

Engenharia de Prompts (IA)

Fundamentos de NLP





Seja a diferença que impulsiona a
inovação além da capacidade da IA

Recapitulando e foco na IA Generativa

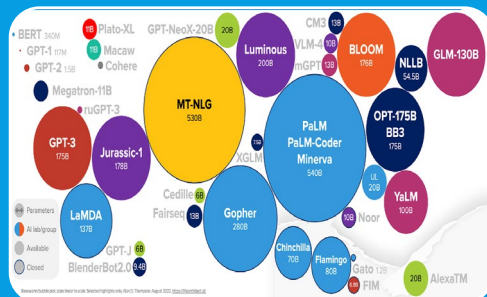
Como essa história começou (IA generativa)



GAN

[Generative Adversarial Network]

Redes Neurais Generativas e Deep Fake.



LLM

(Large Language Models)

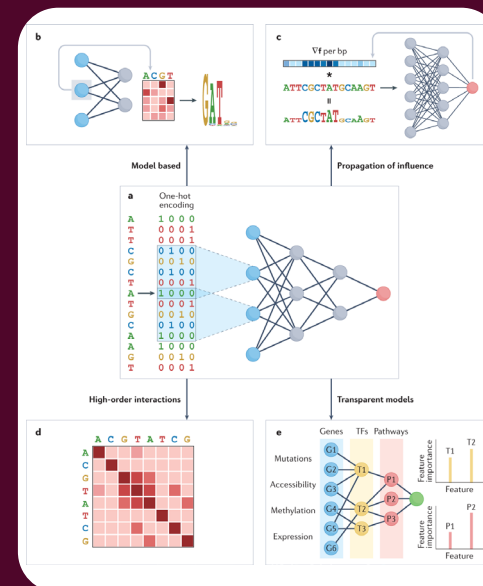
Os modelos Grandes de linguagem, potencializados por RLHF



DDPM

[Denoising Diffusion Probabilistic Models]

As redes neurais de difusão.



XAI:

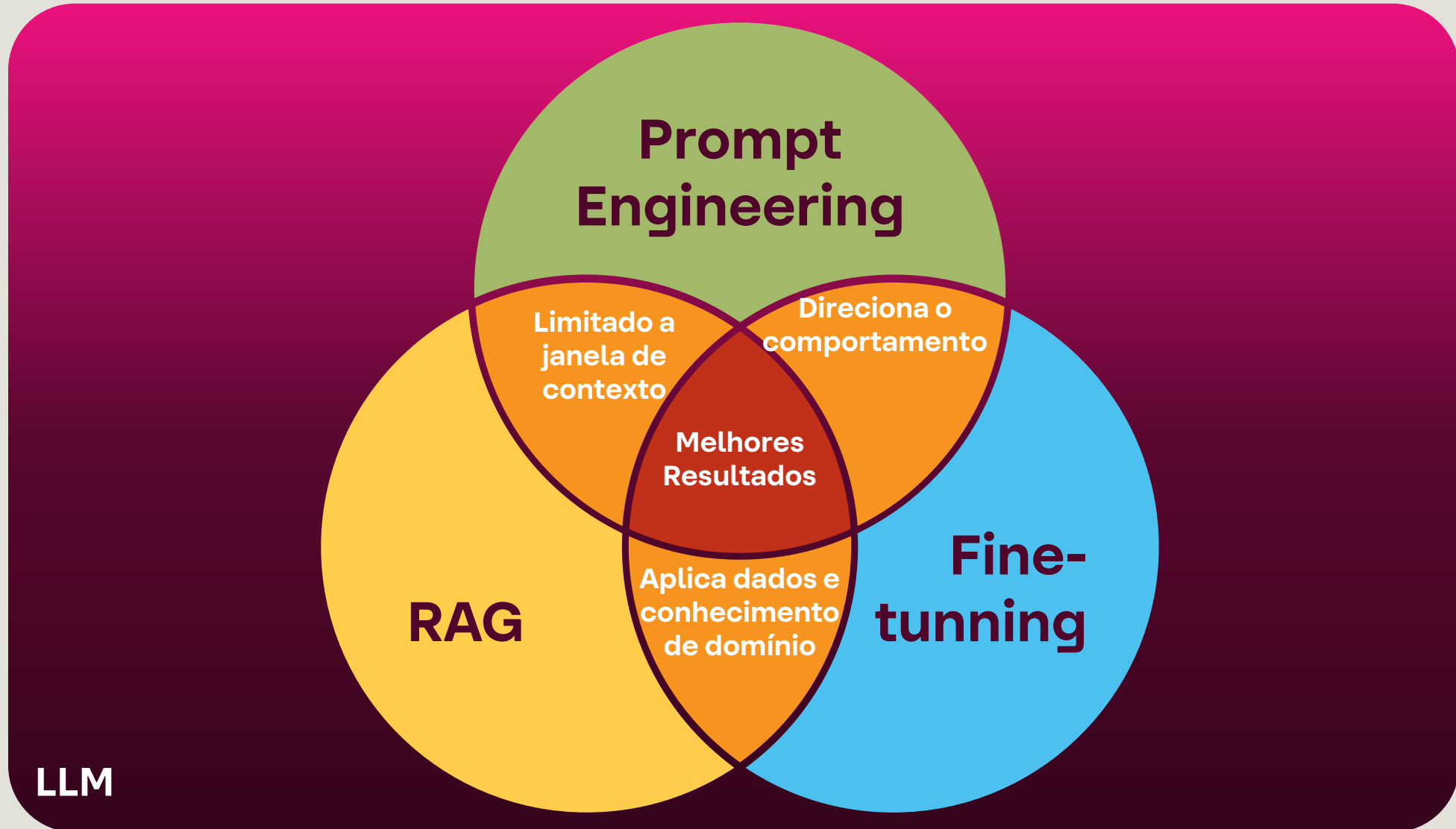
(eXplainable AI)

Modelos que explicam as redes neurais profundas.

