



ESTRATÉGIA DE RACIOCÍNIO EM LLMs

Grupo JARVIS Jr.

CAROLINA LAMARCA, DANIEL RAMÓN,
JOÃO VICTOR BALOGH
JULIANA CATARINA DE SOUZA
MATHEUS RODRIGUES, NICOLE PAES
PAULO ROBERTO OLIVEIRA

I2A2 - Agentes Inteligentes

Ajuste Fino Totalmente Supervisionado – Estratégia RACIEM

O ajuste fino totalmente supervisionado é uma das estratégias centrais no treinamento e especialização de Modelos de Linguagem de Grande Escala (LLMs). Essa abordagem consiste em refinar o modelo pré-treinado com dados rotulados cuidadosamente, onde cada exemplo contém uma entrada e a resposta correta correspondente. No contexto da estratégia RACIEM — Raciocínio, Compreensão, Execução e Memória —, esse tipo de ajuste permite que o modelo desenvolva competências específicas em tarefas complexas, como resolver problemas matemáticos, interpretar textos, ou realizar ações lógicas baseadas em instruções.

O processo envolve:

- Coleta de dados supervisionados: Ex.: perguntas e respostas corretas, problemas com soluções passo a passo, exemplos de código com explicações, etc.
- Treinamento supervisionado: O modelo é exposto repetidamente a esses pares, ajustando seus pesos internos para gerar saídas similares às respostas corretas.
- Avaliação contínua: A performance é avaliada com conjuntos de validação para evitar overfitting (quando o modelo memoriza em vez de generalizar).

Aplicação nas Dimensões do RACIEM

- Raciocínio: Treinamento com dados que requerem múltiplos passos lógicos, como resolução de problemas matemáticos complexos ou raciocínio dedutivo.
- Compreensão: Dados supervisionados de interpretação de texto, inferência e análise semântica.
- Execução: Exemplos de tarefas com comandos ou códigos que o modelo deve completar ou corrigir.
- Memória: Exemplos com dependência de longo prazo, como perguntas baseadas em contextos extensos ou múltiplos turnos de conversa.

Exemplo Prático

Suponha que queremos treinar um modelo para explicar passo a passo como resolver equações do segundo grau. No ajuste fino supervisionado, forneceremos exemplos como:

Entrada: "Resolva $x^2 - 5x + 6 = 0$ passo a passo."

Saída esperada:

"Passo 1: Identifique os coeficientes: $a = 1$, $b = -5$, $c = 6$

Passo 2: Calcule o discriminante: $\Delta = b^2 - 4ac = 25 - 24 = 1$

Passo 3: Calcule as raízes: $x = [5 \pm \sqrt{1}]/2 = (5 \pm 1)/2$

Passo 4: $x_1 = 3$, $x_2 = 2$ "

Comparações com Outras Estratégias

1. RLHF (Reinforcement Learning from Human Feedback)

Diferença principal: O modelo é treinado a partir de preferências humanas em vez de respostas fixas.

Vantagem: Gera respostas mais alinhadas a valores humanos e boas práticas conversacionais.

Desvantagem: Mais complexo e menos direto, e pode ser menos preciso em tarefas com respostas objetivas.

2. In-Context Learning (Aprendizado no Contexto)

Diferença: O modelo não é ajustado com novos pesos, mas "aprende" temporariamente a tarefa com base em exemplos dados no prompt.

Vantagem: Flexível e não exige retreinamento.

Desvantagem: Menos robusto e menos confiável em tarefas complexas ou que exigem consistência.

Vantagens do Ajuste Supervisionado

- Alta precisão em tarefas específicas.
- Controle total sobre o tipo de comportamento que se deseja ensinar.
- Previsibilidade: menos suscetível a variações imprevisíveis de saída.

Desvantagens do Ajuste Supervisionado

- Custo alto (anotação humana, tempo de treinamento).
- Risco de overfitting se a diversidade dos dados for baixa.
- Pouca adaptabilidade se comparado ao aprendizado contextual ou por reforço.

Ajuste Fino Totalmente Supervisionado

TESTES

Prompt 1

"Resolva a equação $3x - 7 = 2x + 5$, explicando cada passo do processo."

Resposta ChatGPT:

Passo 1: Isolar os termos com incógnita de um lado da equação

- Queremos que todos os termos com a variável x fiquem de um lado (por exemplo, do lado esquerdo). Para isso, subtraímos $2x$ dos dois lados da equação:

$$3x - 7 - 2x = 2x + 5 - 2x$$

$$(3x - 2x) - 7 = 5$$

$$x - 7 = 5$$

Passo 2: Isolar o termo com a variável

- Agora, somamos 7 dos dois lados da equação para eliminar o -7 :

$$x - 7 + 7 = 5 + 7$$

$$x = 12$$

Solução final: $x = 12$

Resposta Gemini:

Passo 1: Agrupar os termos com ' x ' em um lado da equação.

- Para fazer isso, vamos subtrair $2x$ de ambos os lados da equação. Isso elimina o termo com ' x ' do lado direito:

$$3x - 2x - 7 = 2x - 2x + 5$$

Simplificando: $x - 7 = 5$

Passo 2: Isolar o termo com ' x '.

- Agora, precisamos nos livrar do -7 que está do lado do ' x '. Para isso, vamos adicionar 7 a ambos os lados da equação. Isso isolará o ' x ' no lado esquerdo:

$$x - 7 + 7 = 5 + 7$$

Simplificando: $x = 12$

Prompt 2

"Escreva uma função em Python que recebe uma lista de números inteiros e retorna a soma dos números pares. Explique o que cada parte do código faz."

Resposta ChatGPT:

Código gerado:

```
def soma_pares(lista):  
    # Inicializa a variável soma com zero  
    soma = 0  
  
    # Percorre cada número na lista fornecida  
    for numero in lista:  
        # Verifica se o número é par (ou seja, se o resto da divisão por 2 é zero)  
        if numero % 2 == 0:  
            # Adiciona o número par à variável soma  
            soma += numero  
  
    # Retorna o valor total da soma dos números pares  
    return soma
```

Explicação de cada parte:

1. `def soma_pares(lista):`
Define uma função chamada `soma_pares` que recebe um parâmetro chamado `lista`, que deve ser uma lista de números inteiros.
2. `soma = 0`
Inicializa uma variável chamada `soma` com o valor 0. Essa variável será usada para acumular a soma dos números pares encontrados na lista.
3. `for numero in lista:`
Um laço `for` que percorre cada elemento da lista. Cada elemento será

acessado pela variável número.

4. `if numero % 2 == 0:`

Verifica se o número atual é par. Isso é feito usando o operador %, que calcula o resto da divisão. Se o resto da divisão do número por 2 for igual a 0, o número é par.

5. `soma += numero`

Se o número for par, ele é adicionado à variável soma.

6. `return soma`

Depois de percorrer toda a lista, a função retorna o valor final da soma dos números pares.

Resposta Gemini:

Código gerado:

```
def somar_pares(numeros):
```

```
    """
```

```
    Recebe uma lista de números inteiros e retorna a soma dos números pares.
```

```
    Args:
```

```
        numeros: Uma lista de números inteiros.
```

```
    Returns:
```

```
        Um inteiro representando a soma dos números pares na lista.
```

```
    """
```

```
    soma = 0 # Inicializa uma variável para armazenar a soma, começando com zero.
```

```
    for numero in numeros: # Itera por cada número na lista de entrada.
```

```
        if numero % 2 == 0: # Verifica se o número atual é par. O operador % (módulo)
            # retorna o resto da divisão. Se o resto da divisão por 2 é 0, o número é par.
```

```
            soma = soma + numero # Se o número for par, adicioná-lo ao valor atual da
            # variável 'soma'.
```

```
return soma # Após percorrer todos os números, retorna o valor final da 'soma'.
```

Exemplo de uso:

```
lista_de_numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
soma_dos_pares = somar_pares(lista_de_numeros)
print(f"A soma dos números pares na lista é: {soma_dos_pares}")
```

Explicação de cada parte do código:

1. def somar_pares(numeros)::

- def é a palavra-chave em Python que define uma nova função.
- somar_pares é o nome que escolhemos para a nossa função. Ele descreve o que a função faz.
- (numeros) indica que a função recebe um argumento chamado numeros. Esperamos que este argumento seja uma lista de números inteiros.

2. """Docstring explicando a função""":

- As três aspas duplas ("""Docstring""") delimitam uma docstring (string de documentação). É uma boa prática incluir docstrings para explicar o que a função faz, quais argumentos ela recebe (Args), e o que ela retorna (Returns). Isso torna o código mais fácil de entender.

3. soma = 0:

- Aqui, criamos uma variável chamada soma e a inicializamos com o valor 0. Essa variável será usada para acumular a soma dos números pares que encontrarmos na lista. Começamos com 0 porque ainda não somamos nenhum número.

4. for numero in numeros::

- Este é um loop for. Ele percorre cada elemento da lista numeros.
- Em cada iteração do loop, o valor do elemento atual da lista é atribuído à variável numero.

5. if numero % 2 == 0::

- Esta é uma instrução condicional if. Ela verifica se o número atual (numero) é par.
- O operador % é o operador de módulo. Ele retorna o resto da divisão de um número por outro.

- Se `numero % 2` for igual a 0, significa que o número é divisível por 2 sem deixar resto, ou seja, ele é par.

6. `soma = soma + numero:`

- Esta linha é executada apenas se a condição no `if` for verdadeira (ou seja, se o número for par).
- Ela pega o valor atual da variável `soma`, adiciona o valor do número par a ela, e então atualiza o valor da variável `soma`. Em outras palavras, estamos acumulando a soma dos números pares.

7. `return soma:`

- Após o loop for terminar de percorrer todos os números na lista, esta linha retorna o valor final da variável `soma`. Este valor representa a soma de todos os números pares encontrados na lista.

8. `# Exemplo de uso::`

- As linhas que começam com `#` são comentários. Elas não são executadas pelo Python e são usadas para explicar o código.

9. `lista_de_numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:`

- Aqui, criamos uma lista de exemplo chamada `lista_de_numeros` para testar nossa função.

10. `soma_dos_pares = somar_pares(lista_de_numeros):`

- Chamamos a função `somar_pares` com a nossa lista de exemplo e armazenamos o valor retornado (a soma dos pares) na variável `soma_dos_pares`.

11. `print(f"A soma dos números pares na lista é: {soma_dos_pares}"):`

- Finalmente, usamos a função `print()` para exibir o resultado na tela. O `f` antes da string permite usar f-strings (strings formatadas), onde podemos inserir o valor de variáveis diretamente dentro da string usando chaves `{}`.

Prompting e In-Context Learning

Com o avanço dos modelos de linguagem como o GPT-4, novas estratégias foram desenvolvidas para melhorar o desempenho desses sistemas em tarefas complexas. Duas das mais importantes são o Raciocínio Prompting e o In-Context Learning. Ambas aproveitam a forma como os modelos processam entradas textuais (prompts) para induzir comportamentos desejáveis, como raciocínio estruturado ou aprendizado temporário.

Raciocínio Prompting e In-Context Learning são estratégias complementares que expandem as capacidades dos modelos de linguagem. Enquanto o primeiro foca em melhorar a qualidade do raciocínio, o segundo permite uma forma flexível e rápida de adaptar o modelo a novas tarefas. Combinadas, essas abordagens tornam os modelos mais versáteis, explicativos e eficientes em uma ampla gama de aplicações.

Raciocínio Prompting

O Raciocínio Prompting (ou Reasoning Prompting) é uma técnica que orienta o modelo a seguir etapas lógicas antes de fornecer uma resposta. Um exemplo clássico dessa abordagem é o uso da frase "Vamos pensar passo a passo" ("Let's think step by step"), que encoraja o modelo a dividir o problema em partes menores e resolver cada uma delas de forma sequencial. Essa abordagem melhora significativamente o desempenho em tarefas que envolvem lógica, matemática, ou interpretação complexa de linguagem.

