Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

A IA Generativa na Engenharia de Software

Cássio Azevedo Cancio

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Paulo Roberto Miranda Meirelles
Cossupervisor: Arthur Pilone Maia da Silva
Cossupervisor: Carlos Eduardo Santos

Aos meus pais, que sempre incentivaram meus estudos.
Aos meus professores, que tornaram este trabalho possível.

Resumo

Cássio Azevedo Cancio. **A IA Generativa na Engenharia de Software**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

Elemento obrigatório, constituído de uma sequência de frases concisas e objetivas, em forma de texto. Deve apresentar os objetivos, métodos empregados, resultados e conclusões. O resumo deve ser redigido em parágrafo único, conter no máximo 500 palavras e ser seguido dos termos representativos do conteúdo do trabalho (palavras-chave). Deve ser precedido da referência do documento. Texto texto. Texto texto.

Palavras-chave: Palavra-chave1. Palavra-chave2. Palavra-chave3.

Abstract

Cássio Azevedo Cancio. **Generative AI in Software Engineering: *A case study***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo.

Elemento obrigatório, elaborado com as mesmas características do resumo em língua portuguesa. De acordo com o Regimento da Pós-Graduação da USP (Artigo 99), deve ser redigido em inglês para fins de divulgação. É uma boa ideia usar o sítio www.grammarly.com na preparação de textos em inglês. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text.

Keywords: Keyword1. Keyword2. Keyword3.

Lista de abreviaturas

IA	Inteligência Artificial (*Artificial Intelligence*)
IME	Instituto de Matemática e Estatística
LLM	Modelo de Linguagem de Grande Escala (*Large Language Model*)
USP	Universidade de São Paulo

Lista de figuras

Lista de tabelas

Lista de programas

Sumário

Introdução

Contexto

A engenharia de *software* é um campo da computação que surgiu e se desenvolveu através da crescente demanda da sociedade do fim do século XX até a atualidade por sistemas computacionais cada vez mais complexos. Neste contexto, ferramentas, métodos e processos foram criados para possibilitar o atendimento dessa demanda.

Nas últimas décadas, os estudos em inteligência artificial (IA) avançaram rapidamente, de modo que um novo paradigma em IA surgiu, a IA generativa. Diferentemente da IA tradicional, a IA generativa pode criar conteúdos novos e originais baseados no que aprendeu, em vez de apenas copiar, imitar e reproduzir algo que já existe.

Dada a flexibilidade e a abertura de diversas possibilidades com essa nova tecnologia, é previsível que uma de suas aplicações fosse a engenharia de *software*. Nos últimos anos, diversos estudos foram publicados de modo a analisar essas aplicações, suas consequências e propor diferentes abordagens para tais aplicações (Johnson e Menzies, 2024 e Terragni *et al.*, 2025).

Neste contexto, este projeto se propõe a levantar e estruturar os resultados observados em diversos artigos que tratam do estudo do impacto da IA generativa na engenharia de *software*. As fases da engenharia em foco são: *design* de *software*, codificação e testes.

Motivação

Este trabalho se faz relevante no contexto em que o uso de ferramentas de IA generativa vem crescendo com o passar dos anos, desde o surgimento de ferramentas como *ChatGPT* e *GitHub Copilot*. Segundo dados da *Stack Overflow 2024 Developer Survey* (Stack Overflow, 2024), 63,2% dos desenvolvedores profissionais já utilizam ferramentas de IA no seu processo de desenvolvimento, enquanto 13.5% deste mesmo grupo de trabalhadores planeja utilizá-las em breve. Além disso, entre os desenvolvedores que afirmaram usar inteligência artificial, 82% a utilizam para escrever código.

Desta maneira, é evidente que uma nova tecnologia com amplo uso no mercado de *software* e que abre possibilidade para diversas aplicações, terá impactos sobre como os desenvolvedores escrevem seus códigos. Assim, é de suma importância buscar avaliar e compreender de que maneira esses impactos vêm ocorrendo nas bases de código e na relação dos engenheiros de *software* com o código.

Objetivos

Os principais objetivos do trabalho são:

- Pesquisar como a IA generativa tem sido adotada pelas empresas de *software*;
- Construir uma evolução temporal do uso da IA pelos engenheiros;
- Avaliar a mudança na qualidade e nas métricas do código após a adoção da IA;
- Levantar impressões, usos e sentimentos dos engenheiros quanto ao uso de IA.

Capítulo 1

Referencial Teórico

1.1 Engenharia de Software

Segundo a definição do *IEEE Standard Glossary of Software Engineering Terminology* (1990), a engenharia de *software* é a aplicação de uma sistemática, disciplinada e quantificável abordagem para o desenvolvimento, operação e manutenção de um *software*. Para que a engenharia de *software* seja viável, dada a complexidade dos sistemas demandados na atualidade, foram desenvolvidas etapas, metodologias e ferramentas que dessem suporte aos atores envolvidos no projeto, como desenvolvedores, analistas, investidores, clientes, entre outros.

