

COMO FUNCIONA O SISTEMA RAG

Índice

- [Visão Geral](#)
 - [Fluxo Completo](#)
 - [1. Retrieval - Busca Vetorial](#)
 - [2. Augmentation - Formatação](#)
 - [3. Generation - LLM](#)
 - [Por que Scores Diferentes?](#)
 - [Exemplo Real](#)
 - [RAG vs LLM Puro](#)
 - [Arquitetura Técnica](#)
 - [Conceitos-Chave](#)
 - [Experimentos](#)
-

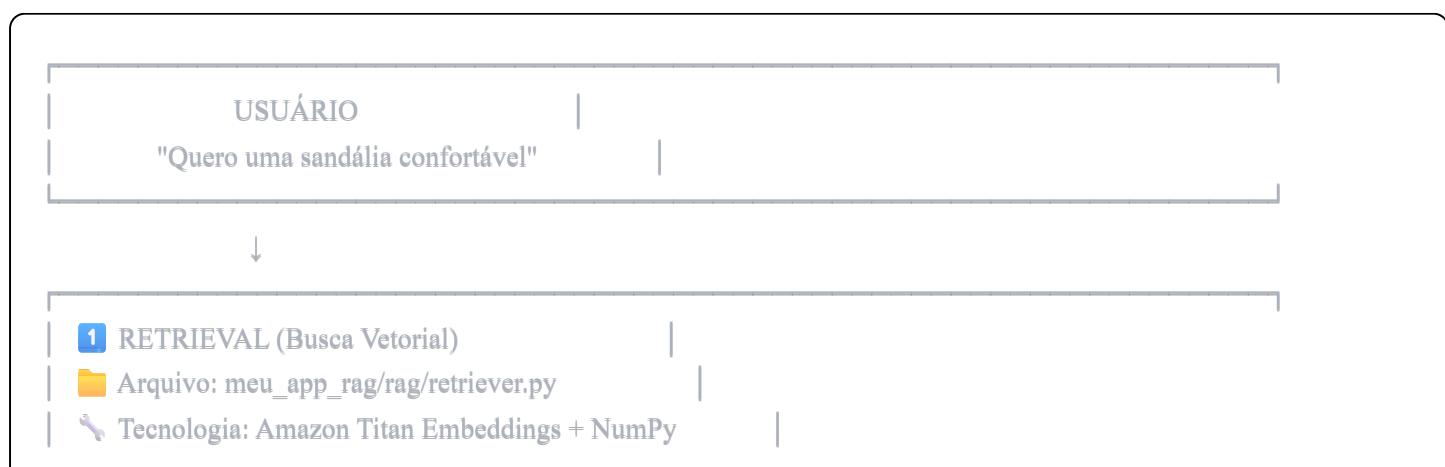
Visão Geral

RAG (Retrieval-Augmented Generation) é uma técnica que combina:

1. **Busca vetorial** (encontrar informações relevantes)
2. **Contexto aumentado** (organizar dados)
3. **LLM** (gerar respostas em linguagem natural)

Objetivo: Fazer o LLM responder com base em dados reais do seu catálogo, evitando "alucinações" (inventar produtos que não existem).

Fluxo Completo



Tempo: ~0.5 segundos

Processo:

1. Normaliza a consulta
2. Gera embedding (vetor de 1024 dimensões)
3. Compara com embeddings dos produtos (similaridade)
4. Retorna top-5 produtos mais relevantes



[Produtos encontrados com scores]



2 AUGMENTATION (Enriquecimento de Contexto)

Arquivo: meu_app_rag/rag/augmenter.py

Tecnologia: Python (formatação de strings)

Tempo: ~0.01 segundos

Processo:

1. Formata cada produto de forma estruturada
2. Adiciona instruções para o LLM
3. Cria contexto rico e organizado



[Contexto estruturado]



3 GENERATION (Geração de Resposta)

Arquivo: meu_app_rag/rag/generator.py

Tecnologia: Claude 3 Sonnet (AWS Bedrock)