Existem variações mais sofisticadas, como o Chain-of-Thought Prompting (CoT), que induz o modelo a gerar cadeias de raciocínio explícitas. Isso é especialmente útil em tarefas onde a simples associação entre entrada e saída não é suficiente, e é necessário simular uma linha de pensamento coerente.

In-Context Learning

In-Context Learning (aprendizado em contexto) é a capacidade dos modelos de aprender padrões e realizar tarefas com base apenas nos exemplos fornecidos no próprio prompt, sem ajustes nos pesos internos da rede neural. Ou seja, o

modelo "aprende" temporariamente durante a conversa ao observar exemplos fornecidos pelo usuário.

Essa técnica funciona como uma forma de demonstração: o usuário apresenta um ou mais exemplos de entrada e saída, e o modelo usa esse contexto para inferir como resolver exemplos semelhantes.

Exemplo:

Exemplo 1:

Pergunta: Qual é a capital da França?

Resposta: Paris

Exemplo 2:

Pergunta: Qual é a capital da Itália?

Resposta:

Com base nesses exemplos, o modelo infere que deve responder "Roma".

Essa estratégia é extremamente poderosa para personalizar o comportamento do modelo sem necessidade de reprogramação ou reentrenamento.

Prompting e In-Context Learning

TESTES

Prompt 1

“Um trem sai de uma cidade A às 8h da manhã, viajando a 60 km/h. Outro trem sai da cidade B às 9h, no mesmo trajeto e direção, a 90 km/h. A distância entre as duas cidades é de 300 km. Em que horário os dois trens se encontram? Vamos pensar passo a passo.”

Resposta ChatGPT:

Passo 1: Entender o problema

- Trem 1 sai da cidade A às 8h, com velocidade de 60 km/h.
- Trem 2 sai da cidade B às 9h, com velocidade de 90 km/h, na mesma direção.
- A distância entre as cidades é de 300 km.
- Pergunta: Em que horário os dois trens se encontram?

Passo 2: Ver o que acontece entre 8h e 9h

Somente o Trem 1 está viajando entre 8h e 9h.

- Ele viaja por 1 hora a 60 km/h.
- Então, ele percorre:

$$Distancia = 60km/h \times 1h = 60km$$

Portanto, às 9h, o Trem 1 está a 60 km da cidade A, e a 240 km da cidade B (300 km - 60 km).

Passo 3: A partir das 9h, os dois trens estão em movimento

A partir de agora, o Trem 1 está 240 km à frente do Trem 2 e ambos estão se movendo na mesma direção:

- Velocidade do Trem 1: 60 km/h
- Velocidade do Trem 2: 90 km/h

Como o Trem 2 está mais rápido, ele vai se aproximar do Trem 1 a uma velocidade relativa de:

$$90 - 60 = 30km/h$$

Passo 4: Calcular o tempo para o Trem 2 alcançar o Trem 1

O Trem 2 precisa percorrer os 240 km de diferença a 30 km/h:

$$Tempo = 240 km / 30 km/h = 8 \text{ horas.}$$

Passo 5: Somar esse tempo ao horário de partida do Trem 2

O Trem 2 parte às 9h. Então, 8 horas depois:

$$9h + 8h = 17h$$

Resposta Final: Os trens se encontram às 17h (5 da tarde).

Resposta Gemini:

Passo 1: Analisar a situação inicial

- O trem que sai de A já viajou por 1 hora quando o trem que sai de B começa a se mover (das 8h às 9h).
- Nessa 1 hora, o trem de A percorreu uma distância de:
 $\text{Distância A} = \text{Velocidade A} \times \text{Tempo} = 60 \text{ km/h} \times 1 \text{ h} = 60 \text{ km}$

Passo 2: Calcular a distância restante entre os trens

- A distância total entre as cidades A e B é de 300 km.
- Após a primeira hora, a distância entre os dois trens é: $\text{Distância restante} = \text{Distância total} - \text{Distância A} = 300 \text{ km} - 60 \text{ km} = 240 \text{ km}$

Passo 3: Calcular a velocidade relativa dos trens

- Como os trens estão viajando na mesma direção, a velocidade relativa com que se aproximam é a diferença entre suas velocidades: $\text{Velocidade relativa} = \text{Velocidade B} - \text{Velocidade A} = 90 \text{ km/h} - 60 \text{ km/h} = 30 \text{ km/h}$

Passo 4: Calcular o tempo para o encontro

- Para encontrar o tempo que levará para os trens se encontrarem, dividimos a distância restante pela velocidade relativa:

$$\text{Tempo} = \frac{\text{Distância restante}}{\text{Velocidade relativa}} = \frac{240 \text{ km}}{30 \text{ km/h}} = 8 \text{ horas}$$

Passo 5: Determinar o horário do encontro

- O trem B saiu às 9h. O encontro acontecerá após 8 horas de viagem do trem B.
- Horário do encontro = Horário de partida do trem B + Tempo para o encontro
- Horário do encontro = 9h + 8 horas = 17h

Resposta: Os dois trens se encontrarão às 17h.

Prompt 2

“Transforme as frases informais em linguagem mais formal:

Exemplo 1:

Informal: "Me manda o relatório aí."

Formal: "Por gentileza, poderia me enviar o relatório?"

Exemplo 2:

Informal: "A gente precisa terminar isso logo."

Formal:"

Resposta ChatGPT: Formal: "Precisamos concluir esta tarefa o quanto antes."

Resposta Gemini: Formal: "Precisamos concluir isto o mais breve possível."

Prompt 3

"Exemplo:

Cachorro está para filhote assim como gato está para:

Resposta: Vamos pensar passo a passo.

- Cachorro é o animal adulto.
- Filhote é a forma jovem do cachorro.
- Gato é o animal adulto.
- A forma jovem do gato é o filhote de gato, ou seja: gatinho.

Resposta: gatinho

Agora tente:

Árvore está para semente assim como flor está para:

Resposta: Vamos pensar passo a passo."

Resposta ChatGPT:

Vamos pensar passo a passo:

- Árvore é o ser vivo adulto.
- Semente é a forma jovem ou inicial da árvore.
- Flor é o ser vivo adulto.
- A forma jovem ou inicial da flor é o que dá origem a ela.

Pensando nisso:

- As flores crescem a partir de sementes também.
- Portanto, a relação entre árvore e semente é a mesma entre flor e semente.

Mas se quisermos evitar repetir "semente" e buscarmos uma relação mais específica para "flor", podemos pensar:

- Flor está para botão (o estágio jovem ou em desenvolvimento da flor).

Resposta: botão

Resposta Gemini:

Árvore é a planta adulta.

Semente é o que dá origem a uma nova árvore.

Flor é a estrutura reprodutiva de muitas plantas.

O que geralmente se desenvolve a partir da flor e contém as sementes é o fruto.

Resposta: fruto

Rational Engineering

Rational Engineering (Engenharia Racional) é uma abordagem estratégica usada em diversos campos, como engenharia, ciência da computação, design de sistemas e inteligência artificial, para estruturar e justificar o raciocínio por trás de decisões técnicas. Essa estratégia é centrada na ideia de tornar explícito o processo de tomada de decisão, utilizando argumentos lógicos e bem fundamentados.

Definição e Objetivo

Rational Engineering refere-se à prática de documentar e estruturar o raciocínio que leva a determinadas decisões durante o desenvolvimento de um sistema, produto ou solução. O objetivo é tornar o processo mais transparente, reproduzível e compreensível, tanto para os envolvidos no projeto quanto para avaliadores externos.

Estratégia de Raciocínio em Rational Engineering

A estratégia de raciocínio em Rational Engineering geralmente envolve:

- Identificação do problema: definição clara do que precisa ser resolvido.
- Coleta de evidências: dados, requisitos, restrições e contexto.
- Geração de alternativas: diferentes formas de abordar o problema.
- Avaliação crítica: análise comparativa entre as alternativas com base em critérios objetivos.
- Justificação da escolha: explicitação da lógica que levou à solução escolhida.
- Documentação do raciocínio: registro estruturado de todo o processo decisório.

Aplicações

- Engenharia de Software: na escolha de arquiteturas, linguagens ou frameworks.
- Design de Sistemas: justificando decisões de usabilidade ou interfaces.

- IA e Aprendizado de Máquina: explicando modelos, parâmetros ou abordagens.
- Gestão de Projetos: registrando decisões estratégicas e operacionais.

Benefícios

- Transparência: todos entendem o “porquê” das decisões.
- Rastreabilidade: facilita revisões e auditorias.
- Melhoria contínua: decisões anteriores podem ser avaliadas e refinadas.
- Facilita colaboração: equipes diferentes conseguem alinhar entendimentos.

A estratégia de raciocínio da Rational Engineering é fundamental para garantir que decisões técnicas sejam bem pensadas, justificadas e comunicadas de forma clara. Isso não só melhora a qualidade dos projetos, como também reduz riscos e conflitos futuros, promovendo um desenvolvimento mais consciente e eficiente.

Rational Engineering

TESTES

Prompt 1

"Você é um engenheiro de software encarregado de escolher entre três linguagens de programação — Python, Java e Rust — para desenvolver um sistema bancário com alta demanda de segurança e desempenho. Utilize a estratégia de Rational Engineering para apresentar sua escolha final, documentando o raciocínio completo: identificação do problema, critérios de escolha, comparação das alternativas e justificativa da decisão."

Resposta Chat GPT

Escolha de Linguagem para Sistema Bancário: Python, Java ou Rust — Aplicando a Estratégia de Rational Engineering

1. Identificação do Problema

Estamos encarregados de desenvolver um sistema bancário de grande porte que exige:

- **Alta segurança:** proteção contra vulnerabilidades, memória segura, criptografia robusta, autenticação, controle de acesso e prevenção de falhas.
- **Alto desempenho:** resposta rápida, gerenciamento eficiente de recursos, suporte a alto volume de transações simultâneas.
- **Manutenibilidade e escalabilidade:** código limpo, fácil de manter e estender ao longo do tempo.

- **Ecossistema confiável:** bibliotecas maduras, ferramentas de desenvolvimento, comunidade de suporte.
- **Equipe capacitada:** disponibilidade de profissionais qualificados para manutenção e evolução da solução.

2. Critérios de Escolha

Com base nos requisitos, definimos os seguintes critérios de avaliação (pontuação de 1 a 5):

Critério	Peso
Segurança	30%
Desempenho	25%
Manutenibilidade	15%
Maturidade do ecossistema	15%
Disponibilidade de talento	15%

3. Comparação das Alternativas

Python

- **Segurança: 3/5**

Possui boas bibliotecas de segurança, mas sendo uma linguagem interpretada e de tipagem dinâmica, está mais suscetível a erros em tempo de execução e vulnerabilidades acidentais.

- **Desempenho: 2/5**

Python é relativamente lento e pouco eficiente em computação intensiva. Pode ser usado com extensões C/C++ para acelerar partes do código, mas isso adiciona complexidade.

- **Manutenibilidade: 5/5**

Código altamente legível, comunidade com boas práticas e muitas ferramentas de teste.

- **Maturidade do ecossistema: 5/5**

Rico em frameworks e bibliotecas, embora poucos sejam otimizados para aplicações bancárias de alta escala em tempo real.

- **Disponibilidade de talento: 5/5**

Python é amplamente ensinado e utilizado; há muitos profissionais disponíveis.

Java

- **Segurança: 4/5**

Forte em segurança (controle de memória automático, sandboxing, frameworks robustos). Muito utilizado em ambientes bancários e financeiros.

- **Desempenho: 4/5**

Embora não tão rápido quanto código nativo, a JVM é altamente otimizada, com garbage collection eficiente e bom suporte para aplicações concorrentes e distribuídas.

- **Manutenibilidade: 4/5**
Código mais verboso, mas com excelente suporte a boas práticas, refatoração e ferramentas empresariais.
- **Maturidade do ecossistema: 5/5**
Um dos ecossistemas mais maduros para aplicações bancárias (Spring, Jakarta EE, Hibernate).
- **Disponibilidade de talento: 5/5**
Java é popular há décadas. Muitos desenvolvedores experientes no setor bancário.