1.1.1 Etapas do Desenvolvimento de Software

O Ciclo de Vida de Desenvolvimento de Software (SDLC) consiste numa sequência de processos pelos quais o desenvolvimento de um *software* ocorre, de modo a produzir um resultado eficaz e de alta qualidade. Existe alguma variação no número de passos descritos por diferentes fontes, mas, em geral, há sete fases essenciais: planejamento, análise de requisitos, *design*, codificação, testes, implantação e manutenção.

Planejamento

A fase inicial envolve definir o propósito e o escopo do *software*. Durante esta etapa, a equipe de desenvolvimento deve levantar as tarefas necessárias, elaborar estratégias para cumpri-las e colaborar de modo a compreender as necessidades dos usuários finais. Neste processo, os objetivos do *software* e qual problema ele se propõe a resolver precisam ficar claros a todos os envolvidos.

Além disso, nesta fase também ocorre o estudo de viabilidade, ou seja, desenvolvedores e outros atores do projeto avaliam desafios técnicos e financeiros que possam impactar a evolução ou o sucesso do *software*. Ao fim desta fase, um plano de projeto é criado, com o intuito de detalhar as funções do sistema, os recursos necessários, possíveis riscos e o cronograma do projeto. Ao definir papéis, responsabilidades e expectativas claras, o planejamento estabelece uma base sólida para um processo eficiente de desenvolvimento.

Análise de Requisitos

A segunda fase do SDLC visa identificar e registrar os requisitos dos usuários finais. Nesta etapa, a equipe de projeto realiza o levantamento dos requisitos, por meio da coleta de informações das partes interessadas, como analistas, usuários e clientes. São empregadas técnicas como entrevistas, pesquisas e grupos de foco para compreender as necessidades e expectativas dos usuários.

Após a coleta, os dados são analisados, diferenciando os requisitos essenciais dos desejáveis. Essa análise possibilita a definição das funcionalidades, desempenho, segurança e interfaces do *software*. Neste momento, são definidos os requisitos funcionais e não funcionais. Os requisitos funcionais especificam as funções que o *software* deve realizar, ou seja, o que o sistema deve fazer. Já os requisitos não funcionais tratam de como o sistema deve se comportar, incluindo aspectos como desempenho, segurança, usabilidade e escalabilidade.

O resultado desse processo é o Documento de Especificação de Requisitos (DER), que descreve o propósito, as funcionalidades e características do *software*, servindo como guia para a equipe de desenvolvimento e fornecendo estimativas de custo, quando necessário. O êxito desta fase é crucial para o sucesso do projeto, pois assegura que a solução desenvolvida atenda às expectativas dos usuários.

Design

A fase de *design* é responsável pela definição da estrutura do *software*, abrangendo sua funcionalidade e aparência. A equipe de desenvolvimento detalha a arquitetura do sistema, a navegação, as interfaces de usuário e a modelagem do banco de dados, assegurando que o *software* tenha boa usabilidade e seja eficiente.

Entre as atividades desta fase, destaca-se a elaboração de diagramas de fluxo de dados, de entidade-relacionamento, de classes, protótipos de interface e diagramas arquiteturais. O objetivo é garantir que as estruturas projetadas sejam suficientes para dar suporte a todas as funcionalidades do sistema. Também são identificadas dependências, pontos de integração e eventuais restrições, como limitações do equipamento físico e requisitos de desempenho.

O resultado desta fase é o Documento de *Design* de Software (DDS) que estrutura formalmente as informações do projeto e trata preocupações de *design*. Neste documento, são adicionados os artefatos produzidos, servindo como guia estável para coordenar equipes grandes e garantir que todos os componentes do sistema funcionem de maneira integrada.

Codificação

Na fase de codificação, os engenheiros e desenvolvedores transformam o *design* do *software* em código executável. O objetivo é produzir um *software* funcional, eficiente e com boa usabilidade. Para isso, utilizam-se linguagens de programação adequadas, seguindo o DDS e diretrizes de codificação estabelecidas pela organização e pela legislação local.

Durante esta fase, são realizadas revisões de código, nas quais os membros da equipe examinam o trabalho uns dos outros para identificar erros ou inconsistências, garantindo

elevados padrões de qualidade. Além disso, testes preliminares internos são conduzidos para garantir que as funcionalidades básicas do sistema foram atendidas.

Ao final da fase de codificação, o *software* passa a existir como um produto funcional, representando a materialização dos esforços das etapas anteriores, mesmo que ainda sejam necessários refinamentos e ajustes subsequentes. O resultado desta fase é o código-fonte.

Testes

A fase de testes consiste em verificar a qualidade e a confiabilidade do *software* antes de sua entrega aos usuários finais. Seu objetivo é identificar falhas, erros e vulnerabilidades, assegurando que o sistema atenda aos requisitos especificados.

Inicialmente, são definidos parâmetros de teste alinhados aos requisitos do *software* e casos de teste que contemplem diferentes cenários de uso. Em seguida, são conduzidos testes de diversos níveis e tipos, incluindo testes de unidade, de integração, de sistema, de segurança e de aceitação, permitindo a avaliação tanto de componentes individuais quanto da operação do sistema na sua totalidade.

