querty = if (isAlt) R.xml.qwerty_alt else R.xml.qwerty

            keyboard = Keyboard( context: this, querty)
            keyboardView!!.keyboard = keyboard
            keyboardView!!.invalidateAllKeys()
        }
        Keyboard.KEYCODE_DELETE -> {
            currentInputConnection.deleteSurroundingText( beforeLength: 1, afterLength: 0)
            updateSuggestions()
        }
        Keyboard.KEYCODE_SHIFT -> {
            isCaps = !isCaps
            keyboard!!.isShifted = isCaps
            keyboardView!!.invalidateAllKeys()
        }
        Keyboard.KEYCODE_DONE -> {
            currentInputConnection.sendKeyEvent(
                KeyEvent(
                    KeyEvent.ACTION_DOWN,
                    KeyEvent.KEYCODE_ENTER
                )
            )
        }
        else -> {
            var charCode = primaryCode.toChar()

            if (Character.isLetter(charCode) && isCaps) {
                charCode = charCode.uppercaseChar()
            }
            currentInputConnection.commitText(charCode.toString(), newCursorPosition: 1)

            updateSuggestions()
        }
    }
}

```

Figura 13. Atualização das sugestões ao digitar. Fonte: Elaboração própria.

```

init {
    viewBind = View.inflate(context, R.layout.candidate_layout, root: this)

    firstPrediction = viewBind.findViewById(R.id.first_prediction)
    firstPrediction.setOnClickListener { it: View!
        service?.pickSuggestion((it as TextView).text.toString())
        service?.updateSuggestions()
    }

    secondPrediction = viewBind.findViewById(R.id.second_prediction)
    secondPrediction.setOnClickListener { it: View!
        service?.pickSuggestion((it as TextView).text.toString())
        service?.updateSuggestions()
    }

    thirdPrediction = viewBind.findViewById(R.id.third_prediction)
    thirdPrediction.setOnClickListener { it: View!
        service?.pickSuggestion((it as TextView).text.toString())
        service?.updateSuggestions()
    }
}

```

**Figura 14. Atualização das sugestões ao escolher uma palavra sugerida. Fonte: Elaboração Própria.**

Para a inclusão das predições no texto, foi desenvolvida um método que, ao ser acionada pelo evento de clique da sugestão selecionada (Figura 14), substitui ou adiciona a palavra. Se o cursor estiver no início do texto ou houver um espaço em branco na posição anterior, a palavra é adicionada. No entanto, se houver o caractere anterior, a função realiza a substituição da cadeia de caracteres anterior ao cursor até o primeiro espaço da direita para a esquerda. Essas abordagens representadas na figura 15 em formato de algoritmo, permite a utilização dos resultados do LLM para funções de predição e correção de palavras.

```

fun pickSuggestion(suggestion: String) {
    val extractedText = currentInputConnection.getExtractedText(ExtractedTextRequest(), flags: 0)
    val currentText = extractedText.text
    val cursorPosition = extractedText.selectionStart
    // If cursor is at the beginning of the text, just add the suggestion
    if (cursorPosition == 0) {
        currentInputConnection.commitText(text: "$suggestion ", newCursorPosition: 1)
        return
    }

    val previousChar = currentText[cursorPosition - 1]
    // If char before cursor is whitespace, just add the suggestion
    if (previousChar.isWhitespace()) {
        currentInputConnection.commitText(text: "$suggestion ", newCursorPosition: 1)
        return
    }

    val charsBeforeCursorUntilWhiteSpace = currentText.subSequence(0, cursorPosition).split(...delimiters: " ").last()
    // Delete characters before the cursor up to the next space, then input the suggested text.
    currentInputConnection.deleteSurroundingText(charsBeforeCursorUntilWhiteSpace.length, afterLength: 0)
    currentInputConnection.commitText(text: "$suggestion ", newCursorPosition: 1)
}

