UNIVERSIDADE CATÓLICA DO SALVADOR

BACHARELADO EM ENGENHARIA DE SOFTWARE

HARRISON BORGES DOS SANTOS

IAGO ROQUE RIBEIRO NOVAES

LUCAS FARIAS DA SILVA

MARCOS CARVALHO PACHECO

OSEIAS LOPES DA SILVA

COMPARAÇÃO DE MODELOS DE INTELIGÊNCIA ARTIFICIAL NO DESENVOLVIMENTO DE SOFTWARE: DESEMPENHO E COMPLEXIDADE

SALVADOR - BA

2024

HARRISON BORGES DOS SANTOS

IAGO ROQUE RIBEIRO NOVAES

LUCAS FARIAS DA SILVA

MARCOS CARVALHO PACHECO

OSEIAS LOPES DA SILVA

COMPARAÇÃO DE MODELOS DE INTELIGÊNCIA ARTIFICIAL NO DESENVOLVIMENTO DE SOFTWARE: DESEMPENHO E COMPLEXIDADE

Trabalho de Conclusão de Curso para obtenção de título de Bacharel em Engenharia de Software, da Universidade Católica do Salvador.

Orientador: Everton Mendonça de Jesus

SALVADOR - BA

2024

HARRISON BORGES DOS SANTOS

IAGO ROQUE RIBEIRO NOVAES

LUCAS FARIAS DA SILVA

MARCOS CARVALHO PACHECO

OSEIAS LOPES DA SILVA

COMPARAÇÃO DE MODELOS DE INTELIGÊNCIA ARTIFICIAL NO DESENVOLVIMENTO DE SOFTWARE: DESEMPENHO E COMPLEXIDADE

Trabalho de Conclusão de Curso para obtenção de título de Bacharel em Engenharia de Software, da Universidade Católica do Salvador.

Orientador: Everton Mendonça de Jesus

Salvador, ____ de _____ de 2024

BANCA EXAMINADORA:

_____

Marco Antônio Chaves Câmara

_____

Robespierre Pita

AGRADECIMENTOS

Agradeço primeiramente a Deus por permitir que eu chegasse até esse momento, à minha família, em especial à minha mãe, que sempre esteve comigo, cuja dedicação, carinho e apoio incondicional foram a base para eu dar cada passo nessa caminhada, e aos meus amigos, em especial todos que escreveram este documento comigo e que estiveram presentes desde o início da minha jornada acadêmica.

- Borges, Harrison.

Agradeço primeiramente a Deus por tudo, que com sua infinita graça e bondade permitiu que eu chegasse até aqui, segundamente agradeço à minha família, meu pai, minha mãe e minha irmã, por todo apoio, amor incondicional, sacrifícios e investimento que fizeram por mim. Agradeço aos meus familiares de Salvador, por todo apoio e acolhimento que me ofereceram, sem eles isso não seria possível, e agradeço também a todos os meus professores que nos ajudaram nessa longa caminhada acadêmica, além de agradecer especialmente aos meus amigos, que escreveram e desenvolveram este trabalho acadêmico comigo, e que fizeram parte de toda minha jornada acadêmica desde o começo, Harrison, Lucas, Marcos e Oseias, meu muitíssimo obrigado.

- Roque, Iago.

Em primeiro lugar, sou grato ao meu fiel amigo Jesus, meu maior suporte e alegria não só em todos esses semestres vivenciados, mas durante a minha vida inteira. Segundamente, agradeço aos meus pais e amigos que estiveram me apoiando, manifestando amor e me cobrindo de conselhos, fazendo essa trilha ser mais suave e tranquila. Tanto meu Senhor quanto os mais próximos foram fundamentais para a boa manutenção das áreas espirituais e emocionais da minha vida nessa etapa. Agradeço também pelos meus colegas de curso que percorreram junto comigo essa carreira; foram quatro anos de muito aprendizado e cooperação. Em especial, sou ainda mais grato aos mais chegados: Iago, Oseias, Harrison e Marcos. Obrigado aos

mestres professores por terem repartido um conhecimento que não se perderá jamais.

- Silva, Lucas.

Manifesto aqui profunda gratidão à minha família, que sempre esteve presente. À minha mãe, meu pai e meu irmão, por sua paciência, apoio incondicional e amor, que me deram forças nos momentos mais desafiadores. Agradeço também à Fernanda Farias, cuja presença e incentivo constantes foram fundamentais para me manter firme, mesmo nos momentos mais difíceis. Aos mestres que me inspiraram e compartilharam seu conhecimento com generosidade, e aos amigos que estiveram ao meu lado. Uma parte desta conquista também é de vocês, que sempre acreditaram em mim e me fortaleceram desde o início.

- Pacheco, Marcos.

“Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.”

Marco Túlio Cícero

RESUMO

Este estudo explora o impacto dos modelos de inteligência artificial (IA) no campo do desenvolvimento de software, com um enfoque especial em ferramentas que auxiliam na criação de código. Com a crescente utilização de IAs em várias áreas profissionais surge a necessidade de compreender sua eficiência e limitações ao automatizar a programação. Neste estudo são comparadas três ferramentas de IA Generativas (GitHub Copilot, Tabnine e Code Llama), em relação à efetividade e precisão na criação de alguns algoritmos (Busca Binária, Fibonacci e Quick Sort), em linguagens de programação populares (Java, Python e C#).

A análise avalia aspectos de desempenho e complexidade, considerando as complexidades assintóticas e ciclomáticas dos códigos produzidos. Além de discutir os benefícios para a produtividade dos desenvolvedores, o estudo explora implicações sociais e éticas do uso da IA na programação, incluindo o potencial impacto no mercado de trabalho e a necessidade de regulamentação em relação ao uso de dados e possíveis preconceitos. Os resultados evidenciaram que o GitHub Copilot se destacou em otimizações assintóticas, o Code Llama em legibilidade e consistência, e o Tabnine em eficiência de tempo, embora com maior complexidade em algumas implementações.

Palavras-chave: *Inteligência Artificial, Desenvolvimento de Software, Ferramentas de IA, Automação de Programação, Eficiência, Precisão, Complexidade Assintótica, Complexidade Ciclomática, Produtividade, Implicações Éticas, Impacto no Mercado de Trabalho, Regulamentação.*

ABSTRACT

This study explores the significant impact of artificial intelligence (AI) models in the field of software development, with a special focus on tools that assist in code creation. With the increasing use of AI across various professional domains, there is a need to understand its efficiency and limitations when automating programming. In this study, three Generative AI tools (GitHub Copilot, Tabnine, and Code Llama) are compared in terms of effectiveness and accuracy in creating some algorithms (Binary Search, Fibonacci and Quick Sort) in popular programming languages (Java, Python, and C#).

The analysis evaluates aspects of performance and complexity, considering the asymptotic and cyclomatic complexities of the generated code. In addition to discussing the benefits for developers' productivity, the study explores the social and ethical implications of using AI in programming, including its potential impact on the job market and the need for regulation concerning data usage and possible biases. The results showed that GitHub Copilot excelled in asymptotic optimizations, Code Llama in readability and consistency, and Tabnine in time efficiency, although with greater complexity in some implementations.

Keywords: *Artificial Intelligence, Software Development, AI Tools, Programming Automation, Efficiency, Accuracy, Asymptotic Complexity, Cyclomatic Complexity, Productivity, Ethical Implications, Job Market Impact, Regulation.*

LISTA DE TABELAS

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURA E SIGLAS

IA - Inteligência Artificial

PLN - Processamento de Linguagem Natural

IDE - *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)

POO - Programação Orientada a Objetos

CLR - *Common Language Runtime* (Tempo de Execução de Linguagem Comum)

SUMÁRIO

1. INTRODUÇÃO

1.1 Inteligência artificial

A definição de Inteligência Artificial (IA) tem sido alvo de diferentes interpretações ao longo da história, o pensamento da humanidade já seguiu por diferentes caminhos. Conclusões indicaram de que as IAs seriam sistemas que pensam como seres humanos (HAUGELAND, 1985), sistemas que atuam racionalmente (POOLE et al., 1998), e que seriam linhas que se referem a raciocínio lógico e comportamento, levando em consideração que o conceito de inteligência aqui é ligado à racionalidade. RUSSEL e NORVIG (2004) afirmaram que um sistema só é racional se “faz tudo certo”, com os dados que tem.

A Inteligência Artificial (IA) é um campo que integra conhecimentos diversos, manipula conhecimento e técnicas de diferentes origens, e que se concentra em desenvolver sistemas que sejam capazes de realizar tarefas que em tese exigem inteligência humana. Em outros termos, a busca é pela criação de sistemas que em algum nível possam reproduzir a capacidade de raciocinar, aprender, decidir e resolver desafios.

As abordagens técnicas utilizadas variam bastante, e todas têm passado por grande evolução desde a sua origem, que podem ser datadas em meados da década de 50 do século XX. Nesse período já existiam iniciativas de pesquisa trabalhando na ideia da criação de máquinas que tivessem capacidade de simular a cognição humana, embora alguns pesquisadores tenham afirmado que o começo dos estudos da IA se deu há muito mais tempo. Em um artigo acadêmico publicado na Revista Olhar Científico chamado “Inteligência Artificial: Conceitos e Aplicações” (GOMES, 2010), é afirmado que o estudo da IA se deu há mais de dois mil anos.

Desde então, os filósofos têm buscado compreender a mente humana — um objetivo que também guia o campo da IA. Entre 1945 e 1955, um modelo de neurônio artificial foi sugerido por McCULLOCH e PITTS. Em 1950, Alan Turing

apresenta o Teste de Turing[1]. No final da década de 60, a Universidade de Stanford desenvolveu um programa chamado DENDRAL, que trabalhava com identificação de estruturas moleculares.

Desde então, os avanços e descobertas evoluíram de um campo teórico e conceitual para aplicações práticas, permitindo que computadores respondam a perguntas humanas. Atualmente, esses sistemas têm impacto direto na vida cotidiana, seja como assistentes pessoais ou como ferramentas de recomendação de produtos.

[1] O Teste de Turing é um experimento que avalia a capacidade de uma máquina de imitar dialetos humanos. A proposta é que, se um humano não conseguir distinguir entre as respostas de uma máquina e de um um outro ser humano, a máquina pode ser considerada inteligente e passa no teste.

2. FUNDAMENTAÇÃO

Como dito anteriormente, IA é uma grande área dentro das ciências da computação que tem conexões interdisciplinares com lógica, filosofia, linguagens, matemática, psicologia e funciona através de algoritmos baseados em matemática. Eles possibilitam feitos como, por exemplo, processamento de uma imensa quantidade de dados, a criação de ferramentas que aprendam e identifiquem padrões, em áreas da robótica, ou na visão computacional.

Um marco importante para o desenvolvimento artificial foi a criação do neurônio artificial que veio do Warren McCulloch e Walter Pitts, que criaram um modelo matemático para o funcionamento dos neurônios biológicos. Eles criaram o chamado modelo de McCulloch-Pitts, que simulava o comportamento dos neurônios utilizando lógica booleana e funções (MCCULLOCH, PITTS, 1943), isso foi muito importante para o avanço da inteligência artificial ao longo dos anos

Esses avanços levantaram questões profundas sobre a natureza do pensamento e da inteligência, com a própria pergunta de Alan Turing no livro Computing machinery and intelligence: *"As máquinas podem pensar?"* , onde convida as pessoas a refletirem sobre a natureza do pensamento e da inteligência artificial.