Tempo: ~5 segundos

Processo:

1. Envia contexto + consulta para o Claude
2. Claude analisa os produtos
3. Gera resposta em linguagem natural
4. Retorna recomendação baseada em dados reais



RESPOSTA FINAL

```
{  
  "query": "Quero uma sandália confortável",  
  "resposta": "A Sandália Feminina Conforto...",  
  "produtos_encontrados": 5,  
  "produtos": [...],  
  "tempo_processamento": 5.829
```

1 RETRIEVAL - Busca Vetorial

Entrada

"Quero uma sandália confortável"

Processo Detalhado

Passo 1: Normalização

python

```
# retriever.py - método _normalize()
```

Original: "Quero uma sandália confortável"

↓ (lowercase)

"quero uma sandália confortável"

↓ (remove acentos - unidecode)

"quero uma sandalia confortavel"

↓ (remove pontuação)

"quero uma sandalia confortavel"

↓ (remove espaços extras)

"quero uma sandalia confortavel"

Passo 2: Geração de Embedding

python

```
# embeddings.py - método embed()
```

Texto: "quero uma sandalia confortavel"

↓ (AWS Bedrock - Titan Embeddings v2)

Vetor: [0.234, -0.127, 0.891, 0.456, ..., -0.321]

↑

1024 dimensões (números float)

O que é um embedding?

- É uma representação matemática do significado do texto
- Textos similares têm vetores similares
- Permite comparação matemática de significados

Passo 3: Comparaçao com Produtos

Os produtos já têm embeddings pré-calculados (gerados pelo comando `(popular_embeddings)`):

python

Consulta (embedding): [0.234, -0.127, 0.891, ...]

VS

Produto 1 - Sandália Feminina:

Embedding: [0.221, -0.130, 0.895, ...]

↓

Similaridade: 0.6241 ★ (ALTA!)

Produto 2 - Sandália Rasteira:

Embedding: [0.198, -0.115, 0.870, ...]

↓

Similaridade: 0.3306 (MÉDIA)

Produto 3 - Tênis:

Embedding: [0.050, 0.230, -0.450, ...]

↓

Similaridade: 0.1790 (BAIXA)

Produto 4 - Camiseta:

Embedding: [-0.120, 0.340, 0.120, ...]

↓

Similaridade: 0.1702 (BAIXA)

Produto 5 - Óculos:

Embedding: [0.020, -0.050, 0.100, ...]

↓

Similaridade: 0.1045 (MUITO BAIXA)

Passo 4: Cálculo de Similaridade (Cosseno)

python

```
# retriever.py - método _cosine_similarity()
```

Fórmula: $\cos(\theta) = (\mathbf{A} \cdot \mathbf{B}) / (\|\mathbf{A}\| \times \|\mathbf{B}\|)$

Onde:

- \mathbf{A} = vetor da consulta
- \mathbf{B} = vetor do produto
- \cdot = produto escalar
- $\|\mathbf{A}\|$ = norma do vetor \mathbf{A}

Resultado: número entre -1 e 1

- 1.0 = vetores idênticos (100% similar)
- 0.0 = vetores perpendiculares (sem relação)
- -1.0 = vetores opostos

⬆️ Saída

Lista de produtos ordenados por relevância:

```
json  
[  
  {  
    "id": 1,  
    "nome": "Sandália Feminina Conforto",  
    "score": 0.6241,  
    "preco": 79.90,  
    ...  
  },  
  {  
    "id": 2,  
    "nome": "Sandália Rasteira Dourada",  
    "score": 0.3306,  
    ...  
  },  
  ...  
]
```

2 AUGMENTATION - Formatação do Contexto

📝 Entrada

```
python
```

```
produtos = [
    {"id": 1, "nome": "Sandália Feminina", "preco": 79.90, ...},
    {"id": 2, "nome": "Sandália Rasteira", "preco": 59.90, ...},
    ...
]
query = "Quero uma sandália confortável"
```