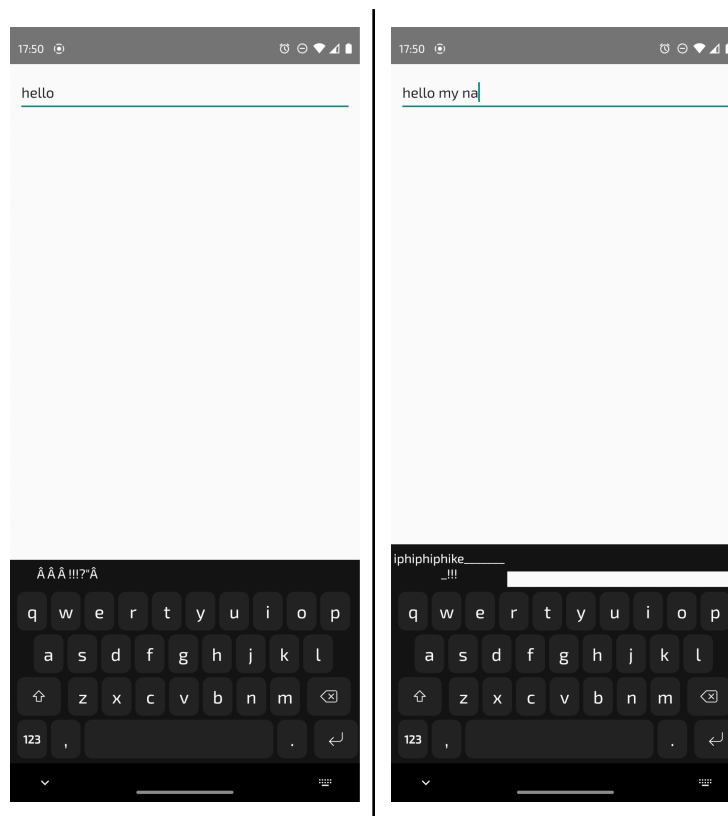
```

**Figura 15. Algoritmo para inserir predição ou correção de palavra. Fonte: Elaboração Própria.**

#### 5.4. Análise do aplicativo

Diante das funcionalidades entregues, é possível analisar a viabilidade de integração de LLM em um teclado Android com os métodos e modelo de IA Generativa proposto. Para forma de avaliação definimos as seguintes exigências: o modelo deve retornar palavras em inglês, e não deve haver congelamentos de tela do smartphone.

Diante da integração realizada, foi necessário averiguar a qualidade dos resultados expelidos pelo modelo. Ao digitar a palavra “*Hello* “ (Figura 16.a), foi obtida uma resposta inesperada, uma sequência de caracteres, que juntos não possuem sentido. Mesmo com a adição de mais palavras, para aumento do contexto (Figura 16.b), não foi possível atingir um resultado minimamente satisfatório, e devido ao tamanho do conteúdo apresentado, ocorreram *bugs* na interface das previsões geradas.



**Figura 16. (a) Sugestão com pouco contexto (b) Sugestão com contexto. Fonte: Elaboração Própria.**

O mesmo comportamento se tornou esperado para a correção de palavras, devido ao uso da mesma metodologia. Diante dos testes realizados, foi notado um tempo de resposta entre as sugestões que atrapalha a digitação. Então foi feito alguns ajustes no algoritmo (Figura 17.a) para identificar o tempo de resposta da predição de palavras, tornando-se visível um atraso de aproximadamente quinhentos milissegundos (Figura 17.b).

```

fun promptToken(prompt: String): List<String> {
    var nextToken: List<String> = listOf()

    val startTime = System.currentTimeMillis()

    nextTokenJob = viewModelScope.launch { this: CoroutineScope
        initJob.join()
        nextTokenJob?.cancelAndJoin()
        nextToken = getNextWord(prompt)
    }

    Log.d( tag: "Prompt", msg: "Prompt: $prompt")
    Log.d( tag: "GenerationTime",
        msg: "Time to get next token: ${System.currentTimeMillis() - startTime} ms"
    )
    Log.d( tag: "GenerationTime", msg: "Next token: $nextToken")
    Log.d( tag: "GenerationTime", msg: "-----")