Foundation Models

...depois de 70 anos de história

Modelos de Linguagens e seus fundamentos básicos





ChatGPT



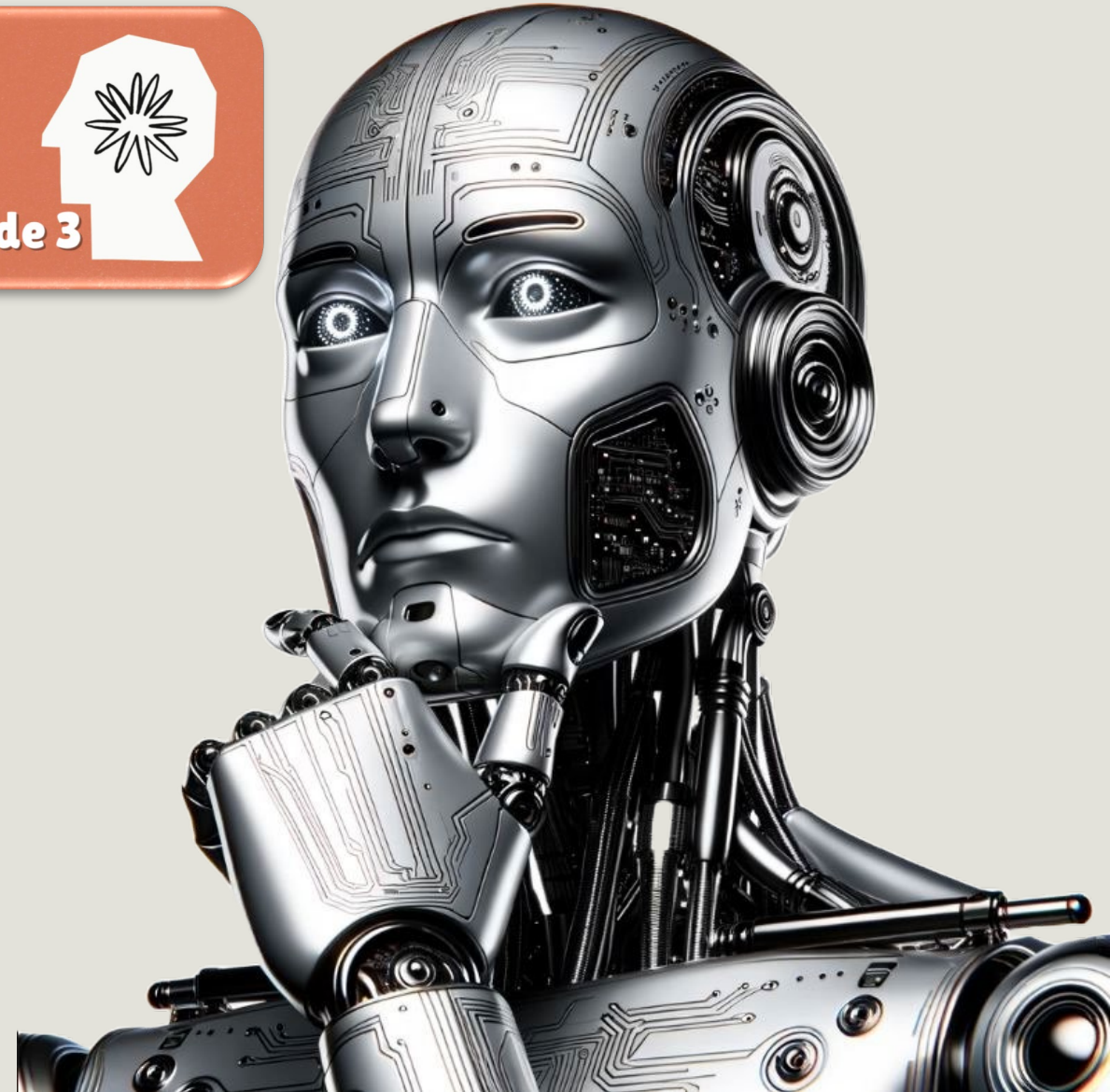
✦
Gemini



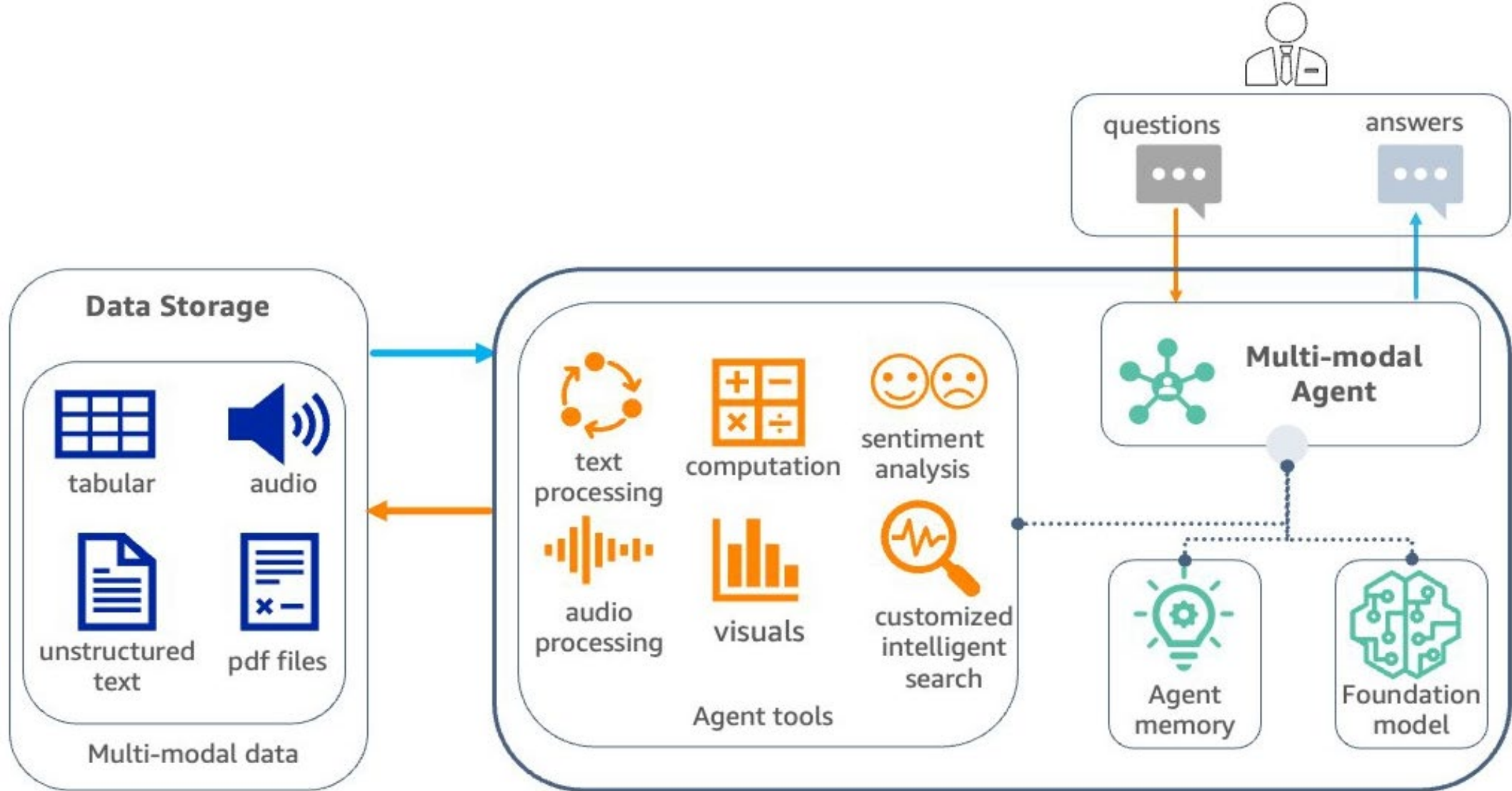
GitHub
Copilot



Copilot



Conceito de Agentes



Engenharia de Prompts

O que é Engenharia de Prompts?

A engenharia de Prompt é o processo de elaboração de instruções claras para guiar os sistemas de IA, como o **GitHub** ou **Microsoft Copilot**, para gerar código adequado ao contexto e adaptado às necessidades específicas do seu projeto. Isso garante que o código seja sintático, funcional e contextualmente correto.

Dessa forma, **Engenharia de Prompts** é como preparar instruções claras para uma IA, semelhante a um professor planejando uma aula. Envolve criar orientações precisas para que a IA entenda exatamente o que queremos e responda de forma útil.

O processo inclui:

- Definir claramente seus objetivos antes de escrever o prompt
- Escolher palavras e estruturas que direcionem a IA adequadamente
- Testar e refinar os prompts baseado nos resultados obtidos

Assim como uma boa aula facilita o aprendizado dos alunos, um prompt bem elaborado permite que a IA forneça respostas de alta qualidade que atendam às suas necessidades.