🔗 Processo

python

augmenter.py - método augment()

Para cada produto:

1. Formata informações de forma estruturada
2. Destaca promoções (se houver)
3. Indica estoque baixo (se < 10)
4. Adiciona score de relevância

Depois:

5. Adiciona instruções para o LLM
6. Define regras de comportamento

⬆️ Saída (Contexto para o LLM)

text

CONSULTA DO USUÁRIO:

"Quero uma sandália confortável"

PRODUTOS ENCONTRADOS (ORDENADOS POR RELEVÂNCIA):

Total de produtos: 5

==== PRODUTO ====

ID: 1

Nome: Sandália Feminina Conforto

Categoria: Calçados

Subcategoria: N/A

💰 Preço: R\$ 79.90

Sem promoção no momento

Marca: Comfort Plus

Cor: Preto

Tamanho: 37

📦 Estoque: 50 unidades

⭐ Avaliação: 4.5 / 5.0

👤 Avaliações: 120 pessoas avaliaram

📝 Descrição: Sandália preta confortável para uso diário

📋 Especificações: N/A

🎯 Relevância: 0.6241

==== PRODUTO ====

ID: 2

Nome: Sandália Rasteira Dourada

...

INSTRUÇÕES PARA O ASSISTENTE:

- Use APENAS os produtos listados acima
- Destaque promoções quando disponíveis
- Mencione estoque baixo se relevante (< 10 unidades)
- Considere as avaliações dos usuários
- Seja objetivo e útil
- NÃO invente informações, marcas, preços ou características
- Se o usuário pedir algo fora dessa lista, responda:

"Não encontrei esse item no catálogo atual"

🎯 Por que isso é importante?

- **Estruturação:** LLM entende melhor dados organizados
- **Grounding:** Força o LLM a usar apenas dados reais
- **Instruções claras:** Define comportamento esperado
- **Evita alucinações:** LLM não inventa produtos

3 GENERATION - Claude no Bedrock

Entrada

python

```
# generator.py - método generate()

system_prompt = """
Você é um assistente de compras...
[contexto dos produtos]
"""

messages = [
    SystemMessage(content=system_prompt),
    HumanMessage(content="Quero uma sandália confortável")
]
```

Processo

Claude 3 Sonnet recebe:

- System Prompt (instruções + contexto)
 - Regras de comportamento
 - Todos os produtos encontrados
 - Scores de relevância
- User Message (consulta original)

Claude analisa:

- Score 0.6241 → Sandália Feminina é a mais relevante
- Avaliação 4.5/5 → Bem avaliada
- Descrição "confortável" → Match direto
- Preço R\$ 79,90 → Razoável
- Estoque 50 unidades → Disponível

Claude gera resposta:

- Recomendação fundamentada em dados

Saída

text

"Com base no catálogo fornecido, a Sandália Feminina Conforto da marca Comfort Plus parece ser a opção mais adequada para você. Ela é descrita como uma 'Sandália preta confortável para uso diário' e tem uma boa avaliação de 4,5/5 por 120 pessoas. O preço é de R\$ 79,90 e há 50 unidades em estoque no tamanho 37.

Outra opção que pode ser confortável é a Sandália Rasteira Dourada da BeachStyle por R\$ 59,90, mas ela tem uma avaliação um pouco menor de 4,3/5 por 85 pessoas..."

🎯 Por que o Score é Diferente para Cada Produto?

Análise dos Scores

json

```
"score": 0.6241 ← Sandália Feminina  
"score": 0.3306 ← Sandália Rasteira  
"score": 0.1790 ← Tênis  
"score": 0.1702 ← Camiseta  
"score": 0.1045 ← Óculos
```

Explicação Detalhada

Produto 1: Sandália Feminina Conforto (0.6241)

Por que score ALTO?

