

O fim da programação, de novo!

Paulo Borba

Centro de Informática

Universidade Federal de Pernambuco

pauloborba.cin.ufpe.br

The End of Programming

By Matt Welsh

Communications of the ACM, January 2023, Vol. 66 No. 1, Pages 34-35

10.1145/3570220

[Comments \(13\)](#)

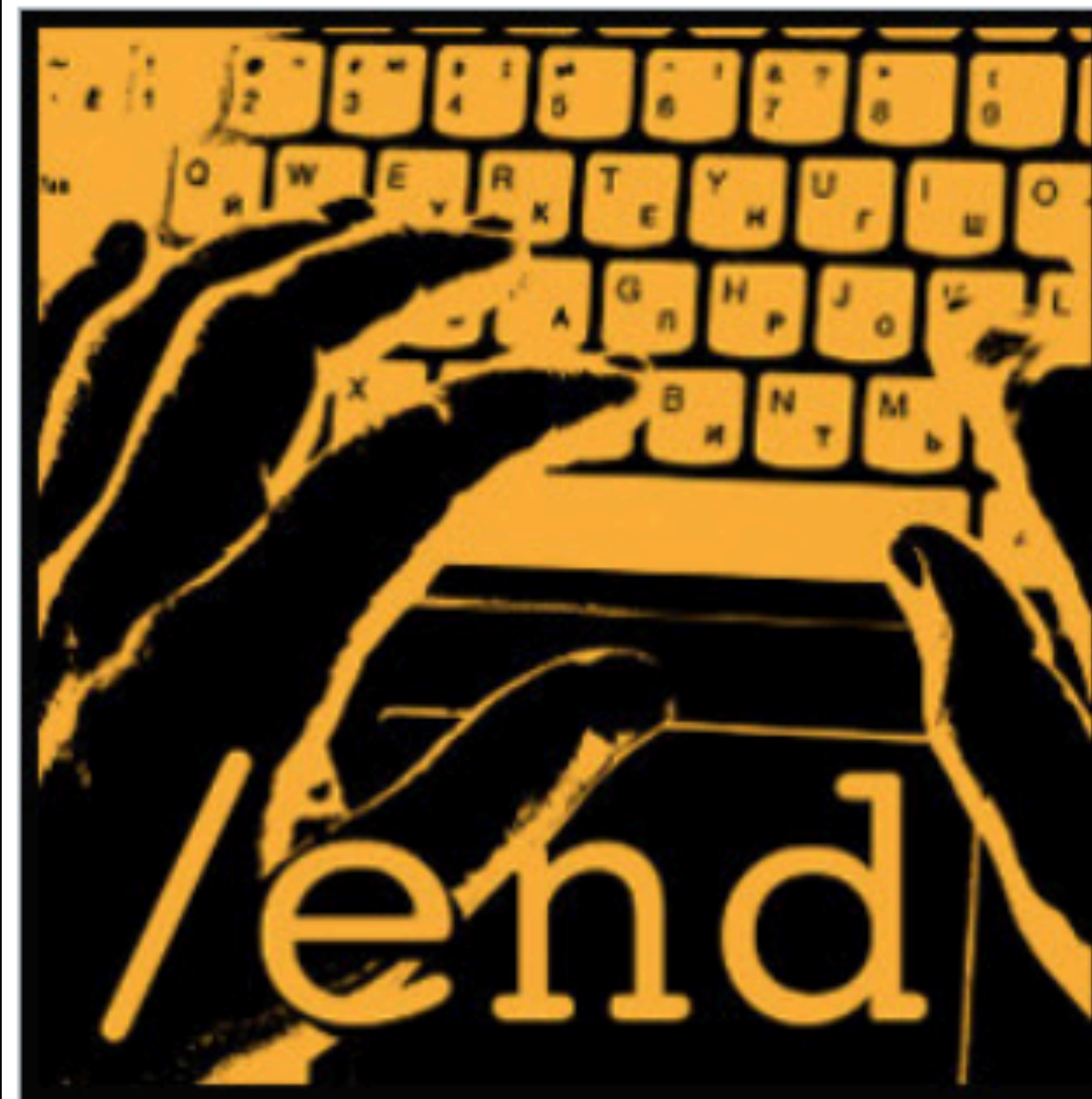
VIEW AS:



SHARE:



O fim da
programação!

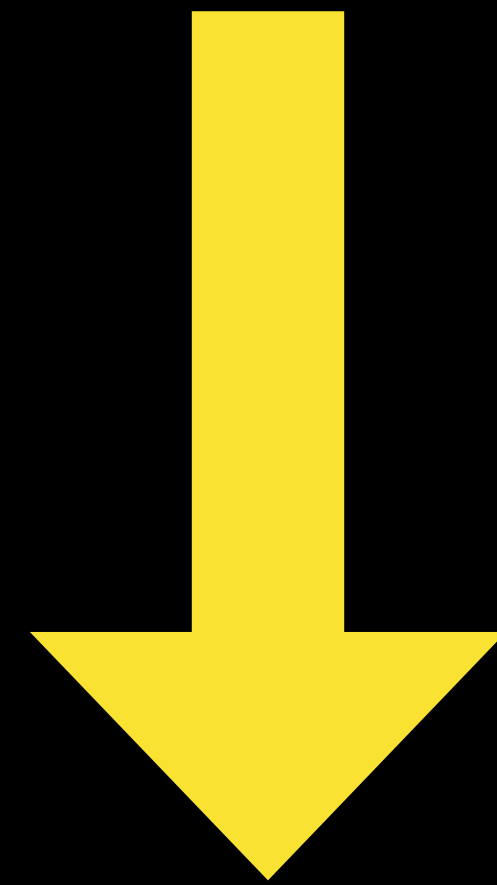


Credit: Ursa Major

I came of age in the 1980s, programming personal computers such as the Commodore VIC-20 and Apple][e at home. Going on to study computer science (CS) in college and ultimately getting a Ph.D. at Berkeley, the bulk of my professional training was rooted in what I will call "classical" CS: programming, algorithms, data structures, systems, programming languages. In Classical Computer Science, the ultimate goal is to reduce an idea to a program written by a human—source code in a language like Java or C++ or Python. Every idea in Classical CS—no matter how complex or sophisticated, from a database join algorithm to the mind-bogglingly obtuse Paxos consensus protocol—can be expressed as a human-readable, human-comprehensible program.