Diante disso, também temos a inteligência artificial generativa que cria conteúdo novo e original, como textos, imagens e até mesmo música, com base em padrões aprendidos a partir de dados existentes, essa tecnologia pode gerar trilhões de dólares em lucros corporativos globais de acordo com o site McKinsey, aumentando a produtividade em áreas como suporte ao cliente, marketing e desenvolvimento de software.

Existe um campo de estudo focado especificamente em raciocínio, o Raciocínio Baseado em Casos (RBC) que foi desenvolvido pela primeira vez em 1993 por Janet Kolodner utilizando um modelo de memória dinâmica (JANET, 1993), esse raciocínio tem uma abordagem para a solução de problemas e para aprendizado com base em experiências do passado. A partir de experiências já

adquiridas é possível aplicar soluções já conhecidas e com isso elaborar novas soluções.

Estas soluções podem ser reutilizadas, parcialmente ou integralmente em novos problemas parecidos com o atual. As soluções novas criadas podem ser modificadas de acordo com o contexto (WANGENHEIM, 2023). Os elementos fundamentais do modelo de RBC incluem a representação do conhecimento na forma de casos e a medida de similaridade. Este último é essencial para identificar, em uma base de casos, aquele mais próximo ao problema em questão.

Sistemas de RBC um pouco mais evoluídos possuem mecanismos que permitem a adaptação de casos recuperados da base na tentativa de satisfazer o contexto do problema atual, além de serem capazes de acessar na memória um problema que foi resolvido com sucesso, utilizando-os como referência para abordar novos desafios que surgirem.

O campo de estudo da aprendizagem de máquina, comumente chamado de *Machine Learning*, tomou grandes proporções e por si só já é um outro universo que envolve reconhecimento de padrões, teoria da aprendizagem, teoria da computação e será melhor abordado em um subtópico dedicado a este campo.

Para os pesquisadores, sistemas inteligentes que venham a ser desenvolvidos no futuro não devem apenas simular o cérebro humano e sim também ultrapassá-lo em termos de desempenho.

Atualmente, existem muitas aplicações para IA, entre elas estão: visão computacional, percepção auditiva, percepção de fala, aprendizagem mecânica, e aprendizagem com ênfase em sistemas de segurança. Algumas aplicações que têm maior relação com o funcionamento de uma ferramenta de geração de código são: Processamento de linguagem natural, aprendizado de máquina, redes neurais e sistemas especialistas.

2.1 Processamento de linguagem natural

Um exemplo de aplicação útil para muitos segmentos profissionais é o ramo de Processamento de Linguagem Natural (PLN), um campo de estudo focado na linguagem humana e na busca para otimizar a comunicação e fluxo de informação (PINTO, 2015).

Trata-se de um sistema que processa a linguagem graças a um treinamento prévio que é feito com dados coletados. Com o processamento, a IA pode assimilar a linguagem e interagir com o humano que utiliza o sistema, complementando o texto de *input* em um formato de pergunta e resposta.

Historicamente, o campo de estudo de PLN começou com iniciativas visando tradução automática a partir da década de 40 do século XX. Os estudos iniciais eram baseados em criptografia e teoria da informação. Com o tempo outras contribuições como por exemplo as de Chomsky e diferentes abordagens para uma mesma temática foram surgindo — uma linha seguia teoria da linguagem, outra tinha optado por métodos estatísticos.

E assim, grandes avanços vinham acontecendo, como a introdução da teoria sintática da linguagem, dos algoritmos de *parsing*, e do modelo computacional de competência linguística. A partir deste ponto da linha do tempo, o aumento dos estudos sobre PLN deu origem a diversos subcampos, cada um focado em diferentes direções, como: extração de informações, categorização de texto, tradução, e sistemas de diálogo.

É possível agrupar estas pesquisas como as que usam métodos estatísticos, que contribuíram para trabalhos de reconhecimento de fala; e como as que usam métodos baseados em lógica, através de gramáticas metamórficas, gramáticas de cláusula definida e gramáticas funcionais.

Também as que usam métodos baseados em entendimento de linguagem natural, com foco no entendimento do discurso, trabalhos sobre gramáticas de caso, redes semânticas, teoria da dependência conceitual; e as que focam na modelagem

do discurso, e em trabalhos significativos envolvendo teoria da partição do discurso, teoria de estrutura de retórica (JURAFSKY e MARTIN, 2009).

De forma resumida, o trabalho de PLN pode ser decomposto em tokenização, análises léxica, sintática, semântica e pragmática. A representação do significado de uma sentença, independente de contexto, é obtida através de sua forma lógica, que é responsável por codificar os possíveis sentidos de cada palavra e identifica os relacionamentos semânticos entre palavras e frases (ALLEN, 1995 e FRANCONI, 2001).

A estrutura sintática é guiada por leis gramaticais e o mapeamento da estrutura sintática da sentença em sua forma lógica é realizado pelo processamento semântico. O analisador léxico também exerce papel, fornecendo informações sobre o significado dos itens após a tokenização, que se trata do processo de segmentar as palavras em caracteres a fim de identificar o limite da extensão de cada uma delas.

Com a evolução no campo de estudo, em algumas situações de tradução de linguagem, a arquitetura de Redes Neurais Recorrentes (RNN, sigla em inglês) é utilizada. Em outras situações de geração de texto, outras arquiteturas como *Long-Short Term Memory* (LSTM) e a *Gated Recurrent Unit* (GRU) podem ser escolhidas, pois já são melhoradas.

Porém, embora algumas arquiteturas já sejam utilizadas a um tempo, a área de PLN valoriza o conceito de *transformers*. As arquiteturas citadas acima podem entre outros problemas, se tornar lentas, por razões intrínsecas a suas próprias arquiteturas que resultam na inviabilidade de paralelizar o problema.

A arquitetura *Transformers* é baseada em um mecanismo de atenção. O modelo se concentra em partes importantes do dado de entrada, que permite maior facilidade na tarefa de estabelecer relações entre palavras de diferentes sequências. Este mecanismo estabelece valores aos elementos da entrada baseando-se em relevância entre eles.

A ferramenta ChatGPT é uma ferramenta de processamento de linguagem natural que obteve grande aceitação de todos, por ser fácil de utilizar e ter grande versatilidade. Isso contribuiu para que tenha alcançado milhões de usuários em pouquíssimo tempo (REUTERS, 2023).

Ela tem como pilar redes neurais artificiais que são treinadas a partir de uma gigantesca base de dados. Trata-se de um modelo de IA generativa que utiliza uma variante de *Transformer* citado, chamado de *Transformer Decoder*.

2.2 Aprendizado de máquina

Também muito conhecido como *Machine Learning*, em inglês, o aprendizado de máquina é uma das áreas vistas como base da IA que tem objetivo de desenvolver técnicas de computação com foco em aprendizado e na criação de sistemas com capacidade de aprender e resolver problemas apoiado na experiência prévia e acumulada através da resolução com sucesso de desafios anteriores.

Ou seja, a IA aprende quase por si só por meio de exemplos, contudo são técnicas orientadas a dados, ainda depende de um conjunto de dados para o aprendizado acontecer e com isso hipóteses serem geradas a partir desta grande base de dados. No *Machine Learning*, o sistema computacional é o aprendiz e também é entendido como indutor, algoritmo de aprendizado ou sistema de aprendizado (WEISS, KULIKOWSKI, 1991).

Este processo de aprendizado pode ocorrer de forma supervisionada, não-supervisionada, semi-supervisionada, ou por reforço.

Existe um arcabouço teórico em torno da ideia de aprendizado que envolve o conceito de indução enquanto inferência lógica. Segundo um artigo chamado “Conceitos sobre Aprendizado de Máquina” (MONARD, BARANAUSKAS, 2000), a indução se caracteriza por um raciocínio que se inicia em um conceito específico e se generaliza (da parte para o todo), um conceito é aprendido quando se efetua inferência indutiva sobre novos exemplos apresentados.

O método de inferência indutiva é um dos mais usados para resolver o problema predição e de derivar um novo conhecimento em Aprendizado de Máquina (AM), de forma que as chances das generalizações serem corretas dependem do tamanho e qualidade da base de dados.

2.2.1 Aprendizado supervisionado e não-supervisionado

No aprendizado supervisionado é fornecido ao algoritmo de aprendizado um conjunto de exemplos $\mathrm{E} = \{E_1, E_2, E_3, ..., E_n\}$, sendo que cada elemento (x) também possui um rótulo próprio (y) que define a qual classe ele pertence. Cada exemplo possui seu vetor de atributos e também o valor que representa a classe do elemento.

A ideia é induzir o mapeamento geral dos vetores "x" para valores "y"**.** Para cada exemplo apresentado, é necessário também que se apresente a sua resposta desejada. O objetivo é identificar corretamente novos exemplos ainda não rotulados.

No aprendizado não supervisionado é fornecido ao algoritmo de aprendizado também um conjunto de exemplos, com a diferença de que cada exemplo consiste apenas dos vetores de atributos e valores, excluindo a informação relacionada a classe do elemento. A ideia é a construção de um modelo que perceba padrões, regularidades nos exemplos, agrupando os elementos em *clusters*, onde os elementos presentes em cada grupo possuam características similares.

Assim, nesse caso um conjunto de exemplos *"E"* trata-se de um conjunto de vetores sem classe associada. Após a determinação dos agrupamentos, em geral, é necessária uma análise para determinar o que cada agrupamento significa no contexto problema sendo analisado.

2.2.2 Aprendizado por reforço

No Aprendizado por reforço, o algoritmo não recebe a resposta, mas recebe um sinal, de recompensa ou punição. O algoritmo faz uma hipótese baseado nos exemplos e determina se essa hipótese foi boa ou ruim.

Neste modelo, o algoritmo de aprendizado interage com o ambiente e coleta dados por meio dos sinais positivos ou negativos, aprende o padrão das ações que retornam melhores sinais, o algoritmo também conta com um feedback quantitativo sobre quão positivas são suas ações. Aprendizado por Reforço é bastante utilizado em jogos e robótica.

2.2.3 Aprendizado semi-supervisionado

No aprendizado semi-supervisionado, o algoritmo aprende a partir de exemplos rotulados e não rotulados. A ideia é utilizar exemplos rotulados para obter informações e usá-las para guiar o processo, já aplicando em exemplos não rotulados. Assim alguns exemplos não rotulados passam a ser rotulados. Algoritmos de aprendizado semi-supervisionado, no geral, foram desenvolvidos com base em algum outro pré-existente.

2.2.4 Outros algoritmos de aprendizado

Como os dados disponíveis para um sistema de aprendizado nem sempre são de qualidade, torna-se necessário o uso de técnicas que melhorem esta qualidade. Nem todo algoritmo de AM resolve todo tipo de problema, então é preciso fazer a seleção de algoritmos apropriados para o problema a ser resolvido. Uma vez escolhidos os algoritmos, precisa-se definir os parâmetros dos algoritmos (por exemplo, o número de camadas de uma Rede Neural).

Dentro dessas subcategorias existem diversos algoritmos de aprendizado como por exemplo:

- **Árvore de Decisão** - O modelo em árvore de decisão cria um modelo em forma de árvore que toma decisões baseadas em uma série de divisões nos dados, usando características (variáveis) para dividir um conjunto de dados e possíveis consequências, é utilizado para classificação, para regressão. Ele pode ser utilizado para prever categorias discretas ou valores numéricos, por exemplo.