Quando um erro é identificado, ele é registrado detalhadamente, incluindo seu comportamento, métodos de reprodução e impacto sobre o sistema. As falhas são encaminhadas para correção e o *software* retorna à fase de testes para validação. Este ciclo de teste e correção se repete até que o sistema esteja conforme os critérios previamente estabelecidos. O resultado desta fase é um código-fonte mais robusto e menos propenso a falhas.

Implantação

A fase de implantação ou *deployment* consiste em disponibilizar o *software* aos usuários finais, garantindo sua operacionalidade no ambiente de produção. Este processo ocorre tanto no primeiro lançamento do sistema, quanto quando ele já está em uso pelos usuários e passando por atualizações. Por isso, o processo deve minimizar interrupções e impactos negativos nos acessos dos usuários.

A escolha da estratégia de implantação é feita conforme as características do sistema e de seus usuários. As estratégias mais comuns são:

- ***Rolling*:** a atualização ocorre de forma gradual, substituindo instâncias antigas por novas até que todo o sistema esteja atualizado;
- ***Blue-Green*:** dois ambientes paralelos são mantidos, um em produção e outro em preparação, permitindo a troca imediata entre eles;
- ***Canary*:** a nova versão é liberada primeiramente para um grupo de usuários, monitorando o comportamento do sistema antes de expandir a implantação para todos.

Além de colocar o *software* em operação, esta fase envolve assegurar que os usuários compreendam seu funcionamento. Para isso, podem ser fornecidos manuais, treinamentos e suporte técnico. Desta maneira, a fase de implantação marca a transição do *software* de projeto para produto, iniciando efetivamente o cumprimento de seus objetivos e a entrega de valor ao usuário.

Manutenção

A fase de manutenção é caracterizada por suporte contínuo e por melhorias incrementais, de modo a garantir que o *software* mantenha seu funcionamento adequado, acompanhe as necessidades dos usuários e as demandas de mercado. Nesta fase, são realizadas atualizações, correções de falhas e suporte ao usuário.

Considerando o horizonte de longo prazo, a manutenção inclui estratégias de modernização ou substituição do *software*, buscando manter sua relevância e adequação às evoluções tecnológicas.

1.1.2 Metodologias de Desenvolvimento

As metodologias de desenvolvimento de *software* consistem em abordagens sistemáticas para organizar, planejar e executar projetos de engenharia de *software*. Cada metodologia apresenta vantagens e limitações, sendo a escolha dependente do tamanho do projeto, grau de complexidade, maturidade da equipe e expectativa de mudanças nos requisitos.

Método de Cascata

A metodologia cascata é uma abordagem sequencial em que cada fase do desenvolvimento de *software* deve ser concluída antes de iniciar a seguinte. Ela é adequada para projetos com requisitos muito definidos e pouca probabilidade de mudanças durante o desenvolvimento. O modelo enfatiza documentação detalhada e planejamento prévio, garantindo controle rígido sobre prazos e entregas. Embora simples de aplicar, pode se mostrar inflexível diante de alterações nos requisitos ou no ambiente de negócios.

Metodologias Ágeis

As metodologias ágeis consistem em práticas iterativas e incrementais, voltadas à entrega contínua de valor ao usuário e à adaptação rápida às mudanças nos requisitos. As abordagens ágeis valorizam colaboração, flexibilidade e melhoria contínua, sendo especialmente adequada para projetos dinâmicos e de alta complexidade.

Além disso, elas são geralmente complementadas por práticas de Integração Contínua e Entrega Contínua (CI/CD), que automatizam a construção, teste e implantação do *software*. Desta maneira, a utilização de CI/CD permite que novas funcionalidades e correções sejam disponibilizadas rapidamente, mantendo a qualidade do sistema e reduzindo o tempo de reposta entre a equipe de desenvolvimento e os usuários.

Alguns exemplos de métodos ágeis incluem:

- ***Scrum*:** método iterativo que organiza o trabalho em ciclos curtos, as iterações (*sprints*), com duração de uma a quatro semanas. Define papéis específicos, como o responsável pelo produto (*product owner*), o facilitador do processo (*scrum master*) e a equipe de desenvolvimento. Inclui ainda reuniões regulares, como o planejamento da iteração (*sprint planning*), reuniões diárias de acompanhamento (*daily*), a revisão da iteração e a retrospectiva, que garantem planejamento, monitoramento e melhoria contínua do processo;

- ***Kanban*:** método visual de gestão do fluxo de trabalho, baseado na utilização de quadros divididos em colunas que representam etapas do processo (por exemplo, "A Fazer", "Em Progresso" e "Concluído"). Cada atividade é representada por um cartão que se movimenta conforme avança nas etapas, promovendo transparência, fluxo contínuo e foco na identificação e eliminação de gargalos;
- **Programação extrema (*extreme programming*):** metodologia ágil que enfatiza práticas de desenvolvimento colaborativo e de qualidade, como programação em par, integração contínua, desenvolvimento orientado a testes e refatoração frequente. Valoriza o envolvimento próximo do cliente, a simplicidade do código e a adaptação rápida às mudanças de requisitos;
- **Desenvolvimento enxuto (*lean software development*):** abordagem inspirada nos princípios da produção enxuta, que busca eliminar desperdícios, reduzir custos e acelerar entregas. Essa metodologia prioriza a criação de valor para o cliente, a melhoria contínua e a capacitação das equipes para tomadas de decisão mais eficientes.