Motivo 1

implementação de algoritmos



treinamento de modelos

A função desejada não é **implementada** por algoritmos e estruturas de dados

Ela é **aproximada** por modelos de aprendizado de máquina devidamente treinados

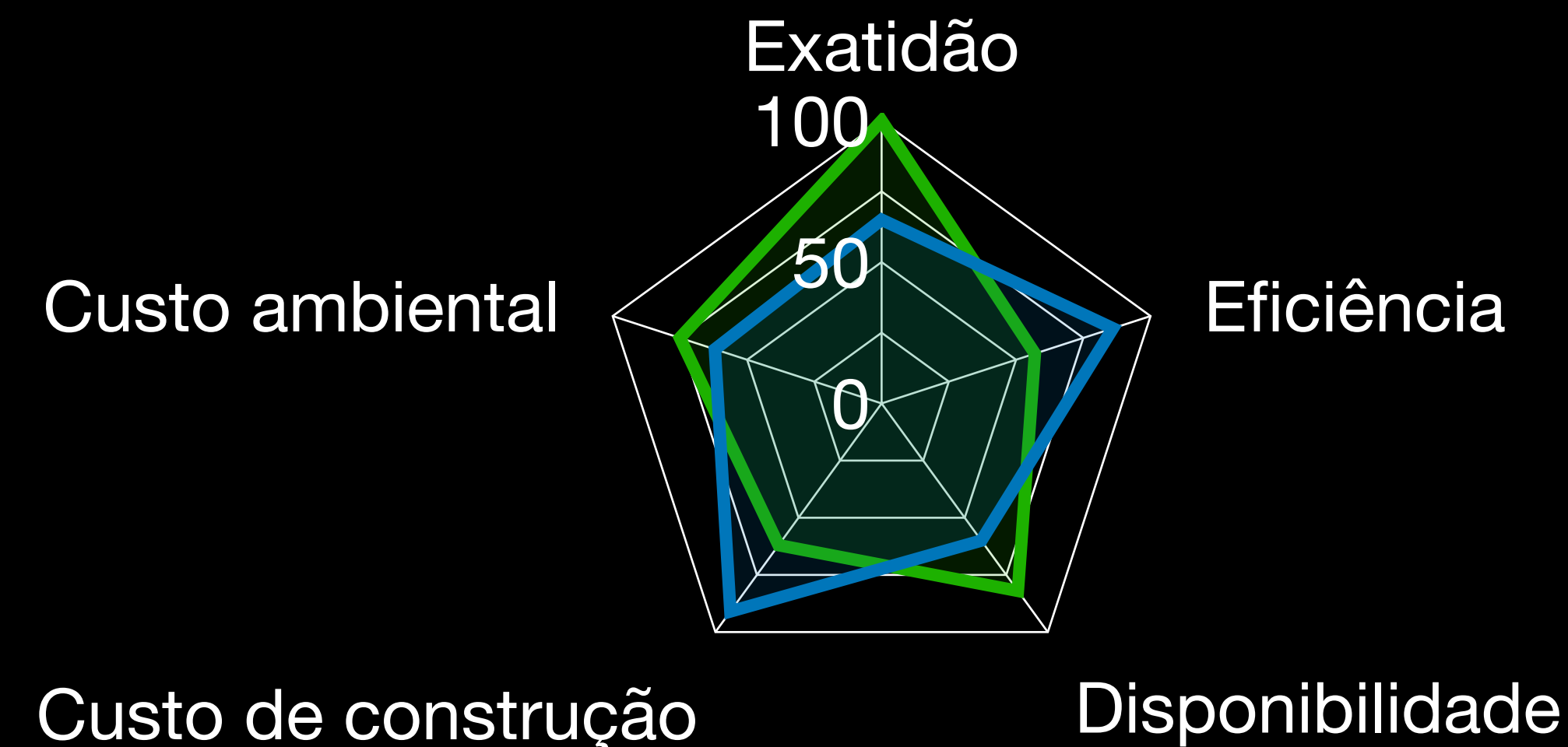
(com dados que registram valores pontuais esperados de entrada e saída da função)

Opção mais adequada para várias situações, mas não para várias outras

(ao contrário do que o artigo dá a entender!)

A decisão de que **técnica** usar é multidimensional, envolve várias forças

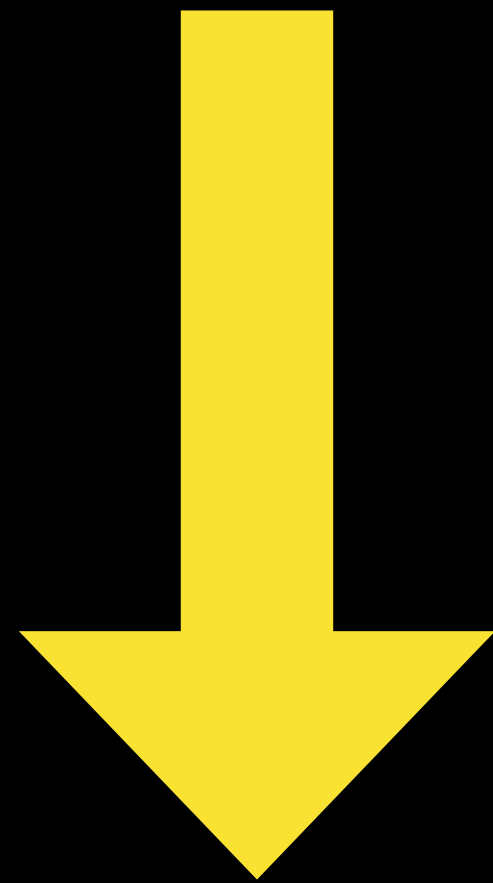
- adequação ao problema!
- exatidão (tolerância a imprecisões)
- eficiência computacional
- disponibilidade de especialistas, e de dados
- custo de construção da solução
- custo ambiental
- privacidade
- etc.



Essencial dominar as duas técnicas, ao invés de esperar que uma vá substituir a outra

Motivo 2

Pessoas programadoras



LLMs
(modelos de linguagem)

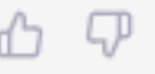
ChatGPT (3, free) quase lá

Por favor, escreva um programa Hello World em Rust que mostra também o horário em que ele é executado

Nem sempre acerta, errou pela proximidade das palavras



Claro! Aqui está um exemplo de programa Hello World em Rust que também exibe o horário em que é executado:



```
rust Copy code

use std::time::SystemTime;

fn main() {
    // Obter a hora atual do sistema
    let current_time = SystemTime::now();

    // Imprimir a mensagem "Hello, world!" na saída padrão
    println!("Hello, world!");

    // Imprimir a hora atual na saída padrão
    match current_time.elapsed() {
        Ok(elapsed) => {
            let seconds = elapsed.as_secs();
            let nanos = elapsed.subsec_nanos();
            println!("Execution time: {:.:09} seconds", seconds, nanos);
        }
        Err(e) => {
            println!("Error: {:?}", e);
        }
    }
}
```

Este programa usa a biblioteca padrão do Rust para obter a hora atual do sistema, que é armazenada na variável `current_time`. Em seguida, ele imprime a mensagem "Hello, world!" na saída padrão usando a macro `println!()`. Finalmente, ele usa a função `elapsed()` na variável `current_time` para obter o tempo decorrido desde o início do programa e imprime esse tempo na saída padrão como uma string formatada.

Espero que isso ajude!

ChatGPT, acertou (quase)!

Eu não gostaria que ele imprimisse o tempo de execução, mas sim o horário em que ele começou a executar

Combinou
códigos repetitivos



Sem problemas! Aqui está o exemplo atualizado que mostra o horário em que o programa começou a ser executado:



```
rust Copy code

use std::time::{SystemTime, UNIX_EPOCH};

fn main() {
    // Obter a hora atual do sistema
    let current_time = SystemTime::now();

    // Obter a hora em UNIX timestamp
    let unix_timestamp = current_time.duration_since(UNIX_EPOCH).unwrap().as_secs();

    // Converter o timestamp em uma data/hora legível
    let datetime = chrono::NaiveDateTime::from_timestamp(unix_timestamp as i64, 0).unwrap();
    let formatted_datetime = datetime.format("%Y-%m-%d %H:%M:%S");

    // Imprimir a mensagem "Hello, world!" na saída padrão
    println!("Hello, world!");

    // Imprimir a hora em que o programa começou a ser executado na saída padrão
    println!("Started at: {}", formatted_datetime);
}
```

Este programa também usa a biblioteca padrão do Rust para obter a hora atual do sistema, que é armazenada na variável `current_time`. Em seguida, ele usa a constante `UNIX_EPOCH` para calcular o tempo decorrido desde o Unix Epoch até a hora atual em segundos. O resultado é armazenado na variável `unix_timestamp`.