Definindo Objetivos

Estabelecendo resultados claros para o prompt



Elaborando o Prompt

Criando instruções precisas e direcionadas



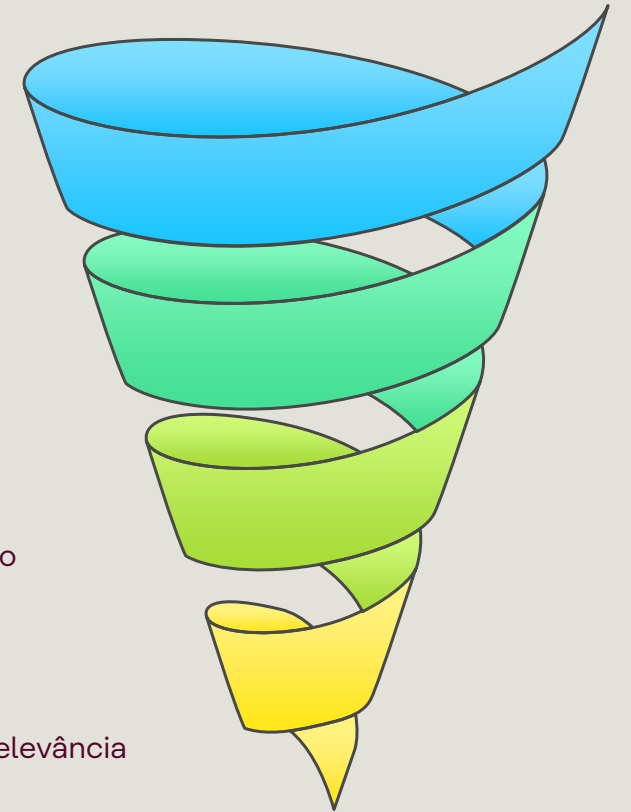
Testando o Prompt

Avaliando o desempenho do modelo

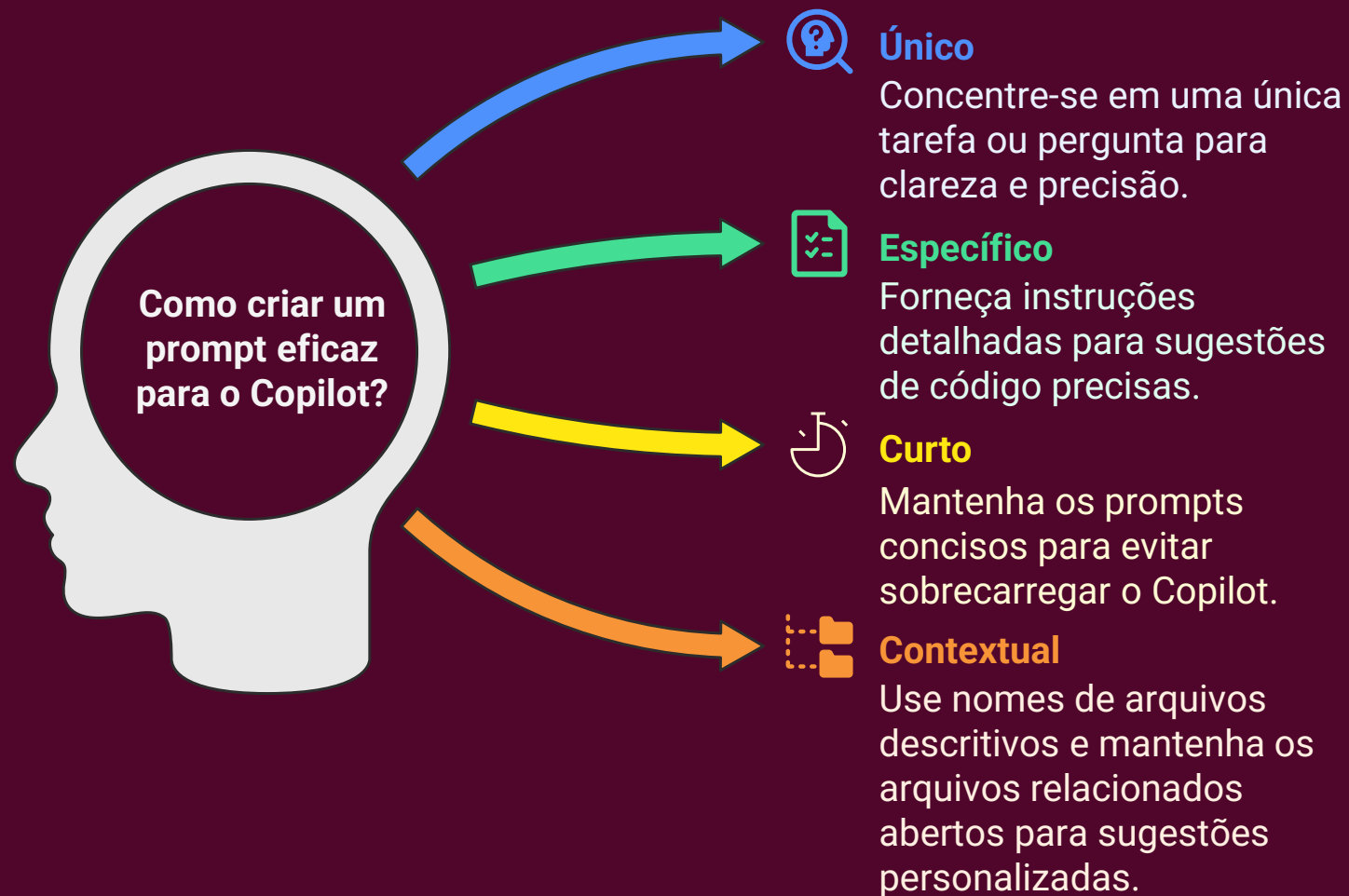


Refinando o Prompt

Ajustando para melhor precisão e relevância



Princípios da engenharia de Prompt



Qual o segredo do Prompt?



Alguém pode dar um PITACO!?

Qual o segredo do Prompt?

- P (Papel)** Atuar/Agir como um Gerente de Conteúdo
- I (Instrução)** Criar uma aula sobre transformação digital e Inteligência Artificial
- TA (Tarefa)** Complementar com exemplos práticos de adoção, em linguagem executiva no melhor português brasileiro que facilite a leitura e gere interesse do público
- CO (Contexto)** Focar nos valores da Minsait que trazem princípios centrado na pessoa humana num trabalho em conjunto, gerando valores para nossos clientes que criam encantamento através das iniciativas e senso de dono.

Para separar as instruções usar “tags” como **###**, **<>** ou **“”**

Exemplos

1. Seleção de novos candidatos a uma vaga de emprego (RH)

- **Papel:** Especialista de Recrutamento
- **Instrução:** Avalie
- **Tarefa:** o currículo de candidatos
- **Contexto:** Para a vaga de desenvolvedor Python sênior, focando em habilidades técnicas, experiência relevante e projetos anteriores.

Prompt completo:

"Como especialista de recrutamento, avalie o currículo de candidatos à vaga de desenvolvedor Python sênior, focando em habilidades técnicas, experiência relevante e projetos anteriores."

2. Análise de riscos contratuais (Jurídico)

- **Papel:** Advogado Corporativo
- **Instrução:** Análise
- **Tarefa:** os riscos associados a um contrato de prestação de serviços
- **Contexto:** Focado em cláusulas de responsabilidade, penalidades por inadimplência e implicações financeiras para o fornecedor de TI.

Prompt completo:

"Como advogado corporativo, analise os riscos associados a um contrato de prestação de serviços, focando em cláusulas de responsabilidade, penalidades por inadimplência e implicações financeiras para o fornecedor de TI."

3. Montagem de um estudo financeiro (Financeiro)

- **Papel:** Analista Financeiro
- **Instrução:** Crie
- **Tarefa:** um estudo de viabilidade financeira
- **Contexto:** Para a implementação de um novo sistema de ERP na empresa, analisando custos iniciais, retorno sobre o investimento (ROI) e economia de custos operacionais a longo prazo.

Prompt completo:

"Como analista financeiro, crie um estudo de viabilidade financeira para a implementação de um novo sistema de ERP, incluindo uma análise de custos iniciais, ROI e economia de custos operacionais a longo prazo."