Rust

- **Segurança: 5/5**
Segurança de memória em tempo de compilação, ausência de garbage collector, elimina classes inteiras de vulnerabilidades. Ideal para sistemas críticos.
- **Desempenho: 5/5**
Equivalente ao C/C++, altamente eficiente e previsível, ideal para sistemas de tempo real e transações de alto desempenho.
- **Manutenibilidade: 3/5**
A curva de aprendizado é íngreme. Código mais complexo, embora altamente confiável a longo prazo.
- **Maturidade do ecossistema: 3/5**
Ainda jovem, especialmente em aplicações corporativas. Falta de frameworks empresariais maduros comparáveis a Java.
- **Disponibilidade de talento: 2/5**
Poucos desenvolvedores experientes, especialmente no setor bancário.

4. Avaliação Quantitativa

Multiplicamos os valores por seus respectivos pesos:

Critério	Peso	Python	Java	Rust
Segurança	30 %	0.9	1.2	1.5
Desempenho	25 %	0.5	1.0	1.25
Manutenibilidade	15 %	0.75	0.6	0.45
Maturidade do ecossistema	15 %	0.75	0.75	0.45
Disponibilidade de talento	15 %	0.75	0.75	0.30
Total		3.65	4.30	3.95

5. Justificativa da Decisão

Com base na análise racional dos critérios definidos, **Java** se destaca como a melhor escolha para o desenvolvimento de um sistema bancário com alta exigência de **segurança, desempenho, maturidade e escalabilidade**.

Embora **Rust** ofereça o melhor desempenho e segurança de memória, ainda não está suficientemente maduro em ecossistemas corporativos, e a escassez de talentos é um risco relevante. Já **Python**, apesar de excelente para prototipagem e automação, não atende bem aos requisitos de desempenho e segurança em ambientes críticos.

✓ Decisão Final: Java

Java é a linguagem mais equilibrada entre segurança, desempenho, ecossistema e mão de obra disponível, sendo a melhor escolha racional para o desenvolvimento de um sistema bancário de missão crítica.

Prompt 2

"Você é um designer UX encarregado de escolher entre uma navegação baseada em abas, um menu lateral ou um menu suspenso para um aplicativo mobile de gerenciamento financeiro. Use a estratégia de Rational Engineering para analisar as opções e justificar sua escolha final, detalhando os critérios utilizados, o contexto do usuário e as implicações de cada alternativa."

Resposta Chat GPT

Escolha do Padrão de Navegação para um App Mobile de Gerenciamento Financeiro — Aplicação da Estratégia de Rational Engineering

1. Identificação do Problema

Estamos projetando a interface de **navegação principal** para um **aplicativo mobile de gerenciamento financeiro pessoal**. O objetivo é proporcionar uma experiência:

- **Intuitiva e eficiente**
- **Acessível com uma só mão**
- **Clara na hierarquia de informações**
- **Escalável** para novos recursos

O aplicativo incluirá funcionalidades como:

- Visão geral de saldo e gastos
- Registro de despesas/receitas
- Orçamentos por categoria
- Metas financeiras
- Relatórios e gráficos
- Configurações da conta

Devemos escolher entre três padrões de navegação:

- **Abas inferiores (bottom tabs)**
- **Menu lateral (drawer)**
- **Menu suspenso (dropdown)**

2. Definição dos Critérios

Cada padrão será avaliado com base nos seguintes critérios (com pesos):

Critério	Peso
	o
Acessibilidade e usabilidade	30
	%

Clareza e visibilidade das opções	25 %
Escalabilidade	20 %
Espaço e impacto visual	15 %
Familiaridade para usuários	10 %

3. Comparação das Alternativas

✓ 1. Navegação por Abas (Bottom Tabs)

- **Acessibilidade:** 5/5
Muito acessível com o polegar; ideal para telas pequenas.
- **Visibilidade:** 5/5
Opções principais estão sempre visíveis, sem interações extras.
- **Escalabilidade:** 3/5
Ideal para 3 a 5 seções principais. Pode ficar congestionado se tiver muitos itens.
- **Impacto visual:** 4/5
Ocupa pouco espaço e mantém interface limpa.
- **Familiaridade:** 5/5
Padrão amplamente reconhecido em apps financeiros (ex: Nubank, PicPay, C6).

Total ponderado: 4.6

2. Menu Lateral (Drawer)

- **Acessibilidade:** 3/5
Requer movimento lateral ou clique em ícone — menos eficiente com uma mão.
- **Visibilidade:** 2/5
Itens ficam escondidos até que o menu seja aberto. Pode gerar descoberta tardia.
- **Escalabilidade:** 5/5
Ideal para muitos itens de navegação. Pode conter submenus e agrupamentos.
- **Impacto visual:** 4/5
Interface principal fica limpa, mas o menu pode sobrepor conteúdo.
- **Familiaridade:** 4/5
Comum em apps de conteúdo e corporativos, mas menos frequente em apps financeiros.

Total ponderado: 3.5

3. Menu Suspenso (Dropdown, geralmente no topo)

- **Acessibilidade:** 2/5
Menos ergonômico — menus superiores exigem movimentação da mão, especialmente em telas grandes.

- **Visibilidade:** 2/5
Opções ocultas por padrão. Baixa descoberta para usuários novos.
- **Escalabilidade:** 4/5
Suporta bem categorias agrupadas, mas exige mais interação.
- **Impacto visual:** 5/5
Interface fica muito limpa. Ideal para apps com foco visual (gráficos, dashboards).
- **Familiaridade:** 3/5
Mais comum em apps B2B ou com foco em filtros, não tanto para navegação estrutural.

Total ponderado: 3.15

4. Contexto do Usuário

- Usuários são **leigos a intermediários em finanças**, mas experientes com smartphones.
 - O app será usado **diariamente ou com frequência**, em contextos rápidos (como registrar um gasto logo após uma compra).
 - **Facilidade de acesso rápido a informações chave** (saldo, gastos, orçamento) é essencial.
 - O app pode crescer, mas inicialmente terá entre **4 a 5 seções principais**.
-

5. Justificativa da Decisão

A navegação por abas inferiores é a melhor escolha para o aplicativo mobile de gerenciamento financeiro, segundo a análise racional.

- **Oferece a melhor combinação de visibilidade, ergonomia e familiaridade.**
- **É ideal para até 5 categorias principais**, o que cobre a maioria das funções previstas.
- Facilita **uso com uma só mão** e minimiza o esforço cognitivo.
- Mantém uma interface visual clara, sem esconder funcionalidades essenciais.

Embora o **menu lateral** seja mais escalável, ele **sacrifica usabilidade e descoberta**, o que prejudica usuários menos experientes. O **menu suspenso** também oculta conteúdo e **reduz a eficiência em interações frequentes**, o que vai contra os objetivos do app.

✓ **Decisão Final: Navegação por Abas Inferiores**

Recomendação: Usar 4 a 5 abas fixas visíveis (ex: *Resumo, Transações, Orçamentos, Metas, Mais*), sendo a última um botão "Mais" ou menu compacto para recursos menos acessados (como Configurações).

Resultado esperado: Navegação simples, intuitiva, eficiente e escalável, proporcionando a melhor experiência ao usuário no contexto do app.

Self-Consistency Sampling

A estratégia de Self-Consistency Sampling (amostragem de autoconsistência) é uma técnica utilizada em modelos de linguagem, como os da família GPT, para melhorar a qualidade e coerência das respostas geradas em tarefas que envolvem raciocínio complexo.

Em vez de gerar apenas uma única resposta por vez, o Self-Consistency Sampling consiste em gerar várias amostras de raciocínio possíveis (ou "caminhos de pensamento") usando amostragem estocástica (por exemplo, com temperatura > 0). Depois, o modelo escolhe a resposta final com base no consenso entre essas múltiplas amostras — normalmente por votação majoritária sobre a conclusão ou resposta final de cada caminho.

Etapas principais

- Geração de múltiplas cadeias de raciocínio: O modelo é executado várias vezes com configurações que permitem variação nas respostas (por exemplo, top-k, temperature, ou nucleus sampling).
- Extração das respostas finais: De cada cadeia de raciocínio, extrai-se a conclusão final (por exemplo, um número, uma decisão ou resposta direta).
- Agregação por consenso: A resposta mais frequente entre as conclusões finais é escolhida como a resposta definitiva.

Vantagens

- Reduz erros de raciocínio: Ao explorar várias trajetórias de pensamento, é mais provável que o modelo descubra um caminho correto e consistente.
- Melhora a robustez: A estratégia ajuda a compensar casos em que um único raciocínio pode parecer plausível, mas está incorreto.

Aplicações

Self-Consistency é especialmente eficaz em tarefas como:

- Resolução de problemas matemáticos
- Perguntas de lógica
- Planejamento de múltiplas etapas
- Explicações que exigem inferência

Exemplo prático (simplificado)

Suponha que a tarefa seja: "Se João tem o dobro da idade de Maria e juntos eles têm 36 anos, qual a idade de cada um?"

O modelo pode gerar várias cadeias de raciocínio (com variações nos detalhes), mas se 7 em 10 caminhos chegam a “João tem 24 anos e Maria 12”, essa resposta será escolhida por autoconsistência.

Teste 1: Robustez da Resolução de Problemas Complexos

Objetivo: Avaliar a capacidade do sistema de IA de chegar à solução correta de um problema complexo, mesmo quando há múltiplas abordagens válidas ou caminhos de raciocínio intermediários que podem levar a erros, utilizando o Self-Consistency Sampling para filtrar as respostas.

Cenário/Instruções: O sistema de IA será apresentado a um conjunto de **problemas de raciocínio matemático e lógico** de nível avançado (e.g., quebra-cabeças lógicos intrincados, problemas de otimização com restrições múltiplas). Para cada problema, o sistema deve:

1. **Gerar N soluções candidatas independentes** (e.g., $N=10$, $N=20$) para o problema, utilizando diferentes "cadeias de pensamento" ou abordagens internas.
2. **Aplicar o Self-Consistency Sampling:** Para cada solução candidata, o sistema deve gerar uma **justificativa detalhada** ou os passos lógicos que levaram àquela resposta.
3. **Avaliar a consistência:** Comparar as N soluções geradas e suas justificativas. A solução final a ser apresentada deve ser a que aparece com maior frequência entre as N candidatas, ou a que possui a justificativa mais coerente e logicamente sólida de acordo com um mecanismo interno de verificação de consistência (mesmo que heurístico).

CrITÉRIOS de Avaliação:

- **Taxa de Sucesso:** Porcentagem de problemas para os quais o sistema forneceu a **solução correta e completa**.
- **Qualidade da Justificativa:** A clareza, a completude e a correção lógica dos passos de raciocínio apresentados para a solução escolhida. Será pontuado a capacidade da justificativa de ser **autoconsistente** (sem contradições internas) e de estar alinhada com a solução final.
- **Melhora em relação à Linha de Base:** Comparar a performance do sistema com e sem a aplicação do Self-Consistency Sampling. Um resultado bem-sucedido implica em uma **melhora significativa na taxa de sucesso**

e/ou na qualidade das justificativas quando o Self-Consistency Sampling é utilizado.

- **Eficiência (secundário):** O tempo computacional gasto para chegar à solução final usando Self-Consistency Sampling. Embora o foco seja a robustez, um aumento exorbitante no tempo pode ser um fator a considerar.
-

Teste 2: Confiança em Geração de Texto Criativo com Restrições

Objetivo: Avaliar a capacidade do sistema de IA de gerar texto criativo que adere a um conjunto de restrições complexas, utilizando o Self-Consistency Sampling para identificar e selecionar as gerações mais coerentes e alinhadas com as instruções.