Depois, o programa usa a biblioteca `chrono` para converter o timestamp em uma data/hora legível, que é armazenada na variável `formatted_datetime`. Por fim, ele imprime a mensagem "Hello, world!" na saída padrão e imprime a hora em que o programa começou a ser executado na saída padrão usando a variável `formatted_datetime`.

Espero que isso ajude!

```
use std::time::{SystemTime, UNIX_EPOCH};
use chrono;

fn main() {
    // Obter a hora atual do sistema
    let current_time = SystemTime::now();

    // Obter a hora em UNIX timestamp
    let unix_timestamp = current_time.duration_since(UNIX_EPOCH).unwrap().as_secs();

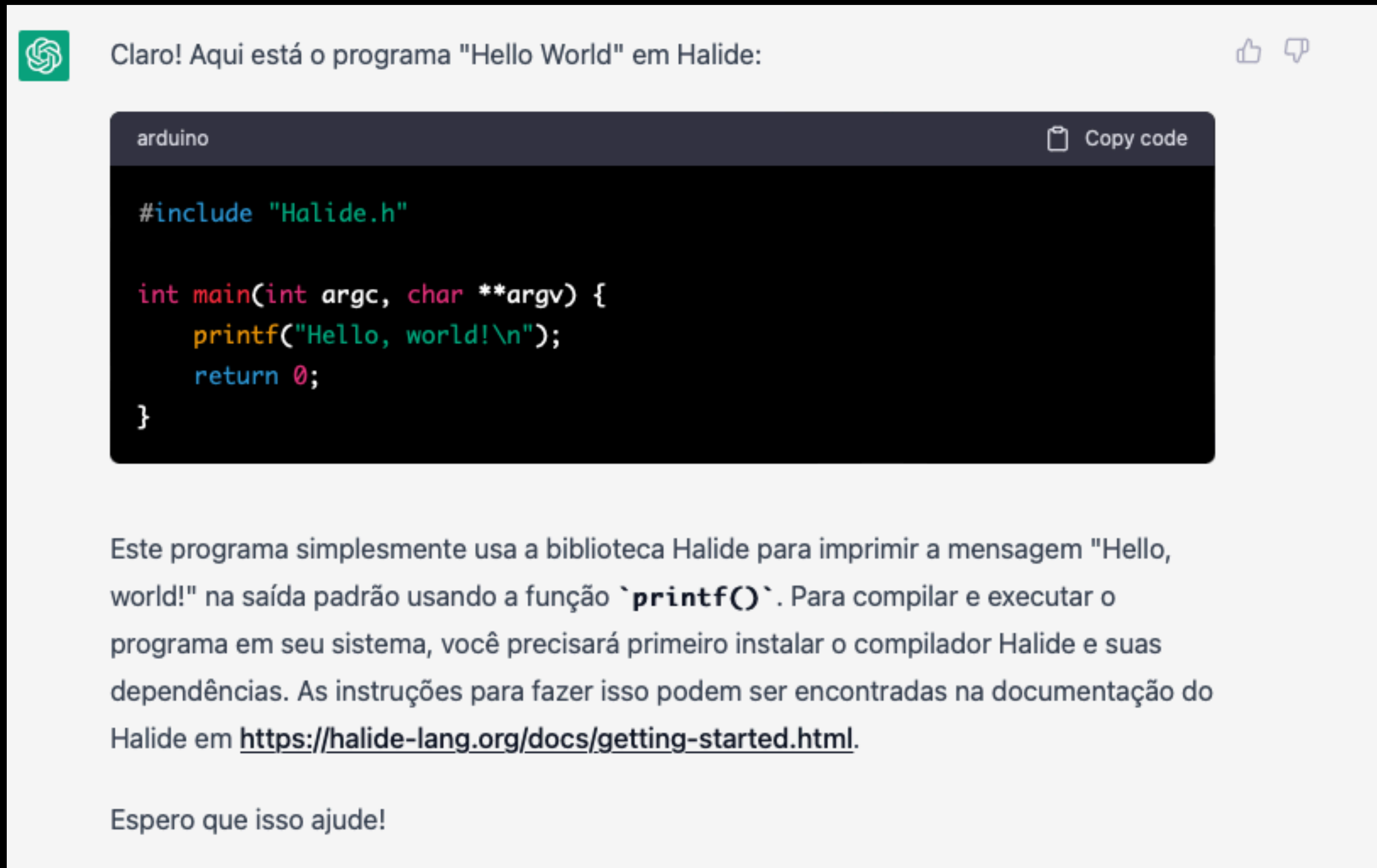
    // Converter o timestamp em uma data/hora legível
    let datetime = chrono::NaiveDateTime::from_timestamp_opt(unix_timestamp as i64, 0);
    let formatted_datetime = datetime.expect("REASON").format("%Y-%m-%d %H:%M:%S");

    // Imprimir a mensagem "Hello, world!" na saída padrão
    println!("Hello, world!");

    // Imprimir a hora em que o programa começou a ser executado na saída padrão
    println!("Started at: {}", formatted_datetime);
}
```

ChatGPT, errou (muito!)

Você pode traduzir este programa para Halide?



Claro! Aqui está o programa "Hello World" em Halide:

```
arduino Copy code  
  
#include "Halide.h"  
  
int main(int argc, char **argv) {  
    printf("Hello, world!\n");  
    return 0;  
}
```

Este programa simplesmente usa a biblioteca Halide para imprimir a mensagem "Hello, world!" na saída padrão usando a função `printf()`. Para compilar e executar o programa em seu sistema, você precisará primeiro instalar o compilador Halide e suas dependências. As instruções para fazer isso podem ser encontradas na documentação do Halide em <https://halide-lang.org/docs/getting-started.html>.

Espero que isso ajude!

Só conhece o que é mais frequente, comum, e
que foi usado para treinamento

ChatGPT, acertou

Mostre o programa Java correspondente ao último programa Rust que você gerou



Claro! Aqui está o programa Java correspondente ao último programa Rust que eu gerei:



java

Copy code

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class HelloWorld {

    public static void main(String[] args) {
        // Obter a hora atual do sistema
        LocalDateTime currentTime = LocalDateTime.now();

        // Converter a hora atual em uma string formatada
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        String formattedDateTime = currentTime.format(formatter);

        // Imprimir a mensagem "Hello, world!" na saída padrão
        System.out.println("Hello, world!");

        // Imprimir a hora em que o programa começou a ser executado na saída padrão
        System.out.println("Started at: " + formattedDateTime);
    }
}
```

Este programa usa as classes `LocalDateTime` e `DateTimeFormatter` da biblioteca padrão do Java para obter a hora atual do sistema e formatá-la como uma string legível. Em seguida, ele usa a função `System.out.println()` para imprimir a mensagem "Hello, world!" na saída padrão e a hora em que o programa começou a ser executado.

Para compilar e executar o programa em sua máquina, você precisará ter o JDK (Java Development Kit) instalado. Em seguida, siga estes passos:

1. Copie o código acima para um arquivo chamado `HelloWorld.java`
2. Abra um terminal ou prompt de comando e navegue até o diretório em que o arquivo `.java` foi salvo
3. Compile o arquivo `.java` digitando `javac HelloWorld.java` no terminal
4. Execute o programa resultante digitando `java HelloWorld` no terminal

Quando você executar o programa, verá a mensagem "Hello, world!" seguida da hora em que o programa começou a ser executado, no formato `YYYY-MM-DD HH:MM:SS`.

Espero que isso ajude!

Pode gerar **código complexo**,
por partes,
sendo **devidamente guiado**,
mas tendo **riscos e problemas**
similares aos ilustrados

