

UNIVERSIDADE DE SÃO PAULO  
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**Percepções em Transformação**

*Os Impactos da IA Generativa na Produção  
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

*O conteúdo deste trabalho é publicado sob a licença CC BY 4.0  
(Creative Commons Attribution 4.0 International License)*

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



## Resumo

Cássio Azevedo Cancio. *Percepções em Transformação: Os Impactos da IA Generativa na Produção 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.

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



## Abstract

Cássio Azevedo Cancio. **Perceptions in Transformation: Impacts of AI on Code Production.** Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo.

**Keywords:** Keyword1. Keyword2. Keyword3.



# Listas de abreviaturas

IA	Inteligência Artificial
IME	Instituto de Matemática e Estatística
LLM	<i>Large Language Model</i> (Modelo de Linguagem de Grande Escala)
OTAN	Organização do Tratado do Atlântico Norte
SDLC	<i>Software Development Life Cycle</i> (Ciclo de Desenvolvimento de Software)
USP	Universidade de São Paulo

# Listas de figuras

3.1	Proporção anual das linhas modificadas classificadas pelo tempo que permaneceram no repositório antes de sofrerem uma nova modificação significativa entre 2020 e 2024 (GITCLEAR, 2025) . . . . .	15
3.2	Proporção anual dos desenvolvedores profissionais que utilizam ferramentas de IA em seu processo de desenvolvimento entre 2023 e 2025 (STACK OVERFLOW, 2023; STACK OVERFLOW, 2024; STACK OVERFLOW, 2025). . . . .	16

# Lista de tabelas

3.1	Histórico de métricas de código de 2020 a 2024 (GITCLEAR, 2025) . . . . .	13
3.2	Histórico de duplicação de código nos commits de 2020 a 2024 (GITCLEAR, 2025). . . . .	14



# Sumário

<b>Introdução</b>	<b>1</b>
<b>1 Referencial Teórico</b>	<b>3</b>
1.1 Engenharia de Software . . . . .	3
1.1.1 Etapas do Desenvolvimento de Software . . . . .	3
1.1.2 Ferramentas de Desenvolvimento . . . . .	5
1.2 Inteligência Artificial . . . . .	6
1.2.1 Aprendizado de Máquina . . . . .	7
1.2.2 IA Generativa . . . . .	7
<b>2 Metodologia</b>	<b>11</b>
2.1 Abordagem de Pesquisa . . . . .	11
2.1.1 Tipo de Pesquisa . . . . .	11
2.1.2 Procedimentos Metodológicos . . . . .	11
2.2 Coleta de Dados . . . . .	11
2.2.1 Fontes de Dados . . . . .	11
2.3 Métodos de Análise . . . . .	12
<b>3 Literatura Cinzenta</b>	<b>13</b>
3.1 GitClear – AI Copilot Code Quality: Evaluating 2024's Increased Defect Rate via Code Quality Metrics . . . . .	13
3.2 Stack Overflow: Developer Survey . . . . .	14
3.3 Gartner: Principais Tendências Tecnológicas Estratégicas para 2025: IA Agêntica . . . . .	15
3.4 Gartner: Magic Quadrant for AI Code Assistants . . . . .	15
<b>4 Literatura formal</b>	<b>17</b>
<b>5 Análise comparativa</b>	<b>19</b>

<b>6 Conclusão</b>	<b>21</b>
<b>Apêndices</b>	
<b>Anexos</b>	
<b>Referências</b>	<b>23</b>
<b>Índice remissivo</b>	<b>25</b>

# Introdução

A engenharia de *software* é um campo da computação que se propõe a produzir e manter sistemas de *software*. Essa definição foi estabelecida em 1968 pela Organização do Tratado do Atlântico Norte (OTAN), direcionando esforços para resolver a chamada “crise do *software*”, um período em que o desenvolvimento de programas se tornava cada vez mais complexo e desorganizado. Desde então, diversas ferramentas, métodos e processos foram criados para possibilitar que programadores organizassem a produção de *software* e realizassem projetos complexos com maior eficiência.