Cenário/Instruções: O sistema de IA receberá uma série de prompts para a **geração de textos criativos** (e.g., contos curtos, poemas, roteiros de cenas), cada um contendo múltiplas e, por vezes, sutis **restrições** (e.g., "um poema sobre a primavera sem usar a palavra 'flor'", "um diálogo entre dois personagens que demonstre sarcasmo sem usar um ponto de interrogação", "um conto onde o protagonista muda de ideia sobre um assunto três vezes em menos de 500 palavras"). Para cada prompt, o sistema deve:

1. **Gerar M rascunhos de texto independentes** (e.g., $M=5$, $M=10$) que tentem satisfazer as restrições.
2. **Aplicar o Self-Consistency Sampling:** Para cada rascunho, o sistema deve realizar uma **autoavaliação interna** que verifique o cumprimento de cada uma das restrições fornecidas no prompt. Isso pode envolver uma "verificação de regras" ou uma "interpretação semântica" interna.
3. **Selecionar o Melhor Rascunho:** Com base nas autoavaliações, o sistema deve identificar e apresentar o rascunho de texto que demonstra a **maior consistência no cumprimento de todas as restrições**, possuindo o menor número de violações detectadas ou a maior "pontuação de conformidade".

CrITÉRIOS de Avaliação:

- **Adesão às Restrições:** Para cada texto gerado, será avaliado o **número e a gravidade das violações das restrições**. Um resultado bem-sucedido implica em uma **taxa de violação significativamente menor** para os textos escolhidos pelo Self-Consistency Sampling em comparação com um método de geração direta.
- **Qualidade Criativa (subjetivo):** Avaliação humana da fluidez, originalidade e apelo criativo do texto escolhido, mesmo dentro das restrições. Embora o Self-Consistency Sampling não garanta criatividade per se, ele deve selecionar as opções que melhor equilibram a criatividade com a conformidade.
- **Confiança da Seleção:** A capacidade do sistema de **"explicar" por que o texto escolhido é o mais consistente** (e.g., listando as restrições cumpridas e/ou as quebras de restrições em outras opções).
- **Redução de Erros por Ambiguidade:** A eficácia do Self-Consistency Sampling em **mitigar erros ou inconsistências que surgiriam de prompts ambíguos ou com restrições difíceis de conciliar**, levando a uma produção final mais robusta e confiável.

Chain of Thought Prompting

Chain of Thought Prompting (ou Raciocínio em Cadeia de Pensamento) é uma estratégia em que o modelo é instruído a "pensar em voz alta", ou seja, gerar uma sequência de passos intermediários antes de chegar a uma conclusão. Em vez de fornecer diretamente a resposta final, o modelo explicita o processo de raciocínio, o que muitas vezes resulta em maior precisão e coerência.

Exemplo:

Pergunta sem Chain of Thought:

“Quantos pés tem uma aranha?”

Resposta:

“8.”

Pergunta com Chain of Thought:

“Quantos pés tem uma aranha? Vamos pensar passo a passo.”

Resposta:

“Aranhas são aracnídeos, e aracnídeos normalmente têm 8 patas. Como patas são equivalentes a pés nesse contexto, uma aranha tem 8 pés.”

Benefícios da técnica:

- Maior precisão: ao explicitar o raciocínio, o modelo evita pular etapas e reduz erros.
- Transparência: facilita a interpretação e revisão do processo de pensamento do modelo.
- Melhor desempenho em tarefas complexas: especialmente útil em problemas que envolvem várias etapas ou lógica.

Tipos de aplicação:

- Resolução de problemas matemáticos
- Perguntas de múltipla escolha com raciocínio lógico
- Diagnóstico clínico ou análise de sintomas

- Argumentações e justificativas éticas ou legais

Como aplicar Chain of Thought Prompting:

- Instrução explícita: Adicione frases como “Vamos pensar passo a passo” ou “Explique seu raciocínio”.
- Exemplos guiados: Inclua exemplos com raciocínios detalhados como parte do prompt (exemplos encadeados).
- Treinamento por few-shot prompting: Forneça exemplos de raciocínios corretos antes da tarefa principal.

Chain of Thought Prompting

TESTES

Prompt 1

“João tem o dobro da idade de Ana. Daqui a 5 anos, a soma das idades deles será 55 anos. Quantos anos cada um tem hoje?

Vamos pensar passo a passo.”

Resposta ChatGPT:

Passo 1: Definir as variáveis

Seja:

- A a idade atual de Ana
- J a idade atual de João

Sabemos que:

- **João tem o dobro da idade de Ana**, ou seja: $J=2A$
- **Daqui a 5 anos**, as idades de Ana e João serão $A+5$ e $J+5$, respectivamente.

A soma dessas idades será 55: $(A+5)+(J+5)=55$

Passo 2: Substituir a equação usando a relação entre as idades

Sabemos que $J=2A$, então substituímos na equação da soma: $(A+5)+(2A+5)=55$

Passo 3: Resolver a equação

$$A+5+2A+5=55 \Rightarrow 3A+10=55 \Rightarrow 3A=45 \Rightarrow A=15$$

Passo 4: Calcular a idade de João

$$J=2A=2 \times 15=30$$

Resposta:

- Ana tem **15 anos**
- João tem **30 anos**

Resposta Gemini:

Passo 1: Definir as variáveis

Para facilitar a nossa jornada, vamos usar letras para representar as idades:

- Vamos chamar a idade atual de Ana de A .
- Como João tem o dobro da idade de Ana, a idade atual de João será $2A$.

Passo 2: Pensar no futuro

O problema nos diz o que acontecerá daqui a 5 anos:

- Daqui a 5 anos, a idade de Ana será $A + 5$.
- Daqui a 5 anos, a idade de João será $2A + 5$.

Passo 3: Montar a equação

A informação chave é que a soma das idades deles daqui a 5 anos será 55 anos.

Podemos escrever isso como uma equação:

$$(A+5)+(2A+5)=55$$

Passo 4: Resolver a equação

Agora, vamos resolver essa equação para encontrar o valor de A :

- Combine os termos semelhantes: $A+2A+5+5=55$
- Simplifique: $3A+10=55$
- Subtraia 10 de ambos os lados: $3A=55-10$
- Simplifique: $3A=45$
- Divida ambos os lados por 3: $A=15$
- Encontre o valor de A: $A=15$

Passo 5: Encontrar a idade de cada um hoje

Agora que sabemos a idade atual de Ana ($A = 15$ anos), podemos encontrar a idade atual de João:

- Idade de Ana hoje: $A=15$ anos
- Idade de João hoje: $2A=2 \times 15=30$ anos

Resposta: Hoje, Ana tem 15 anos e João tem 30 anos.

Prompt 2

“Três amigos — Lucas, Marina e Pedro — têm profissões diferentes: médico, engenheiro e professor, mas não necessariamente nessa ordem.

Lucas não é engenheiro.

Pedro não é professor.

Marina não é médica.

Quem é o médico, quem é o engenheiro e quem é o professor?

Vamos resolver isso passo a passo, analisando cada informação.”

Resposta ChatGPT:

Três amigos — Lucas, Marina e Pedro — têm profissões diferentes: médico, engenheiro e professor, mas não necessariamente nessa ordem.

Informações dadas:

- Lucas não é engenheiro.
- Pedro não é professor.
- Marina não é médica.

Etapas da Resolução:

1. Listamos as pessoas e profissões:

- Pessoas: Lucas, Marina, Pedro
- Profissões: médico, engenheiro, professor

2. Aplicamos as restrições:

Pessoa	Médico	Engenheiro	Professor
Lucas	?	×	?
Marina	×	?	?
Pedro	?	?	×

3. Testamos possibilidades:

Suponha que Marina seja engenheira:

- Marina não pode ser médica
- Marina = engenheira

Sobra médico e professor para Lucas e Pedro:

- Pedro não pode ser professor \Rightarrow Pedro = médico
- Lucas não pode ser engenheiro, e Pedro já é médico \Rightarrow Lucas = professor

Resposta Final:

Pessoa	Profissão
a	

Lucas	Professor
-------	-----------

Marina	Engenheira
--------	------------

Pedro	Médico
-------	--------

Todas as condições foram satisfeitas.

Resposta Gemini:

1. Lucas não é engenheiro.

Isso nos deixa duas possibilidades para Lucas: ele pode ser médico ou professor.

2. Pedro não é professor.

Para Pedro, restam as opções de médico ou engenheiro.

3. Marina não é médica.

Marina pode ser engenheira ou professora.

Agora, vamos combinar essas informações para chegar à solução:

- Se Lucas fosse médico, então Pedro teria que ser engenheiro (já que não pode ser professor), e Marina seria professora (já que não pode ser médica). Essa combinação é possível.
- Se Lucas fosse professor, então Pedro teria que ser médico (já que não pode ser professor), e Marina seria engenheira (já que não pode ser médica). Essa combinação também é possível.

Vamos analisar um pouco mais a fundo. Se Pedro é médico, então Lucas não pode ser médico (profissões são diferentes). Isso nos leva de volta à primeira possibilidade que exploramos.

Portanto, a única combinação que respeita todas as informações é:

- Lucas é o médico.
- Pedro é o engenheiro.
- Marina é a professora.

Decomposição de Problemas – Estratégia RACIEM

A decomposição de problemas é uma estratégia de raciocínio que visa transformar uma tarefa complexa em partes menores e mais simples, facilitando o processamento por modelos de linguagem. Em LLMs, essa abordagem se mostrou eficaz para melhorar a precisão, a interpretabilidade e controle das respostas geradas.

No contexto da estratégia **RACIEM** — **R**aciocínio, **C**ompreensão, **E**xecução e **M**emória —, a decomposição auxilia na organização lógica da tarefa, no foco em partes específicas do problema e na recuperação eficiente de soluções anteriores. Esse tipo de raciocínio é particularmente útil para tarefas que exigem múltiplos passos, tomada de decisão encadeada ou uso de conhecimento especializado.

O processo envolve:

- **Least-to-Most Prompting (do menos para o mais):**
Inicia com um subproblema simples e vai progredindo gradualmente para os mais complexos.
- **Decomposed Prompting:**
Divide um problema grande em subtarefas independentes e trata cada uma com uma instância especializada de LLM.
- **Successive Prompting:**
Resolve iterativamente subproblemas em sequência, aproveitando os resultados anteriores para resolver os seguintes.

Aplicação nas Dimensões do RACIEM

- **Raciocínio:** Permite separar etapas lógicas como identificar o objetivo, definir dados, processar resultados e comparar alternativas.

- **Compreensão:** Ajudas no entendimento detalhado de partes do problema, facilitando a inferência contextual e semântica.
 - **Execução:** Divide comandos longos em etapas curtas, reduzindo erros e facilitando a revisão.
 - **Memória:** Em tarefas sequenciais, permite acumular e referenciar as soluções anteriores ao longo do fluxo.
-

Exemplo Prático

Problema completo a ser resolvido:

"Quero montar uma planilha de orçamento mensal que me ajude a organizar meus gastos, prevendo economia mínima de R\$ 300,00 no mês."

Prompt com decomposição (Successive Prompting):

Passo 1: Identifique os principais grupos de despesas mensais.

LLM: "Aluguel, alimentação, transporte, saúde, lazer, contas fixas (água, luz, internet), emergências e extras."

Passo 2: Pergunte o valor médio para cada categoria com base em histórico ou expectativa.

LLM:

- Aluguel: R\$ 1.200
- Alimentação: R\$ 600
- Transporte: R\$ 300
- Saúde: R\$ 200

- Lazer: R\$ 150
- Contas Fixas: R\$ 350
- Emergência: R\$ 100
- Extras: R\$ 250

Passo 3: Calcule o total previsto de despesas.

LLM:

Total = 1.200 + 600 + 300 + 200 + 150 + 350 + 100 + 250 = **R\$ 3.150**

Passo 4: Determine se há possibilidade de economizar R\$ 300.

LLM:

Se a renda for R\$ 3.500, o restante seria:

3.500 - 3.150 = **R\$ 350 de economia** — meta alcançada.

Passo 5: Sugira cortes ou ajustes para aumentar a economia.

LLM:

Reduzir lazer para R\$ 100 e extras para R\$ 150:

Nova economia = R\$ 450

Comparações com Outras Estratégias

1. Chain-of-Thought Prompting

- Diferença: CoT explicita um raciocínio linear e contínuo, mas não necessariamente separa em subproblemas especializados.
- Vantagem: Clareza na argumentação.
- Desvantagem: Pode se perder em problemas multidimensionais.