1.1.3 Ferramentas de Desenvolvimento

As ferramentas de desenvolvimento de software oferecem suporte às etapas do ciclo de vida, desde planejamento até manutenção. O uso combinado dessas ferramentas contribui para maior produtividade, qualidade e confiabilidade no desenvolvimento de software. Elas incluem ferramentas de:

- **Controle de Versão (como *Git* e *SVN*):** permitem registrar e gerenciar alterações no código-fonte temporalmente, possibilitando a colaboração simultânea entre desenvolvedores, a recuperação de versões anteriores e o rastreamento completo do histórico de mudanças;
- **Ambientes de Desenvolvimento Integrados (IDEs) (como *Visual Studio, IntelliJ IDEA* e *Eclipse*):** oferecem um conjunto de ferramentas em um único ambiente, incluindo edição de código, depuração, testes, gerenciamento de dependências e integração com sistemas de controle de versão;
- **Gerenciamento de Projetos (como *Jira, Trello* e *Asana*):** auxiliam na organização e priorização de tarefas, acompanhamento do progresso e comunicação entre membros da equipe, fornecendo transparência e facilitando a coordenação do trabalho;
- **Integração e Entrega Contínua (CI/CD) (como *Jenkins, GitHub Actions* e *GitLab CI*):** automatizam processos de compilação, testes e implantação, promovendo maior qualidade e agilidade nas entregas de software;
- **Teste (como *Selenium, JUnit* e *Postman*):** permitem a execução de testes automatizados e manuais para validar funcionalidades, desempenho e segurança do sistema, contribuindo para a detecção precoce de falhas e a melhoria da qualidade do software.
- **Virtualização e Monitoramento (como *Docker, Prometheus* e *Grafana*):** permi-

tem criar ambientes isolados e consistentes para execução do software, garantindo que ele funcione de maneira idêntica em diferentes máquinas. Além disso, possibilitam acompanhar o desempenho e a saúde dos sistemas em produção, auxiliando na detecção precoce de problemas e na manutenção da qualidade do software.

1.2 Inteligência Artificial

Inteligência Artificial (IA) é o campo da ciência da computação que se dedica a criar sistemas capazes de executar tarefas que normalmente exigem inteligência humana, como reconhecimento de padrões, raciocínio, tomada de decisão, resolução de problemas e aprendizado a partir de dados. Segundo a posição da IEEE (*Artificial Intelligence* 2019), a inteligência artificial envolve tecnologias computacionais inspiradas, mas funcionam de forma diferente, no modo como pessoas e outros organismos biológicos percebem, aprendem, raciocinam e agem.

As aplicações de IA afetam cada vez mais todos os aspectos da sociedade, incluindo defesa e segurança nacional, sistemas de justiça civil e criminal, comércio, finanças, manufatura, saúde, transporte, educação, entretenimento e interações sociais. Aplicações como essas estão se expandindo pela combinação de processadores avançados, grandes volumes de dados e novos algoritmos. Segundo um estudo da McKinsey Global Institute (2018), a IA contribuirá com cerca de 13 trilhões de dólares para o PIB global até 2030.

1.2.1 Classificações por Funcionalidade

Máquinas Reativas

As máquinas reativas representam o tipo mais rudimentar de inteligência artificial. Estes sistemas não possuem memória e não conseguem usar experiências passadas para influenciar decisões futuras, portanto, são projetados para executar uma tarefa muito específica e reagem apenas a um conjunto limitado de estímulos atuais, com base em sua configuração inicial.

Por consequência, não aprendem ou se adaptam às novas circunstâncias, operando com base em um conjunto de regras rígidas e pré-definidas. Um exemplo clássico de um sistema reativo é um programa de xadrez do século XX que, embora possa jogar uma partida inteira, reage apenas aos movimentos no tabuleiro e não tem capacidade de lembrar ou aprender com partidas anteriores para melhorar seu jogo. A principal limitação desta IA é a sua falta de contexto e a incapacidade de generalizar ou se adaptar.

IA de Memória Limitada

A IA de memória limitada marca um avanço significativo sobre as máquinas reativas, pois ela permite que dados históricos sejam armazenados, por um curto período, de modo que essas informações promovam decisões mais eficientes no futuro. Entretanto, sua "memória" é transitória e específica para a tarefa em questão, sem formar uma compreensão completa ou duradoura do mundo. As aplicações práticas desta IA são vastas, incluindo

sistemas de recomendação de serviços de streaming e de redes sociais, direção de carros autônomos, assistentes virtuais, *chatbots* de atendimento ao cliente, entre outros.

IA de Teoria da Mente

Nesta categoria, a IA conseguiria entender as emoções, crenças, intenções e estados mentais dos seres humanos com quem interage. A ideia é que, ao compreender o estado mental de uma pessoa, a IA poderia adaptar suas ações e respostas para permitir interações sociais mais naturais e fluidas.