No mesmo período, a humanidade presenciou diversos avanços tecnológicos, como a produção de processadores cada vez mais potentes, o barateamento do *hardware*, tornando computadores e celulares muito mais acessíveis, e a inclusão de bilhões de pessoas na *internet*, gerando uma enorme quantidade de dados sobre os diversos aspectos da vida cotidiana e virtual. Com todo esse poder computacional e a quantidade massiva de dados disponíveis, a inteligência artificial (IA) pôde se desenvolver a passos largos, culminando no surgimento da IA generativa. Diferentemente da IA tradicional, a IA generativa é capaz de “criar” conteúdos com base no que aprendeu.

Dada sua flexibilidade, a IA generativa pode ser utilizada para diversos fins, e era natural que uma de suas aplicações fosse a engenharia de *software*. Nos últimos anos, vários estudos foram publicados com o objetivo de analisar essas aplicações, discutir suas consequências e propor abordagens seguras e responsáveis para seu uso (JOHNSON e MENZIES, 2024 e TERRAGNI *et al.*, 2025).

Segundo dados da STACK OVERFLOW (2025), 80,7% dos desenvolvedores profissionais utilizam ferramentas de IA em seu processo de desenvolvimento de *software* e 4,6% desse grupo planeja utilizá-las em breve. É evidente que uma tecnologia amplamente adotada entre desenvolvedores tende a causar impactos significativos na produção de código e, nesse contexto, este trabalho faz-se relevante. Seus objetivos são: reunir dados sobre os impactos das ferramentas de IA generativa na produção de *software*, analisar a evolução da percepção dos desenvolvedores sobre o uso dessas ferramentas e comparar os resultados provenientes da literatura cinzenta e da literatura formal.

O Capítulo 1 apresenta os conceitos fundamentais para o desenvolvimento do trabalho. O Capítulo 2 descreve a metodologia adotada e as fontes utilizadas. No Capítulo 3, é apresentada uma revisão da literatura cinzenta sobre o tema, já no Capítulo 4, a revisão foca na literatura formal. O Capítulo 5 traz uma análise que compara e relaciona os estudos das duas frentes. Por fim, o Capítulo 6 conclui o trabalho.



# 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* precisam ficar claros a todos os envolvidos.

Além disso, nesta fase ocorre o estudo de viabilidade, ou seja, desenvolvedores e outros atores do projeto avaliam desafios técnicos e financeiros que possam impactar a evolução e o sucesso do sistema. 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 de execução. Ao definir papéis, responsabilidades e expectativas claras, o planejamento estabelece uma base sólida para um processo eficiente de desenvolvimento.

## Análise de Requisitos

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 ter, 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. 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, são utilizadas 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 sejam 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 contemplam 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, neste caso, o processo deve minimizar interrupções e impactos no acesso dos usuários.

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 Ferramentas de Desenvolvimento

As ferramentas de desenvolvimento de *software* oferecem suporte às etapas do SDLC. 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, integração com sistemas de controle de versão e ferramentas de IA generativa;
- **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 *Github 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*)**: permitem criar ambientes isolados e consistentes para execução do *software*, garantindo que ele funcione de maneira idêntica em diferentes máquinas. Além disso, possuem a capacidade de monitorar o desempenho e a saúde dos sistemas em produção, auxiliando na detecção precoce de problemas na 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 exigiriam inteligência humana, como reconhecimento de padrões, raciocínio, tomada de decisão, resolução de problemas e aprendizado a partir de dados. Segundo uma declaração da IEEE (*Artificial Intelligence 2019*), a inteligência artificial inclui tecnologias computacionais inspiradas no modo como pessoas e outros organismos biológicos percebem, aprendem, raciocinam e agem.

As aplicações de IA afetam cada vez mais diversos aspectos da sociedade, incluindo defesa e segurança nacional, sistemas de justiça, 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 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 importante para o aprendizado de máquina são as redes neurais, modelos computacionais inspirados na estrutura do cérebro humano, 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 Profundo

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

O processo de treinamento de uma rede neural profunda envolve a alimentação de um grande conjunto de dados. Conforme a rede processa esses dados, ela ajusta os "pesos" nas conexões entre os nós, utilizando o algoritmo de *backpropagation* para minimizar o erro e melhorar sua precisão. 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 "lembra" o que aconteceu em camadas anteriores, tornando-a ideal para tarefas de linguagem natural e análise de sentimentos.