```
def scan_string(source, line):
    string = ""
    while (len(source) > 0 and source[0] != '\n'):
        c = source[0]
        source = source[1:]
        if c == '\n':
            line += 1
        else:
            string += c
    if len(source) == 0:
        raise ValueError("Unterminated string at line {}".format(line))
    source = source[1:]
    return Token(TokenType.STRING, string, line), source, line
```

```
class TokenType(Enum):
    # Single character tokens
    LEFT_PAREN = '('
    RIGHT_PAREN = ')'
    LEFT_BRACE = '{'
    RIGHT_BRACE = '}'
    COMMA = ','
    DOT = '.'
    MINUS = '-'
    PLUS = '+'
    SEMICOLON = ';'
    SLASH = '/'
    STAR = '*'
```


Um tipo de conflito semântico de merge ocorre quando um programador altera um comando de um método, outro programador altera outro comando do mesmo método, e o primeiro comando atualiza o valor de uma variável que é lida pelo segundo. Escreva um programa Java que detecta esse tipo de conflito semântico. Isso possivelmente pode ser implementado mais facilmente utilizando uma análise estática de fluxo de dados.

ChatGPT 4

Detectar conflitos semânticos de merge pode ser um desafio, pois envolve a análise do fluxo de dados e a compreensão do código em um nível mais profundo. Neste exemplo, utilizaremos a biblioteca WALA, que fornece uma infraestrutura para análise estática de programas em Java. Antes de começar, adicione as dependências necessárias ao seu projeto. Para o Maven, por exemplo, adicione o seguinte ao seu arquivo pom.xml:

```
xml Copy code

<dependencies>
  <dependency>
    <groupId>com.ibm.wala</groupId>
    <artifactId>com.ibm.wala.core</artifactId>
    <version>1.5.6</version>
  </dependency>
  <dependency>
    <groupId>com.ibm.wala</groupId>
    <artifactId>com.ibm.wala.shrike</artifactId>
    <version>1.5.6</version>
  </dependency>
  <dependency>
    <groupId>com.ibm.wala</groupId>
    <artifactId>com.ibm.wala.cast.java</artifactId>
    <version>1.5.6</version>
  </dependency>
</dependencies>
```

Em seguida, siga o exemplo abaixo para implementar um programa Java que detecta conflitos semânticos de merge:

```
java Copy code

import com.ibm.wala.cast.java.translator.jdt.JDTJava2CAstTranslator;
import com.ibm.wala.cast.java.translator.jdt.ecj.ECJClassLoaderFactory;
import com.ibm.wala.classLoader.*;
import com.ibm.wala.ipa.callgraph.*;

Regenerate response
```

```
import com.ibm.wala.ipa.callgraph.AnalysisOptions;
import com.ibm.wala.ipa.callgraph.AnalysisScope;
import com.ibm.wala.ipa.callgraph.CallGraphBuilder;
import com.ibm.wala.ipa.callgraph.CallGraphBuilderCancelException;
import com.ibm.wala.ipa.callgraph.impl.Util;
import com.ibm.wala.ipa.cha.ClassHierarchy;
import com.ibm.wala.ipa.cha.ClassHierarchyException;
import com.ibm.wala.ipa.cha.IClassHierarchy;
import com.ibm.wala.ssa.*;
import com.ibm.wala.types.ClassLoaderReference;
import com.ibm.wala.util.config.FileOfClasses;
import java.io.IOException;
import java.util.Collection;
import java.util.Collections;

public class SemanticMergeConflictDetector {

    public static void main(String[] args) throws IOException, ClassHierarchyException {
        if (args.length < 1) {
            System.err.println("Usage: java SemanticMergeConflictDetector <path_to_jar>");
            System.exit(1);
        }

        String javaClassPath = args[0];

        // Create an analysis scope and configure it
        AnalysisScope scope = AnalysisScope.createJavaAnalysisScope();
        scope.addToScope(ClassLoaderReference.Primordial, new FileOfClasses());
        scope.addToScope(ClassLoaderReference.Application, new ECJClassLoaderFactory());

        // Build class hierarchy
        IClassHierarchy cha = ClassHierarchy.make(scope);

        // Set up analysis options and build call graph
        AnalysisOptions options = new AnalysisOptions(scope, null);
        CallGraphBuilder<?> builder = Util.makeZeroOneContainerCFABuilder(Language.JAVA, options);
        builder.buildCallGraph();

        // Iterate over methods in the call graph
        for (IClass clazz : cha) {
            for (IMethod method : clazz.getDeclaredMethods()) {
                if (method instanceof SSAConversion) {
                    SSAConversion ssa
```

```
java Copy code

        SSAConversion ssaMethod = (SSAConversion) method;
        IR ir = builder.getAnalysisCache().getIRFactory().makeIR(ssaMethod);
        SSACFG cfg = ir.getControlFlowGraph();

        // Analyze data dependencies in the method
        analyzeDataDependencies(cfg);
    }
}

private static void analyzeDataDependencies(SSACFG cfg) {
    for (SSACFG.BasicBlock block : cfg) {
        for (SSAInstruction instruction : block.getAllInstructions()) {
            if (instruction instanceof SSAPutInstruction || instruction instanceof SSAStoreInstruction) {
                // If the instruction writes to a variable, track the variable's value
                analyzeWriteInstruction(instruction, cfg);
            } else if (instruction instanceof SSAGetInstruction || instruction instanceof SSAFieldInstruction) {
                // If the instruction reads from a variable, check if the value has been updated
                analyzeReadInstruction(instruction, cfg);
            }
        }
    }
}

private static void analyzeWriteInstruction(SSAInstruction instruction, SSACFG cfg) {
    // TODO: Track the variable's value and the instruction that updates it
}