Uma pesquisa publicada na revista *Nature Human Behaviour* constatou que alguns modelos de linguagem de grande porte (LLMs) apresentaram desempenho equivalente ou superior ao de seres humanos quando submetidos a testes de capacidade de rastreamento de estados mentais de pessoas (Strachan *et al.*, 2024).

IA Autoconsciente

A IA autoconsciente é o ápice da progressão da inteligência artificial. Nesta categoria, a IA seria um sistema consciente de si mesmo e do mundo ao seu redor, o que lhe permitiria entender e reagir a estímulos externos de forma análoga à consciência humana. Além da capacidade de compreender as emoções e pensamentos humanos, ela também teria seu próprio conjunto de emoções, necessidades e crenças. A IA autoconsciente é estritamente teórica até o momento.

1.2.2 Classificações por Capacidade

Inteligência Artificial Restrita (*Narrow AI*)

A Inteligência Artificial Restrita, também conhecida como IA fraca, refere-se a sistemas projetados para executar uma tarefa específica ou um conjunto limitado de tarefas. Esses sistemas de IA são altamente especializados, portanto, apresentam baixa capacidade de generalização.

Devido ao seu alto nível de especialização, a IA fraca apresenta desempenho de alto nível nas funções para as quais foi projetada, eventualmente superando o desempenho humano nesta tarefa. A IA fraca está presente em ferramentas de reconhecimento facial, tradução automática de idiomas, recomendação de redes sociais e *engines* de xadrez.

Inteligência Artificial Geral (*General AI*)

A Inteligência Artificial Geral, também chamada *IA Forte*, refere-se a sistemas que possuiriam a capacidade de compreender, aprender e aplicar conhecimento em uma ampla variedade de tarefas, de forma semelhante às habilidades cognitivas humanas. Diferentemente da IA Restrita, a IA Geral teria a capacidade de generalizar seu conhecimento e aplicá-lo em diferentes contextos.

Características Principais

- **Inteligência Ampla:** seria capaz de executar múltiplas tarefas, tornando-se versátil e adaptável.
- **Raciocínio Semelhante ao Humano:** teria a habilidade de raciocinar, resolver problemas e tomar decisões de forma comparável a um ser humano.
- **Aprendizado Autônomo:** seria capaz de aprender e melhorar continuamente, adaptando-se a novas situações e adquirindo novas habilidades sem intervenção humana.

Atualmente, a IA Geral permanece no campo teórico e ainda não foi alcançada. Pesquisadores trabalham para desenvolver sistemas que possam atingir esse nível de capacidade, mas trata-se de um objetivo de longo prazo no desenvolvimento da IA.

Inteligência Artificial Superinteligente (*Superintelligent AI*)

A Inteligência Artificial Superinteligente representa a forma mais avançada de IA, superando a inteligência humana em todos os aspectos, incluindo criatividade, resolução de problemas e inteligência emocional. Esse tipo de IA seria capaz de ultrapassar as mentes humanas mais brilhantes em qualquer área, desde a ciência até a arte e as habilidades sociais.

Características Principais

- **Supera a Inteligência Humana:** possuiria habilidades cognitivas muito além das capacidades humanas, tornando-se potencialmente a ferramenta mais poderosa — ou a maior ameaça — já criada.
- **Tomada de Decisão Autônoma:** teria capacidade de decidir sem intervenção humana, com processos de raciocínio possivelmente além da nossa compreensão.
- **Questões Éticas e Existenciais:** o desenvolvimento de uma IA Superinteligente levanta preocupações éticas profundas, incluindo os riscos que poderia representar à humanidade se não for devidamente controlada.

Assim como a IA Geral, a IA Superinteligente ainda é um conceito teórico, explorado em discussões acadêmicas e obras de ficção científica. Seu possível desenvolvimento é tema de intensos debates entre pesquisadores, especialistas em ética e futuristas.

1.2.3 Aprendizado de Máquina

Aprendizado de máquina (Machine Learning) é um subcampo da IA que se concentra em criar algoritmos capazes de aprender e fazer previsões a partir de quantidades expressivas de dados, geralmente estruturados ou rotulados, sem serem explicitamente programados para cada tarefa. Um conceito essencial para o aprendizado de máquina são as redes neurais, modelos computacionais inspirados na estrutura do cérebro humano. Elas são compostas por camadas de nós (neurônios artificiais) interconectados, capazes de processar informações e aprender padrões a partir de exemplos. Redes neurais são amplamente utilizadas em tarefas como classificação, reconhecimento de imagens e processamento de linguagem natural.

Aprendizado Supervisionado

O aprendizado supervisionado opera com dados rotulados, onde cada entrada de dados é pareada com uma saída correta correspondente. O objetivo do modelo é aprender o mapeamento entre a entrada e a saída, para então ser capaz de fazer previsões ou classificações precisas em novos dados não vistos. Este paradigma é ideal para cenários onde a resposta correta é conhecida.