### 1.2.2 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*” por Vaswani et al., 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 massivas 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

Esta pesquisa caracteriza-se como descritiva, dado que se fundamenta na análise de produções teóricas previamente publicadas sobre o tema. Para isso, foram utilizados livros, artigos científicos, relatórios técnicos e trabalhos acadêmicos que abordem o impacto da inteligência artificial generativa na engenharia de software, bem como suas aplicações nas fases de design, codificação e testes.

#### 2.1.2 Procedimentos Metodológicos

Quanto à sua finalidade, trata-se de uma pesquisa **básica**, pois tem como objetivo aprofundar o conhecimento científico existente sobre o tema, explorando especificamente os efeitos e implicações do uso de ferramentas de IA generativa no desenvolvimento de software. Dessa forma, não se busca propor soluções práticas imediatas, mas compreender e organizar criticamente o conhecimento já produzido.

### 2.2 Coleta de Dados

#### 2.2.1 Fontes de Dados

A seleção do material bibliográfico será feita em bases de dados científicas, como ACM Digital Library, IEEE Xplore, Scopus e Google Scholar, utilizando termos relacionados à inteligência artificial generativa, engenharia de software, ferramentas de apoio ao desenvolvimento e impactos na prática profissional.

## 2.3 Métodos de Análise

No que se refere à abordagem, este trabalho adota uma pesquisa qualitativa. Os estudos selecionados serão analisados criticamente, considerando seus objetivos, métodos, resultados e conclusões. A interpretação dos dados terá caráter descritivo e analítico, buscando identificar padrões, benefícios, desafios e lacunas nas aplicações da IA generativa no contexto da engenharia de software.

# Capítulo 3

## Literatura Cinzenta

Literatura cinzenta é o nome dado para publicações, textos e produções os quais não passaram pelo mesmo processo de revisão por pares que a chamada literatura branca, que inclui, por exemplo, publicações formais, artigos científicos e periódicos. Neste capítulo, estão listados os relatórios de literatura cinzenta utilizados na pesquisa.

### 3.1 GitClear – AI Copilot Code Quality: Evaluating 2024's Increased Defect Rate via Code Quality Metrics

Este relatório foi escrito pela empresa GitClear, que desenvolve ferramentas de análise de código-fonte para empresas de desenvolvimento de software. O relatório realizou uma análise de métricas de qualidade de código de 211 milhões de linhas alteradas em 2024, sendo dois terços destas linhas advindos do compartilhamento de dados anonimizados de empresas privadas e o restante de projetos de código livre. Os pesquisadores criaram um histórico comparando as métricas observadas em pesquisas anteriores com as métricas de 2024 e a análise é feita com foco no impacto de ferramentas de IA generativa no processo de desenvolvimento de software, considerando 2022 como o ano em que a ampla adoção da IA na programação se iniciou.

Ano	Adicionado	Deletado	Atualizado	Movido	Cópia	Substituído	Churn
2020	39,2%	19,1%	5,2%	24,1%	8,3%	2,9%	3,1%
2021	39,5%	19,3%	5,0%	24,8%	8,4%	3,4%	3,3%
2022	40,9%	19,8%	5,2%	20,5%	9,4%	3,7%	3,3%
2023	42,3%	21,1%	5,6%	15,8%	10,6%	3,6%	4,5%
2024	46,2%	21,9%	5,9%	9,5%	12,3%	4,2%	5,7%

Tabela 3.1: Histórico de métricas de código de 2020 a 2024 ([GITCLEAR, 2025](#)).

Na tabela Tabela 3.1, são apresentados os resultados das análises entre 2020 e 2024. Observa-se que a porcentagem de linhas movidas, algo associado a refatorações estruturais,

reorganização de trechos e melhorias arquiteturais, apresentou uma queda significativa, passando de 24,1% em 2020 para apenas 9,5% em 2024. Por outro lado, a tendência para as linhas copiadas foi inversa, partindo de 8,3% em 2020 para 12,3% em 2024. Este aumento de duplicações também pode ser observado na [Tabela 3.2](#), que mostra que, em 2020, apenas 0,7% dos commits analisados continham blocos de código duplicados e, em 2024, esse número subiu para 6,66%.