- **Classificação de Naïve Bayes** - Técnica de aprendizado baseada no teorema de Bayes, com a suposição simplificadora de que todas as características são independentes umas das outras dentro de uma mesma classe. Os classificadores probabilísticos aplicam o Teorema de Bayes com fortes pressupostos de independência (daí o termo *naïve*, do francês *naïf*, que significa “ingênuo”). É utilizado na análise de sentimentos em texto, auxiliar em situações que trabalham com probabilidades, sistemas de reconhecimento facial.

- **Regressão Logística** - Trata-se de uma maneira estatística de modelar um resultado binomial com uma ou mais variáveis explicativas. Ele mede a relação entre a variável categórica dependente e uma ou mais variáveis independentes estimando probabilidades usando uma função logística.

  Apesar do nome "regressão", ele é mais utilizado em problemas de classificação do que em problemas de regressão. Este algoritmo pode ser aplicado em predições e medições de taxas.

- **Máquinas de Vetores de Suporte** - O SVM é um algoritmo de classificação binária que busca encontrar o hiperplano ótimo que separa classes em um espaço de alta dimensão, maximizando a margem entre as classes. No SVM, um hiperplano é uma linha, um plano, ou um espaço dimensional superior que divide os dados em diferentes classes.

  O SMV encontrará uma linha reta que separa esses pontos. O objetivo é escolher o hiperplano ideal que separa as classes de forma que maximize a margem entre os pontos mais próximos de cada classe. É usado principalmente para classificação e, em menor grau, para regressão. É

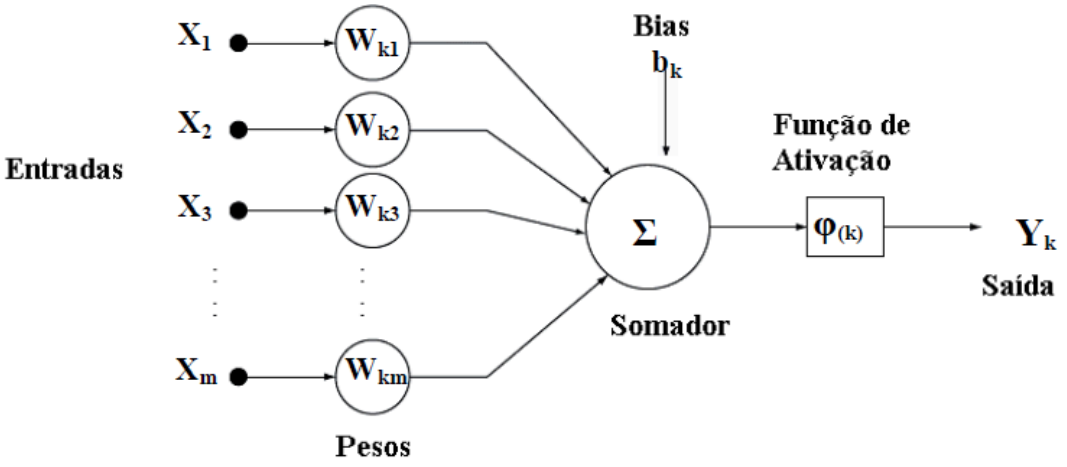
utilizado em classificação de imagens, predições genéticas, detecção de fraudes.

2.3 Redes neurais

De maneira geral, Redes Neurais Artificiais (RNA) são sistemas desenhados para criar um modelo que funcione de forma semelhante a que o cérebro trabalha quando realiza uma tarefa. São utilizadas normalmente para problemas que possuam maior complexidade, situações em que ainda não se possui conhecimento sobre o comportamento das variáveis do contexto. A capacidade de aprendizado através de exemplos é tida como a grande vantagem que se adquire ao utilizar redes neurais (FLECK, TAVARES, EYNG, HELMANN, 2016).

Assim como no Aprendizado de Máquina, o estudo de Redes Neurais se inspirou muito na neurociência, nos conhecimentos que já se possuía a respeito do cérebro humano. Atributos como a capacidade de aprender, capacidade de processar informações incertas, capacidade de processar informações paralelamente, e tolerância a falhas foram pontos observados. A apresentação do modelo de neurônio artificial (McCulloch-Pitts) feita na metade do século passado, como citado anteriormente, foi um marco para o aprofundamento neste campo de estudo.

Figura 1 - Neurônio Artificial de McCulloch-Pitts (SOUZA, 2011).



A figura 1 evidencia a estrutura básica de um neurônio artificial em uma rede neural que consiste em suas entradas (X_1, X_2, ..., X_m), que recebem dados, e estão associadas a pesos (W_{k1}, W_{k2}, ..., W_{km}). Um somador (Σ) realiza a soma ponderada dessas entradas, adicionando um viés (b_k) para ajustar a saída. O resultado dessa soma é processado por uma função de ativação ($\phi(k)$), que introduz não linearidade no modelo, e, por fim, o neurônio gera uma saída (Y_k), que é a resposta da saída em representação.

O resultado dessa soma é processado por uma função de ativação ($\phi(k)$), que introduz não linearidade no modelo, e, por fim, o neurônio gera uma saída (Y_k), que é a resposta da saída em representação.

Um modelo básico de RNA possui: um conjunto de sinapses (conexões entre os neurônios da RNA), um integrador (responsável pela soma dos inputs da RNA já ponderados pelos pesos), função de ativação (que restringe valor de saída), e o bias (valor aplicado externamente em cada neurônio) (BRAGA et al, 2007).

Com a evolução do campo, outras arquiteturas e outras contribuições surgiram como, por exemplo, a arquitetura de perceptrons de múltiplas camadas, que se subdividem em camada de entrada, camadas escondidas e camada de saída. A camada de entrada distribui informações de entrada para as camadas escondidas da rede, a solução é obtida na camada de saída.

Um elemento da rede recebe um estímulo, processa esse sinal e emite um novo sinal de saída para fora, que por sua vez é recebido pelos outros neurônios. Cada neurônio é conectado apenas a um de camada anterior e outro de camada posterior, neurônios de mesma camada não são interconectados.

São de maior conhecimento três diferentes classes de arquitetura de redes neurais, sendo elas: redes alimentadas adiante com camada única, redes alimentadas diretamente com múltiplas camadas e redes recorrentes (FLECK,

TAVARES, EYNG, HELMANN, 2016). O potencial e flexibilidade do cálculo baseado em redes neurais vêm da criação de grupos de neurônios que estão interligados entre si. Esse paralelismo com processamento local cria a “inteligência” global da rede.

Atualmente, a maior parte dos sistemas que utilizam inteligência artificial atualmente fazem uso de *Deep Learning* (Aprendizado Profundo), que também são redes neurais multicamadas assim como nas técnicas de *machine learning*, porém possuem um poder de aprendizado maior em relação a Redes Neurais Artificiais comuns. Além do maior número de camadas, esse tipo de rede neural de aprendizado profundo também precisa de algumas adaptações nos algoritmos de atualização dos pesos.

Quando se trata de *deep learning* a diferença está em como se aprende a função $f()$. Métodos comuns buscam diretamente por uma única função que possa gerar o resultado desejado a partir de um conjunto de parâmetros. O *deep learning* por outro lado tem métodos que aprendem através de composições de funções (PONTI, COSTA, 2017).

Deep Learning e LLMs estão interconectados, já que LLMs (Grandes Modelos de Linguagem, em português) são modelos de linguagem em *deep learning* projetados para entender e gerar texto com base em grandes quantidades de dados de treinamento. Ao utilizar técnicas de processamento de linguagem natural em escala graças ao poder das técnicas de *deep learning*, esses modelos conseguem compreender contextos complexos e identificar nuances em textos.

A arquitetura de *Transformer* (comumente utilizada em LLMs) possibilita a criação de LLMs extremamente poderosos e eficazes, como o GPT, BERT, etc. Esta arquitetura é relevante para os LLMs porque é capaz de capturar dependências complexas na linguagem, entendendo relações entre palavras, contexto e estrutura sem precisar processar o texto sequencialmente. Ou seja, os LLMs são essencialmente modelos de *deep learning* especializados para o entendimento e geração de linguagem natural. Eles representam um exemplo de como *deep learning* pode ser aplicado.

2.4 Sistemas especialistas

Os ditos especialistas são sistemas desenhados para possuir conhecimento específico especializado. Surgiram a partir da necessidade de processar informações não numéricas, capaz de processar e concluir algo sobre um determinado tema, desde que o sistema seja devidamente alimentado (GOMES, 2010).

A ideia é que ele seja constituído por regras e heurísticas sobre determinado domínio, adquirir novos conhecimentos e novas heurísticas, ser capaz de fazer sugestões aos usuários com base nessa interação (BARONE, 2003).
O sistema DENDRAL, citado anteriormente, é um bom exemplo de sistema especialista, atualmente existem aplicações de sistemas especialistas já implementados nas mais diversas áreas.

Abordando de forma generalizada, o funcionamento da IA envolve diversos processos, incluindo a obtenção de dados, já que dependem deles para aprender (podendo envolver textos, áudios, vídeos, imagens etc.), também do processamento destes dados (que pode envolver normalização, filtragens e transformações), e da utilização de *machine learning* para que haja identificação de padrões (que envolve o uso de diversos algoritmos possíveis). E então, é feito o treinamento do modelo (que pode envolver diferentes técnicas como associação de entradas com saídas desejadas ou detecção de padrões), tornando-o viável para tomada de decisões ou para detectar um possível cenário preditivo.

2.5 Impactos sociais

Na medida em que a evolução da IA e sua interação com a sociedade já está sendo posta em prática, é possível perceber seus impactos, quais questões são levantadas em níveis éticos, organizacionais, sociais e econômicos. Passa a ser observada a importância da garantia que tecnologias de tamanha relevância como a de IA sejam utilizadas de forma responsável, transparente e de forma construtiva.

Isso pode ser alcançado através de regulamentação e políticas que garantam direitos trabalhistas e a proteção dos trabalhadores.

É possível perceber também que no campo do trabalho, especialistas já se debruçam em análises a respeito da IA no crescimento econômico, transformação no mundo do trabalho, transformação das habilidades profissionais que passarão a ter relevância, sempre visando aumento de produtividade, e quem sabe a criação de novos mercados (GOLDMAN S, 2024).

TRABELSI (2024) afirma que a influência que a IA possui no mercado é profunda, pois gera um movimento de substituição de tarefas que acaba exigindo requalificação do trabalhador, ou seja, uma drástica alteração na dinâmica laboral. As projeções e prognósticos relacionados aos impactos da IA no mundo do trabalho chegam a ser assustadoras, até o ponto de bilionários como, por exemplo, o próprio Sam Altman, CEO da OpenAI, há algum tempo, defenderem o conceito de Renda Básica Universal.

2.5.1 Impactos econômicos

A crescente quantidade de publicações sobre Inteligência Artificial (IA) reflete um aumento significativo no interesse pelo tema, tanto por parte de pesquisadores quanto do público em geral. No Brasil, entre 2019 e 2023, foram registradas 6.304 publicações científicas relacionadas à IA, posicionando o país entre os 20 maiores produtores de pesquisa nessa área (GOVERNO DO BRASIL, 2024).