private static void analyzeReadInstruction(SSAInstruction instruction, SSACFG cfg) {
    // TODO: Check if the value has been updated by another instruction
}
}
```

In the code above, we have implemented a basic structure for analyzing data dependencies within methods. The `analyzeDataDependencies` method iterates through the basic blocks of the control flow graph (CFG) and analyzes each instruction. If an instruction writes to a

Pode **ajudar**, mas não **substituir**
pessoas programadoras,
principalmente as mais experientes

(ao contrário do que o artigo dá a entender!)

É preciso **verificar** (corretude,
eficiência, segurança, etc.),
complementar, refinar,
modularizar, refatorar, integrar, e
alterar o código gerado

Principalmente, é preciso **criar** abstrações do domínio, arquitetura, estrutura modular

Importante aprender a usar como
um **co-piloto**, o seu par da
programação em pares, mas ainda
não está claro até que ponto...

ferramenta útil, mas imprecisa e sem garantia de
critério de qualidade, para ajudar na escrita de
pequenas partes do código

**Substituto
do stack
overflow**

**Gerador de
código
repetitivo**

**Interpretador de
linguagem natural
para programação
passo-a-passo**

risco

**Criar APIs,
abstrações, etc.**

(ainda sem evidência científica)

Escrever código é difícil, mas o mais difícil da programação é **entender** como o programa executa, e **decidir** o que mudar para que ele funcione como esperado

Tendo um LLM para **escrever** código para você, pode tornar o **entendimento** mais difícil!

Possivelmente com menor **produtividade** devido a longos ciclos de **tentativas e erros**, ao invés de **planejamento e verificação**

The End of Programming

By Matt Welsh

Communications of the ACM, January 2023, Vol. 66 No. 1, Pages 34-35

10.1145/3570220

[Comments \(13\)](#)

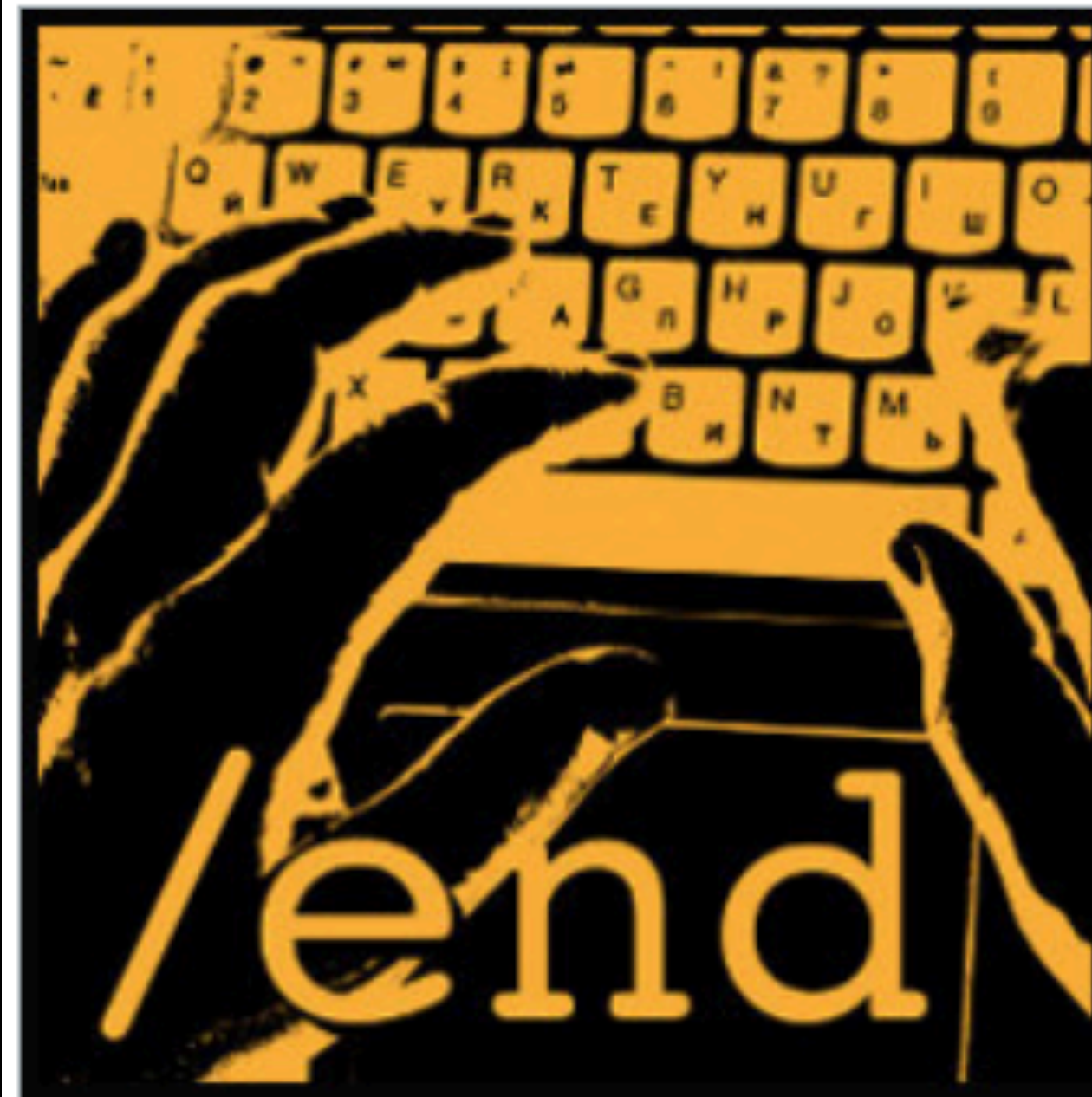
VIEW AS:



SHARE:



O fim da
programação,
de novo!



Credit: Ursa Major

I came of age in the 1980s, programming personal computers such as the Commodore VIC-20 and Apple][e at home. Going on to study computer science (CS) in college and ultimately getting a Ph.D. at Berkeley, the bulk of my professional training was rooted in what I will call "classical" CS: programming, algorithms, data structures, systems, programming languages. In Classical Computer Science, the ultimate goal is to reduce an idea to a program written by a human—source code in a language like Java or C++ or Python. Every idea in Classical CS—no matter how complex or sophisticated, from a database join algorithm to the mind-bogglingly obtuse Paxos consensus protocol—can be expressed as a human-readable, human-comprehensible program.

For almost 40 years, people have been anticipating and writing about “automatic programming”, the generation of a program for solving a problem from a statement of the problem specifications.

For almost 40 years, people have been anticipating and writing about “automatic programming”, the generation of a program for solving a problem from a statement of the problem specifications.

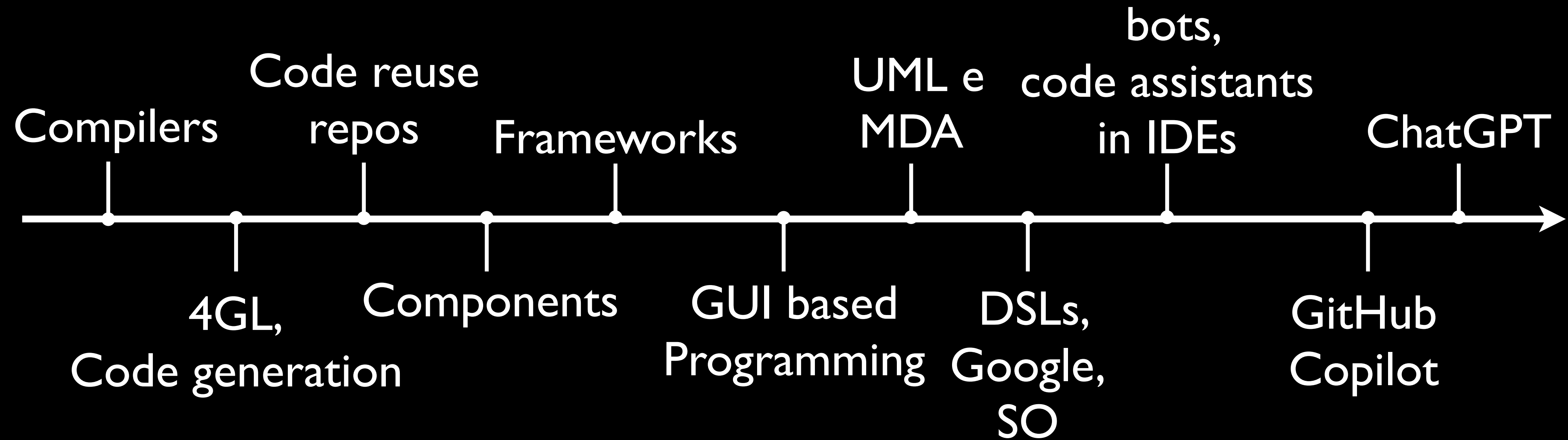
No Silver Bullet
—Essence and Accident in Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

1986

Narrativa de quase 70 anos!



**As previsões estavam
erradas?**

A nova previsão está certa?

Sinalizam apenas **mudança na forma** como a programação é feita...

aumentando o **nível de abstração**

LLMs trazem maior abstração via...

Escolha de
modelos,
coleta e
limpeza de
dados,
treinamento

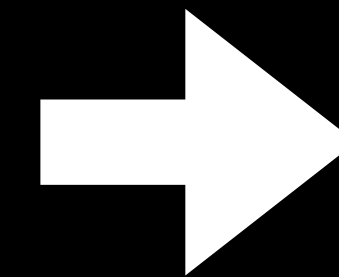