Ano	Tot. commits	Tot. duplicações	Commits com duplicação	Commits com duplicação (%)
2020	19.805	9.227	139	0,70%
2021	29.912	9.295	143	0,48%
2022	40.010	10.685	182	0,45%
2023	41.561	20.448	747	1,80%
2024	56.495	63.566	3.764	6,66%

**Tabela 3.2:** Histórico de duplicação de código nos commits de 2020 a 2024 ([GITCLEAR, 2025](#)).

Além disso, outra métrica relevante neste contexto é o *churn*, que busca medir quanto retrabalho é efetuado em uma base de código. Neste relatório, o *churn* é calculado medindo quanto das linhas escritas e enviadas ao repositório foram revertidas ou substancialmente revisadas nas duas semanas seguintes. A [Tabela 3.1](#) mostra que o *churn* passou de 3,1% em 2020 para 5,7% em 2024, evidenciando o aumento do retrabalho no código.

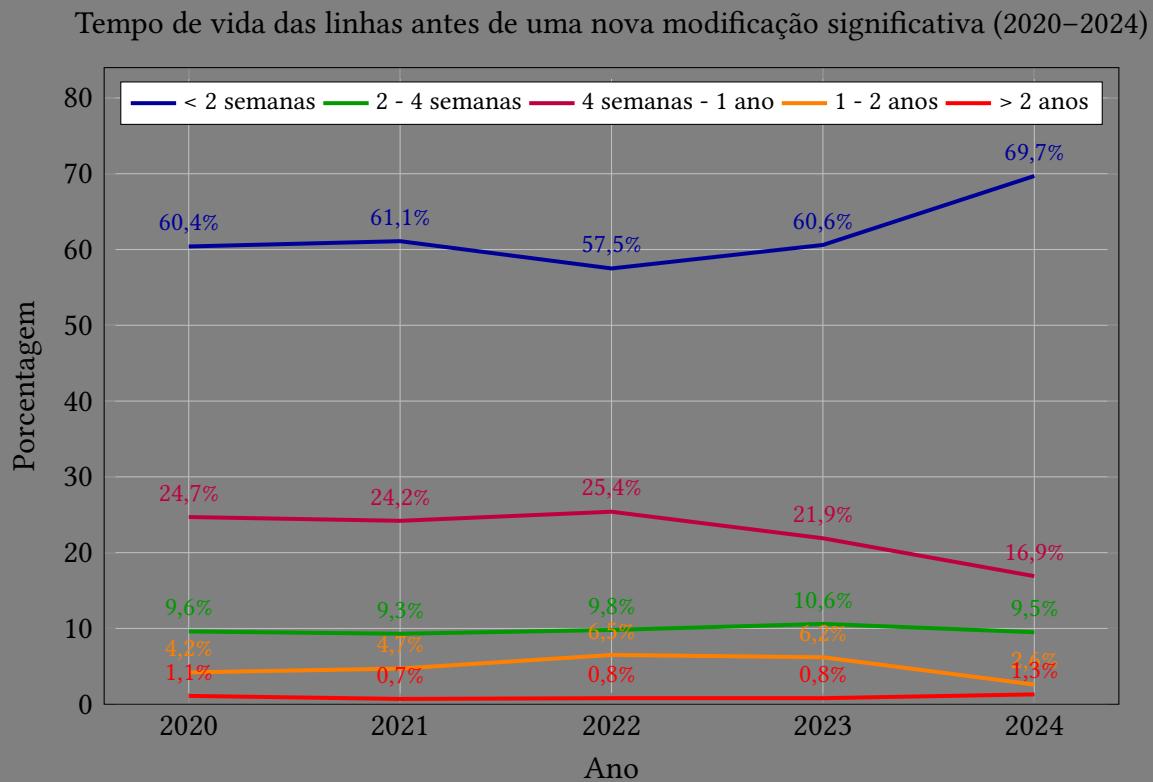
O relatório fez também uma análise de quanto tempo se passou entre o momento em que os códigos analisados foram criados e em quanto tempo eles receberam sua próxima mudança significativa. Na [Figura 3.1](#), observa-se que a proporção de linhas que em menos de duas semanas foram alteradas aumentou de 60,4% em 2020 para 69,7% em 2024, em contrapartida, o grupo das linhas que precisaram de alterações apenas depois de 4 semanas, mas antes de um ano, diminuiu de 24,7% para 16,9% no mesmo período.