Exemplos

4. Montagem de um script Python para consumo de uma API REST de conversão de moedas (Programação)

- **Papel:** Desenvolvedor Python
- **Instrução:** Escreva
- **Tarefa:** um script em Python
- **Contexto:** Que consuma uma API REST de conversão de moedas, com a capacidade de receber como input uma moeda base e converter para outras moedas específicas, retornando a taxa de câmbio atualizada.

Prompt completo:

"Como desenvolvedor Python Sênior com mais de 10 anos de experiência, escreva um script em Python que consuma uma API REST de conversão de moedas, permitindo receber como input uma moeda base e converter para outras moedas específicas, retornando a taxa de câmbio atualizada."

5. KAM

- **Papel:** Consultor
- **Instrução:** Desenvolva
- **Tarefa:** um pitch de vendas
- **Contexto:** Para uma reunião com um potencial cliente do setor bancário, focando nas vantagens da automação com IA para a segurança de dados e otimização de processos.

Prompt completo:

"Como consultor comercial, desenvolva um pitch de vendas para uma reunião com um cliente do setor bancário, destacando as vantagens da automação com IA para a segurança de dados e otimização de processos."

6. Tradução de Textos (Comunicação com Espanha)

- **Papel:** Especialista de Tradução
- **Instrução:** Traduza
- **Tarefa:** um e-mail corporativo
- **Contexto:** Enviando uma demonstração de resultados de uma formação de IA para um cliente na Espanha, destacando métricas de desempenho e feedbacks recebidos.

Prompt completo:

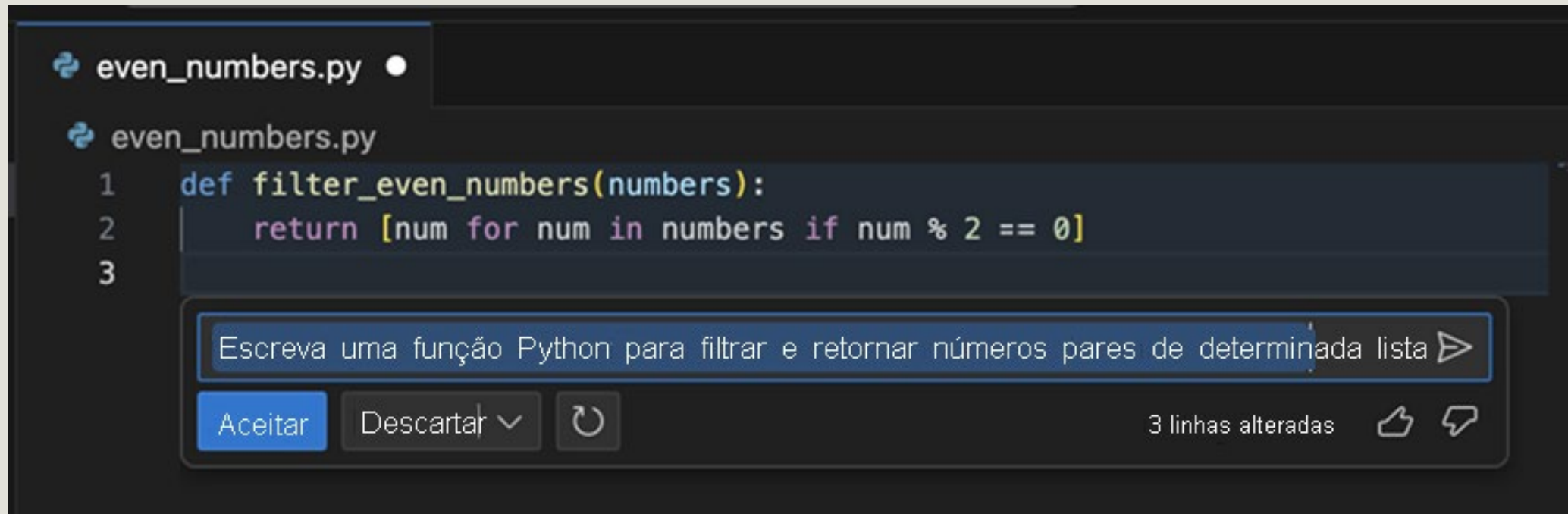
"Como especialista de tradução, traduza um e-mail corporativo para o espanhol, enviando uma demonstração de resultados de uma formação de IA para nossa matriz na Espanha, com destaque para métricas de desempenho e feedbacks recebidos."

Melhores práticas na engenharia de Prompts

As seguintes práticas avançadas, baseadas nesses princípios, refinam e aprimoram seu envolvimento com o **Copilot**, garantindo que o código gerado não seja apenas preciso, mas perfeitamente alinhado às necessidades e contextos específicos do seu projeto.

Forneça clareza suficiente

Com base nos princípios "Único" e "Específico", sempre busque a explicitação em seus prompts. Por exemplo, um prompt como "Grave uma função Python para filtrar e retornar números pares de uma lista fornecida" é específico e tem um foco único.



The screenshot shows a code editor with a file named `even_numbers.py`. The code defines a function `filter_even_numbers` that takes a list of numbers and returns a list of even numbers. A Copilot suggestion box is overlaid on the code, displaying the prompt: "Escreva uma função Python para filtrar e retornar números pares de determinada lista". Below the prompt are buttons for "Aceitar" (Accept), "Descartar" (Discard), and a refresh icon. The suggestion also indicates "3 linhas alteradas" (3 lines changed) and includes thumbs up and thumbs down icons for feedback.

```
1 def filter_even_numbers(numbers):  
2     return [num for num in numbers if num % 2 == 0]  
3
```

Escreva uma função Python para filtrar e retornar números pares de determinada lista ➤

Aceitar Descartar ↕ ↺ 3 linhas alteradas 👍 👎

Melhores práticas na engenharia de Prompts

Fornecer contexto suficiente com detalhes

Enriqueça o reconhecimento do Copilot com o contexto, seguindo o princípio "Contexto". Quanto mais informações contextuais forem fornecidas, mais ajustadas serão as sugestões de código geradas. Por exemplo, ao adicionar alguns comentários na parte superior do código para fornecer mais detalhes sobre o que deseja, você pode fornecer mais contexto para o Copilot entender seu prompt e fornecer sugestões de código melhores.

```
even_numbers.py •
even_numbers.py
1  # write a simple flask app that returns a list of even numbers from a list of numbers
2  # Create a function that takes a list of numbers and returns only the even values.
3  # create a sample list of numbers
4  # create a list of even numbers from the sample list
5  # return the list of even numbers
6
7
```