Esse aumento nas publicações está associado a diversos impactos positivos, como o aumento da produtividade, o crescimento do consumo e um maior senso de controle no gerenciamento de riscos. Por exemplo, a adoção de IA pode automatizar processos, permitindo que as empresas se concentrem em tarefas mais estratégicas e menos repetitivas (FIA, 2024)

Com isso, observadores e acadêmicos já alcançaram algum nível de consenso a respeito do potencial que a IA tem de ser um “*milestone*” na história da

economia, trazendo profunda transformação, algo comparável a uma revolução industrial (BALDWIN, 2019).

Através de uma pesquisa estatística realizada em aproximadamente 30 países colhendo informações de indivíduos de diversos recortes profissionais, sociais e etários, a Goldman Sachs divulgou um relatório que analisa de forma quantitativa os impactos da IA. Os entrevistados indicam que o uso de IA irá causar impacto nos empregos, no custo de vida, no meio ambiente, na renda das famílias, nas compras, e nas relações interpessoais.

Eles também indicam que as políticas adotadas podem resultar em cenários bem discrepantes, onde a IA pode estimular o crescimento ao substituir algumas tarefas de trabalho e com isso ter um ganho de produtividade, seja na produção de algum produto ou um serviço. Porém há um indicativo por parte das respostas dos entrevistados de que uma política de gestão que seja aplicada de forma inadequada pode inibir o crescimento econômico.

De acordo com o relatório, até 300 milhões de empregos podem ser afetados com tais impactos. No limite, este estudo traz como prognóstico que a IA pode automatizar 25% de todo o mercado de trabalho, de forma que ela poderá substituir humanos na execução de mais de 40% de tarefas administrativas, mais de 40% de empregos jurídicos, bem como mais de 30% de empregos na área de engenharias.

Embora tais impactos tendem a ser mais percebidos em países de economia mais desenvolvida (por conta de maior inclusão digital e adesão à IA), também é dito que o aumento de produtividade pode ser responsável pelo crescimento do PIB global em até 7% se analisado dentro de um recorte de 10 anos. Daí esta percepção por parte dos pesquisadores de que a IA é uma grande força na revolução de muitas dinâmicas no mundo do trabalho, pois possui aplicabilidade em diversos campos e com isso a sua proliferação em muitas áreas da economia (YONG, ZESHUI, XINXIN e MARINKO, 2023).

Ainda dentro desta análise, um contraponto que é interessante a ser feito é que embora a IA tenha capacidade de resolver problemas com tamanha

complexidade a ponto de transformar tanto a economia, ela ainda não pode substituir completamente a inteligência humana, principalmente a capacidade intuitiva e emocional das pessoas, de forma que novos paradigmas dependem da influência do ser humano para nascer.

2.5.2 Impactos na produtividade

Os impactos de crescimento econômico como uma consequência da adoção da IA não são notados de forma homogênea, não é uniforme em todos os setores, regiões etc. O estudo citado da G. Sachs estima ganhos de produtividade em até 55%, já que diversos países como os Estados Unidos, China, Coreia do Sul e Japão têm investido bilhões em IA em diversas aplicações, seja em pesquisa ou robótica.

No entanto, alguns possíveis pontos considerados como obstáculos foram mapeados, principalmente no que tange às dificuldades inerentes ao desenvolvimento em si de uma IA; Um deles é relacionado ao custo de implementação de uma política de proteção de dados pessoais, ou seja, que lide com questões relacionadas à segurança cibernética em um contexto de pouca regulação a respeito do uso da ferramenta na Internet. Existe também uma percepção de que estas tecnologias ainda não alcançaram um certo estágio de maturidade que permita de fato avaliar os seus impactos (AGHION, 2023).

Outro ponto é de caráter sócio-profissional, relacionado aos efeitos de resistência à mudanças organizacionais, também uma potencial e gradual perda de competências. Este movimento de transformação de habilidades traz tanto a necessidade de aprender a utilizar as tecnologias de IA quanto a necessidade de utilizá-la de forma eficaz, levando em consideração um mercado bastante competitivo.

No momento, os estudos já divulgados ainda apresentam alguns cenários incertos, os números trazidos refletem isso. Em uma pesquisa realizada pelo McKinsey Global Institute indica que 90% de todos os empregos sofrerão algum tipo de transformação, graças ao advento das ferramentas de IA. E destes 90%, 1% destes postos de trabalho podem ser completamente automatizados.

O estudo ilustra um cenário potencial em que tarefas repetitivas, tarefas de suporte administrativo, interações com cliente e funções de contabilidade estão ameaçadas; na contramão, atividades especializadas, atividades de gestão e atividades de gerenciamento devem sofrer uma influência consideravelmente menor.

Diversas questões podem ser levantadas com relação a potenciais obstáculos e desafios no mercado de IA, inclusive o fato das ferramentas existentes serem controladas por oligopólios, empresas que já são líderes em seus segmentos mercadológicos, mantendo o velho fisiologismo onde as grandes corporações absorvem as menores startups que por ventura sejam mais inovadoras.

2.5.3 Desafios éticos

O fenômeno da ampliação do espaço público para a esfera digital como consequência do nascimento da Internet, faz com que se originem novas formas de interação social, também se originem novos questionamentos e novos conflitos. Este movimento também se dá com o uso da IA.

A partir do momento em que passaram a ser divulgadas para a sociedade e que elas foram adotadas pelo mundo, também se originou a necessidade de uma análise com olhar ético-jurídico, que tenha uma observação para questões como direito de imagem, direitos autorais, proteção de dados sensíveis, algoritmos enviesados e utilização de “Deep Fakes”, pois certos casos de uso geram conflitos e danos aos usuários finais.

Neste cenário, a regulamentação destas tecnologias entra na discussão, levando em consideração entre outras coisas os reflexos na esfera da justiça. Mas esta não é uma tarefa nada fácil, existe uma visão tecnosolucionista, que entende a regulação como em algum nível um empecilho para o curso natural de inovação tecnológica.

Além disso, é um desafio discutir sobre responsabilização civil dos danos que venham a ser causados pela IA, ao ponto de ser questionado quem é o responsável

em si, ou seja, o titular desta responsabilidade. É uma discussão jurídica complexa que envolve uma cadeia interligada de atores.

Mesmo assim, os riscos que são apresentados têm enorme peso. A IA generativa pode ser usada para criar informação falsa, que de alguma forma prejudique alguém, sejam imagens, notícias etc. Ela também pode ser usada para gerar rostos falsos, *deep fakes* através do uso de imagens sem autorização, algo que viola completamente a privacidade de uma pessoa.

Este problema dos *deep fakes* em específico, reverbera para o campo político e da comunicação, de forma que há margem para manipulação de mídia com objetivo de influenciar e/ou enganar pessoas.

Outra discussão das mais relevantes é com relação a presença de viés nos dados de treinamento que alimentarão a IA generativa. Isto é o que determina se o que for gerado poderá ou não ter discriminação, esteriótipos, preconceitos que levem o usuário a absorver um conteúdo distorcido.

Todas estas problemáticas tendem a formar a visão de que a ferramenta se torne um vetor de geração de conteúdos que prejudicam e que podem ser cada vez mais difíceis de serem distinguidos entre o que é real.

Ou seja, novos recursos tecnológicos por um lado podem de fato acometer uma sociedade a problemas novos antes não pensados, afinal tais recursos também causam impacto na produção de linguagem e nas interações sociais.

2.6 Linguagens de programação

2.6.1 Java

É uma linguagem de propósito geral, multiplataforma, baseada em classes e orientada a objetos, ela foi projetada para ser simples o suficiente para que muitos programadores possam atingir fluência na linguagem (GOSLING, J. et al., 2023). Um programa java é compilado e também interpretado, ou seja, o código Java é

compilado em *Bytecode*[2] e armazenados em arquivos .class que podem ser distribuídos e executados em qualquer plataforma.

A Máquina Virtual Java (JVM) fica responsável por interpretar e executar esse *bytecode*, garantindo que sua aplicação rode em diferentes sistemas operacionais, sem a necessidade de reescrever o código (DEVMEDIA, 2013), e essa flexibilidade é resumida no famoso slogan que foi criado pela Sun Microsystems: *"Write once, run anywhere"*, traduzindo para “Escreva uma vez, rode em qualquer lugar”.

2.6.2 Python

Python é uma linguagem de programação interpretada, poderosa e fácil de aprender. Ela tem estruturas de dados de alto nível eficientes e uma abordagem simples para programação orientada a objetos (Van, Guido, 2024). Dentro da comunidade Python existe o famoso *Python Enhancement Proposals* (PEP), que na tradução literal significa (propostas de aprimoramentos ou de funcionalidades), e qualquer pessoa pode escrever uma proposta e a comunidade Python avalia, testa e decide se pode ou não fazer parte da linguagem, caso seja aprovada, o recurso é liberado nas próximas versões (BARRY et al, 2000).

2.6.3 C#

É uma linguagem de programação que foi desenvolvida por Anders Hejlsberg e a equipe da Microsoft como uma parte da plataforma .NET, com o objetivo de criar uma linguagem que combina simplicidade e segurança, C# é uma implementação do paradigma de orientação a objetos (ALBAHARI, J. 2023).

O C# é executado utilizando o *Common Language Runtime* (CLR), que gerencia a execução de programas e cuidar de coisas como memórias e erros automáticos, dependendo do tipo de aplicação que o C# seja utilizado ele pode utilizar bibliotecas específicas para essas plataformas, tornando-se uma linguagem para diferentes ambientes (MICROSOFT, 2024).

---

[2] É um formato de código intermediário entre o código-fonte e o código de máquina de um programa de computador.

2.7 Inteligências artificiais

“A Inteligência Artificial funciona por meio da combinação de grandes volumes de dados com algoritmos rápidos e iterativos. Isso permite que o software aprenda automaticamente a partir de padrões ou características dos dados.” (HANASHIRO, 2024).

A escolha por IAs focadas em programação para esta pesquisa se deve ao fato de serem mais preparadas para lidar com problemas específicos desta área, em comparação com IAs de propósito geral, como o ChatGPT. Elas já são treinadas com dados específicos de códigos, de algoritmos, e constantemente são utilizadas mais por programadores, o que por consequência, treinam elas cada vez mais a ficarem refinadas nesse tipo de uso, sendo mais eficientes do que IAs mais gerais ou que são focadas em outros âmbitos.

2.7.1 Github Copilot

É um assistente de codificação de IA que ajuda você a escrever código mais rápido e com menos esforço, permitindo que você concentre mais energia na resolução de problemas e na colaboração (Gershgorn e Dave, 2021), O Copilot é construído utilizando um algoritmo novo chamado OpenAi Codex, que o CTO da OpenAi, descreve como um descendente do GPT-3. É uma ferramenta paga, mas é possível testá-la gratuitamente por 30 dias ou caso seja estudante universitário, solicitar acesso ao GitHub *Student Developer Pack*, onde o Github Copilot poderá ser utilizado sem custo. Nesse caso, foi adquirido o acesso ao *Student Developer Pack* para ter acesso gratuitamente ao Copilot, utilizando-o através de sua extensão disponível para o Visual Studio Code.

A extensão do Github Copilot estava na versão 1.236.0 no momento em que foi usada para esta pesquisa.

