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

Makley Tibola Trichez[1], Márcio Nicolau[2]

[1]ATITUS Educação
Passo Fundo - RS - Brasil

[2] 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. The proposal was to develop a virtual keyboard application for Android. The proposed application looks 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. 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

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[1]. 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[2], 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. 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.

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

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

[2] "[...] 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)

computadores. Isso se deve principalmente à limitação do teclado virtual, 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 dispositivos móveis mais rápida e precisa.

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[3] e o Microsoft SwiftKey[4] 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 possui uma interface mais simples, enquanto o Microsoft SwiftKey valoriza a personalização do teclado. Ambos os teclados estão disponíveis para dispositivos Android e iOS[5].

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, 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.

A precisão de ambos teclados é altamente eficaz para prever palavras, mas as preferências podem variar dependendo das necessidades do usuário. Por exemplo, pessoas 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.

---

[3] Disponível em: <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

[4] Disponível em: <https://www.microsoft.com/pt-br/swiftkey?activetab=pivot_1%3aprimaryr2>. Acesso em: 13 nov. 2023

[5] 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.

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). 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.

Dentre os algoritmos que preveem palavras com inteligência artificial, podem ser citados 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 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 (LEVENSHTEIN, 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.

2.4. Desenvolvimento Android

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.

Quando se compara o desenvolvimento nativo e híbrido, a vantagem tende ao uso de *frameworks*, porém, quando se aborda a criação de um aplicativo que necessite de métodos mais específicos do SO, é mais simples utilizar a Interface de Programação de Aplicativos (API) nativa do Android.

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. 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 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. 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 precisas e relevantes.

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:** 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. No seu formato de correção de palavras, é apresentado uma notificação de erros de escrita a serem corrigidas no canto superior esquerdo do teclado, permitindo que a qualquer momento, possa ser realizada a correção. 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 traz 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, diversas tecnologias foram minuciosamente avaliadas e selecionadas com o objetivo de garantir um desenvolvimento eficaz e eficiente 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 a arquitetura Transformers (VASWANI, 2017) para realizar o processamento de linguagem natural (NLP), o que lhes permitem 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, gerando 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 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 modelos NLP, pois fornece suporte a interoperabilidade entre TensorFlow e PyTorch (TRANSFORMERS, 2023).
- **Frameworks TensorFlow e PyTorch:** Ambos são amplamente utilizados para aplicações de inteligência artificial, incluindo visão computacional e NLP.

Permitem a criação, treinamento e implantação eficaz de modelos de aprendizado de máquina e redes neurais. O TensorFlow, possui uma outra versão chamada de TensorFlow Lite, que é otimizada para a execução de modelos em dispositivos com recursos limitados de processamento e memória, ou sem necessidade de acesso à internet.

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).

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, os testes foram conduzidos em dispositivos virtuais e físicos. O dispositivo virtual possuía uma configuração com heapSize[6] e RAM, respectivamente de 4 e 8 gigabytes, Android versão 13 e API 33, proporcionando um ambiente representativo para avaliação. Essa configuração se assemelha ao dispositivo físico Motorola Edge 30 Fusion, utilizado nos testes.

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.

Durante o estudo da documentação do Android, foi possível encontrar uma classe essencial para a continuidade do desenvolvimento. O *InputMethodService*[7], 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.

---

[6] 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.

[7] Disponível em: <https://developer.android.com/reference/kotlin/android/inputmethodservice/InputMethodService>. Acesso em: 11 nov. 2023

Para tornar possível a utilização da aplicação em qualquer campo do sistema operacional, foi necessário a implementação de um componente *Service*[8]. Os serviços desempenham um papel crucial na execução de tarefas em segundo plano. Ao realizar a integração, é assegurada uma maior flexibilidade e eficiência na manipulação de entradas de texto.

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)

A escolha do modelo BERT disponibilizado pela *Hugging Face* 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.

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[9]. 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.

Durante o estudo mais aprofundado, sobre a arquitetura 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 NLP, como classificação de texto e reconhecimento de entidades.
- **Decoder**: é uma parte da arquitetura Transformer que é responsável pela geração autorregressiva de texto, levando em consideração o contexto à esquerda. O *Decoder* utiliza a camada de auto atenção com máscara para impedir que a

[8] Disponível em: <https://developer.android.com/reference/android/app/Service>. Acesso em: 11 nov. 2023

[9] 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

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 na análise do trabalho de Alammar (2019), é possível concluir que o modelo abordado neste tópico, embora possua a arquitetura Transformer, difere dos LLM utilizados para fins generativos. O BERT utiliza apenas o *Encoder* para aprender representações contextuais de palavras em uma sequência de texto. No entanto, devido à ausência do *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.

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. Em sua documentação, é incluído um exemplo[10] de integração do LLM em um ambiente Android, visando a autocompletar textos.

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. Além disso, buscou-se modificar o parâmetro *top_k*, que é responsável por determinar a quantidade de resultados com as melhores probabilidades.