Entre as aplicações práticas do aprendizado supervisionado, destacam-se a detecção de spam em e-mails e a análise de sentimentos do cliente, onde os modelos são treinados para classificar textos com base em categorias predefinidas. No campo da visão computacional, ele é utilizado no reconhecimento de imagens e objetos, permitindo que algoritmos localizem, isolem e classifiquem elementos em vídeos ou imagens. Um exemplo clássico e cotidiano é o teste CAPTCHA, que treina modelos a reconhecerem veículos em imagens para detectar bots. Outras aplicações incluem a criação de mecanismos de recomendação personalizados, que analisam o histórico de escolhas e compras de clientes para oferecer sugestões relevantes , e a análise de dados preditiva para estimar salários de funcionários ou prever necessidades de estoque.

Aprendizado Não Supervisionado

Diferentemente do aprendizado supervisionado, o aprendizado não supervisionado trabalha com dados não rotulados. Seu propósito é descobrir padrões, estruturas e agrupamentos ocultos nos dados sem qualquer intervenção humana prévia. Este modelo se destaca por sua capacidade de encontrar semelhanças e diferenças nas informações, o que o torna uma solução poderosa para a análise exploratória de dados.

Entre as tarefas principais do aprendizado não supervisionado estão o clustering (agrupamento), a associação e a redução de dimensionalidade. O clustering é uma técnica de mineração de dados que agrupa informações com base em suas semelhanças. Um exemplo prático é a segmentação de clientes, onde o algoritmo agrupa automaticamente clientes com comportamentos de compra ou características semelhantes, permitindo que as empresas criem estratégias de marketing mais direcionadas. Outra aplicação é a categorização de feedback de clientes para grandes modelos de linguagem (LLMs), permitindo que o algoritmo identifique tópicos como "questão de cobrança"ou "feedback positivo"a partir de textos não estruturados. A análise de associação, por sua vez, é usada na detecção de fraude, examinando padrões de compras e relacionamentos entre contas para identificar comportamentos suspeitos. A capacidade de identificar padrões ocultos nos dados é a principal contribuição deste paradigma.

Aprendizado por Reforço

O Aprendizado por Reforço é um processo de aprendizado dinâmico em que um "agente"aprende a tomar decisões ao interagir com um ambiente. O objetivo do agente é maximizar uma recompensa cumulativa, tomando ações que geram feedback positivo e evitando aquelas que resultam em feedback negativo. O processo de aprendizado se dá por tentativa e erro, sem a necessidade de exemplos rotulados.

Este paradigma é particularmente útil para resolver problemas complexos e sequenciais

em ambientes incertos. Aplicações reais incluem a robótica e o desenvolvimento de carros autônomos, onde o agente aprende a navegar e interagir com o ambiente para realizar tarefas específicas. No universo dos jogos, algoritmos de aprendizado por reforço, como o AlphaGo Zero, alcançaram desempenho super-humano ao aprender a jogar contra si mesmos, aperfeiçoando suas estratégias através do feedback de cada movimento. Além disso, a técnica é aplicada em cenários de tomada de decisão, como no gerenciamento de recursos em clusters de computadores, no controle de semáforos para otimizar o fluxo de tráfego, na reconfiguração autônoma de sistemas web, e na otimização de reações químicas. O Aprendizado por Reforço representa uma mudança de paradigma da "aprendizagem passiva"(onde o modelo simplesmente processa dados estáticos) para a "aprendizagem ativa", na qual o agente interage com um ambiente dinâmico, aprendendo com as consequências de suas ações. Essa capacidade de aprender a agir é um pilar de uma IA mais sofisticada.

Aprendizado Profundo

O Aprendizado Profundo, como um subconjunto do Aprendizado de Máquina, distingue-se pelo uso de Redes Neurais Artificiais (RNAs), estruturas computacionais inspiradas no funcionamento do cérebro humano. Essas redes são compostas por camadas de nós interconectados, onde cada nó é responsável por aprender um atributo específico dos dados. O termo "profundo"se origina da utilização de múltiplas camadas ocultas entre a camada de entrada e a de saída, o que permite que a rede construa representações hierárquicas e complexas dos dados.

O processo de treinamento de uma rede neural profunda envolve a alimentação de um grande conjunto de dados rotulados. Conforme a rede processa esses dados, ela ajusta os "pesos"nas conexões entre os nós, usando técnicas como o

backpropagation, para que o algoritmo aprenda e melhore sua precisão na classificação de novos dados. A capacidade das redes neurais profundas de reter ou esquecer informações seletivamente, processar grandes volumes de dados e realizar tarefas complexas, como processamento de linguagem natural e reconhecimento de fala, as torna uma das tecnologias mais poderosas da IA contemporânea.

Existem arquiteturas de redes neurais profundas especializadas para diferentes tipos de dados. As Redes Neurais Convolucionais (CNNs), por exemplo, são particularmente eficazes para processar imagens e vídeos. Elas utilizam camadas convolucionais para extrair informações dos dados de entrada, camadas de agrupamento para reduzir a dimensionalidade e camadas totalmente conectadas para fazer previsões de alto nível. As Redes Neurais Recorrentes (RNNs) são projetadas para dados sequenciais, como texto ou séries temporais. Sua arquitetura permite que a rede combine a entrada atual com um estado oculto anterior, conferindo-lhe uma forma de "memória"para capturar dependências e padrões ao longo do tempo. Uma forma avançada de RNN é a memória de curto prazo longa (LSTM), que aprimora a capacidade da rede de "lembrar"o que aconteceu em camadas anteriores, tornando-a ideal para tarefas de linguagem natural e análise de sentimentos.