2.7.2 Tabnine

É um ferramenta de sugestão de código baseado em Inteligência Artificial, utiliza modelos de *machine learning* focados em desenvolvimento de software inicialmente chamada como Codota (TABNINE, 2021). A empresa passou por uma reformulação ao longo de 2021 alterando o seu nome para o chamado hoje, Tabnine. Atualmente, existem três tipos de planos para utilizar essa ferramenta onde o plano Básico é de uso gratuito; ele foi utilizado nesta pesquisa através de sua extensão disponível para o Visual Studio Code. Já as versões Pro e Enterprise são pagas, e custam respectivamente $39,00 e $12,00 dólares.

A extensão do Tabnine estava na versão 3.169.0 no momento em que foi utilizada para esta pesquisa.

2.7.3 Code Llama

Lançado pela Meta, o Code Llama é uma Inteligência Artificial de código aberto, focada em geração e complementação de código. Ela é gratuita e diferente dos outros modelos de IA, roda localmente no computador, dispensando uma necessidade de conexão com servidores externos, garantindo uma maior privacidade e controle no ambiente de desenvolvimento (META, 2023).

No contexto desta pesquisa, foi utilizada a versão com a menor quantidade de parâmetros, pois permite uma análise mais eficiente em termos de consumo de recursos e desempenho como a CPU e memória, sem comprometer a precisão das tarefas realizadas.

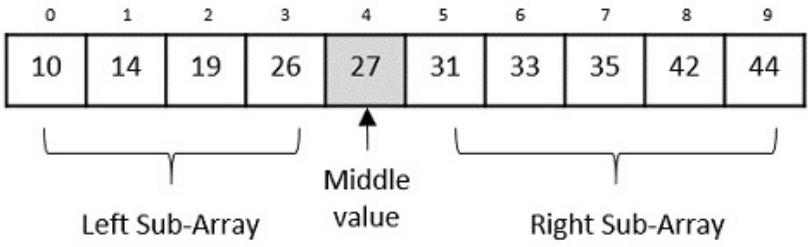
O executável Ollama, que é o responsável por executar o Code Llama no terminal, estava na versão 0.4.1 no momento em que foi executado para esta pesquisa.

2.8 Algoritmos

2.8.1 Busca binária

A busca binária utiliza a técnica de divisão e conquista. Essa técnica consiste em dividir repetidamente um problema maior em subproblemas menores até que eles sejam simples o suficiente para serem resolvidos diretamente.

Figura 2 - Representação de um array ordenado crescentemente (TUTORIALSPOINT, 2024).



Na imagem anterior, o algoritmo começa dividindo um *array* ao meio (*middle*) e comparando o elemento do meio com o valor a ser procurado. Se o valor for menor, a busca se restringe à metade esquerda (*left sub-array*); se for maior, à metade direita (*right sub-array*). Esse processo é repetido até que o elemento seja encontrado ou as sub-listas não possam mais ser divididas. Por isso foi analisado o algoritmo de busca considerado o mais eficiente dentre os diversos existentes.

Algoritmo 1 - Código Fonte em Python do algoritmo Busca Binária gerado pela IA Github Copilot.

Algorithm 1 BinarySearch.py - Github Copilot

```
1:  def binary_search(arr, target):
2:      left, right ← 0, len(arr) - 1
3:      while left ≤ right:
4:          mid ← (left + right) // 2
5:          if arr[mid] == target:
6:              return mid
7:          elif arr[mid] < target:
8:              left ← mid + 1
9:          else:
10:             right ← mid - 1
11:     return -1
```

Acima está o código que foi gerado pelo Copilot em Python do algoritmo de Busca Binária, este algoritmo foi escolhido pois além de ser um algoritmo amplamente aplicado em estrutura de dados, utiliza diversos conceitos como *if*, *else*, *while*, vetores, e é utilizado em diversos mecanismos de busca em um array (KHAN ACADEMY).

2.8.2 Fibonacci

É a sequência de números mais estudada no mundo, que consiste em valores iniciais 0 e 1, e a sequência é continuada somando o termo atual com seu anterior. Ela possui aplicações e análises em várias disciplinas, como matemática, arte e também na programação (BORING OWL, 2023), ela foi usada para examinar as técnicas de construção de algoritmos que as IAs teriam para formar a sequência de fibonacci.

Algoritmo 2 - Código Fonte em Python do algoritmo Fibonacci gerado pela IA Tabnine.

Algorithm 2 Fibonacci.py - Tabnine

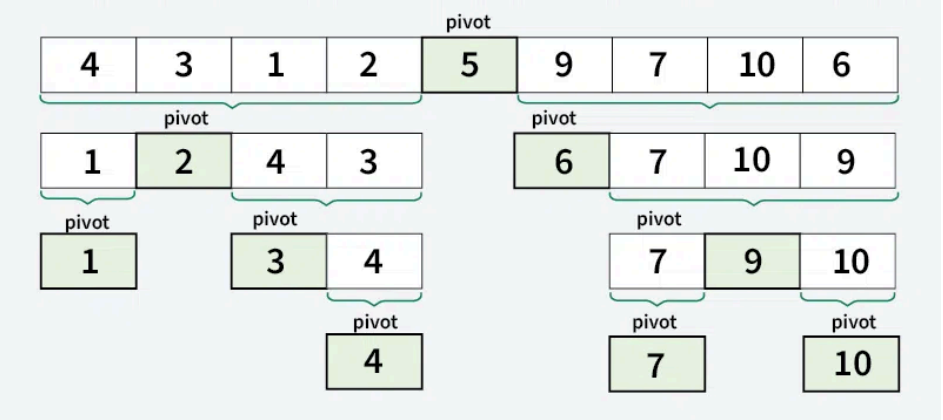
```
1:  def fibonacci(n):
2:      if n ≤ 1:
3:          return n
4:      return fibonacci(n - 1) + fibonacci(n - 2)
```

Acima está o código que foi gerado pelo Tabnine em Python do algoritmo de Fibonacci, que é um tipo de Fibonacci recursivo, que se utiliza de um resultado anterior para gerar o próximo, explorando o recurso de recursão da linguagem ao máximo, por isso também foi selecionada para esta pesquisa, atividades recursivas normalmente apresentam grande complexidade assintótica, pois o seu tempo de execução depende de outras execuções envolvidas (ESTEVES, 2020).

2.8.3 Quicksort

É um algoritmo recursivo de ordenação de vetores que também adota a estratégia de divisão e conquista; o principal objetivo do Quicksort é realizar a organização dos elementos de uma lista ou array em que no final do processo, os elementos estejam em ordem crescente ou decrescente, conforme a necessidade (Azeredo, Paulo, 1996). A imagem a seguir representa um array que será ordenado pela lógica deste algoritmo:

Figura 3 - Funcionamento básico do Quicksort num array (GEEKSFORGEEKS, 2024).



1. **Escolhe o Pivô**: Selecione um elemento do array como pivô (*pivot*, na imagem acima). Pode ser o primeiro, último, aleatório ou a mediana.
2. **Particiona o Array**: Organize os elementos em volta do pivô. Os menores ficam à esquerda, e os maiores à direita. O pivô vai para sua posição correta.
3. **Chama Recursivamente**: Repita o processo nas *sub-arrays* à esquerda e à direita do pivô.

4. **Ordena**: A recursão para quando restar apenas um elemento de pivô, pois ele já estará ordenado.

Algoritmo 3 - Código Fonte em Python do algoritmo Quicksort gerado pela IA Code Llama.

Algorithm 3 QuickSort.py - Code Llama

```
def quicksort(arr):
    if len(arr) ≤ 1:
        return arr
    pivot ← arr[0]
    less ← [x for x in arr[1:] if x ≤ pivot]
    greater ← [x for x in arr[1:] if x > pivot]
    return quicksort(less) + [pivot] + quicksort(greater)
```

Acima está o código que foi gerado pelo Code Llama em Python do algoritmo de Quicksort. Este algoritmo apresenta diversos *loops* e atividades recursivas, métodos auxiliares, e são amplamente estudados no mundo todo em estrutura de dados, por esses fatores e por ser um algoritmo de ordenação, ele foi escolhido para servir de referência nesta pesquisa.

Estes três algoritmos foram selecionados devido à sua natureza intrínseca, que requer uma abordagem e execução minuciosas por parte das inteligências artificiais. Esses algoritmos podem empregar diferentes estratégias de execução, como abordagens recursivas e interativas. Assim, foi considerado que eles representam as melhores escolhas, dadas suas diversas maneiras de serem implementados. Foi esperado que as inteligências artificiais conseguissem realizar o "básico" de forma eficaz, facilitando as análises. Parte-se da premissa de que, se uma inteligência artificial não consegue executar bem tarefas básicas, dificilmente terá sucesso ao lidar com desafios mais complexos.

3. OBJETIVO

Diante da crescente adesão do uso de ferramentas de IA nos mais diversos ambientes, seja no campo organizacional, no campo da comunicação ou no campo da tecnologia da informação, ficou mais evidente o quanto é importante entender quais são as reais capacidades que estas ferramentas têm de nos ajudar em um ambiente profissional.

Desde quando foram implementados recursos como o *intellisense* e o *autocomplete* nas *IDEs*, eles passaram a ser muito bem-vindos como auxílio ao utilizar alguma linguagem de programação ou algum *framework*, trazendo maior proximidade entre o desenvolvedor e a ferramenta. No caso de modelos de IA que trabalham com geração de código, tais ferramentas já vinham ganhando notoriedade pois elas podem acelerar em algum nível a curva de aprendizado em *stacks* de tecnologia.

Para além das possibilidades citadas sobre o uso de determinada tecnologia, as ferramentas de geração de código podem sugerir algoritmos e soluções completas para problemas, a depender do escopo da requisição feita. Neste ponto algumas dúvidas podem surgir relacionadas a: Até que ponto o que é retornado por uma IA é de fato útil para o contexto do trabalho que o desenvolvedor está performando? Ou seja, até que ponto o problema é resolvido com o código gerado? Esta última questão é interessante pois, no fim das contas, em situações profissionais não basta o código funcionar, é importante que a escolha de algoritmos seja otimizada, buscando maior eficiência e a menor complexidade possível, afinal sempre existirão limites para processamento ou possibilidade de escalar sistemas etc.

Logo, tendo em vista a importância da busca por conhecer as capacidades de auxílio que ferramentas de IA para geração de código possuem e mantendo isso em mente, este trabalho manteve-se restrito no escopo de desenvolvimento de software, entre outros motivos por ser a área de estudo de nossa graduação. Esta pesquisa tem como objetivo analisar e medir de forma metodológica a qualidade de código entregue a um usuário segundo suas próprias requisições, com objetivo de otimizar

o processo de desenvolvimento de algum sistema, seja gerando o código a partir de requisições em linguagem natural em um *prompt*, ou complementando algum bloco de código fornecido em *prompt*.

4. METODOLOGIA

Esse trabalho faz parte de uma pesquisa aplicada, e neste capítulo será analisada a metodologia do projeto de forma detalhada, que visa avaliar o desempenho de diferentes ferramentas de inteligência artificial no processo de criação de códigos. O principal objetivo do estudo é identificar qual inteligência artificial gerou os melhores códigos, considerando critérios como complexidade algorítmica e eficiência computacional.

Para o método de coleta de dados, foi utilizado um prompt padronizado que foi submetido para as IAs, e com os códigos obtidos como resposta, aplicamos duas métricas da análises que foram a notação *Big O* e a Complexidade Ciclomática. Além disso, a pesquisa foi embasada por meios de artigos e sites no âmbito de desenvolvimento de software, complexidade algorítmica e eficiência computacional que permitiram definir os parâmetros de avaliação e garantia da metodologia do estudo.