Desta maneira, o relatório conclui que houve aumento da duplicação de linhas nos repositórios, violando a boa prática DRY (*Don't Repeat Yourself*), a diminuição da proporção de linhas movidas, que indicam refatoração, o aumento da instabilidade do código, já que o *churn* aumentou, e uma redução da durabilidade do código, no sentido dele precisar ser reescrito ou modificado mais cedo. O relatório credita este cenário de piora da qualidade do código e de sua manutenibilidade no longo prazo à adoção de IA no desenvolvimento de *software*.

## 3.2 Stack Overflow: Developer Survey

Esta pesquisa foi realizada pelo *Stack Overflow*, plataforma reconhecida no campo de tecnologia como meio de encontrar respostas para dúvidas relacionadas a códigos, linguagens de programação, frameworks, algoritmos e tecnologias em geral. O *Stack Overflow* realiza desde XXXX uma pesquisa anual aberta com usuários do sistema de modo a levantar dados sobre as tendências no mercado de tecnologia, impressões dos desenvolvedores, entre outros. Os entrevistados incluem desde programadores profissionais experientes até pessoas que estão iniciando seus estudos sobre programação.

3.3 | GARTNER: PRINCIPAIS TENDÊNCIAS TECNOLÓGICAS ESTRATÉGICAS PARA 2025: IA AGÊNTICA



**Figura 3.1:** Proporção anual das linhas modificadas classificadas pelo tempo que permaneceram no repositório antes de sofrerem uma nova modificação significativa entre 2020 e 2024 ([GITCLEAR, 2025](#)).

Os resultados da pesquisa em 2024 incluem uma seção específica para tratar a IA e seus impactos no mercado. Dentre os respondentes, 61,8% utiliza ferramentas de IA em seus processos de desenvolvimento de código e outros 13,8% planeja utilizar, além disso, 72% é favorável ou muito favorável ao uso de IA.

4343,268,3

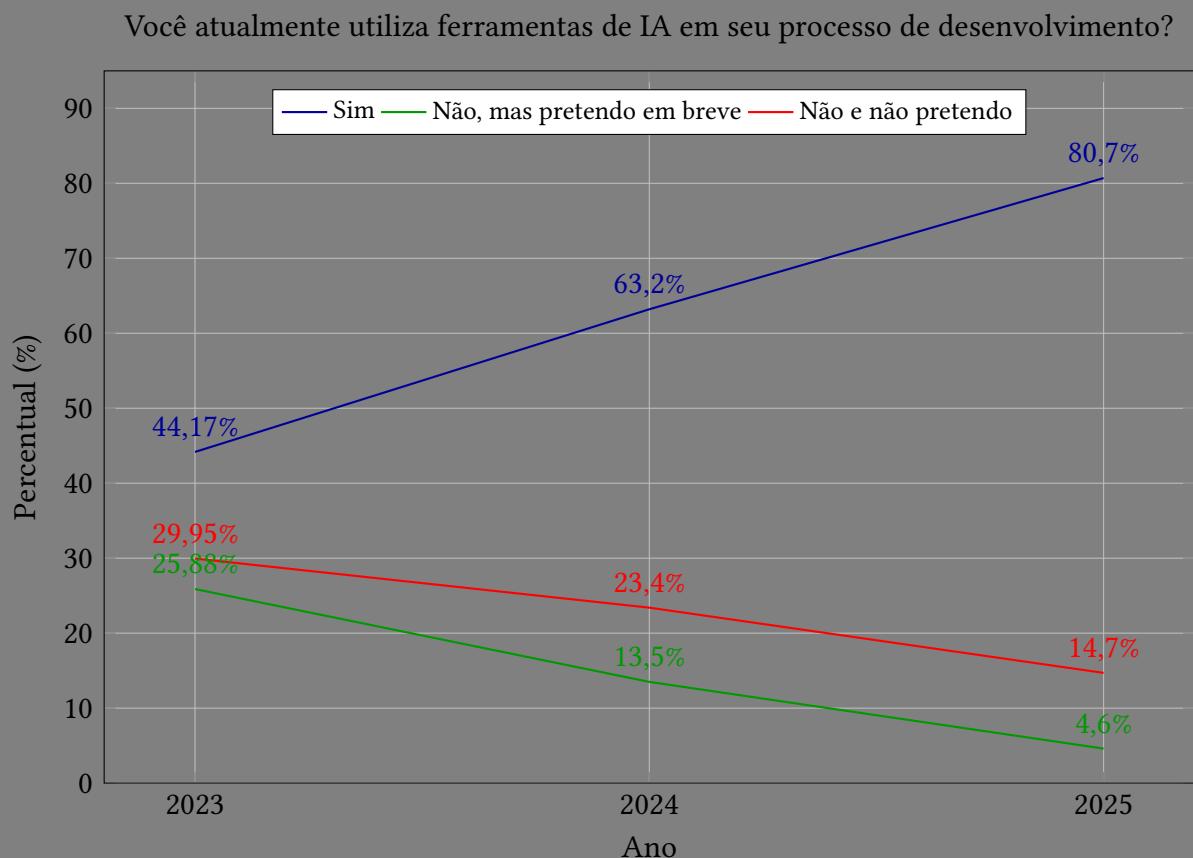
Descrição do uso Benefícios Principais usos