1.2.4 IA Generativa

A Inteligência Artificial Generativa, ou IA Generativa, é um subconjunto da IA que desenvolveu a capacidade de criar conteúdos novos e originais, como texto, imagens, vídeos, áudio e código, em resposta a prompts de texto, que podem variar de simples a altamente complexos. Este campo emergiu como um dos mais proeminentes da IA nos últimos anos, marcando uma transição da capacidade da IA de apenas processar e classificar dados para a de criar ativamente.

O avanço da IA Generativa está intimamente ligado à inovação arquitetural dos modelos de transformação, ou Transformers. Antes de sua ascensão, as Redes Neurais Recorrentes (RNNs) eram a tecnologia de ponta para processamento de dados sequenciais. No entanto, sua natureza sequencial impedia a paralelização e limitava o treinamento em grandes conjuntos de dados. A arquitetura Transformer, introduzida no artigo de 2017 "Attention is All You Need,"revolucionou este processo ao permitir que a rede processasse uma sequência de dados inteira simultaneamente. Essa capacidade de paralelização foi a tecnologia fundamental que permitiu o treinamento de modelos em conjuntos de dados massivos, utilizando o poder de processamento das GPUs.

Essa inovação arquitetural levou ao desenvolvimento dos Grandes Modelos de Linguagem (LLMs), que são modelos de linguagem pré-treinados em gigantescas quantidades de texto e que se tornaram a base da maioria das aplicações modernas de embedding de palavras. Exemplos notáveis de LLMs incluem o BERT do Google (que usa o codificador do Transformer) e a série GPT da OpenAI (que usa o decodificador). A capacidade dos LLMs de gerar texto coerente e de alta qualidade foi o que, de fato, catalisou a era moderna da IA Generativa.

A explosão da IA Generativa que se observa hoje não é um evento isolado, mas um resultado direto de uma inovação arquitetural específica. A capacidade de paralelização (a causa) permitiu o treinamento em escala sem precedentes (o efeito imediato), que, por sua vez, desbloqueou a capacidade de gerar conteúdo novo e coerente (o efeito subsequente). Portanto, o sucesso da IA Generativa é um triunfo da engenharia e escalabilidade tanto quanto do algoritmo em si.

As aplicações da IA Generativa se estendem por diversas modalidades. Na geração de texto, os LLMs impulsionam chatbots, assistentes virtuais e ferramentas de geração de código. Na geração de imagens e vídeos, ferramentas como o Adobe Firefly permitem aos usuários criar conteúdo visual a partir de comandos de texto. O processo é intuitivo: o usuário insere um comando de texto detalhado, e o modelo de IA, que utiliza Aprendizado de Máquina e Processamento de Linguagem Natural, converte a descrição em um clipe de vídeo profissional, geralmente com cinco segundos de duração.

Modelos de Linguagem

Modelos de linguagem são algoritmos estatísticos e computacionais projetados para compreender, gerar e manipular texto em linguagem natural. O objetivo central desses modelos é capturar padrões linguísticos, tanto de natureza sintática quanto semântica, a partir de grandes coleções de dados textuais (*corpora*). A partir desses padrões, o modelo consegue prever a probabilidade de uma palavra ocorrer em determinada sequência,

construir sentenças coerentes, interpretar perguntas e até mesmo inferir significados implícitos no discurso.

Historicamente, os primeiros modelos de linguagem eram baseados em abordagens probabilísticas relativamente simples, como modelos de *n-gramas*, que calculavam a probabilidade de uma palavra com base nas anteriores. Com o avanço da aprendizagem profunda (*deep learning*), redes neurais recorrentes (RNNs) e arquiteturas como LSTMs (*Long Short-Term Memory*) e GRUs (*Gated Recurrent Units*) passaram a ser empregadas para capturar dependências de longo prazo em sequências textuais. Apesar disso, tais arquiteturas enfrentavam dificuldades em lidar com textos extensos devido a problemas como o desvanecimento do gradiente.

Transformers

Introduzidos em 2017 por Vaswani et al. no artigo *"Attention is All You Need"*, os *Transformers* representam uma das arquiteturas mais influentes e transformadoras no campo do processamento de linguagem natural (PLN). Diferentemente de RNNs e LSTMs, os Transformers não processam o texto de forma sequencial; em vez disso, utilizam mecanismos de *atenção* que permitem analisar todas as palavras de uma sequência em paralelo. Isso possibilita capturar relações contextuais entre termos que estão distantes no texto, algo que modelos anteriores realizavam de forma limitada e ineficiente.

O componente central dessa arquitetura é o mecanismo de *self-attention*, que calcula pesos relacionando cada palavra da sequência com todas as outras, permitindo ao modelo focar seletivamente nas informações mais relevantes para determinada tarefa. Essa capacidade de processamento paralelo e contextualização eficiente revolucionou não apenas o PLN, mas também áreas como visão computacional e bioinformática, sendo hoje a base de modelos de ponta em inteligência artificial.