Durante essa análise, identificou-se um parâmetro chamado *max_length*, o qual poderia ser especificado um tamanho maior que a quantidade de palavras do conteúdo para definir o retorno do texto gerado. Entretanto, não foi possível encontrar uma forma de obter o tamanho do texto de entrada de forma dinâmica. Além da impossibilidade de

[10] Disponível em: <https://www.tensorflow.org/lite/examples/auto_complete/overview?hl=pt-br>. Acesso em: 16 out. 2023

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 probabilidade.

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 disponível. Apesar de existirem modelos convertidos para TensorFlow Lite no próprio repositório, ainda assim foi necessário fazer os ajustes que foram propostos anteriormente.

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 probabilidade de escolha (*logits*), e então utilizar o *tokenizer* para mapear as palavras representadas. 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*[11], que possui dois métodos principais, *loadTFModel* responsável pela inicialização do modelo, e *getNextToken* possui a obrigação de gerar e retornar as sugestões.

Os resultados são retornados em uma *string* separada por vírgula, sendo necessário realizar a divisão, transformando em uma lista. Para utilizar o modelo, é necessário fazer a inicialização juntamente da interface através do *onCreateInputView* da classe *MyInputMethod*[12] que é requisitado pelo sistema quando 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* na mesma classe, que é executado pelo SO, quando notificado que o serviço não está sendo mais utilizado.

Para garantir a consistência contextual, o método *updateSuggestions* da classe *MyInputMethod* é desenvolvido para lidar com diferentes posições 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.

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

[11] Disponível em: https://github.com/Makley-Tibola-Trichez/Integration_LLM_Android_Keyboard/blob/master/app/src/main/java/com/example/keyboard_learning/ml/LLMNextWord.kt

[12] Disponível em: https://github.com/Makley-Tibola-Trichez/Integration_LLM_Android_Keyboard/blob/master/app/src/main/java/com/example/keyboard_learning/service/MyInputMethod.kt

utilizado em dois momentos distintos: primeiro, à medida que as teclas do teclado são pressionadas; e segundo, quando uma sugestão é selecionada no topo da interface da aplicação.

Para a inclusão das predições no texto, foi desenvolvida um método que, ao ser acionada pelo evento de clique da sugestão selecionada, 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 um caractere anterior, a função realiza a substituição da cadeia de caracteres até o primeiro espaço da direita para a esquerda. Essas abordagens representadas no método *pickSuggestions*, permite a utilização dos resultados do LLM para funções de predição e correção de palavras.

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 1.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 1.b), não foi possível atingir um resultado minimamente satisfatório, e devido ao tamanho do conteúdo apresentado, ocorreram *bugs* na interface das previsões geradas.

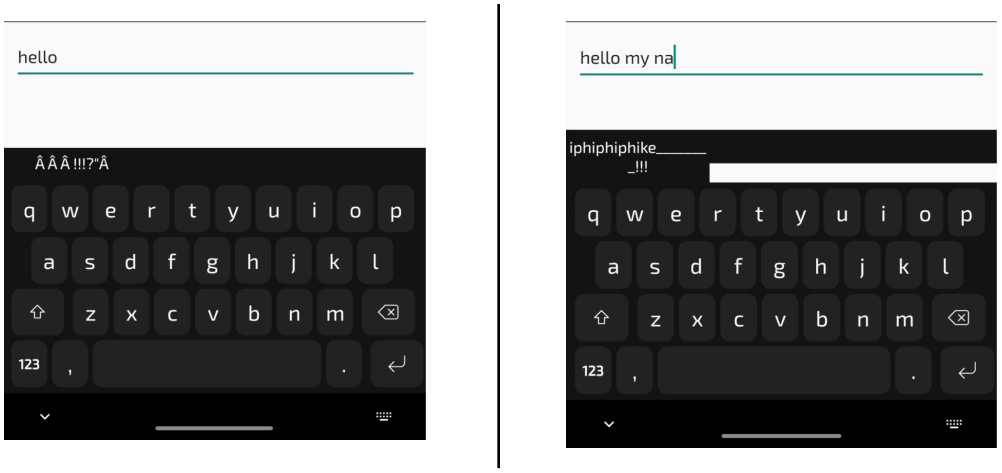

Figura 1. (a) Sugestão com pouco contexto (b) Sugestão com contexto. Fonte: Elaboração Própria.

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 método *promptToken* para identificar o tempo de resposta da predição de palavras, tornando-se visível um atraso de aproximadamente quinhentos milissegundos (Figura 2).

Figura 2. (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 ChatGPT, incentivando os desenvolvedores a explorarem novas soluções baseadas nessa tecnologia. 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. Foram introduzidos 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 revelou desafios significativos, resultando em limitações na capacidade da ferramenta de gerar predições e correções de palavras satisfatórias, indicando a necessidade de abordagens mais refinadas e aprofundadas em trabalhos futuros.

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 caminhos de pesquisa e desenvolvimento futuros, visando a complementação e a continuidade do

presente trabalho. Para a evolução deste projeto, é essencial realizar estudos mais aprofundados sobre os modelos de linguagem utilizados, identificando 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.

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.

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.

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.

LEVENSHTEIN, 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.

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.

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.