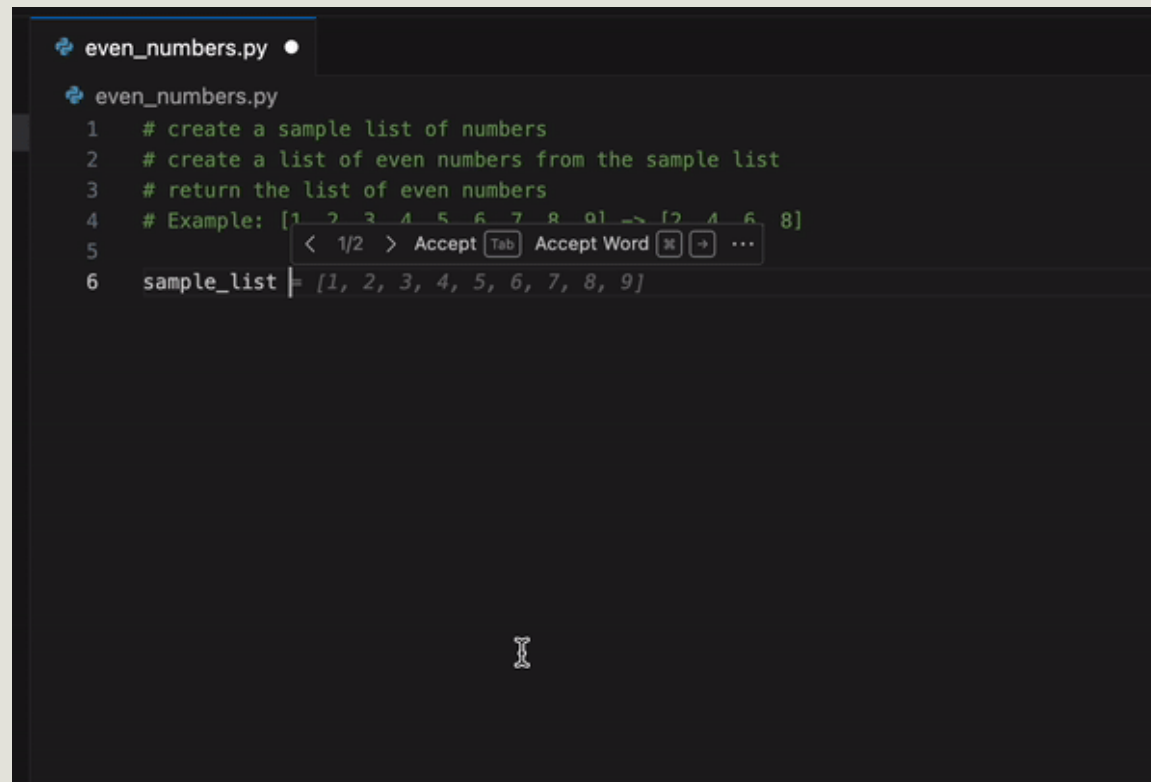
Nesse exemplo, usamos etapas para fornecer mais detalhes e, ao mesmo tempo, mantê-lo curto. Essa prática segue o princípio "Curto", equilibrando detalhes com concisão para garantir a clareza e a precisão na comunicação.

⚠ O Copilot também usa guias abertas paralelamente em seu editor de código para obter mais contexto sobre os requisitos do seu código.

Melhores práticas na engenharia de Prompts

Fornecer exemplos para aprendizado

O uso de exemplos pode esclarecer seus requisitos e expectativas, ilustrando conceitos abstratos e tornando os prompts mais tangíveis para o Copilot.



```
even_numbers.py
even_numbers.py
1 # create a sample list of numbers
2 # create a list of even numbers from the sample list
3 # return the list of even numbers
4 # Example: [1, 2, 3, 4, 5, 6, 7, 8, 9] -> [2, 4, 6, 8]
5
6 sample_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Uma das chaves para desbloquear todo o potencial do GitHub Copilot é a prática da iteração. Seu primeiro prompt nem sempre produzirá o código perfeito, e isso é perfeitamente aceitável. Se a primeira saída não for exatamente o que você está procurando, trate-a como uma etapa de um diálogo. Apague o código sugerido, enriqueça seu comentário inicial com detalhes e exemplos adicionais e prompt o Copilot novamente.



Como o Copilot aprende com seus Prompts

Como o Copilot aprende com seus Prompts

O GitHub Copilot opera com base em modelos de IA treinados em grandes quantidades de dados. Para aprimorar seu reconhecimento de contextos de código específicos, os engenheiros geralmente fornecem exemplos a ele. Essa prática, normalmente encontrada no aprendizado de máquina, levou a diferentes abordagens de treinamento, como:

Fornecer contexto suficiente com detalhes

Aqui, o GitHub Copilot gera o código sem nenhum exemplo específico, confiando apenas em seu treinamento básico. Por exemplo, suponha que você deseje criar uma função para converter temperaturas entre Celsius e Fahrenheit. Você pode iniciar escrevendo apenas um comentário descrevendo o que deseja, e o Copilot poderá gerar o código para você, com base no seu treinamento anterior, sem escrever nenhum outro exemplo.

```
1  # Preciso de uma função para converter de Celsius  
2  para Fahrenheit  
3  def celsius_to_fahrenheit(celsius):  
4      |  
5      fahrenheit = (celsius * 9/5) + 32  
6      return fahrenheit  
7  |
```

Como o Copilot aprende com seus Prompts

Aprendizado One-shot

Com essa abordagem, um único exemplo é fornecido, ajudando o modelo a gerar uma resposta mais sensível ao contexto. Com base no exemplo anterior de zero-shot, você pode fornecer um exemplo de uma função de conversão de temperatura e, em seguida, pedir ao Copilot que crie outra função semelhante. Veja como poderia ser:

```
# Exemplo: função para converter de Fahrenheit para Celsius
def fahrenheit_to_celsius(fahrenheit):
    celsius = (fahrenheit - 32) * 5/9
    return celsius

# Agora preciso de uma função para converter de
# Celsius para Fahrenheit
def celsius_to_fahrenheit(celsius):
    fahrenheit = celsius * 9/5 + 32
    return fahrenheit
```


Como o Copilot aprende com seus Prompts

Few-shot learning

Nesse método, o Copilot é apresentado com vários exemplos, que atingem um equilíbrio entre a imprevisibilidade do zero-shot e a precisão do ajuste fino. Digamos que você deseje gerar um código que lhe envie uma saudação dependendo da hora do dia. Aqui está uma versão em poucas fotos desse prompt:

```
greeting.py
1  # Example 1: Greeting message for morning
2  # Input: 9 AM
3  # Output: "Good morning!"
4
5  # Example 2: Greeting message for afternoon
6  # Input: 2 PM
7  # Output: "Good afternoon!"
8
9  # Example 3: Greeting message for evening
10 # Input: 7 PM
11 # Output: "Good evening!"
12
13 # Now, generate a python code that takes the current time as input using the datetime module
14 # and returns the appropriate greeting message
15
16 # Solution:
17 # Import datetime module
18 import datetime
19
20 # Get current time
21 current_time = datetime.datetime.now()
22
23 # Get current hour
24 current_hour = current_time.hour
25
26 # Check if it is morning (before 12 PM)
27 if current_hour < 12:
28     print("Good morning!")
29
30 # Check if it is afternoon (between 12 PM and 4 PM)
31 elif current_hour < 16:
32     print("Good afternoon!")
33
34 # Check if it is evening (after 4 PM)
35 elif current_hour < 21:
36     print("Good evening!")
37
38 # Else it is night time
39 else:
40     print("Good night!")
41
```

Estrutura do Prompt

Zero-Shot Learning

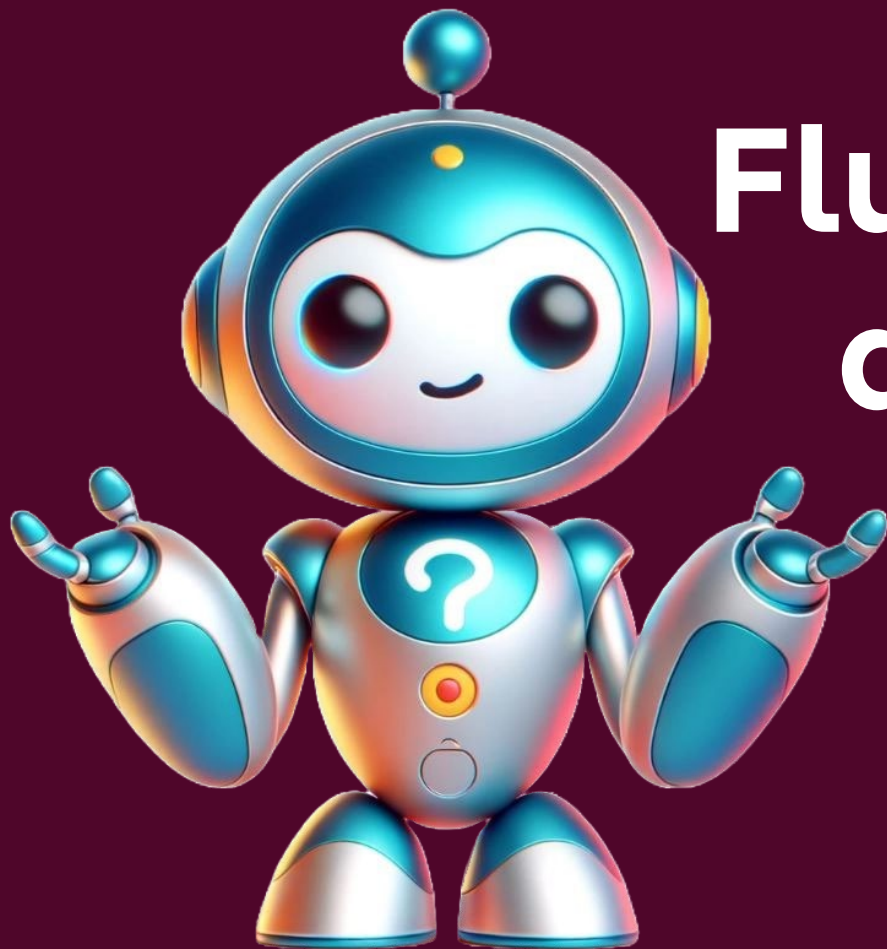
Um modelo que pode responder perguntas ou executar tarefas sem nenhum treinamento específico nesses prompts particulares.

One-Shot Learning

Inferência **one-shot** toma um exemplo. O modelo pode entender a essência da tarefa e gerar a saída desejada.

Few-Shot Learning

A inferência **few-shot** permite que você forneça um pequeno número de exemplos para guiar o comportamento do modelo. É como dar ao modelo uma mini sessão de treinamento com apenas um punhado de prompts.



Fluxo do processo de prompts do usuário

Como o Copilot aprende com seus Prompts

1. Transmissão segura de prompts e coleta de contexto

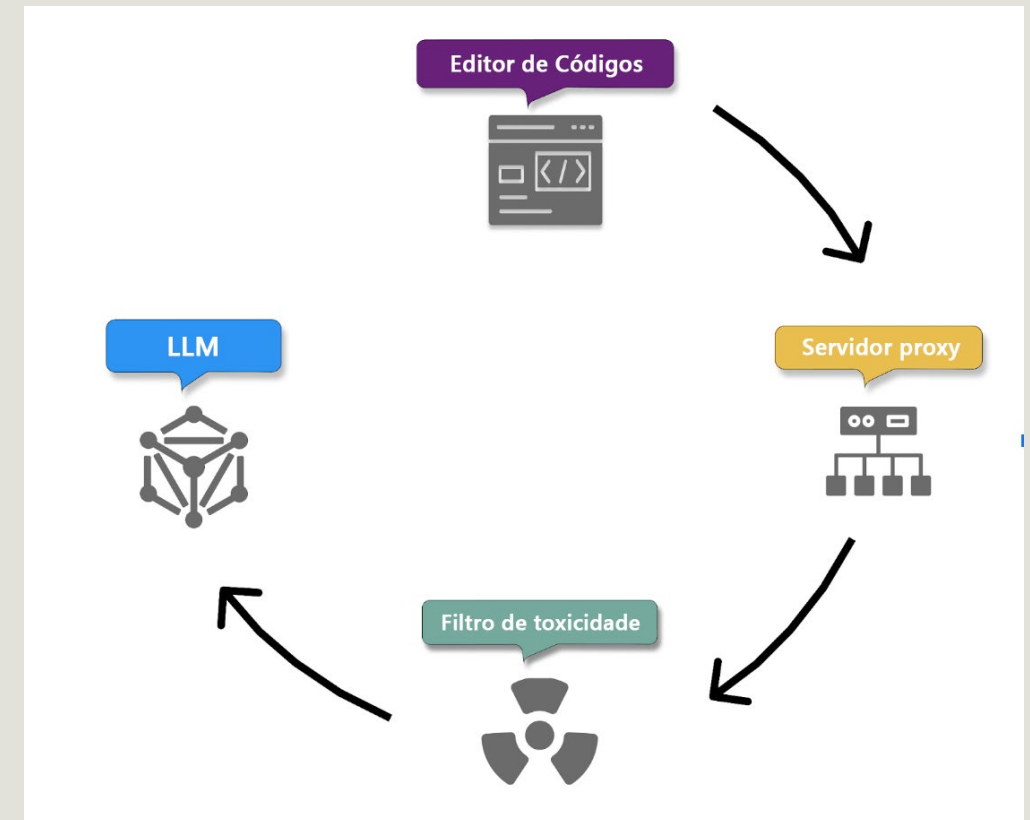
O processo começa com a transmissão segura do prompt do usuário por HTTPS. Isso garante que seu comentário em linguagem natural seja enviado aos servidores do GitHub Copilot de forma segura e confidencial, protegendo as informações confidenciais.

O GitHub Copilot recebe com segurança o prompt do usuário, que pode ser um chat do Copilot ou um comentário em linguagem natural fornecido por você no seu código.

Simultaneamente, o Copilot coleta detalhes de contexto:

- O código antes e depois da posição do cursor, o que o ajuda a entender o contexto imediato do prompt.
- O nome e tipo do arquivo que está sendo editado, permitindo que adapte sugestões de código ao tipo de arquivo específico.
- Informações sobre as guias abertas adjacentes, garantindo que o código gerado se alinhe a outros segmentos de código no mesmo projeto.
- Informações sobre a estrutura do projeto e caminhos de arquivo
- Informações sobre linguagens de programação e frameworks
- Pré-processamento usando a técnica Fill-in-the-Middle (FIM) para considerar o contexto do código anterior e posterior, expandindo efetivamente a compreensão do modelo, permitindo que o Copilot gere sugestões de código mais precisas e relevantes aproveitando um contexto mais amplo.

Essas etapas traduzem a solicitação de alto nível do usuário em uma tarefa de codificação concreta.



Como o Copilot aprende com seus Prompts

2. Filtro de proxy

Depois que o contexto é coletado e o prompt é criado, ele passa com segurança para um servidor proxy hospedado em um locatário do Microsoft Azure de propriedade do GitHub. O proxy filtra o tráfego, bloqueando tentativas de hackear o prompt ou manipular o sistema para revelar detalhes sobre como o modelo gera sugestões de código.

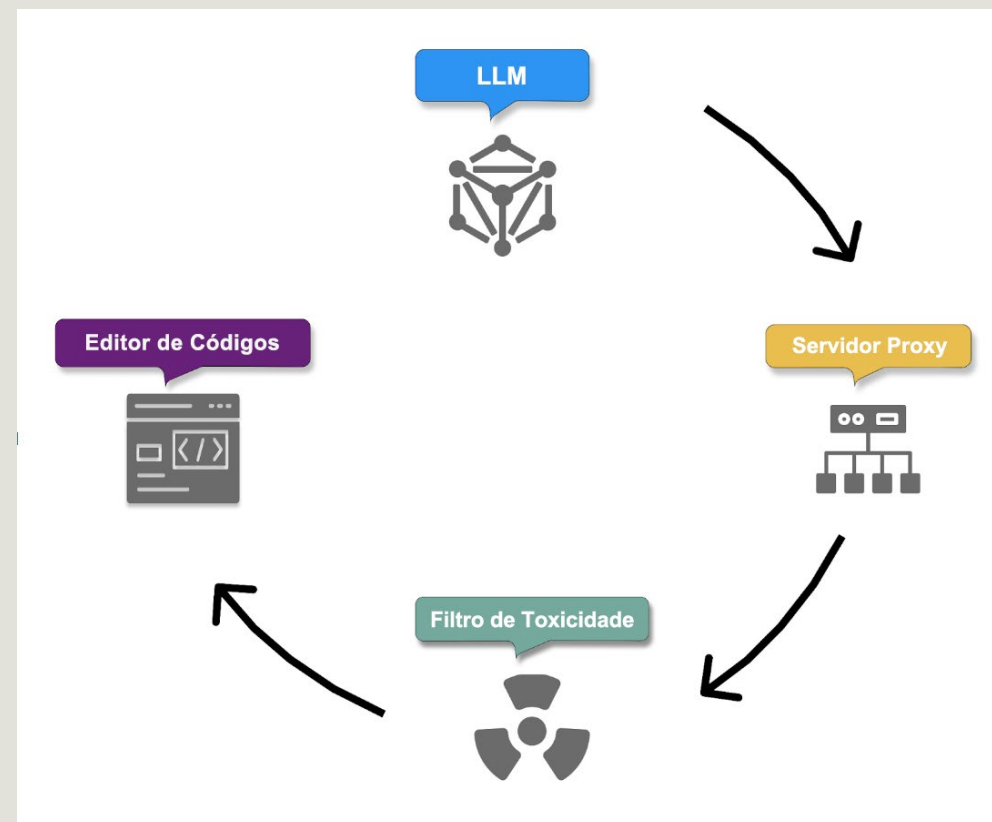
3. Filtragem de toxicidade

O Copilot incorpora mecanismos de filtragem de conteúdo antes de prosseguir com a extração de intenções e geração de código, para garantir que as respostas e o código gerado não incluam nem promovam:

- **Discurso de ódio e conteúdo inadequado:** O Copilot emprega algoritmos para detectar e prevenir a ingestão de discurso de ódio, linguagem ofensiva ou conteúdo inadequado que possa ser prejudicial ou ofensivo.
- **Dados pessoais:** O Copilot filtra o conteúdo de modo a deixar de fora todos os dados pessoais, como nomes, endereços ou números de identidade, para proteger a privacidade e a segurança de dados do usuário.

4. Geração de código com LLM

Por fim, o prompt filtrado e analisado é passado para os Modelos LLM, que geram sugestões de código apropriadas. Essas sugestões são baseadas no entendimento do Copilot sobre o prompt e o contexto circundante, garantindo que o código gerado seja relevante, funcional e alinhado com os requisitos específicos do projeto.



Como o Copilot aprende com seus Prompts

5. Pós-processamento e validação de resposta