Modelos de Linguagem de Grande Escala (*Large Language Models – LLMs*)

Modelos de linguagem de grande escala (*Large Language Models* – LLMs) são uma evolução direta da arquitetura Transformer, caracterizados pelo uso de quantidades massivas de dados de treinamento e pela presença de bilhões (ou até trilhões) de parâmetros ajustáveis. Essa escala permite que os modelos capturem nuances linguísticas extremamente sofisticadas, possibilitando não apenas a geração de texto coerente, mas também a realização de tarefas complexas como tradução automática, sumarização de documentos, resposta a perguntas, geração de código e até raciocínio lógico em determinados contextos.

Entre os exemplos mais representativos de LLMs estão o *GPT* (Generative Pre-trained Transformer), da OpenAI, o *BERT* (Bidirectional Encoder Representations from Transformers), do Google, e o *T5* (Text-to-Text Transfer Transformer). Cada um desses modelos introduziu contribuições significativas: o GPT popularizou a abordagem de pré-treinamento seguido de ajuste fino para múltiplas tarefas; o BERT trouxe o treinamento bidirecional, permitindo maior compreensão de contexto; e o T5 propôs unificar diferentes tarefas de PLN no paradigma *text-to-text*.

Devido ao seu poder e versatilidade, os LLMs vêm sendo amplamente aplicados em produtos tecnológicos, assistentes virtuais, motores de busca e sistemas de suporte à

decisão. Entretanto, seu uso também levanta desafios importantes relacionados a viés nos dados, consumo energético elevado durante o treinamento e riscos de uso inadequado, tornando essencial a discussão ética e responsável sobre seu desenvolvimento e aplicação.

Capítulo 2

Metodologia

2.1 Abordagem de Pesquisa

2.1.1 Tipo de Pesquisa

2.1.2 Procedimentos Metodológicos

2.2 Coleta de Dados

2.2.1 Fontes de Dados

2.2.2 Instrumentos de Coleta

2.2.3 Processo de Coleta

2.3 Análise de Dados

2.3.1 Métodos de Análise

2.3.2 Ferramentas Utilizadas

2.3.3 Critérios de Avaliação

Capítulo 3

Resultados

3.1 Análise dos Dados

3.2 Avaliação do Sistema

3.2.1 Desempenho

3.2.2 Eficiência

3.2.3 Usabilidade

3.3 Discussão

3.3.1 Limitações Identificadas

3.3.2 Melhorias Propostas

Capítulo 4

Conclusão

4.1 Resumo dos Resultados

4.1.1 Principais Descobertas

4.1.2 Objetivos Alcançados

4.1.3 Contribuições

4.2 Trabalhos Futuros

4.2.1 Direções de Pesquisa

4.2.2 Melhorias Propostas

4.2.3 Desafios Identificados

4.3 Considerações Finais

Referências

[*Artificial Intelligence* 2019] *Artificial Intelligence*. Position Statement. Piscataway, NJ, USA: IEEE, jun. de 2019. URL: https://globalpolicy.ieee.org/wp-content/uploads/2019/06/IEEE18029.pdf (citado na pg. 8).

[*IEEE Standard Glossary of Software Engineering Terminology* 1990] *IEEE Standard Glossary of Software Engineering Terminology*. Rel. técn. 1990, pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064 (citado na pg. 3).

[Johnson e Menzies 2024] Brittany Johnson e Tim Menzies. " AI Over-Hype: A Dangerous Threat (and How to Fix It) ". *IEEE Software* 41.06 (nov. de 2024), pp. 131–138. ISSN: 1937-4194. DOI: 10.1109/MS.2024.3439138. URL: https://doi.ieeecomputersociety.org/10.1109/MS.2024.3439138 (citado na pg. 1).

[McKinsey Global Institute 2018] McKinsey Global Institute. *Notes from the AI frontier: Modeling the impact of AI on the world economy*. Acesso em: 25 ago. 2025. Set. de 2018. URL: https://www.mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-ai-frontier-modeling-the-impact-of-ai-on-the-world-economy (citado na pg. 8).

[Stack Overflow 2024] Stack Overflow. *Stack Overflow Developer Survey 2024*. 2024. URL: https://survey.stackoverflow.co/2024/ (acesso em 19/08/2025) (citado na pg. 1).

[Strachan *et al.* 2024] James W. A. Strachan *et al.* "Testing theory of mind in large language models and humans". *Nature Human Behaviour* 8.7 (2024), pp. 1285–1295. ISSN: 2397-3374. DOI: 10.1038/s41562-024-01882-z. URL: https://doi.org/10.1038/s41562-024-01882-z (citado na pg. 9).

[Terragni *et al.* 2025] Valerio Terragni, Annie Vella, Partha Roop e Kelly Blincoe. "The future of ai-driven software engineering". *ACM Trans. Softw. Eng. Methodol.* 34.5 (mai. de 2025). ISSN: 1049-331X. DOI: 10.1145/3715003. URL: https://doi.org/10.1145/3715003 (citado na pg. 1).

Índice remissivo