### 3.3 Gartner: Principais Tendências Tecnológicas Estratégicas para 2025: IA Agêntica

Este relatório

### 3.4 Gartner: Magic Quadrant for AI Code Assistants

Este relatório



**Figura 3.2:** Proporção anual dos desenvolvedores profissionais que utilizam ferramentas de IA em seu processo de desenvolvimento entre 2023 e 2025 (*STACK OVERFLOW, 2023; STACK OVERFLOW, 2024; STACK OVERFLOW, 2025*).

# **Capítulo 4**

## **Literatura formal**



## Capítulo 5

### Análise comparativa



## **Capítulo 6**

### **Conclusão**



# 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. 6).
- [GITCLEAR 2025] GITCLEAR. *AI Copilot Code Quality: Evaluating 2024's Increased Defect Rate via Code Quality Metrics*. Rel. técn. Acessado em: 17 nov. 2025. GitClear, 2025. URL: <https://gitclear-public.s3.us-west-2.amazonaws.com/AI-Copilot-Code-Quality-2025.pdf> (citado nas pgs. 13–15).
- [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](https://doi.org/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](https://doi.ieeecomputersociety.org/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. 6).
- [STACK OVERFLOW 2023] STACK OVERFLOW. *Stack Overflow Developer Survey 2023*. 2023. URL: <https://survey.stackoverflow.co/2023/> (acesso em 19/08/2025) (citado na pg. 16).
- [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. 16).
- [STACK OVERFLOW 2025] STACK OVERFLOW. *Stack Overflow Developer Survey 2025*. 2025. URL: <https://survey.stackoverflow.co/2025/> (acesso em 19/08/2025) (citado nas pgs. 1, 16).

[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](https://doi.org/10.1145/3715003). URL: <https://doi.org/10.1145/3715003> (citado na pg. 1).

# Índice remissivo

## C

Captions, *veja* Legendas

Código-fonte, *veja* Floats

## E

Equações, *veja* Modo matemático

## F

Figuras, *veja* Floats

Floats

Algoritmo, *veja* Floats, ordem

Fórmulas, *veja* Modo matemático

## I

Inglês, *veja* Língua estrangeira

## P

Palavras estrangeiras, *veja* Língua es-

trangeira

## R

Rodapé, notas, *veja* Notas de rodapé

## S

Subcaptions, *veja* Subfiguras

Sublegendas, *veja* Subfiguras

## T

Tabelas, *veja* Floats

## V

Versão corrigida, *veja* Tese/Dissertação,  
versões

Versão original, *veja* Tese/Dissertação,  
versões