2. Tool-Augmented Reasoning

- Diferença: Utiliza ferramentas externas para resolver subpartes.
 - Vantagem: Garante precisão matemática ou factual.
 - Desvantagem: Requer acesso a APIs ou integração de ferramentas.
-

Vantagens da Decomposição de Problemas

- Modularidade: fácil manutenção, revisão e depuração.
- Adaptação: funciona bem em fluxos interativos ou pipelines com múltiplos agentes.
- Robustez: reduz falhas ao limitar o escopo de cada subtarefa.

Desvantagens da Decomposição de Problemas

- Demanda controle de contexto e ordem das subtarefas.
- Exige mais iterações e coordenação entre prompts.
- Pode causar redundância se subproblemas forem mal definidos.

Teste 1

Prompt:

“Um investidor aplica R\$ 5.000,00 em um fundo que rende juros compostos de 1,2% ao mês. Ele pretende resgatar após 8 meses. Calcule o valor final e explique cada passo do cálculo.”

Resposta ChatGPT

Passo 1: Identificar a fórmula de juros compostos

A fórmula é:

$$VF = VP \times (1 + i)^n,$$

onde:

- VF = valor futuro
 - VP = valor principal (R\$ 5.000)
 - i = taxa de juros (1,2% = 0,012)
 - n = número de períodos (8 meses)
-

Passo 2: Substituir os valores na fórmula

$$VF = 5.000 \times (1 + 0,012)^8$$

Passo 3: Calcular a base

$$1 + 0,012 = 1,012$$

Passo 4: Elevar à potência de 8

$$1,012^8 \approx 1,104486$$

Passo 5: Multiplicar pelo valor principal

$$5.000 \times 1,104486 \approx \text{R\$ } 5.522,43$$

Resultado Final:

O investidor terá aproximadamente **R\$ 5.522,43** ao final de 8 meses.

Resposta Gemini

Etapas semelhantes com linguagem alternativa:

1. Fórmula identificada corretamente.
 2. Utilização da base 1,012 com 8 expoentes.
 3. Valor futuro aproximado de **R\$ 5.522,43**.
 4. Sugestão de usar Python para verificação matemática.
-

Teste 2

Prompt:

“Você tem 15 lápis, 9 canetas e 6 borrachas. Deseja montar kits escolares com exatamente 1 lápis, 1 caneta e 1 borracha por kit. Quantos kits completos é possível formar? E o que fazer com os itens restantes?”

Resposta ChatGPT

Passo 1: Avaliar os conjuntos possíveis com base nas menores quantidades disponíveis

- Lápis: 15
- Canetas: 9
- Borrachas: 6

Para formar 1 kit, precisa-se de 1 unidade de cada. O número de **kits completos** será limitado pelo item com menor quantidade:

mínimo(15, 9, 6) = 6 kits

Passo 2: Calcular os itens restantes

- Lápis restantes: $15 - 6 = 9$
 - Canetas restantes: $9 - 6 = 3$
 - Borrachas restantes: $6 - 6 = 0$
-

Resultado Final:

- **6 kits completos** podem ser formados.
 - Restam: **9 lápis, 3 canetas, 0 borrachas.**
-

Resposta Gemini

- Utiliza lógica de `min()` para definir o limitador dos kits.
 - Recomenda registrar o resultado como uma função matemática, caso o problema fosse recorrente.
 - Sugere: `def kits_completos(l, c, b): return min(l, c, b)`
-

Prompt 1 – Juros Compostos: Crescimento de Investimento

“Um investidor aplica R\$ 5.000,00 em um fundo que rende juros compostos de 1,2% ao mês. Ele pretende resgatar após 8 meses. Calcule o valor final e explique cada passo do cálculo.”

Resposta ChatGPT:

Passo 1: Identificar a fórmula de juros compostos

$$VF = VP \times (1 + i)^n$$

Passo 2: Substituir os valores na fórmula

$$VF = 5.000 \times (1 + 0,012)^8$$

Passo 3: Calcular a base: $1 + 0,012 = 1,012$

Passo 4: Elevar à potência de 8: $1,012^8 \approx 1,104486$

Passo 5: Multiplicar pelo principal: $5.000 \times 1,104486 \approx \text{R\$ } 5.522,43$

Resultado Final: O investidor terá aproximadamente R\$ 5.522,43 ao final de 8 meses.

Resposta Gemini:

Mesmos passos e valor final, com sugestão de validação por código Python.

Prompt 2 – Combinação de Itens: Montagem de Kits Escolares

“Você tem 15 lápis, 9 canetas e 6 borrachas. Deseja montar kits escolares com exatamente 1 lápis, 1 caneta e 1 borracha por kit. Quantos kits completos é possível formar? E o que fazer com os itens restantes?”

Resposta ChatGPT:

Passo 1: Identificar o item com menor quantidade: $\min(15, 9, 6) = 6$ kits

Passo 2: Calcular itens restantes:

- Lápis restantes: $15 - 6 = 9$
- Canetas restantes: $9 - 6 = 3$
- Borrachas restantes: $6 - 6 = 0$

Resultado Final: 6 kits completos, sobram 9 lápis e 3 canetas.

Resposta Gemini:

Lógica semelhante, com sugestão de transformar o cálculo em uma função Python.

Tool-Augmented Reasoning

O **Raciocínio Aumentado por Ferramentas** (em inglês, *Tool-Augmented Reasoning*) é uma área em crescimento na Inteligência Artificial (IA) que busca combinar as habilidades avançadas de reconhecimento de padrões e linguagem de **Grandes Modelos de Linguagem (LLMs)** com a precisão e o conhecimento específico de **ferramentas computacionais externas**.

Em vez de esperar que os LLMs possuam todo o conhecimento e capacidade de raciocínio internamente, essa abordagem os capacita a:

- **Identificar** quando uma tarefa requer funções especializadas (como uma calculadora, um motor de busca ou uma consulta a um banco de dados).
 - **Utilizar** essas ferramentas de forma autônoma.
 - **Integrar** os resultados dessas ferramentas de volta ao processo de raciocínio para chegar a uma resposta mais precisa e completa.
-

Como Funciona

O processo geralmente segue estes passos:

1. **Compreensão da Tarefa:** O LLM recebe uma pergunta ou comando.
2. **Raciocínio e Identificação da Ferramenta:** O LLM analisa a tarefa e determina se alguma ferramenta externa é necessária para resolvê-la. Muitas vezes, ele gera um "plano" ou "cadeia de pensamento" para isso.
3. **Seleção da Ferramenta:** Com base na necessidade identificada, o LLM escolhe a ferramenta mais apropriada de seu conjunto de ferramentas disponível.
4. **Geração da Entrada da Ferramenta:** O LLM cria os argumentos de entrada necessários para a ferramenta selecionada, traduzindo as perguntas em linguagem natural para comandos ou parâmetros específicos.
5. **Execução da Ferramenta:** A ferramenta externa é ativada e executa sua função.

6. **Integração da Saída:** O resultado da ferramenta é enviado de volta ao LLM.
 7. **Raciocínio Adicional/Resposta Final:** O LLM integra a saída da ferramenta ao seu processo de raciocínio, podendo realizar mais raciocínio em linguagem natural, chamar outras ferramentas ou gerar a resposta final.
-

Benefícios Principais

- **Precisão Aprimorada:** LLMs podem cometer erros em cálculos exatos ou informações factuais. O uso de ferramentas externas garante maior precisão.
 - **Melhor Explicabilidade:** Ao mostrar explicitamente o uso das ferramentas e os passos intermediários, o processo de tomada de decisão do LLM se torna mais transparente e compreensível.
 - **Adaptabilidade e Escalabilidade:** Em vez de treinar LLMs em enormes conjuntos de dados para cada domínio, essa abordagem permite que eles se adaptem a novas tarefas e domínios simplesmente fornecendo acesso a conjuntos de ferramentas relevantes.
 - **Resolução de Problemas Complexos:** Muitas tarefas do mundo real exigem raciocínio em várias etapas, combinando compreensão da linguagem natural com computação precisa. O raciocínio aumentado por ferramentas capacita os LLMs a lidar com esses desafios de forma mais eficaz.
-

Exemplos de Ferramentas e Aplicações

Diversas ferramentas podem ser integradas, incluindo:

- **Calculadoras/Solucionadores Simbólicos:** Para equações matemáticas complexas, aritmética ou problemas algébricos (por exemplo, em ciência ou finanças).
- **Motores de Busca/Sistemas de Recuperação de Conhecimento:** Para acessar informações factuais atualizadas, bases de conhecimento externas ou documentos específicos (por exemplo, para responder a perguntas ou para pesquisa).

- **Interpretadores de Código/Geradores de Programas:** Para executar código, manipular dados ou simular processos (por exemplo, interpretadores Python para análise de dados).
- **APIs (Application Programming Interfaces):** Para interagir com vários serviços externos, bancos de dados ou softwares especializados (por exemplo, APIs de clima, APIs de dados de mercado financeiro).
- **Ferramentas de Manipulação de Tabelas:** Para extrair, organizar e raciocinar sobre dados tabulares (por exemplo, em relatórios financeiros ou conjuntos de dados científicos).

Essa abordagem representa um avanço significativo na criação de sistemas de IA mais capazes, confiáveis e transparentes, que podem utilizar efetivamente recursos externos para resolver problemas complexos do mundo real.

Tool-Augmented Reasoning

O **Raciocínio Aumentado por Ferramentas** (em inglês, *Tool-Augmented Reasoning*) é uma área em crescimento na Inteligência Artificial (IA) que busca combinar as habilidades avançadas de reconhecimento de padrões e linguagem de **Grandes Modelos de Linguagem (LLMs)** com a precisão e o conhecimento específico de **ferramentas computacionais externas**.

Em vez de esperar que os LLMs possuam todo o conhecimento e capacidade de raciocínio internamente, essa abordagem os capacita a:

- **Identificar** quando uma tarefa requer funções especializadas (como uma calculadora, um motor de busca ou uma consulta a um banco de dados).
 - **Utilizar** essas ferramentas de forma autônoma.
 - **Integrar** os resultados dessas ferramentas de volta ao processo de raciocínio para chegar a uma resposta mais precisa e completa.
-

Como Funciona

O processo geralmente segue estes passos:

1. **Compreensão da Tarefa:** O LLM recebe uma pergunta ou comando.
2. **Raciocínio e Identificação da Ferramenta:** O LLM analisa a tarefa e determina se alguma ferramenta externa é necessária para resolvê-la. Muitas vezes, ele gera um "plano" ou "cadeia de pensamento" para isso.
3. **Seleção da Ferramenta:** Com base na necessidade identificada, o LLM escolhe a ferramenta mais apropriada de seu conjunto de ferramentas disponível.
4. **Geração da Entrada da Ferramenta:** O LLM cria os argumentos de entrada necessários para a ferramenta selecionada, traduzindo as perguntas em linguagem natural para comandos ou parâmetros específicos.
5. **Execução da Ferramenta:** A ferramenta externa é ativada e executa sua função.

6. **Integração da Saída:** O resultado da ferramenta é enviado de volta ao LLM.
 7. **Raciocínio Adicional/Resposta Final:** O LLM integra a saída da ferramenta ao seu processo de raciocínio, podendo realizar mais raciocínio em linguagem natural, chamar outras ferramentas ou gerar a resposta final.
-

Benefícios Principais

- **Precisão Aprimorada:** LLMs podem cometer erros em cálculos exatos ou informações factuais. O uso de ferramentas externas garante maior precisão.
 - **Melhor Explicabilidade:** Ao mostrar explicitamente o uso das ferramentas e os passos intermediários, o processo de tomada de decisão do LLM se torna mais transparente e compreensível.
 - **Adaptabilidade e Escalabilidade:** Em vez de treinar LLMs em enormes conjuntos de dados para cada domínio, essa abordagem permite que eles se adaptem a novas tarefas e domínios simplesmente fornecendo acesso a conjuntos de ferramentas relevantes.
 - **Resolução de Problemas Complexos:** Muitas tarefas do mundo real exigem raciocínio em várias etapas, combinando compreensão da linguagem natural com computação precisa. O raciocínio aumentado por ferramentas capacita os LLMs a lidar com esses desafios de forma mais eficaz.
-

Exemplos de Ferramentas e Aplicações

Diversas ferramentas podem ser integradas, incluindo:

- **Calculadoras/Solucionadores Simbólicos:** Para equações matemáticas complexas, aritmética ou problemas algébricos (por exemplo, em ciência ou finanças).
- **Motores de Busca/Sistemas de Recuperação de Conhecimento:** Para acessar informações factuais atualizadas, bases de conhecimento externas ou documentos específicos (por exemplo, para responder a perguntas ou para pesquisa).