Depois que o modelo produz suas respostas, o filtro de toxicidade remove qualquer conteúdo gerado nocivo ou ofensivo. O servidor proxy então aplica uma camada final de verificações para garantir a qualidade do código, a segurança e os padrões éticos. Essas verificações incluem:

- **Qualidade do código:** As respostas são verificadas quanto a bugs ou vulnerabilidades comuns, como cross-site scripting (XSS) ou injeção de SQL, garantindo que o código gerado seja robusto e seguro.
- **Código público correspondente (opcional):** Opcionalmente, os administradores podem habilitar um filtro que impede o Copilot de retornar sugestões com mais de ~150 caracteres caso elas sejam muito semelhantes a códigos públicos existentes no GitHub. Isso evita que correspondências coincidentes sejam sugeridas como conteúdo original. Se qualquer parte da resposta falhar nessas verificações, ela será truncada ou descartada.

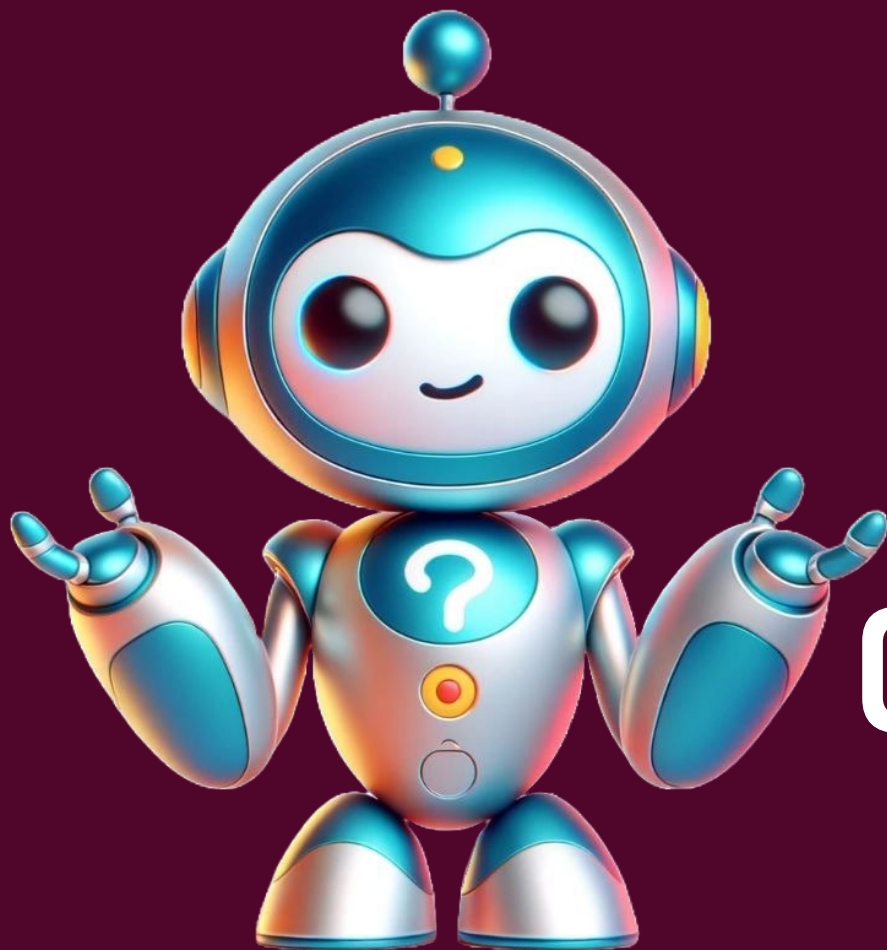
6. Entrega de sugestões e início do ciclo de feedback

Somente as respostas que passam por todos os filtros são entregues ao usuário. O Copilot então inicia um ciclo de feedback com base em suas ações para alcançar o seguinte:

- Aumentar seu próprio conhecimento com base nas sugestões aceitas.
- Aprender e se aprimorar por meio de modificações e rejeições das sugestões.

7. Repita para os prompts subsequentes

O processo é repetido à medida que você fornece mais prompts, com o Copilot lidando continuamente com as solicitações do usuário, entendendo sua intenção e gerando um código como resposta. Com o tempo, o Copilot aplica os dados cumulativos de feedback e de interação, incluindo os detalhes de contexto, para aprimorar seu entendimento da intenção do usuário e refinar seus recursos de geração de código.



Cadeias avancadas (Pensamento)

Cadeia de Pensamento (COT – Chain of Thought)

O **COT** é uma técnica que guia os modelos de linguagem a abordar problemas complexos de maneira estruturada e sequencial. Em vez de fornecer uma resposta direta e imediata, o modelo é incentivado a descrever seu raciocínio passo a passo, permitindo que o processo de pensamento seja transparente e verificável. Isso é particularmente útil em tarefas que requerem lógica, cálculo ou etapas múltiplas para alcançar a resposta correta.

Uma das principais vantagens do **CoT** é que ele explora e expande a capacidade dos modelos de linguagem de realizar inferências mais profundas e elaborar explicações mais robustas. Ao detalhar o raciocínio, o modelo pode identificar subproblemas e resolver cada um separadamente antes de combinar os resultados para formar uma resposta final. Esse método pode aumentar significativamente a precisão das respostas, especialmente em problemas que envolvem múltiplas etapas ou componentes.

PROMPT:

P: Roger tem 5 bolas de tênis. Ele compra mais 2 latas de bolas de tênis. Cada lata contém 3 bolas de tênis. Quantas bolas de tênis ele tem agora?

R: A resposta é 11.

P: cafeteria tinha 23 maçãs. Se elas usaram 20 para fazer o almoço e compraram mais 6, quantas maçãs elas têm agora?

RESPOSTA: A resposta é 27.

⚠ Ao utilizar o CoT, a resposta está correta e detalha cada etapa do processo de raciocínio, proporcionando uma explicação clara e verificável. Esta abordagem não só melhora a precisão das respostas, mas também facilita a compreensão e a verificação do raciocínio do modelo.

PROMPT:

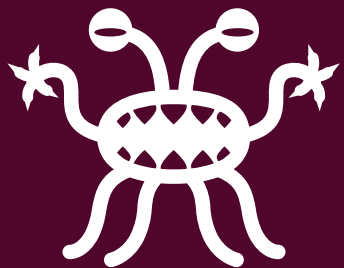
P: Roger tem 5 bolas de tênis. Ele compra mais 2 latas de bolas de tênis. Cada lata contém 3 bolas de tênis. Quantas bolas de tênis ele tem agora?

R: **Roger começou com 5 bolas. 2 latas de 3 bolas de tênis cada resultam em 6 bolas de tênis. $5 + 6 = 11$.** A resposta é 11. [CoT aplicado]

P: A cafeteria tinha 23 maçãs. Se elas usaram 20 para fazer o almoço e compraram mais 6, quantas maçãs elas têm agora?

RESPOSTA: A cafeteria tinha originalmente 23 maçãs. Elas usaram 20 para fazer o almoço. Então, $23 - 20 = 3$. Elas compraram mais 6 maçãs, então têm $3 + 6 = 9$. A resposta é 9.

Alucinação



Cadeia de Verificação (CoVe – Chain of Verification)

CoVe é uma técnica projetada para mitigar as "alucinações" em modelos de linguagem de grande porte (LLMs). As alucinações referem-se à geração de informações plausíveis, mas incorretas. O método CoVe envolve um processo de múltiplas etapas, onde o modelo primeiro gera uma resposta inicial, depois planeja questões de verificação para checar a precisão dos fatos, responde essas perguntas de forma independente e, finalmente, produz uma resposta verificada.

CoVe é particularmente útil em contextos onde a precisão dos dados é crucial, como na verificação de fatos em documentos legais, relatórios científicos, e em tarefas que exigem respostas longas e detalhadas, como a geração de biografias ou a resposta a perguntas em formato de lista. Implementações típicas incluem gerar uma resposta inicial, criar perguntas de verificação, respondê-las sem viés e produzir uma resposta final baseada nas verificações.