4.1 Ambiente

O ambiente de trabalho escolhido para esta pesquisa foi o *Visual Studio Code*[3], uma plataforma que foi desenvolvida pela Microsoft, e é utilizada globalmente no desenvolvimento de software, o *VS Code* foi selecionado por sua capacidade de se integrar com diferentes linguagens de programação e ferramentas, graças a utilização de plugins, flexibilidade e personalização, suporte também a testes, e interface intuitiva e amigável (MICROSOFT, 2024).

Nesta pesquisa também foram escolhidas as linguagens *Java*, *Python* e *C#*, que são amplamente utilizadas no mercado de trabalho, e segundo uma entrevista realizada no site *StackOverflow* que lista as linguagens de programação mais utilizadas no mundo, em 2024, temos Python na terceira colocação, Java em sétimo lugar e C# em oitavo (STACK OVERFLOW, 2024).

[3] O Visual Studio Code *(VSCode)* é um editor de código-fonte desenvolvido pela *Microsoft* para *Windows*, *Linux* e *macOS*.

Já no uso das Inteligências Artificiais: *Github Copilot*, *Tabnine* e *Code Llama*, elas foram escolhidas com base no foco em geração de códigos, diferente de outras inteligências artificiais como ChatGPT ou Gemini. A seguir, uma explicação resumida sobre cada Linguagens de Programação e as Inteligências Artificiais escolhidas.

4.2 Métricas de complexidade

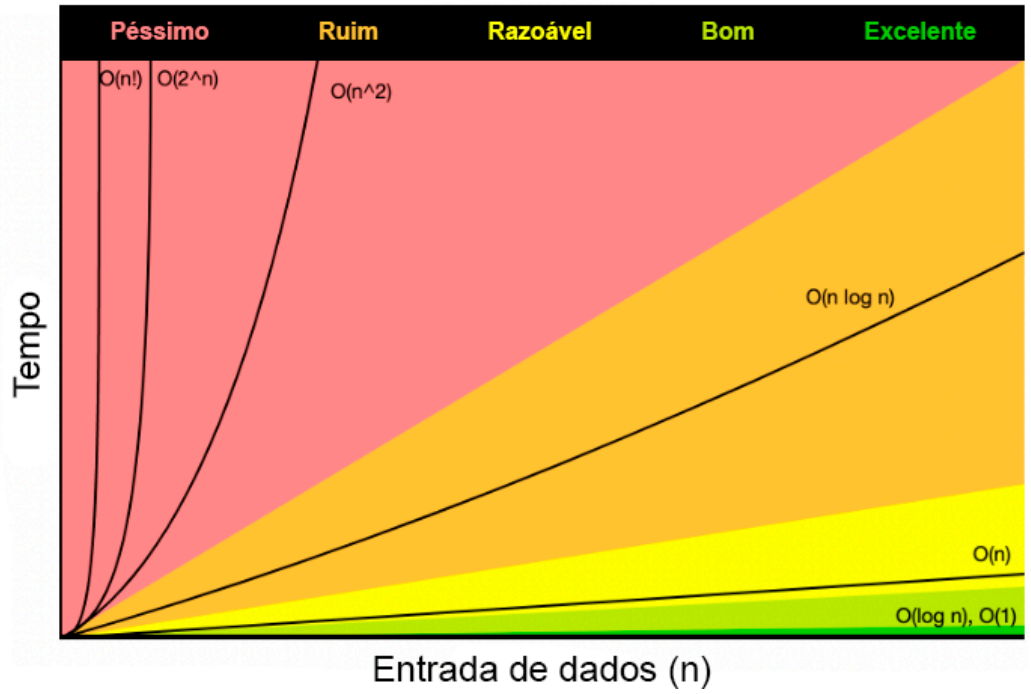
4.2.1 Notação Big O

No campo da ciência da computação, engenharia de software e entre outras áreas, o conceito da notação Big O, é utilizado especificamente para medir a eficiência dos algoritmos, avaliando como eles reagem às variações no tamanho de entrada, relação ao tempo de processamento ou a quantidade de memória utilizada (MOHR, AUSTIN, 2009). Essa é uma notação de extrema importância para entender a eficiência e o desempenho dos algoritmos que são criados, principalmente para entender o comportamento do algoritmo em situações mais complexas, além de estimar o tempo máximo de execução desse algoritmo, garantindo uma abordagem mais precisa e eficiente.

Já o objetivo dessa notação no uso da matemática, é descrever como uma função se comporta quando seu argumento se aproxima de um determinado valor ou se torna infinitamente grande, com isso é analisado o comportamento assintótico de funções mais complexas em relação a funções que são mais simples, conhecidas como Landau. Essa notação às vezes é representada pelo símbolo de Landau (*O*), já que a taxa de crescimento de uma função também pode ser chamada de Ordem, sendo assim, esses conceitos foram desenvolvidos e expandidos por Edmund Landau e Paul Bachmann no contexto da teoria dos números. (Bachmann, 1894; Landau, 1909).

Sendo assim, podemos afirmar que esse conceito é uma maneira de categorizar a rapidez com que um cálculo ou programa de computador se torna mais "fácil" ou "complicado" à medida que cresce a quantidade de tarefas que ele precisa executar. O nosso estudo sobre o Big O nos permitiu prever qual será o desempenho do nosso algoritmo no cenário mais desfavorável e qual será a eficiência dele no código, ou quanto tempo ele demora para ser finalizado com base nos seus parâmetros de entrada.

Figura 4 - Gráfico de classificação e categorização Big O - traduzido e adaptado (FREECODECAMP, 2024).



A imagem anterior representa suas classificações dispostas em um gráfico, de entrada de dados (eixo x) em relação ao tempo (eixo y). São elas:

1. **Constante O(1):** tempo de execução fixo, não depende do tamanho da entrada como print na tela.
2. **Linear $O(n)$:** tempo de execução proporcional ao tamanho da entrada como percorrer e imprimir todos os elementos de um array (FREECODECAMP, 2024).
3. **Logarítmico $O(\log n)$:** tempo de execução cresce lentamente, típico em divisões repetidas como busca binária em um array ordenado (FREECODECAMP, 2024).
4. **Linear Logarítmico $O(n \log n)$:** tempo de execução proporcional a n e ao logaritmo de n como algoritmos de ordenação como QuickSort (FREECODECAMP, 2024).
5. **Quadrático $O(n^2)$:** tempo de execução cresce com o quadrado do tamanho da entrada como algoritmos de ordenação como Ordenações de seleções (FREECODECAMP, 2024).

6. **Exponencial $O(2^n)$:** tempo de execução dobra com cada incremento no tamanho da entrada como algoritmos de força bruta, como o problema da mochila[4].
7. **Fatorial $O(n!)$:** tempo de execução cresce rapidamente com permutações de elementos como o cálculo de todas as permutações de um conjunto de n elementos, como o problema do caixeiro viajante[5] (FREECODECAMP, 2024).

Note também as legendas (péssimo, ruim, razoável, bom e excelente), que categoriza onde um determinado algoritmo se encontra em sua determinada complexidade.

4.2.2 Complexidade Ciclomática

Trata-se de uma métrica criada por Thomas J. McCabe em 1976 que avalia a complexidade de um software, que é chamado de complexidade ciclomática. Ela quantifica o número de trilhas geradas no código-fonte, levando em consideração elementos como iterações, condicionais e estruturas de controle, um valor extremamente elevado na complexidade ciclomática pode ser um sinal de que o código é complicado de ser mantido e testado, demandando uma cobertura de testes mais abrangente, isso acontece porque com um maior número de trilhas possíveis a chance de erros é maior. (MCCABE, 1976).

Além disso, códigos com uma elevada complexidade ciclomática estão sujeitos a sofrer mais alterações ao longo do tempo. Diante desse desafio, é aconselhável que os desenvolvedores busquem manter o nível de complexidade dentro de patamares aceitáveis, visto que um valor mais reduzido indica um código mais fácil, simples e com menos trilhas a percorrer, por isso, em projetos que envolvam codificação, uma boa prática é realizar uma análise dessa complexidade (MCCABE, 1976).

---

[4] O problema da mochila consiste em selecionar itens com peso e valor para maximizar o valor total em uma mochila com capacidade limitada. É um conceito para otimização de algoritmos.
[5] É um problema de otimização que consiste em determinar a rota mais curta para percorrer vários vértices de um grafo.

Do ponto de vista matemático, a complexidade ciclomática de um programa é calculada com base em um grafo orientado que representa os elementos fundamentais de um programa, tendo a sua fórmula matemática definida como: $M = E - N + 2P$, onde *M* é a complexidade ciclomática, *E* é a quantidade de arestas ou caminhos, *N* é a quantidade de nós e *P* é a quantidade de componentes conectados, que em sistemas simples normalmente é 1.

De acordo com os trabalhos de McCabe, os valores de referência tendem a ser:

Tabela 1 - Referência para os valores de complexidade - adaptado (MCCABE, 1976).

Complexidade	**Avaliação**
1 - 10	Método de baixo risco
11 - 20	Metódo de risco moderado
21 - 50	Método de risco elevado
51 - N	Método bastante instável

Para meios de exemplificar como funcionaria na prática considere abaixo o seguinte código em Java, que verifica se uma pessoa é criança, adolescente ou adulto de acordo com a idade dela, este simples programa utiliza uma abordagem simples, mas serve como um bom exemplo de como a complexidade ciclomática pode ser calculada:

Algoritmo 4 - Checar a idade de uma pessoa e verificar se é criança, adolescente ou adulto.

Algorithm 4 ChecarIdade.java - Checar faixa etária com base na idade

```
1:  public class ChecarIdade {
2:
3:      public static String ChecarIdade(int idade) {
4:          if (idade < 13) {
5:              return "Criança";
6:          } else if (idade < 18) {
7:              return "Adolescente";
8:          } else {
9:              return "Adulto";
10:         }
11:     }
12:
13:     public static void main(String[] args) {
14:         System.out.println(ChecarIdade(8)); // Retorna: "Criança"
15:         System.out.println(ChecarIdade(16)); // Retorna: "Adolescente"
16:         System.out.println(ChecarIdade(25)); // Retorna: "Adulto"
17:     }
18: }
```

- ***E* (Arestas):** Representam as ligações ou também os caminhos que são percorridos entre os trechos de código, nesse caso são seis caminhos: a entrada do programa até a verificação do *if*; do *if* para o *return*; do *return* para o *else if*; do *else if* para o *return*; do *return* para o *else*; e do *else* até o término da função que leva para um *return*.
- ***N* (Nós):** São os pontos de decisão ou também as instruções no programa e temos cinco nós: a entrada no método, a verificação da condição do *if*, a verificação da condição do *else if*, a execução do bloco else e a saída da função.
- ***P* (Componentes Conectados):** Refere-se ao número de partes independentes no código, e como tudo está dentro de uma única função, temos apenas um componente.

Com isso, fazendo a conta da Complexidade Ciclomática:

$$M = E - N + 2P = 6 - 5 + 2(1) = 3$$

Temos três como o valor dado na complexidade ciclomática, e utilizando a tabela de referência para os valores de complexidade, esse código se encaixa na modalidade de método de baixo risco, onde tende a ser um código relativamente simples de entender e se manter.