- **Interpretadores de Código/Geradores de Programas:** Para executar código, manipular dados ou simular processos (por exemplo, interpretadores Python para análise de dados).
- **APIs (Application Programming Interfaces):** Para interagir com vários serviços externos, bancos de dados ou softwares especializados (por exemplo, APIs de clima, APIs de dados de mercado financeiro).
- **Ferramentas de Manipulação de Tabelas:** Para extrair, organizar e raciocinar sobre dados tabulares (por exemplo, em relatórios financeiros ou conjuntos de dados científicos).

Essa abordagem representa um avanço significativo na criação de sistemas de IA mais capazes, confiáveis e transparentes, que podem utilizar efetivamente recursos externos para resolver problemas complexos do mundo real.

(CHATGPT) Cenário 1: Previsão de crescimento de investimento com juros compostos

Problema:

Você investiu R\$ 10.000,00 em uma aplicação com rendimento de 0,8% ao mês. Quanto você terá acumulado ao final de 5 anos?

Vamos usar o modelo para:

1. **Compreender o problema**
2. **Escolher a fórmula certa (juros compostos)**
3. **Usar a ferramenta de código para calcular com precisão**



Fórmula de juros compostos:

$$M = P \cdot (1+i)^n$$

- M: montante final
- P: valor inicial (10.000)
- i: taxa de juros mensal (0,8% = 0.008)
- n: número de períodos (5 anos = 60 meses)

Agora, vou usar a ferramenta **python** para calcular.



Resultado:

Se você investir **R\$ 10.000,00** a uma taxa de **0,8% ao mês** por **5 anos**, ao final você terá aproximadamente:

 **R\$ 16.129,91**

(CHATGPT) Cenário 2: Quero saber a cotação atual do dólar comercial

O modelo precisa:

1. Entender que se trata de uma informação atual e variável.
2. Acionar a **ferramenta de busca na web**.
3. Trazer uma resposta precisa e atualizada para você.

Aqui está um exemplo de **Tool-Augmented Reasoning** utilizando a ferramenta de busca na web para obter informações atualizadas:

Cenário: Cotação Atual do Dólar Comercial

Pergunta: Qual é a cotação atual do dólar comercial no Brasil?

Processo:

1. **Compreensão do Problema:** Reconhecer que a cotação do dólar é uma informação que varia constantemente e requer dados atualizados.

2. **Utilização de Ferramenta Externa:** Acionar a ferramenta de busca na web para obter a cotação mais recente do dólar comercial.
3. **Análise e Apresentação dos Dados:** Interpretar os dados obtidos e apresentar a informação de forma clara e precisa.

Resultado:

Conforme informações atualizadas em 20 de maio de 2025, a cotação do dólar comercial está em **R\$ 5,6692**, com uma variação positiva de 0,26% em relação ao dia anterior .

(GEMINI) Cenário 3: Planejamento de Viagem com Dados em Tempo Real

Problema: "Quero planejar uma viagem de Franca, SP, para o Rio de Janeiro no próximo sábado. Qual a previsão do tempo para o Rio de Janeiro nesse dia, qual o horário do voo mais barato saindo do aeroporto mais próximo e qual o tempo estimado de carro para chegar lá?"

Raciocínio Aumentado por Ferramentas do LLM:

1. **Análise e Identificação de Ferramentas:** A pergunta exige dados em tempo real e informações específicas sobre transporte e clima.
 - Previsão do tempo: API de clima.
 - Voos: API de passagens aéreas.
 - Tempo de carro: API de mapas/rotas.
 - Localização do aeroporto mais próximo: pode ser inferido do contexto (Franca) ou pesquisado.
2. **Passo 1: Encontrar o Aeroporto Mais Próximo (Inferência/Conhecimento Geral)**
 - **LLM "pensa":** "O aeroporto mais próximo de Franca, SP, com voos comerciais relevantes é o de Ribeirão Preto (RAO)."
3. **Passo 2: Obter Previsão do Tempo (usando API de Clima)**
 - **LLM "pensa":** "Preciso da previsão para o Rio de Janeiro no próximo sábado."
 - **Chamada da Ferramenta:** `api_clima.get_previsao(local='Rio de Janeiro', data='2025-05-24')` (assumindo o próximo sábado como 24 de maio de 2025)
 - **Saída da Ferramenta:** `{'temperatura': '28°C', 'condicao': 'Parcialmente nublado', 'prob_chuva': '20%'}`
 - **LLM Recebe:** Temperatura: 28°C, Condição: Parcialmente nublado, Probabilidade de chuva: 20%.
4. **Passo 3: Pesquisar Voos (usando API de Passagens Aéreas)**
 - **LLM "pensa":** "Buscar voos de Ribeirão Preto (RAO) para Rio de Janeiro (GIG/SDU) no próximo sábado, o mais barato."
 - **Chamada da Ferramenta:**
`api_passagens.pesquisar_voos(origem='RAO', destino='RIO', data='2025-05-24', ordenar_por='preco_asc')`
 - **Saída da Ferramenta:** `{'voo_mais_barato': {'companhia': 'Gol', 'horario_saida': '14:30', 'preco': 'R$ 380,00', 'aeroporto_destino': 'GIG'}}`
 - **LLM Recebe:** Voo mais barato: Gol, 14:30, R\$ 380,00 para GIG.

5. Passo 4: Calcular Tempo de Carro (usando API de Mapas/Rotas)

- **LLM "pensa":** "Calcular o tempo de carro de Franca, SP, para o Rio de Janeiro, RJ."
- **Chamada da Ferramenta:**
`api_mapas.get_tempo_rota(origem='Franca, SP', destino='Rio de Janeiro, RJ', modo='carro')`
- **Saída da Ferramenta:** `{'tempo_estimado': 'aproximadamente 6h 30min'}`
- **LLM Recebe:** Tempo de carro: Aproximadamente 6h 30min.

6. Geração da Resposta Final: O LLM sintetiza os resultados.

- **Resposta:** "Para sua viagem de Franca, SP, para o Rio de Janeiro no próximo sábado (24 de maio):
 - A previsão do tempo para o Rio de Janeiro é de **28°C**, **parcialmente nublado**, com 20% de chance de chuva.
 - O voo mais barato saindo do aeroporto de Ribeirão Preto (RAO) é da **Gol**, às **14:30**, para o Aeroporto do Galeão (GIG), custando **R\$ 380,00**.
 - O tempo estimado de carro de Franca para o Rio de Janeiro é de aproximadamente **6 horas e 30 minutos**."
-

(GEMINI) Cenário 5: Análise de Texto e Geração de Código

Problema: "Tenho a seguinte lista de nomes: 'Ana Silva, João Pereira, Maria Santos, Pedro Oliveira'. Crie um código Python que filtre apenas os nomes que contêm a letra 'o' (maiúscula ou minúscula) e, para cada um, me diga quantos caracteres ele tem (incluindo espaços)."

Raciocínio Aumentado por Ferramentas do LLM:

1. **Análise e Identificação de Ferramentas:** A tarefa envolve manipulação de texto e a criação/execução de código.
 - Geração e execução de código Python: Interpretador de código.
2. **Passo 1: Gerar o Código Python (usando o conhecimento de programação do LLM)**
 - **LLM "pensa":** "Preciso iterar sobre a lista, verificar a presença de 'o' ou 'O' em cada nome e, se presente, calcular o comprimento do nome."

Chamada da Ferramenta (Instrução para o Interpretador de Código):

Python

```
nomes = ['Ana Silva', 'João Pereira', 'Maria Santos', 'Pedro Oliveira']
```

```
resultados = {}
```

```
for nome in nomes:
```

```

if 'o' in nome.lower(): # Verifica 'o' ou 'O'
    resultados[nome] = len(nome)
print(resultados)

```

-
-

3. **Passo 2: Executar o Código e Capturar a Saída (usando Interpretador de Código)**

- **Saída da Ferramenta:** {'João Pereira': 12, 'Pedro Oliveira': 15}
- **LLM Recebe:** Um dicionário com os nomes filtrados e seus cumprimentos.

4. **Geração da Resposta Final:** O LLM formata a saída do código de forma legível.

- **Resposta:** "Claro! Com base na lista fornecida, o código Python filtra os seguintes nomes e seus respectivos cumprimentos:

- **João Pereira:** 12 caracteres
- **Pedro Oliveira:** 15 caracteres

Este é o código utilizado:

```

Python
nomes = ['Ana Silva', 'João Pereira', 'Maria Santos', 'Pedro Oliveira']
resultados = {}
for nome in nomes:
    if 'o' in nome.lower():
        resultados[nome] = len(nome)
print(resultados)
"""

```

Memory and Contextual Reasoning

É uma abordagem que permite a um sistema de IA "pensar com contexto" e "lembrar informações importantes", para responder de forma mais inteligente e personalizada. Essa estratégia é composta por dois elementos que trabalham juntos:

Memory (Memória)

A memória em IA funciona como a nossa própria memória, só que em diferentes níveis:

a) Memória de curto prazo (contextual)

Guarda as informações do diálogo atual, ou seja, tudo o que foi dito recentemente. Permite à IA lembrar o que você perguntou antes e manter a conversa fluida. Essa memória desaparece ao final da conversa (a menos que seja armazenada).

Exemplo:

Se você pergunta:

"Qual a capital da França?"

E depois diz:

"E qual a moeda lá?"

A IA entende que "lá" se refere à França — graças à memória de curto prazo.

b) Memória de longo prazo (persistente)

Guarda informações entre sessões, como preferências, interesses ou dados pessoais.

É ativada em modelos que possuem um recurso de memória configurável (como no ChatGPT com memória ativada).

Ajuda a manter um perfil contínuo do usuário.

Exemplo:

Se você disser em uma conversa:

"Sou professor de matemática e gosto de respostas técnicas."

Na próxima sessão, a IA pode se lembrar disso e ajustar o estilo da resposta, mesmo que você não mencione novamente.

Contextual Reasoning (Raciocínio Contextual)

O raciocínio contextual é a capacidade do modelo de interpretar corretamente o significado das palavras com base no que já foi dito, no ambiente ou no tema da conversa. Ele envolve:

a) Referência e continuidade

Entender pronomes, tempos verbais e termos implícitos.

Exemplo:

Saber que “ele” se refere a “Napoleão” em “Napoleão foi imperador. Ele governou a França.”

b) Inferência lógica

Fazer deduções com base em dados fornecidos, mesmo que não estejam explicitamente conectados.

Exemplo:

Se o usuário diz “Estou com febre e dor no corpo”, o sistema pode inferir que ele está doente, mesmo sem essa afirmação direta.

c) Relações entre conceitos

Ligar ideias diferentes dentro do mesmo assunto ou entre assuntos.

Exemplo:

Relacionar “mercado de ações” com “volatilidade”, “risco” e “investimentos”.

Por que essa estratégia é importante?

Sem memória e raciocínio contextual, a IA funcionaria como um robô que responde a uma pergunta de cada vez, sem conexão entre elas. Com essa estratégia:

A conversa parece mais natural e inteligente.

A IA adapta a linguagem, o tom e o conteúdo às suas necessidades.

As respostas são mais precisas e relevantes.

Exemplo prático em uma conversa com IA

Usuário: “Estou planejando viajar para a Itália.”

IA: “Legal! Já sabe quais cidades quer visitar?”

Usuário: “Roma, Florença e Veneza.”