    return nextToken
}

```

```

D Prompt: hello
D Time to get next token: 2 ms
D Next token: []
D -----
D Prompt: hello
D Time to get next token: 1322 ms
D Next token: [Ã Ã Ã ?"É~"]
D -----
D Prompt: hello
D Time to get next token: 548 ms
D Next token: [Ã Ã Ã Ã ?"]
D -----
D Prompt: hello my
D Time to get next token: 498 ms
D Next token: [iphiphiphke_____Ã ]
D -----
D Prompt: hello my
D Time to get next token: 509 ms
D Next token: [iphiphiphÃ ian_____]
D -----
D Prompt: hello my
D Time to get next token: 470 ms
D Next token: [iphiphiphiph_____!!!]
D -----
D Prompt: hello my
D Time to get next token: 471 ms
D Next token: [iphiphiphiph*****_____]
D -----
D Prompt: hello my
D Time to get next token: 490 ms
D Next token: [iphiphiph_____iced]
D -----
D Prompt: hello my name
D Time to get next token: 453 ms
D Next token: [!!!!!!!_____Éirl]
D -----

```

**Figura 17. (a) Geração das palavras com a depuração de tempo de execução (b) Registros do tempo de execução. Fonte: Elaboração Própria.**

Diante das análises realizadas, verifica-se que, embora a aplicação desenvolvida esteja funcional, a metodologia aplicada revela que o uso de IA generativa como modelo para predição e correção de palavras em teclados virtuais Android não é viável. As limitações identificadas durante o estudo, especialmente relacionadas à obtenção de predições satisfatórias e à complexidade inerente à tarefa, levantam questionamentos sobre a eficácia e a necessidade de outras metodologias para a sua implementação.

## 6. Conclusão

Em resumo, este artigo buscou evidenciar a crescente relevância da Inteligência Artificial Generativa em 2023, especialmente impulsionada pelo avanço do ChatGPT, incentivando os desenvolvedores a explorarem novas soluções baseadas nessa tecnologia promissora. O foco foi direcionar a atenção para os potenciais aplicações e avanços que a IA Generativa pode oferecer, notadamente no cenário dos teclados virtuais Android.

Durante o desenvolvimento deste estudo, foi criada uma ferramenta que desempenhou um papel importante na validação do problema de pesquisa proposto. Introduzimos funcionalidades de predição e correção de palavras como métodos de avaliação da viabilidade do uso de Modelos Grandes de Linguagem (LLMs) em teclados virtuais Android.

Apesar de ter atendido a todas as propostas de funcionalidades apresentadas, é crucial reconhecer que os resultados obtidos não atingiram as expectativas estabelecidas. A complexidade associada ao uso de uma IA Generativa com dimensões reduzidas neste contexto específico revelou desafios significativos, resultando em

limitações na capacidade da ferramenta de gerar previsões e correções de palavras satisfatórias.

Essas descobertas proporcionam insights valiosos para o campo, indicando a necessidade de abordagens mais refinadas e aprofundadas na integração de LLMs em teclados virtuais Android. O desafio em aberto reside na busca por soluções mais eficientes e eficazes, impulsionando futuras pesquisas e desenvolvimentos na direção de uma implementação bem-sucedida dessa tecnologia inovadora.

## **6.1. Trabalhos Futuros**

Após a realização do presente trabalho e, com base em possíveis desenvolvimentos que podem ser derivados do caminho trilhado até aqui, pode-se definir possíveis caminhos de pesquisa e desenvolvimento futuros, visando a complementação e a continuidade do presente trabalho. Para a evolução futura deste projeto, é essencial realizar estudos mais aprofundados sobre os modelos de linguagem utilizados, identificando possíveis falhas no processo de treinamento. Uma análise mais detalhada pode fornecer novas diretrizes para aprimorar a eficácia dos modelos, resultando em uma reavaliação da viabilidade do uso de LLMs em teclados virtuais para Android.

Outro ponto relevante para futuros trabalhos é a otimização da Interface do Usuário (UI). Modificações significativas podem ser implementadas para garantir uma experiência mais amigável e livre de bugs. A busca por uma interface intuitiva e eficiente pode aumentar ainda mais a aceitação e satisfação dos usuários.

Considerando a limitação atual relacionada à capacidade de gerar respostas em português, seria altamente benéfico explorar modelos LLMs que ofereçam suporte específico para a língua portuguesa. A implementação de um modelo LLM capaz de gerar respostas contextualmente relevantes em português abriria novas possibilidades para atender a uma base de usuários mais ampla e diversificada.

Essas sugestões representam direções para futuros trabalhos, com o intuito de aprimorar tanto a eficácia técnica quanto a avaliação da viabilidade do uso de LLMs em teclados para dispositivos Android. O foco está em refinamentos que contribuam diretamente para a análise da viabilidade dessa tecnologia nesse contexto específico, buscando otimizar a integração e o desempenho dos modelos de linguagem, sem necessariamente enfatizar a experiência do usuário. O objetivo central é garantir a continuidade da aplicabilidade e relevância do teclado virtual no ecossistema Android, sob a perspectiva da eficácia técnica e da utilização efetiva de modelos de linguagem.