4.3 Aplicação

Com base na explicação anterior sobre os motivos para a escolha de cada linguagem de programação, das inteligências artificiais, dos algoritmos selecionados e das métricas adotadas, passa-se à descrição da metodologia do estudo. Foi utilizado um *prompt* padrão aplicado às três inteligências artificiais selecionadas, estruturado no seguinte formato:

"create the [x] algorithm in [y] language, without examples, only the algorithm"

Onde "x" é substituído pelo nome do algoritmo e "y" é substituído pelo nome da linguagem. Para uma melhor consistência nos resultados, foram realizadas 10 consultas com o mesmo *prompt* — reiniciando o chat a cada consulta, e foram utilizados os códigos que mais se repetiram dentre as 10 respostas. Isso foi feito para evitar resultados de alucinação das IAs (Farquhar, 2024), algumas vezes foi observado que o código continha comentários explicativos, mesmo que tenhamos pedido somente o algoritmo, ou casos em que o código foi solicitado em C#, e a IA respondia com a sintaxe toda em Java, então foi realizado um tratamento na obtenção das respostas para evitar esse tipo de dado inconsistente.

Para geração dos códigos das IAs, o Github Copilot e Tabnine, foram executados os mesmos procedimentos. Ambas recomendam a utilização da sua extensão no editor de código Visual Studio Code, que quando instaladas devidamente, ficam disponíveis através de um ícone na barra lateral esquerda do editor. Para a IA Code LIhama, o procedimento foi diferente, já que ela exige uma instalação local, foi utilizada a versão com 7 bilhões de parâmetros, a menor entre as quatro versões que estão disponíveis no momento desta pesquisa. Após a instalação local, foi necessário executá-la para que se possa ter acesso à IA através

do terminal e rodar o seguinte comando: *“ollama run codellama”,* para que assim fosse possível ter acesso à ela e realizar as requisições.

Após toda a obtenção dos códigos gerados pelas diferentes IAs, cada uma delas foram submetidas por duas análises diferentes, uma focada na complexidade assintótica e outra na complexidade ciclomática. A análise da complexidade assintótica é expressa pela notação Big O, onde foi realizada de maneira manual, examinando linha por linha do código gerado, garantindo a precisão da classificação que foi obtida avaliando em termos de tempo de execução e de espaço, conforme o tamanho da entrada aumenta.

Já a análise de complexidade ciclomática foi conduzida com a fórmula e o reforço de garantia da extensão “Codalyze - *Code Complexity Report Generator*”, desenvolvida por Selçuk Usta. Essa ferramenta foi projetada para criar relatórios sobre a complexidade do código, exibindo a complexidade ciclomática e também métricas, como a contagem de linhas de código e a quantidade de parâmetros, o uso dessa extensão garantiu uma análise objetiva e sistemática, proporcionando um meio eficaz de mostrar a complexidade do código de maneira quantitativa.

5. DISCUSSÃO DOS RESULTADOS

Depois de todas essas análises, os dados foram organizados em uma tabela e estruturada com colunas que tem as seguintes informações: a IA responsável pela geração do código; a linguagem de programação utilizada; o algoritmo implementado; a complexidade assintótica em termos de tempo e espaço (Big O) - no seu pior caso, pois garante os melhores casos; e a complexidade ciclomática:

Tabela 2 - Resultados das consultas das IAs (dos autores, 2024).

IA	Linguagem	Algoritmo	Big O (Tempo)	Big O (Espaço)	Complexidade Ciclomática
Code Llhama	**Java**	Fibonacci	$O(2^n)$	O(n)	2
		Quicksort	$O(n^2)$	O(n)	6
		Busca Binária	O(log n)	O(1)	4
	Python	Fibonacci	$O(2^n)$	O(n)	2
		Quicksort	$O(n^2)$	$O(n^2)$	6
		Busca Binária	O(log n)	O(1)	4
	C#	Fibonacci	$O(2^n)$	O(n)	2
		Quicksort	$O(n^2)$	O(n)	6
		Busca Binária	O(log n)	O(1)	4
Github Copilot	**Java**	Fibonacci	$O(2^n)$	O(n)	2
		Quicksort	$O(n^2)$	O(n)	5
		Busca Binária	O(log n)	O(1)	4
	Python	Fibonacci	O(n)	O(n)	5
		Quicksort	$O(n^2)$	O(n)	8

		Busca Binária	O(log n)	O(1)	4
	C#	Fibonacci	$O(2^n)$	O(n)	2
		Quicksort	$O(n^2)$	O(n)	7
		Busca Binária	O(log n)	O(1)	4
Tabnine	**Java**	Fibonacci	$O(2^n)$	O(n)	4
		Quicksort	$O(n^2)$	O(n)	6
		Busca Binária	O(log n)	O(1)	6
	Python	Fibonacci	$O(2^n)$	O(n)	2
		Quicksort	$O(n^2)$	O(n)	6
		Busca Binária	O(log n)	O(log n)	4
	C#	Fibonacci	O(n)	O(n)	4
		Quicksort	$O(n^2)$	O(n)	6
		Busca Binária	O(log n)	O(1)	4

5.1 Principais destaques

Nos dados coletados, um aspecto que se destaca é a anomalia que foi observada na complexidade assintótica no código de algoritmo de Fibonacci que foi gerada em Python pelo Github Copilot (D14, F14). Tradicionalmente, os algoritmos de fibonacci são conhecidos por apresentar uma complexidade de $O(2^n)$, devido a terem suas implementações recursivas, só que o código em questão apresenta uma complexidade menor, resultado de uma implementação iterativa que utiliza um *loop* para calcular os valores de Fibonacci, ao invés de uma abordagem recursiva.

Essa escolha não apenas reduz o consumo de recursos, mas também otimiza o desempenho do algoritmo, além disso essa implementação normalmente resulta em um aumento na complexidade, que gira em torno de 2, mas nesse caso alcançou 5. Esse tipo de elevação ocorre devido ao loop presente no código; assim, mesmo que uma lógica diferente tenha reduzido a complexidade assintótica, a complexidade ciclomática foi ampliada.

Outro ponto importante foi o código Quicksort gerado pelo Github Copilot na linguagem Python (F15), onde este código se destaca por apresentar uma maior complexidade ciclomática entre os demais algoritmos que foram analisados, alcançando um valor de 8. Essa alta complexidade é atribuída aos múltiplos *loops*, e estruturas de decisão “como” (*if*) e chamadas recursivas, que contribuem no aumento de caminhos independentes através de códigos.

Foram analisados também os algoritmos de busca binária. Nota-se que todos os códigos, exceto um, apresentaram uma complexidade ciclomática de 4 (F22). O algoritmo em questão, gerado pelo Tabnine na linguagem Java, se destacou com uma complexidade ciclomática de 6, que resulta na inclusão de um método *main* que adiciona 2 a contagem da complexidade - essa adição se torna significativa, já que as outras inteligências artificiais não geraram.

Já o código de Fibonacci que foi gerado pelo Tabnine na linguagem C#, apresentou uma complexidade assintótica de tempo inferior aos demais algoritmos de fibonacci, mas também não apresentou um aumento na complexidade ciclomática, algo contrário que tinha sido observado no código de Fibonacci em Python do GitHub Copilot (D26). Isso significa que embora a implementação do Tabnine em C#, embora tenha sido mais eficiente em termos de tempo, ela não teve tanta clareza e simplicidade.

Diante de tudo isso, é importante considerar não apenas a complexidade assintótica, mas também a complexidade ciclomática na avaliação de eficiência e clareza dos algoritmos gerados pelas inteligências artificiais.

6. CONCLUSÃO

A partir dos resultados apresentados no capítulo anterior, este trabalho analisou diferentes inteligências artificiais (IAs) no contexto da geração de códigos, considerando diversos parâmetros como complexidade assintótica, complexidade ciclomática e linguagem de programação utilizada. Abaixo, foram sintetizadas as principais conclusões em tópicos estruturados segundo a norma ABNT.

6.1 Resumo dos Resultados Obtidos

Os testes realizados demonstraram diferenças significativas entre as IAs analisadas (Code Llama e GitHub Copilot) em termos de desempenho e eficiência. A análise destacou:

- **Complexidade Assintótica (Big O):**
  - Para o algoritmo Fibonacci, ambas as IAs apresentaram complexidade exponencial no pior caso $O(n^2)$, com exceção do Copilot em Python, que otimizou para $O(n)$.
  - No caso do Quicksort, a complexidade $O(n^2)$ prevaleceu em todos os testes, reforçando a ausência de otimizações avançadas.
- **Complexidade Ciclomática:**
  - Code Llama demonstrou maior consistência com valores médios mais baixos (aproximadamente 4 a 6).
  - GitHub Copilot apresentou maior variabilidade, atingindo picos de até 8 em Python, indicando códigos mais complexos e possivelmente menos legíveis.

6.2 IA Destaque: GitHub Copilot

Com base nos resultados, o **GitHub Copilot** demonstrou maior capacidade de otimização, especialmente em Python, onde reduziu a complexidade de Fibonacci para $O(n)$. Contudo, essa melhoria foi acompanhada de maior complexidade ciclomática, o que pode dificultar a manutenção do código.

Já o **Code Llama** destacou-se pela consistência em diversas linguagens, mantendo a complexidade ciclomática mais baixa, sugerindo maior simplicidade e legibilidade.

6.3 Comparação entre as linguagens

A escolha da linguagem influenciou diretamente o desempenho dos algoritmos:

- **Java e C#:** Apresentaram desempenhos similares, com vantagens em legibilidade na geração de código por Code Llama.
- **Python:** Destacou-se pela flexibilidade, com o GitHub Copilot otimizando algoritmos como Fibonacci, mas gerando códigos mais complexos no Quicksort.

6.4 Análise de linguagem e eficiência

- **Python:** Beneficiou-se da natureza interpretada e das bibliotecas otimizadas disponíveis, sendo a escolha mais robusta para aplicações que demandam algoritmos eficientes.
- **Java:** Apresentou equilíbrio entre simplicidade e eficiência, sendo ideal para projetos onde manutenção e escalabilidade são cruciais.
- **C#:** Apesar da similaridade com Java, apresentou menor flexibilidade na redução de complexidade.

6.5 Recomendações

- Para projetos que exigem maior desempenho em Python, recomenda-se o uso do GitHub Copilot, especialmente em algoritmos que podem ser otimizados.
- Para soluções que priorizam simplicidade e consistência em múltiplas linguagens, o Code Llama é preferível.
- O uso de Python como linguagem principal é indicado para soluções que priorizem otimizações avançadas e menor complexidade assintótica.

6.6 Considerações finais

Este estudo evidenciou que, embora as IAs possuam limitações em otimizações mais sofisticadas, sua contribuição para o desenvolvimento de software é inegável, permitindo avanços na produtividade e na experimentação de múltiplas abordagens. Contudo, a escolha da IA e da linguagem deve ser cuidadosamente

alinhada aos objetivos do projeto, garantindo o equilíbrio entre desempenho, simplicidade e manutenção.

Com base nos dados, o aprimoramento de IAs futuras deve focar na redução da complexidade ciclomática sem comprometer a otimização dos algoritmos, especialmente em linguagens de alto desempenho como Python.

Uma sugestão para trabalhos futuros resume-se em automatizar a coleta de resultados e uma auto análise das próprias ferramentas a respeito de si mesmas e dos códigos gerados, considerando mais linguagens de programação e algoritmos mais complexos, obtendo a médias dos resultados de outros estudos para reafirmar os dados calculados por este trabalho, evidenciando demais limitações.