IA: “Ótimas escolhas! Você quer dicas culturais, gastronômicas ou de transporte?”

Usuário: “Gastronômicas.”

IA: “Em Roma, experimente o cacio e pepe. Em Florença, a bistecca alla fiorentina é imperdível. E em Veneza, o risotto al nero di seppia é tradicional.”

Aqui, a IA usou memória de curto prazo para lembrar as cidades citadas e raciocínio contextual para saber que você queria dicas gastronômicas e associar isso a pratos típicos por cidade.

Memory and Contextual Reasoning

TESTES

Prompt 1

“Vamos jogar um jogo de memória. Vou te contar três fatos sobre mim. Depois, vou fazer perguntas para ver se você lembra.

1. Tenho um gato chamado Imohtep.
2. Sou analista de dados.
3. Minha comida favorita é macarronada.

Agora me diga: qual é o nome do meu gato?”

“Qual é a minha profissão?”

“Você lembra dos três fatos que eu te contei?”

Resposta Chat GPT

1- O nome do seu gato é **Imohtep**. 🐱

2 - Você é **analista de dados**. 📊

3 - Sim, lembro sim! Aqui estão os três fatos que você me contou:

1. Você tem um gato chamado **Imohtep**.
2. Você é **analista de dados**.
3. Sua comida favorita é **macarronada**. 🍝

Pode mandar as próximas perguntas!

Prompt 2

“Ana trabalha como enfermeira em um hospital público. Nos últimos dias, ela tem feito plantões noturnos consecutivos. Ela reclama frequentemente de cansaço, dificuldade de concentração e dores de cabeça. Hoje, ela chegou em casa e esqueceu uma panela no fogo.

O que pode estar acontecendo com Ana? Como você explicaria esse comportamento?”

Resposta Chat GPT

O que pode estar acontecendo com Ana é um quadro de privação de sono e fadiga crônica, muito comum entre profissionais da saúde que enfrentam jornadas longas e plantões noturnos consecutivos.

Explicação do comportamento de Ana

1. Cansaço e dores de cabeça:

- A falta de sono reparador e o acúmulo de horas de trabalho desgastam o corpo. Dores de cabeça são sintomas frequentes de fadiga física e mental.

2. Dificuldade de concentração:

- O sono insuficiente compromete funções cognitivas básicas como atenção, memória de curto prazo e capacidade de tomar decisões.

3. Esquecimento (como a panela no fogo):

- A exaustão mental e a falta de atenção podem levar a esquecimentos perigosos. Isso é um sinal claro de que Ana está cognitivamente sobrecarregada.
-

O que pode estar por trás disso?

- Privação de sono: Plantões noturnos desequilibram o ritmo circadiano (relógio biológico), afetando o sono e a recuperação do corpo.
 - Estresse ocupacional: O ambiente hospitalar é exigente emocional e fisicamente.
 - Síndrome de burnout: Se os sintomas persistirem e se agravarem (apatia, sensação de inutilidade, irritabilidade), é possível que Ana esteja em risco de desenvolver esse quadro.
-

O que pode ser feito?

- Reorganizar a escala de trabalho, se possível, para garantir períodos adequados de descanso.

- Praticar higiene do sono, com um ambiente silencioso e escuro para dormir durante o dia.
- Procurar apoio médico ou psicológico, se os sintomas estiverem prejudicando a qualidade de vida ou se houver suspeita de burnout.

Resumo:

O comportamento de Ana pode ser explicado pela exaustão física e mental causada por privação de sono. Isso compromete sua atenção, memória e saúde geral — sinais que não devem ser ignorados, especialmente em profissionais que lidam com vidas humanas.

Model Context Protocol: Uma Explicação Detalhada

Definição

O **Model Context Protocol** refere-se ao conjunto de regras, mecanismos e estruturas que um sistema de Inteligência Artificial, particularmente um Large Language Model (LLM), utiliza para **gerenciar, armazenar e recuperar informações relevantes de interações anteriores ou dados auxiliares**, a fim de manter a coerência, relevância e compreensão em suas respostas e gerações. Sua finalidade principal é permitir que o modelo "lembre" e utilize o histórico da conversação ou informações contextuais fornecidas para produzir saídas mais precisas e contextualmente apropriadas, superando a limitação inerente de que os modelos de linguagem, por si só, não possuem memória persistente de interações anteriores.

Componentes/Elementos Chave

O Model Context Protocol é composto por vários elementos interconectados que trabalham em conjunto para construir e manter o contexto:

1. Janela de Contexto (Context Window):

- **Função:** É o tamanho máximo da sequência de tokens (palavras, subpalavras, caracteres) que um LLM pode processar em uma única passagem. É a "memória imediata" do modelo. Tudo o que está dentro desta janela é o que o modelo "vê" e considera para gerar sua próxima saída.
- **Detalhe:** Esta janela é geralmente limitada por restrições computacionais e arquitetônicas do modelo (e.g., o número máximo de tokens que a arquitetura Transformer pode lidar eficientemente).

2. Histórico de Conversação (Conversation History):

- **Função:** O registro cronológico de todas as interações (perguntas do usuário e respostas do modelo) dentro de uma única sessão. Este histórico é a base do contexto para conversas.

- **Detalhe:** Frequentemente, o histórico é concatenado e alimentado ao modelo dentro da janela de contexto, com as interações mais recentes geralmente recebendo maior peso ou prioridade.

3. Memória de Curto Prazo:

- **Função:** Representa o contexto imediato e temporário da conversa. É essencialmente o conteúdo dentro da janela de contexto.
- **Detalhe:** É volátil e limitado pelo tamanho da janela. Se a conversa se estender além da capacidade da janela, informações mais antigas podem ser "esquecidas" ou descartadas.

4. Memória de Longo Prazo (via Retrieval Augmented Generation - RAG ou Bases de Conhecimento):

- **Função:** Permite que o modelo acesse informações que estão além da sua janela de contexto imediata ou do seu conhecimento pré-treinado. Isso pode incluir documentos extensos, bases de dados, ou informações específicas do usuário que não cabem na janela de contexto.
- **Detalhe:** Geralmente implementada através de técnicas como **Retrieval Augmented Generation (RAG)**, onde um sistema de recuperação (e.g., um vetorizador e um banco de dados vetorial) busca informações relevantes que são então adicionadas ao prompt do modelo como contexto.
- **Embeddings:** Vetores numéricos que representam o significado semântico de palavras, frases ou documentos. São cruciais para a memória de longo prazo, pois permitem a busca eficiente de informações semanticamente similares em grandes volumes de dados.

5. Instruções do Sistema/Prompt Inicial (System/Initial Prompt):

- **Função:** Define o papel do modelo, suas características, restrições e o estilo de suas respostas desde o início da interação. Fornece um contexto base para todas as futuras interações.

- **Detalhe:** É geralmente colocado no início do contexto e pode incluir persona, regras de formatação, ou informações específicas sobre o domínio.

Mecanismos de Funcionamento

O Model Context Protocol opera através de um fluxo contínuo de gerenciamento de informações:

1. **Recepção da Entrada:** O sistema de IA recebe uma nova entrada do usuário.

2. **Construção do Contexto:**

- O histórico de conversação relevante é recuperado.
- Se a memória de longo prazo for utilizada (e.g., via RAG), informações relevantes (documentos, fatos) são buscadas usando a entrada atual ou o histórico como query. Essas informações são então inseridas no prompt.
- As instruções do sistema/prompt inicial são sempre incluídas.
- Todos esses elementos (instruções, histórico, informações recuperadas, nova entrada) são concatenados para formar a sequência de entrada completa que será alimentada ao LLM.

3. **Tokenização e Embeddings:** A sequência de entrada é tokenizada (dividida em unidades menores, como palavras ou subpalavras) e convertida em embeddings (representações vetoriais).

4. **Processamento pelo LLM (Mecanismo de Atenção):**

- O LLM processa a sequência de embeddings usando sua arquitetura Transformer, com destaque para o **mecanismo de atenção (self-attention)**.
- O mecanismo de atenção permite que o modelo pese a importância de diferentes partes da sequência de entrada ao gerar cada token de saída. Isso significa que ele pode "focar" nas informações mais

relevantes dentro do contexto fornecido, independentemente de sua posição na sequência.

5. **Geração da Resposta:** Com base no contexto processado e em seu conhecimento interno, o LLM gera a próxima sequência de tokens, formando a resposta.
6. **Atualização do Contexto:** A nova entrada do usuário e a resposta gerada pelo modelo são adicionadas ao histórico de conversação para serem utilizadas na próxima interação. Se o histórico exceder a janela de contexto, estratégias de poda (remover as informações mais antigas ou menos relevantes) são aplicadas.

Importância e Benefícios

O Model Context Protocol é crucial por diversas razões:

- **Coerência e Consistência:** Permite que as interações sejam fluidas e lógicas, evitando que o modelo "esqueça" o que foi dito anteriormente.
- **Relevância:** Garante que as respostas do modelo sejam pertinentes ao tópico e ao histórico da conversa.
- **Personalização:** Permite que o modelo adapte suas respostas com base em informações específicas fornecidas pelo usuário ou recuperadas de uma base de conhecimento.
- **Resolução de Ambiguidade:** O contexto ajuda o modelo a interpretar corretamente entradas ambíguas, utilizando informações anteriores para desambiguar intenções ou referências.
- **Habilidade de Continuação:** Essencial para tarefas como geração de código, escrita criativa ou sumarização, onde o modelo precisa continuar um texto ou um conceito iniciado.
- **Melhora da Experiência do Usuário:** Interações mais naturais e inteligentes resultam em maior satisfação do usuário.
- **Resolução de Problemas Complexos:** Permite que o modelo lide com cadeias de raciocínio que exigem múltiplas etapas e referências a informações anteriores.

Desafios e Limitações

Apesar de sua importância, o Model Context Protocol enfrenta desafios significativos:

- **Custo Computacional Elevado:**
 - **Quadrático com o Comprimento do Contexto:** A complexidade do mecanismo de atenção nos Transformers cresce quadraticamente com o comprimento da sequência de entrada. Contextos muito longos requerem memória e tempo de processamento exponencialmente maiores.
 - **Inferência Lenta:** Modelos que operam com janelas de contexto muito grandes podem ter latência de inferência inaceitável para aplicações em tempo real.
- **Degradação de Performance com Contextos Muito Longos (Fenômeno "Lost in the Middle"):** Estudos mostram que, em janelas de contexto muito extensas, os LLMs podem ter dificuldade em focar nas informações mais relevantes que não estão no início ou no fim da sequência. A atenção do modelo pode "diluir-se".
- **Gerenciamento da Relevância:** Decidir quais partes do histórico são mais relevantes para a próxima interação pode ser complexo. Simplesmente concatenar tudo pode sobrecarregar o modelo.
- **Alucinações Aumentadas (com RAG mal implementado):** Se as informações recuperadas (via RAG) forem imprecisas, irrelevantes ou conflitantes, o modelo pode gerar respostas incorretas ou alucinar com base nesses dados "ruins".
- **Latência na Recuperação (RAG):** A busca em uma base de conhecimento externa pode introduzir latência adicional, impactando a experiência do usuário.
- **Segurança e Privacidade:** O armazenamento e gerenciamento de informações sensíveis do usuário no contexto levantam preocupações com segurança e privacidade.