PROMPT:

Por favor, responda inicialmente de forma concisa à minha pergunta. Em seguida, faça perguntas sobre a resposta inicial e verifique os fatos apresentados nessas respostas. Exponha as perguntas e respostas do processo de verificação detalhadamente e, com base nessa análise, reformule uma resposta final mais precisa e fundamentada para a minha <pergunta> [CoVE aplicado]

<pergunta> Quando o Ayrton Senna morreu? <pergunta>

RESPOSTA: Ayrton Senna morreu em 1º de maio de 1994.

Cadeia de Verificação [CoVe – Chain of Verification]

LLM: Verificando a informação:

Pergunta de verificação: Em que circunstâncias Ayrton Senna morreu em 1º de maio de 1994?

Resposta de verificação: Ayrton Senna morreu devido a um acidente trágico durante o Grande Prêmio de San Marino, em Ímola, na Itália. Ele colidiu com o muro de concreto na curva Tamburello enquanto liderava a corrida.

Pergunta de verificação: Existem outras fontes que confirmem a data e as circunstâncias de sua morte?

Resposta de verificação: Sim, a data e as circunstâncias da morte de Senna são amplamente relatadas por fontes confiáveis, incluindo arquivos de notícias esportivas, documentários sobre sua vida e carreira, e publicações especializadas em automobilismo.

Pergunta de verificação: Qual foi o impacto imediato de sua morte no mundo do automobilismo?

Resposta de verificação: A morte de Senna teve um impacto profundo no mundo do automobilismo, resultando em mudanças significativas nas medidas de segurança dos carros e pistas de Fórmula 1 para melhorar a segurança dos pilotos, incluindo mudanças nos designs dos carros e nas regulamentações das corridas.

LLM: Verificando a informação:

Resposta final reformulada:

Ayrton Senna, um dos maiores pilotos de Fórmula 1 de todos os tempos, morreu em um trágico acidente em 1º de maio de 1994 durante o Grande Prêmio de San Marino, em Ímola, Itália. Sua morte ocorreu devido a um choque fatal contra o muro de concreto na curva Tamburello, o que levou a uma revisão abrangente e a melhorias substanciais nas normas de segurança na Fórmula 1. A data e as circunstâncias de sua morte são bem documentadas e consistentemente relatadas por várias fontes confiáveis, destacando não só a tragédia, mas também o legado duradouro de mudanças que sua morte provocou no esporte.

Desafio

- Problematização PBL
- Literatura livre assim como IAs disponíveis
- Não tem restrição de tecnologia [quem não programa pode fazer conceitual]

Desafio Prático: Sistema Inteligente de Atendimento ao Cliente Minsait

Contexto Empresarial

Imagine que a Minsait foi contratada por um grande banco brasileiro para revolucionar sua experiência de atendimento digital. O cliente enfrenta um cenário crítico: mais de 60% das interações de suporte são repetitivas, os tempos de resposta estão impactando a satisfação do cliente, e há uma necessidade urgente de personalização em escala.

O Desafio

Vocês são a equipe de arquitetos de IA responsável por projetar uma solução que combine o melhor da tecnologia conversacional moderna com a expertise bancária acumulada.

⚠️ Atividade é para avaliação, pode ser em grupo de até 3 pessoas, tempo para dúvidas e escopo básico será de **45 minutos**, vamos trabalhar nos próximos encontros aportando conhecimento para apresentação da solução na última aula.

Desafio Prático: Sistema Inteligente de Atendimento ao Cliente Minsait

Exemplo

Desenvolver um **Assistente Inteligente Bancário** capaz de:

1. **Receber e processar** consultas em linguagem natural dos clientes
2. **Analisar o contexto emocional** da interação (frustração, urgência, satisfação)
3. **Consultar a base de conhecimento** bancária existente (produtos, regulamentações, FAQ)
4. **Gerar respostas personalizadas** que demonstrem compreensão tanto técnica quanto emocional
5. **Escalar automaticamente** para atendimento humano quando necessário

Desafio Prático: Sistema Inteligente de Atendimento ao Cliente Minsait

Componentes técnicos esperados

- Pipeline de NLP Clássico:
- Pré-processamento e normalização de textoAnálise de sentimentos e detecção de intenções
- Extração de entidades financeiras (valores, contas, produtos)

Arquitetura RAG (Retrieval-Augmented Generation):

- Sistema de busca semântica na base de conhecimento
- Ranking e seleção de informações relevantes
- Geração contextualizada de respostas

Agentificação Inteligente:

- Orquestração de múltiplos modelos especializados
- Sistema de decisão para escalação
- Memória conversacional e contexto de sessão

... são só os primeiros Passos