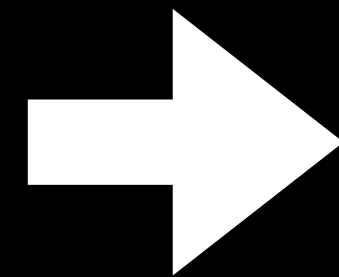
Uso de
linguagem
natural para
gerar, reusar e
combinar
código

e outras
automações

Para entender o **potencial** e as **limitações** dos LLMs, precisamos entender um pouco sobre eles...

LLMs (Large Language Models), um *token* por vez

The best thing about AI
is its ability to



learn	4.5%
predict	3.5%
make	3.2%
understand	3.1%
do	2.9%

De onde vem as probabilidades?

Bilhões de documentos, programas, etc.

algoritmo de aprendizado

LLM

learn	4.5%
predict	3.5%
make	3.2%
understand	3.1%
do	2.9%

The best thing about AI is its ability to

De onde vem as probabilidades?

Bilhões de documentos, programas, etc.

algoritmo de aprendizado

ocorrências observadas de palavras

tokens

semanticamente relacionados

The best thing about AI is its ability to

LLM

learn	4.5%
predict	3.5%
make	3.2%
understand	3.1%
do	2.9%

**Não tem mágica, nem consciência,
justifica os problemas que ilustramos
com ChatGPT**

Stochastic parrot

PH

Eu vou ministrar uma palestra técnica e preparei o seguinte resumo: O fim da programação já foi anunciado várias vezes. Dessa vez, o fim é sugerido pelo surgimento dos LLMs (Large Language Models) e de suas variações para código, disponibilizados através de ferramentas como ChatGPT e GitHub Copilot, que mostram significativa capacidade de gerar código a partir de descrições em linguagem natural. Nesta palestra, vamos apresentar a ideia geral dessas ferramentas, comparar com abordagens anteriores para geração de código, discutir como as mesmas podem ser usadas, e se o fim da programação está de fato próximo.



Sua palestra parece ser bastante interessante e atual, abordando um tema que desperta curiosidade em muitas pessoas: o impacto dos LLMs na programação.



É importante destacar que, embora as ferramentas como ChatGPT e GitHub Copilot possam ser úteis para gerar código a partir de descrições em linguagem natural, elas não substituem completamente a necessidade de programação tradicional. Os LLMs ainda estão em desenvolvimento e têm limitações em relação à complexidade e à precisão do código gerado.

PH

Você poderia, por favor, melhorar este resumo?

Nesta palestra, iremos apresentar a ideia geral dos LLMs e discutir sua evolução até os dias atuais, bem como comparar com abordagens anteriores para geração de código. Vamos analisar as possibilidades que essas ferramentas oferecem para o desenvolvimento de software, além de discutir seus limites atuais e o que pode ser esperado do futuro. Também iremos explorar como as habilidades tradicionais de programação e a capacidade de solucionar problemas ainda são fundamentais para lidar com sistemas complexos e garantir a qualidade do software.

Em resumo, esta palestra busca explorar o impacto dos LLMs na programação, bem como suas possíveis implicações para o futuro do desenvolvimento de software. Iremos destacar os prós e contras de cada abordagem e fornecer uma visão abrangente e imparcial sobre o tema.

Nem sempre escolhe o *token*
mais provável... depende da
temperatura

Não-determinismo

“Criatividade”,
possibilidade de
geração



Claro! Aqui está uma sugestão de estrutura para a sua palestra:



1. Introdução: Apresentação do tema e dos objetivos da palestra. Contextualização sobre a evolução da programação e a chegada dos LLMs.
2. O que são LLMs: Explicação sobre o conceito de Large Language Models e como eles funcionam. Exemplos de LLMs voltados para a geração de código, como o ChatGPT e o GitHub Copilot.