Estratégias de Otimização

Para mitigar os desafios, diversas estratégias de otimização são empregadas:

- **Sumarização (Summarization):**
 - **Para Histórico:** Em vez de manter todo o histórico, um resumo conciso das interações anteriores é gerado periodicamente e adicionado ao contexto.
 - **Para Documentos:** Sumarizar documentos longos antes de inseri-los no contexto para o LLM.
- **Compressão de Contexto:** Técnicas como o uso de representações mais compactas para o histórico (e.g., embeddings do histórico, em vez dos tokens brutos).
- **Técnicas de Atenção Eficientes (Efficient Attention Mechanisms):**
 - Desenvolvimento de arquiteturas de atenção que escalam melhor que o $O(N^2)$ padrão (onde N é o comprimento da sequência), como atenção esparsa, atenção linear, ou atenção baseada em kernel.
 - **FlashAttention:** Uma técnica recente que otimiza o cálculo da atenção para reduzir o uso de memória e aumentar a velocidade.
- **Windowing e Chunking:** Dividir documentos ou históricos longos em "chunks" menores e processar apenas os chunks mais relevantes ou uma janela deslizante.
- **Filtragem de Contexto (Context Filtering):** Utilizar modelos menores ou heurísticas para identificar e selecionar apenas as partes mais críticas do histórico ou dos dados recuperados para incluir no prompt.
- **Retrieval Augmented Generation (RAG) Otimizado:**
 - **Melhoria na Recuperação:** Usar modelos de embeddings mais avançados e técnicas de busca vetorial mais eficientes.
 - **Re-ranking:** Após a recuperação inicial, usar um modelo menor ou uma técnica de re-ranking para classificar a relevância dos documentos recuperados e selecionar os top-K.
 - **Query Expansion/Rewriting:** Expandir ou reescrever a query de busca para o sistema de recuperação para obter resultados mais precisos.

- **Cache de Atenção (KV Cache):** Reutilizar os valores computados para as chaves (K) e valores (V) do mecanismo de atenção de tokens anteriores para evitar recomputações, economizando tempo e memória.
- **Fine-tuning em Contextos Longos:** Treinar ou fazer fine-tuning de modelos especificamente para lidar com contextos maiores, embora isso seja computacionalmente intensivo.
- **Prompt Engineering Avançado:** Estruturar o prompt de forma a guiar o modelo a focar nas informações mais relevantes ou a sintetizar informações complexas.

Exemplos de Aplicação

O Model Context Protocol é onipresente em sistemas de IA modernos:

- **Chatbots de Suporte ao Cliente:** Mantêm o histórico da conversa para responder a perguntas de acompanhamento, resolver problemas em várias etapas e evitar que o usuário precise repetir informações.
- **Assistentes Virtuais (Siri, Google Assistant, Alexa):** Lembram-se de preferências do usuário, compromissos anteriores, ou detalhes de uma tarefa em andamento para fornecer ajuda contextual.
- **Geração de Código (GitHub Copilot):** Analisa o código já escrito no editor (contexto) para sugerir linhas de código, funções ou blocos inteiros que se encaixem logicamente com o que está sendo desenvolvido.
- **Ferramentas de Escrita Criativa:** Mantêm o enredo, personagens e estilo de escrita de um texto para gerar continuações coerentes de histórias ou poemas.
- **Sistemas de Perguntas e Respostas (Q&A):** Utilizam documentos ou bases de conhecimento (memória de longo prazo via RAG) para responder a perguntas complexas que exigem informações específicas.
- **Tradutores em Tempo Real:** Consideram frases anteriores para garantir a consistência e a nuance da tradução em uma conversa.
- **Educação Personalizada:** Modelos que adaptam o conteúdo de ensino com base no progresso e nas dificuldades anteriores do aluno.

Exemplos de Testes Práticos para Avaliar o Model Context Protocol

Aqui estão dois exemplos de testes práticos, focando em diferentes aspectos do Model Context Protocol:

Teste 1: Consistência do Contexto em Conversação Longa (Foco: Gerenciamento da Janela de Contexto e Coerência)

Objetivo: Avaliar a capacidade do sistema de IA de manter a coerência e a consistência em uma conversa prolongada, verificando como ele gerencia a janela de contexto e se informações antigas, mas relevantes, são "esquecidas".

Cenário: Simular uma interação de suporte ao cliente complexa que se estende por várias trocas de mensagens e introduz informações cruciais no início da conversa.

Passos do Teste:

1. Início da Conversa (Contexto Inicial):

- **Usuário:** "Olá, estou com um problema no meu pedido número #XYZ123. A entrega foi marcada para hoje, mas o rastreamento não atualizou."
- **Sistema de IA:** "Entendido. O pedido #XYZ123 está com data de entrega para hoje, 20/05/2025. Vou verificar o status. Poderia me confirmar seu nome completo e e-mail cadastrado?"
- **Usuário:** "Meu nome é João Silva e meu e-mail é joao.silva@email.com. Ah, e a forma de pagamento que usei foi cartão de crédito."
- **Sistema de IA:** "Obrigado, João. Já localizei seu pedido. Ele está em processo de despacho. Aguarde novas atualizações. Precisa de algo mais por enquanto?"

2. Desenvolvimento da Conversa (Introdução de Distratores e Nova Informação):

- **Usuário:** "Sim, aproveitando que estou aqui, queria saber se vocês têm alguma promoção para clientes novos no serviço de streaming? Minha irmã quer assinar."
- **Sistema de IA:** (Responde sobre promoções de streaming, fornecendo informações relevantes)
- **Usuário:** "Ok, obrigado. E sobre o meu pedido #XYZ123, qual o prazo limite para eu reportar se não receber ele hoje?"
- **Sistema de IA:** (Responde sobre o prazo de reporte, por exemplo, "Você tem até 24 horas após o prazo de entrega para reportar o não recebimento.")

3. Retorno ao Contexto Inicial (Teste de "Esquecimento"):

- **Usuário:** "Certo. Voltando ao meu pedido #XYZ123. Qual foi a forma de pagamento que usei?"

Critérios de Avaliação:

- **Aprovado:** O sistema de IA responde corretamente a forma de pagamento ("cartão de crédito"), demonstrando que reteve a informação relevante do início da conversa, apesar das interações intermediárias.
- **Reprovado:** O sistema de IA:
 - Pergunta novamente sobre a forma de pagamento.
 - Dá uma resposta genérica ou incorreta.
 - Indica que não possui essa informação.

Teste 2: Utilização de Informações Contextuais para Geração Específica (Foco: Aplicação de Contexto Recuperado/RAG e Personalização)

Objetivo: Avaliar a capacidade do sistema de IA de integrar e utilizar informações contextuais *fornecidas externamente ou explicitamente no prompt* para gerar uma resposta específica, precisa e personalizada, em vez de recorrer apenas ao seu conhecimento geral.

Cenário: Simular a criação de um e-mail de marketing onde o sistema de IA precisa incorporar detalhes fornecidos em um "briefing" contextual.

Passos do Teste:

1. Fornecimento do Contexto/Briefing:

- **Usuário (ou Sistema que alimenta o contexto):** "Crie um rascunho de e-mail de marketing para o lançamento de um novo produto. Utilize as seguintes informações contextuais:
 - **Nome do Produto:** 'EcoSphere Mini'
 - **Recurso Principal:** Purificador de ar portátil com tecnologia de filtragem HEPA.
 - **Público-alvo:** Profissionais urbanos que buscam bem-estar em ambientes internos.
 - **Benefício Chave:** Ajuda a melhorar a qualidade do ar em escritórios e pequenos apartamentos.
 - **Call-to-Action (CTA):** 'Compre agora na loja online e ganhe 10% de desconto usando o código ECOLAUNCH10.'
 - **Data de Lançamento:** 15 de junho de 2025.
 - **Tonalidade:** Entusiasmada e profissional."

2. Solicitação de Geração:

- **Usuário:** "Com base no briefing acima, por favor, gere o rascunho do e-mail de marketing."

Critérios de Avaliação:

- **Aprovado:** O e-mail gerado pelo sistema de IA:
 - Menciona o nome do produto "EcoSphere Mini".
 - Destaca o recurso principal (purificador de ar portátil, tecnologia HEPA).
 - Direciona-se ao público-alvo e ressalta o benefício (melhora a qualidade do ar em escritórios/apartamentos).

- Inclui o CTA exato ("Compre agora na loja online e ganhe 10% de desconto usando o código ECOLAUNCH10").
 - Menciona a data de lançamento (15 de junho de 2025).
 - A tonalidade é entusiasmada e profissional.
- **Parcialmente Aprovado:** O e-mail inclui a maioria das informações, mas falha em um ou dois pontos cruciais (e.g., o CTA não é exato, ou a data de lançamento está ausente).
- **Reprovado:** O e-mail gerado é genérico, omite várias informações essenciais do briefing, ou "alucina" detalhes que não foram fornecidos.

Conclusão

Este documento explora a crescente sofisticação e as diversas abordagens empregadas no aprimoramento das capacidades de raciocínio de modelos de linguagem de grande escala (LLMs). Ao longo do texto, são apresentadas e analisadas várias estratégias, cada uma com suas peculiaridades e aplicabilidades distintas, visando otimizar o desempenho desses modelos em tarefas complexas.

Inicialmente, este documento aborda o **ajuste fino totalmente supervisionado**, uma técnica fundamental que consiste em refinar modelos pré-treinados com dados rotulados, permitindo que desenvolvam competências específicas em áreas como resolução de problemas matemáticos, interpretação textual e execução lógica baseada em instruções. A estratégia RACIEM (Raciocínio, Compreensão, Execução e Memória) é utilizada como estrutura para detalhar a aplicação do ajuste fino, evidenciando a importância de dados supervisionados para o desenvolvimento de cada uma dessas dimensões.

Em seguida, este documento discute as estratégias de **Prompting e In-Context Learning**, destacando o "Chain of Thought Prompting" e o "Self-Consistency Sampling" como métodos eficazes para melhorar a qualidade e a coerência das respostas geradas pelos LLMs. O "Chain of Thought" induz o modelo a explicitar seus passos de raciocínio, aumentando a transparência e a precisão, enquanto o "Self-Consistency Sampling" gera múltiplas amostras de raciocínio e escolhe a resposta final com base no consenso entre elas, promovendo robustez e reduzindo erros.

A **Decomposição de Problemas** é apresentada como uma estratégia que transforma tarefas complexas em partes menores e mais simples, facilitando o processamento por LLMs. Essa abordagem, integrada à estratégia RACIEM, auxilia na organização lógica da tarefa, no foco em partes específicas do problema e na recuperação eficiente de soluções anteriores. As técnicas de "Least-to-Most

Prompting," "Decomposed Prompting," e "Successive Prompting" são detalhadas como exemplos práticos dessa estratégia.

Por fim, este documento explora o **Tool-Augmented Reasoning**, que combina as habilidades dos LLMs com a precisão e o conhecimento especializado de ferramentas computacionais externas. Essa abordagem permite que os modelos identifiquem quando uma tarefa requer funções especializadas, utilizem essas ferramentas de forma autônoma e integrem os resultados ao seu processo de raciocínio, resultando em respostas mais precisas e completas.

Os testes práticos apresentados ao longo deste documento demonstram a eficácia dessas estratégias em cenários variados, desde cálculos financeiros complexos até o planejamento de viagens com dados em tempo real e a resolução de problemas lógicos. As comparações entre diferentes modelos, como ChatGPT e Gemini, ilustram como cada um pode abordar as tarefas de maneiras distintas, mas eficazes.

Em conclusão, este documento contribui significativamente para a compreensão das técnicas avançadas que estão moldando o futuro dos LLMs. Ao detalhar cada estratégia e fornecer exemplos práticos, o texto oferece uma visão abrangente e aprofundada das ferramentas e métodos que podem ser utilizados para otimizar o desempenho, a precisão e a confiabilidade desses modelos, ressaltando a importância de uma abordagem multifacetada e adaptável para lidar com a complexidade das tarefas que os LLMs são chamados a resolver. A constante evolução dessas estratégias promete transformar ainda mais a forma como interagimos e utilizamos a inteligência artificial.

Referências Bibliográficas

OPENAI. ChatGPT – assistente de linguagem com raciocínio contextual e memória. Disponível em: <https://chat.openai.com/>. Acesso em: 19 maio 2025.

GOOGLE DEEPMIND. Gemini – modelo de linguagem multimodal com suporte a raciocínio avançado. Disponível em: <https://deepmind.google/technologies/gemini/>. Acesso em: 19 maio 2025.

XAI. Grok 3 – modelo de linguagem desenvolvido por xAI, integrado ao X (Twitter). Disponível em: <https://x.ai/>. Acesso em: 19 maio 2025.

Google Gemini 2.5 Flash - Model Context Protocol. Acesso em 20 maio 2025.