## **7. Bibliografia**

ALAMMAR, J. **The Illustrated GPT-2 (Visualizing Transformer Language Models)**, 2019. Disponível em: <<https://jalammar.github.io/illustrated-gpt2/>>. Acesso em: 6 nov. 2023.

AGÊNCIA IBGE NOTÍCIAS. **Nove entre dez usuários de Internet no país utilizam aplicativos de mensagens**, 2018. Disponível em: <<https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/20077-nove-entre-dez-usuarios-de-internet-no-pais-utilizam-aplicativos-de-mensagens>>. Acesso em: 8 abr. 2023.

AGÊNCIA IBGE NOTÍCIAS. **PNAD: De 2005 para 2011, número de internautas cresce 143,8% e o de pessoas com celular, 107,2%**, 2013. Disponível em: <https://agenciadenoticias.ibge.gov.br/agencia-sala-de-imprensa/2013-agencia-de-noticias/releases/14404-asi-pnad-de-2005-para-2011-numero-de-internautas-cresce-1438-e-o-d-e-pessoas-com-celular-1072>>. Acesso em: 16 abr. 2023.

ANDROID. **Android**. Disponível em: <<https://developer.android.com/kotlin>>. Acesso em: 07 nov. 2023.

BEAUFAYS, F.; RILEY, M. **The Machine Intelligence Behind Gboard**, 2017. Disponível em: <<https://ai.googleblog.com/2017/05/the-machine-intelligence-behind-gboard.html>>. Acesso em: 28 abr. 2023.

BOSE, S.; ADITI, K.; MUKHERJEE, M. A comparative study: Java vs kotlin programming in android application development. **International journal of advanced research in computer science**, v. 9, n. 3, p. 41–45, 2018.

CHUI, M. et al. **The state of AI in 2023: Generative AI's breakout year**, 2023. Disponível em: <<https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year>>. Acesso em: 13 nov. 2023.

ELASTIC. **O que é um grande modelo de linguagem (LLM)?**. Disponível em: <<https://www.elastic.co/pt/what-is/large-language-models>>. Acesso em 07 nov. 2023.

GOOGLE. **Aprendizado federado no Google Cloud**, 2022. Disponível em: <<https://cloud.google.com/architecture/federated-learning-google-cloud?hl=pt-br>>. Acesso em: 13 maio. 2023.

GOOGLE. **Large language models (LLMs)**. <<https://cloud.google.com/ai/llms>>. Acesso em: 07 nov. 2023.

GRADLE. **Gradle**. Disponível em: <<https://docs.gradle.org/current/userguide/userguide.html>>. Acesso em: 07 nov. 2023.

HARD, A. et al. **Federated learning for mobile keyboard prediction**, 2018. Disponível em: <<https://www.semanticscholar.org/paper/b72b6fae30561a7e29392e04e82ed1ad7bce8e78>>. Acesso em: 11 maio. 2023.

HINTON, G. **Geoff Hinton: On Radiology**, 2016. Disponível em: <<https://www.youtube.com/watch?v=2HMPRXstSvQ>>. Acesso em: 8 maio. 2023.

HUNNICUTT, S. et al. **Design and implementation of a probabilistic word prediction program**. 1997. Disponível em: <<https://www.semanticscholar.org/paper/Design-and-Implementation-of-a-Probabilistic-Word-Hunnicutt-Magnuson/8440c797c799e77bcd187cca75f58e967a92637d>>. Acesso em: 11 maio. 2023.



KRAJCI, I.; CUMMINGS, D. **Android on x86: An introduction to optimizing for Intel architecture**. 1. ed. Berlim, Germany: Apress, 2013.

LEVENSHTAIN, V. I. Binary codes capable of correcting deletions, insertions, and reversals. **Soviet Physics Doklady**, v. 10, n. 8, p. 707–710, 1966.

MALTBY, H. et al. **Markov chains**. Disponível em: <<https://brilliant.org/wiki/markov-chains/>>. Acesso em: 9 maio. 2023.

NOCERA, R. L. et al. Testes de usabilidades entre os sistemas Android e iOS para identificar o melhor sistema para os idosos. **Journal of Chemical Information and Modeling**, v. 4, n. 1, p. 1–15, 2018.

SANTOS, D. S. DOS. **Sistema de recomendação de frameworks para desenvolvimento multiplataforma em dispositivos móveis**. Sergipe: Universidade Federal de Sergipe, 2018.

SINGH, S.; SINGH, S. Systematic review of spell-checkers for highly inflectional languages. **Artificial intelligence review**, v. 53, n. 6, p. 4051–4092, 2020.

SULTONOV, S. M. Importance of Python Programming Language In Machine Learning. **International Bulletin of Engineering And Technology**. v. 3, n. 9, p. 28-30, 2023.

TENSORFLOW. **TensorFlow**. Disponível em <[www.tensorflow.org](http://www.tensorflow.org)>. Acesso em: 08 nov. 2023.

TRANSFORMERS. **Hugging Face**. Disponível em: <<https://huggingface.co/docs/transformers/index>>. Acesso em: 08 nov. 2023.

VASWANI, A. et al. **Attention is All You Need**, 2017. Disponível em: <<https://research.google/pubs/pub46201/>>. Acesso em: 17 nov. 2023.

VERMA, M. Artificial intelligence and its scope in different areas with special reference to the field of education. **International Journal of Advanced Educational Research**, v. 3, n. 1, p. 5–10, 201.