3. Comparação com abordagens anteriores: Discussão sobre como os LLMs se comparam a abordagens anteriores para geração de código, como o uso de templates ou algoritmos de geração de código. Análise das vantagens e desvantagens de cada abordagem.
4. Capacidades e limitações dos LLMs: Análise das capacidades atuais dos LLMs e das limitações em relação à precisão e complexidade do código gerado. Discussão sobre como os LLMs podem ser utilizados na prática.
5. O futuro dos LLMs: Perspectivas sobre o futuro dos LLMs e como eles podem evoluir nos próximos anos. Discussão sobre os possíveis impactos dos LLMs na programação e no mercado de trabalho.
6. Conclusão: Recapitulação dos principais pontos da palestra e reflexão sobre a importância da programação tradicional e das habilidades de solução de problemas. Encerramento com espaço para perguntas e discussão com o público.



Sim, aqui está uma sugestão de estrutura para sua palestra:



I. Introdução

- Breve apresentação do tema e dos objetivos da palestra

II. O que são LLMs?

- Conceituação básica de LLMs
- Exemplos de ferramentas LLM para geração de código
- Demonstração de como funcionam as ferramentas ChatGPT e GitHub Copilot

III. Evolução dos LLMs e comparação com abordagens anteriores

- Breve histórico da evolução dos LLMs
- Comparação com abordagens anteriores para geração de código
- Limitações atuais dos LLMs em relação à precisão e complexidade do código gerado

IV. Possibilidades e implicações dos LLMs para o desenvolvimento de software

- Exemplos de casos de uso para ferramentas LLM
- Debate sobre as possíveis implicações para o futuro do desenvolvimento de software
- Discussão sobre como as habilidades tradicionais de programação e a capacidade de solucionar problemas ainda são fundamentais

V. Conclusão

- Recapitulação dos principais pontos abordados na palestra
- Considerações finais e reflexão sobre o papel dos LLMs na programação

VI. Perguntas e respostas

- Espaço para perguntas e debate com a plateia

VII. Encerramento

- Agradecimentos e considerações finais.

PH

Você pode preparar o resumo do Profile do palestrante, o professor Paulo Borba



Com certeza! Segue abaixo uma sugestão de resumo atualizado do perfil do professor Paulo Borba, incluindo informações sobre sua formação:



O professor Paulo Borba é um renomado especialista em engenharia de software, com mais de 30 anos de experiência em pesquisa e ensino. Ele é bacharel em Ciência da Computação pela Universidade Federal de

Pernambuco (UFPE), mestre em Ciência da Computação pela Universidade de Maryland (EUA) e doutor em Ciência da Computação pela Universidade de Lancaster (Reino Unido).



Com certeza! Aqui está uma versão atualizada do resumo com a formação do professor Paulo Borba:

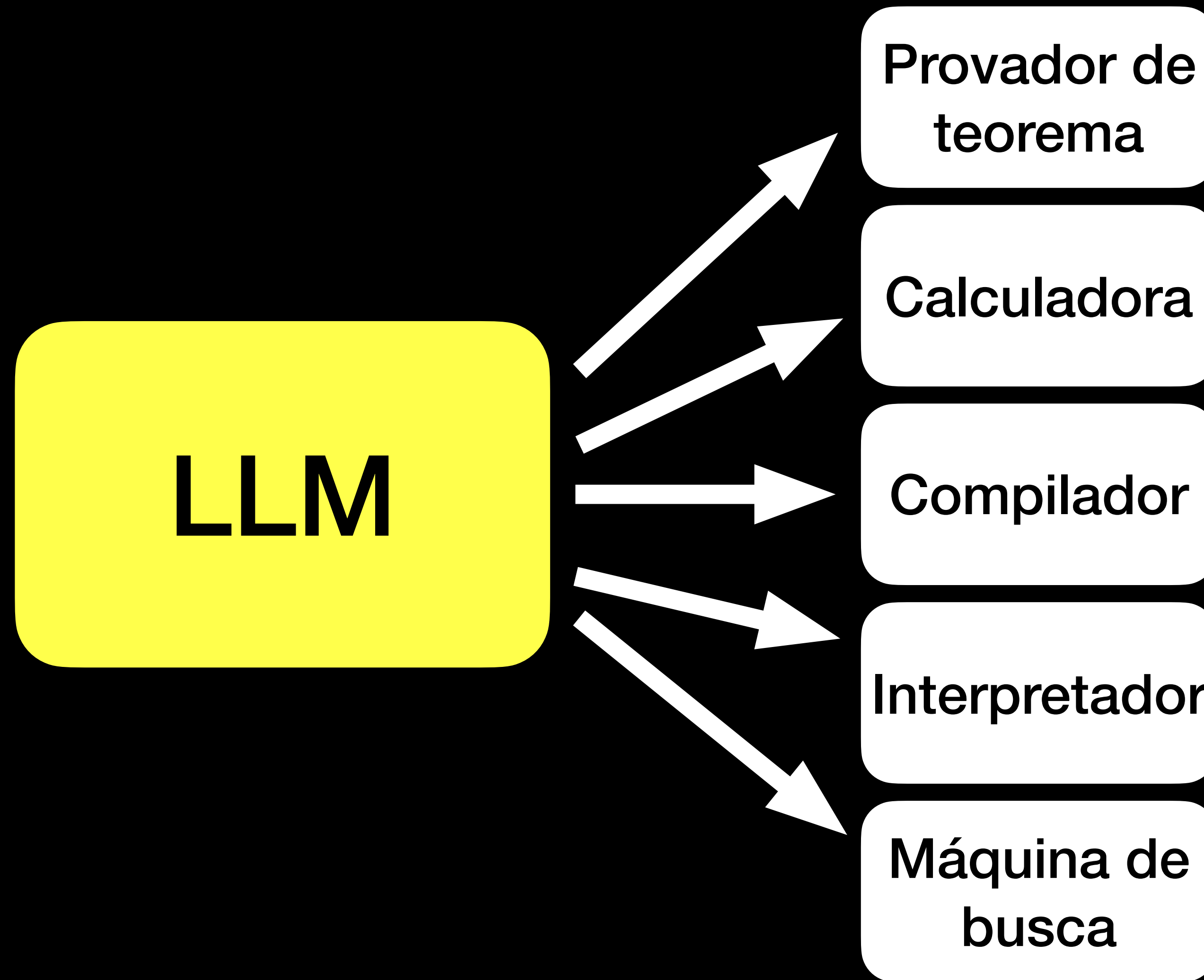


O professor Paulo Borba é um renomado especialista em engenharia de software, com mais de 30 anos de experiência em pesquisa e ensino. Ele

possui doutorado em ciência da computação pela Universidade de York, no Reino Unido, e pós-doutorado pela Universidade de Maryland, nos Estados Unidos. Atualmente, ele é professor titular da Universidade Federal de Pernambuco, onde lidera o grupo de pesquisa em Engenharia de Software e Sistemas (GESS).

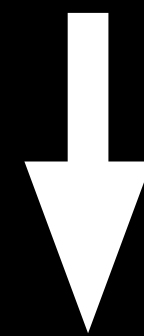
Muito cuidado com a **ilusão** do entendimento, e de que as respostas são corretas!

Augmented LLMs



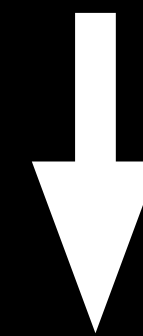
Arquitetura: componente ou controlador

**GUI +
Features**



Modelo

Modelo



Features

**Mas como isso funciona para
código também?**

Código também é natural

Suscetível ao mesmo tipo de análise estatística