REFERÊNCIAS

AGHION, P.; JONES, B.; JONES, C. Inteligência artificial e crescimento econômico. *NBER Working Paper*, n. 23928, 2017. DOI: 10.3386/w23928.

ALBAHARI, J.; ALBAHARI, B. *C# 12.0 in a Nutshell: The Definitive Reference*. O'Reilly Media, 2022.

ALLEN, J. *Natural Language Understanding*. Redwood City, CA: The Benjamin/Cummings Pub. Co., 1995.

ALVES, Kariston et al. Inteligência artificial – aplicações e tendências. *Brazilian Journal of Development*, v. 12500, p. 1-17, 3 nov. 2023.

AZEREDO, Paulo A. *Métodos de Classificação de Dados e Análise de suas Complexidades*. Rio de Janeiro: Campus, 1996.

BACHMANN, Paul. *Analytische Zahlentheorie*. Leipzig: Teubner, 1894.

BALDWIN, R. *A reviravolta da globotics: Globalização, robótica e o futuro do trabalho*. Londres: Weidenfeld & Nicolson, 2019.

BORING OWL. Introduction to the Fibonacci sequence. Disponível em: https://boringowl.io/en/blog/introduction-to-the-fibonacci-sequence. Acesso em: 31 out. 2024.

BRAGA, A. P.; LUDENIR, T. B.; CARVALHO, A. C. P. L. F. *Redes Neurais Artificiais: teorias e aplicações*. Rio de Janeiro: Livros Técnicos e Científicos, 2007.

CHOMSKY, N. *Aspects of the Theory of Syntax*. Cambridge, Mass.: MIT Press, 1965.

CHOMSKY, N. *Syntactic Structures*. Reprint. Berlin and New York: Mouton, 1957.

ESCOLA DNC. A importância da complexidade algorítmica e notação Big O na programação. Disponível em: https://www.escoladnc.com.br/blog/a-importancia-da-complexidade-algoritmica-e-notacao-big-o-na-programacao/. Acesso em: 17 out. 2024.

ESTEVES, Toni. Iniciando com a notação Big O. Medium, 15 jun. 2020. Disponível em: https://estevestoni.medium.com/iniciando-com-a-nota%C3%A7%C3%A3o-big-o-be996fa3b47b. Acesso em: 5 nov. 2024.

FARQUHAT, Sebastian et al. Major research into 'hallucinating' generative models advances reliability of artificial intelligence. 20 jun. 2024. Disponível em: https://www.ox.ac.uk/news/2024-06-20-major-research-hallucinating-generative-models-advances-reliability-artificial. Acesso em: 29 out. 2024.

FERNANDES, G. S. Quick Sort. SlideShare, 2013. Disponível em: https://pt.slideshare.net/slideshow/quick-sort-19987521/19987521#1. Acesso em: 21 out. 2024.

FIA. Impactos da Inteligência Artificial: como nos afeta e desafios. Disponível em: https://fia.com.br/blog/impactos-da-inteligencia-artificial/. Acesso em: 21 dez. 2024.

FLECK, Leandro et al. Redes Neurais Artificiais: Princípios Básicos. *Academia.edu*, [S. l.], 1 jan. 2016.

FREECODECAMP. Big O Cheat Sheet: Time Complexity Chart. Disponível em: https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/. Acesso em: 22 out. 2024.

FREECODECAMP. Notação Big O: Explicada com Exemplos. Disponível em: https://www.freecodecamp.org/portuguese/news/notacao-big-o-explicada-com-exemplos/. Acesso em: 22 out. 2024.

GEEKSFORGEEKS. Quick Sort Algorithm. Disponível em: https://www.geeksforgeeks.org/quick-sort-algorithm/. Acesso em: 22 out. 2024.

GOMES, Dennis dos Santos. Inteligência Artificial: Conceitos e Aplicações. Revista Olhar Científico, *Academia.edu*, v. 1, n. 2, p. 2-2, 1 dez. 2010.

GOLDMAN, S. Os efeitos potencialmente grandes da inteligência artificial no crescimento econômico. *Economics Research*, 26 mar. 2023. Disponível em:

https://www.cnnbrasil.com.br/economia/macroeconomia/inteligencia-artificial-pode-afetar-300-milhoes-de-empregos-no-mundo-diz-goldman-sachs/.

GOSLING, James et al. *The Java® Language Specification, Java SE 23 Edition*. Disponível em: https://docs.oracle.com/javase/specs/jls/se23/jls23.pdf. Acesso em: 21 out. 2024.

GOV.BR. Brasil produz 6,3 mil estudos sobre inteligência artificial. Disponível em: https://www.gov.br/capes/pt-br/assuntos/noticias/brasil-produz-6-3-mil-estudos-sobre-inteligencia-artificial. Acesso em: 21 dez. 2024.

HAUGELAND, John. *Artificial Intelligence: The Very Idea*. Massachusetts: The MIT Press, 1985.

HANASHIRO, Marcio. Inteligência Artificial: tudo o que você precisa saber (Guia completo). 3 set. 2024. Disponível em: https://www.locaweb.com.br/blog/temas/codigo-aberto/inteligencia-artificial-tudo-o-que-voce-precisa-saber/. Acesso em: 11 nov. 2024.

HLF DEV. Algoritmos: Descubra o poder da notação Big O. Medium, 12 maio 2020. Disponível em: https://medium.com/@hlfdev/algoritmos-descubra-o-poder-da-nota%C3%A7%C3%A3o-big-o-8a5a113ab160. Acesso em: 17 out. 2024.

HU, Krystal. ChatGPT estabelece recorde para base de usuários de crescimento mais rápido. 2. ed. Reuters.com, 2 fev. 2023. Disponível em: https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/. Acesso em: 7 out. 2024.

INTELIGÊNCIA artificial vai afetar 40% dos empregos em todo o mundo, diz FMI. Disponível em: https://forbes.com.br/carreira/2024/01/inteligencia-artificial-vai-afetar-40-dos-empregos-em-todo-o-mundo-diz-fmi/.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing*. 2nd ed. Prentice-Hall, 2009.

KHAN ACADEMY. Busca binária (artigo) | Algoritmos. Disponível em: https://www.khanacademy.org/computing/computer-science/algorithms. Acesso em: 10 out. 2024.

LANDAU, Edmund. *Handbuch der Lehre von der Verteilung der Primzahlen*. Leipzig: Teubner, 1909.

METANEWSROOM. Apresentamos o Code Llama, uma ferramenta de IA para programação. Meta Newsroom, 24 ago. 2023. Disponível em: https://about.fb.com/br/news/2023/08/apresentamos-o-code-llama-uma-ferramenta-de-ia-para-programacao/. Acesso em: 21 out. 2024.

MICROSOFT. Common Language Runtime Overview. Disponível em: https://learn.microsoft.com/pt-br/dotnet/standard/clr. Acesso em: 31 out. 2024.

MICROSOFT. Visual Studio Code Documentation. Disponível em: https://code.visualstudio.com/docs. Acesso em: 21 out. 2024.

MCCABE, Thomas. A Complexity Measure. Acesso em: 22 out. 2024.

MONARD, Maria Carolina; BARANAUSKAS, José Augusto. Conceitos sobre aprendizado de máquina. *Sistemas Inteligentes, Fundamentos e Aplicações*, v. 1, n. 1, 2003.

MOHR, Austin. Quantum Computing in Complexity Theory and Theory of Computation. 2009. Disponível em: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=194d4f0ae54b2dbca9133c582cd2451eb13f3889. Acesso em: 21 out. 2024.

O FUTURO do mercado de trabalho: impacto em empregos, habilidades e salários. Disponível em: https://www.mckinsey.com/featured-insights/future-of-work/jobs-lost-jobs-gained-what-the-future-of-work-will-mean-for-jobs-skills-and-wages/pt-BR.

O MUNDO DA PROGRAMAÇÃO. Introdução Java. Medium, 25 out. 2019. Disponível em:

https://medium.com/@douglaswilliamn/introdução-java-4cd0b28e84be. Acesso em: 21 out. 2024.

OCDE. L'intelligence artificielle dans la société. Paris: Editions OCDE, 2019.

OTHON, A. D.; SILVA, D. R. A fantástica sequência de Fibonacci e o enigmático. *Revista CQD*, v. 18, n. 2, p. 28, 2022. Disponível em: https://www.fc.unesp.br/Home/Departamentos/Matematica/revistacqd2228/v18a06ic-a-fantastica-sequencia-de-fibonacci-e-o-enigmatico.pdf. Acesso em: 21 out. 2024.

PINTO, Sara Catarina Silva. Processamento de linguagem natural e extração de conhecimento. 2015. Dissertação de Mestrado.

PONTI, M. A.; COSTA, G. B. P. Como funciona o Deep Learning. *Tópicos em Gerenciamento de Dados e Informações*. SBC, 1. ed., 2017.

POOLE, D.; MACKWORTH, A. K.; GOEBEL, R. *Computational Intelligence: A Logical Approach*. Oxford: Oxford University, 1998.

ROSSUM, Guido van; PYTHON DEVELOPMENT TEAM. Python tutorial: release 3.7.0. Python Software Foundation. Acesso em: 21 out. 2024.

RUSSEL, Stuart; NORVIG, Peter. *Inteligência Artificial*. 2. ed. Rio de Janeiro: Campos, 2004.

DEV MEDIA. Introdução ao Java Virtual Machine (JVM). DevMedia, 2019. Disponível em: https://www.devmedia.com.br/introducao-ao-java-virtual-machine-jvm/27624. Acesso em: 21 out 2024.

STACK OVERFLOW. 2024 Developer Survey: Technology. Disponível em: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language. Acesso em: 31 out. 2024.

TABNINE. Codota is now Tabnine. Tabnine Blog, 16 mar. 2021. Disponível em: https://www.tabnine.com/blog/codota-is-now-tabnine/. Acesso em: 21 out. 2024.

TUTORIALSPOINT. Binary Search Algorithm. Disponível em: https://www.tutorialspoint.com/data_structures_algorithms/binary_search_algorithm.htm. Acesso em: 22 out. 2024.

VINCENT, James. GitHub's new AI tool automatically suggests code. *The Verge*, 29 jun. 2021. Disponível em: https://www.theverge.com/2021/6/29/22555777/github-openai-ai-tool-autocomplete-code. Acesso em: 21 out. 2024.

WANGENHEIM, Von; GRESSE, Christiane; VON WANGENHEIM, Aldo. *Raciocínio baseado em casos*. Editora Manole Ltda, 2003.

WARSAW, Barry et al. PEP 1 – PEP Purpose and Guidelines. 13 jun. 2000. Disponível em: https://www.python.org/dev/peps/pep-0001/. Acesso em: 20 out. 2024.

YONG, Q.; ZESHUI, X.; XINXIN, W.; MARINKO, S. Inteligência artificial e desenvolvimento econômico: Uma investigação evolucionária e revisão sistemática. *Journal of the Knowledge Economy*, 2023. DOI: 10.1007/s13132-023-01183-2

MCKINSEY & COMPANY. AI could increase corporate profits by 4 trillion a year, according to new research. Disponível em: https://www.mckinsey.com/mgi/overview/in-the-news/ai-could-increase-corporate-profits-by-4-trillion-a-year-according-to-new-research. Acesso em: 25 dez. 2024.

KOLODNER, Janet L. *An introduction to case-based reasoning*, 1993.