```
x = x + 1;
```

é muito mais comum, frequente, que

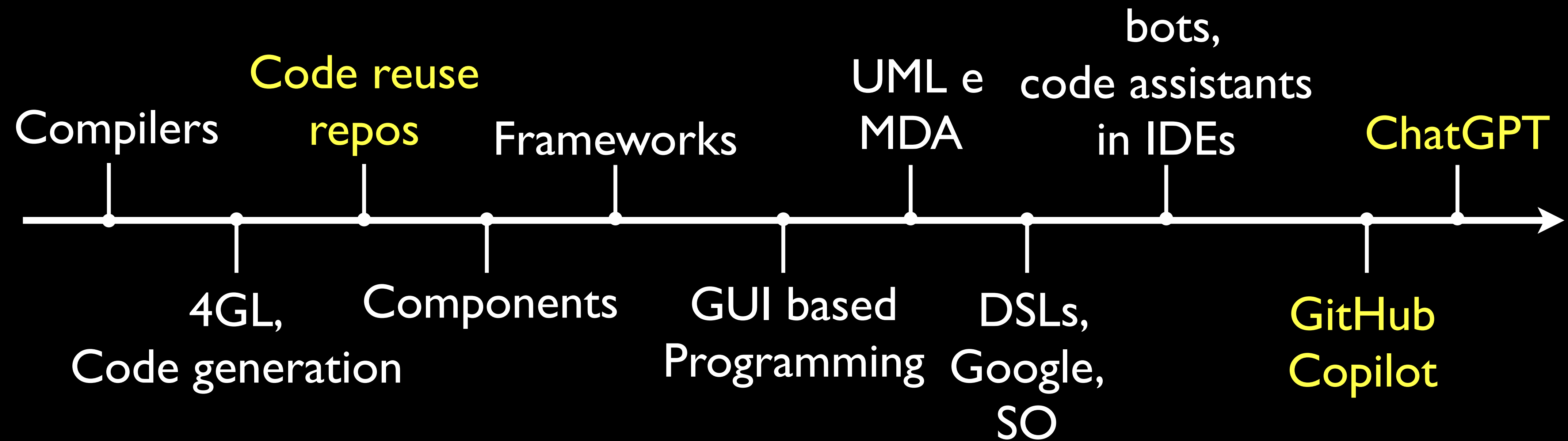
```
x = 1 + x;
```

Modelos para programação

$P(\text{code} \mid \text{text})$	Recuperação, geração
$P(\text{text} \mid \text{code})$	Documentação
$P(\text{code})$	Verificação
$P(\text{new code} \mid \text{old code})$	Correção de bug
$P(\text{next-token} \mid \text{context})$	Sugestão
$P(\text{code-L1} \mid \text{code-L2})$	Tradução

Abordagem mais promissora

Não resolver o problema, mas encontrar a solução previamente escrita



Reuso e geração de código

Code Reuse Repos	Copilot e Chat GPT
Repositório criado, limitado e curado	O repositório é a web
Busca estruturada, com base em especificações	Busca não estruturada, com base em LN
Funções, componentes, sistemas	Trechos de código e suas combinações

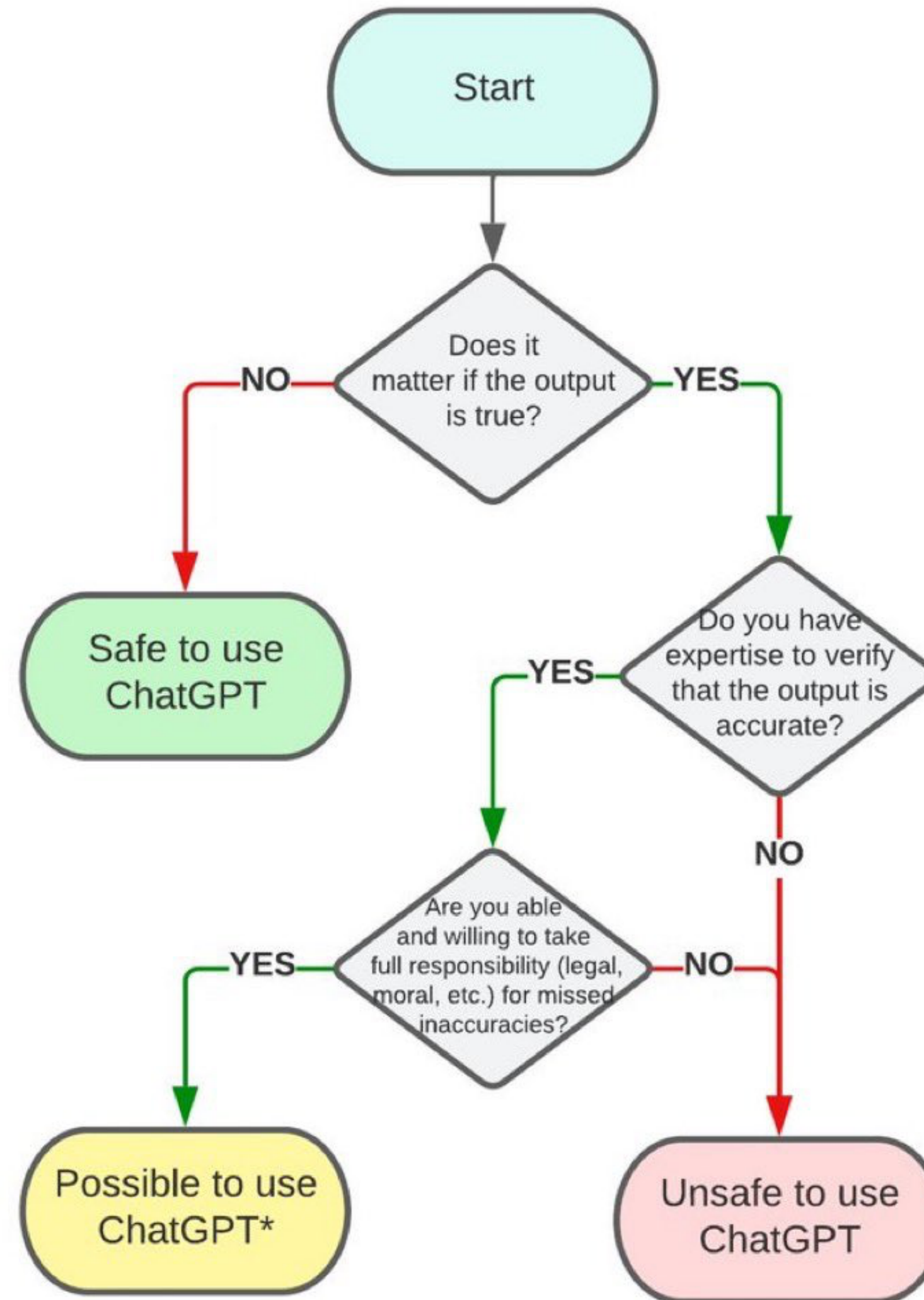
**Mas ainda precisamos de muita
pesquisa de alto nível!**

Desafios e necessidades de pesquisa

- Parte do treinamento é feito com dados que podem deixar de existir devido ao uso dos LLMs (Stack Overflow)
- Amplificação da criação de código com bugs, vulnerabilidades, problemas de usabilidade, e débito técnico (perda de diversidade)
- Curadoria de *datasets*
- Falta de apoio de geração para novas linguagens, tecnologias, frameworks, APIs, etc.
- Garantia de que código gerado respeita as licenças dos programas usados para treinamento
- Custos financeiros e ambientais

Is it safe to use ChatGPT for your task?

Aleksandr Tiulkanov | January 19, 2023



* but be sure to verify each output word and sentence for accuracy and common sense



Haverá cada vez menos necessidade de
profissionais menos qualificados
(gambiarras, código rápido e sujo, etc.)...

e cada vez mais necessidade de
profissionais mais qualificados (modelagem,
arquitetura, código limpo, visão de produto,
etc.)...

que serão mais produtivos com LLMs!

Importante refletir como cada área de **pesquisa**, e cada atividade de **ensino** e **extensão** pode ser impactada por LLMs

Referências

- O Fim da Programação: impactos da IA na programação, Paulo Borba e Felipe Calegario, Podcast do CIn/UFPE, 2023 (21/1).
- Large language models will change programming... a little, Amy Ko, 2023 (21/2).
- Large language models will change programming ... a lot, Amy Ko, 2023 (24/2).
- On the foolishness of natural language programming, Edsger Dijkstra.
- No Silver Bullet: Essence and Accident in Software Engineering, Fred Brooks, 1986.
- On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?, Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, Shmargaret Shmitchell, 2021.
- Codex (and GPT-4) can't beat humans on smart contract audits, Artem Dinaburg, Josselin Feist and Riccardo Schirone, 2023 (22/3).
- The Premature Obituary of Programming, Daniel M. Yellin, CACM 2023 (2).
- The End of Programming, Matt Welsh, CACM 2023 (1).
- Writing a tokenizer with ChatGPT, Crista Lopes, 2023 (21/1).
- The End of Programming (as we know it), Crista Lopes, 2023 (22/1).
- What Is ChatGPT Doing ... and Why Does It Work?, Stephen Wolfram, 2023 (14/2).
- ChatGPT, COVID, and the Almighty Underwater Chicken, Stefan Hanenberg, 2023 (24/2).



**Centro de
Informática**
UFPE

Podcast do Centro de Informática - UFPE


By Podcast CIn

Podcast organizado pela Coordenação de Extensão do Centro de Informática. O objetivo é ser um canal de diálogo relevante entre o que é feito no CIn e o mundo fora da universidade.

[Listen on Spotify](#) [Message](#)

<https://podcasters.spotify.com/pod/show/podcast-cin>



 **Centro de
Informática**
UFPE

O Fim da Programação - impactos da IA na progra...

Podcast do Centro de Informática - UFPE • Jan 21, 2023

 00:00  **Share** 37:46

O fim da programação, de novo!

Paulo Borba

Centro de Informática

Universidade Federal de Pernambuco

pauloborba.cin.ufpe.br