Embedding contém conceitos:

- ├── "sandália" ✓
- ├── "feminina" ✓
- ├── "conforto" ✓
- ├── "uso diário" ✓
- └── "calçados" ✓

Consulta: "sandália confortável"

- ├── "sandália" → MATCH PERFEITO
- └── "confortável" → MATCH PERFEITO

Resultado: ALTA similaridade (0.6241)

Produto 2: Sandália Rasteira Dourada (0.3306)

Por que score MÉDIO?

Embedding contém conceitos:

- └─ "sandália" ✓
- └─ "rasteira" ✓
- └─ "leve" ! (relacionado a conforto)
- └─ "macia" ! (relacionado a conforto)

Consulta: "sandália confortável"

- └─ "sandália" → MATCH
- └─ "confortável" → PARCIAL (inferido)

Resultado: MÉDIA similaridade (0.3306)

Produto 3: Tênis Corrida (0.1790)

Por que score BAIXO?

Embedding contém conceitos:

- └─ "tênis" ✗ (não é sandália)
- └─ "corrida" ✗
- └─ "leve" !
- └─ "respirável" ✗

Consulta: "sandália confortável"

- └─ "sandália" → SEM MATCH
- └─ "confortável" → MUITO INDIRETO

Resultado: BAIXA similaridade (0.1790)

Produto 4 e 5: Camiseta e Óculos (~0.17 e 0.10)

Por que score MUITO BAIXO?

Categorias completamente diferentes:

- └─ Roupas ✗
- └─ Acessórios ✗

Consulta: "sandália confortável"

- └─ Sem relação semântica

Resultado: MUITO BAIXA similaridade

Visualização Geométrica

Espaço Vetorial (simplificado em 2D):

Sandália Feminina (0.6241)

```
•  
/ |  
/ |  
Consulta |  
• |  
\ |  
\ | Sandália Rasteira (0.3306)  
\|•  
\ |  
•  
Tênis (0.1790)
```

```
• Óculos (0.1045)  
(muito distante)
```

Quanto mais próximo no espaço vetorial,
maior a similaridade semântica!

📝 Exemplo Real de Consulta

Entrada do Usuário

```
json  
  
{  
  "query": "Quero uma sandália confortável",  
  "limit": 5  
}
```

Resposta Completa

```
json
```

```
{  
  "query": "Quero uma sandália confortável",  
  "resposta": "Com base no catálogo fornecido, a Sandália Feminina Conforto da marca Comfort Plus parece ser a opção mais  
  \"produtos_encontrados\": 5,  
  \"produtos\": [  
    {  
      \"id\": 1,  
      \"nome\": \"Sandália Feminina Conforto\",  
      \"categoria\": \"Calçados\",  
      \"preco\": 79.9,  
      \"marca\": \"Comfort Plus\",  
      \"cor\": \"Preto\",  
      \"tamanho\": \"37\",  
      \"estoque\": 50,  
      \"avaliacao\": 4.5,  
      \"num_avaliacoes\": 120,  
      \"score\": 0.6241247653961182  
    },  
    {  
      \"id\": 2,  
      \"nome\": \"Sandália Rasteira Dourada\",  
      \"categoria\": \"Calçados\",  
      \"preco\": 59.9,  
      \"marca\": \"BeachStyle\",  
      \"cor\": \"Dourado\",  
      \"tamanho\": \"38\",  
      \"estoque\": 35,  
      \"avaliacao\": 4.3,  
      \"num_avaliacoes\": 85,  
      \"score\": 0.3306025266647339  
    },  
    {  
      \"id\": 3,  
      \"nome\": \"Tênis Corrida Pro Run\",  
      \"categoria\": \"Calçados\",  
      \"preco\": 199.9,  
      \"preco_promocional\": 149.9,  
      \"marca\": \"SportPro\",  
      \"cor\": \"Branco/Azul\",  
      \"tamanho\": \"42\",  
      \"estoque\": 30,  
      \"avaliacao\": 4.8,  
      \"num_avaliacoes\": 250,  
      \"score\": 0.17908956110477448  
    }  
  ],  
}
```

"tempo_processamento": 5.829

}

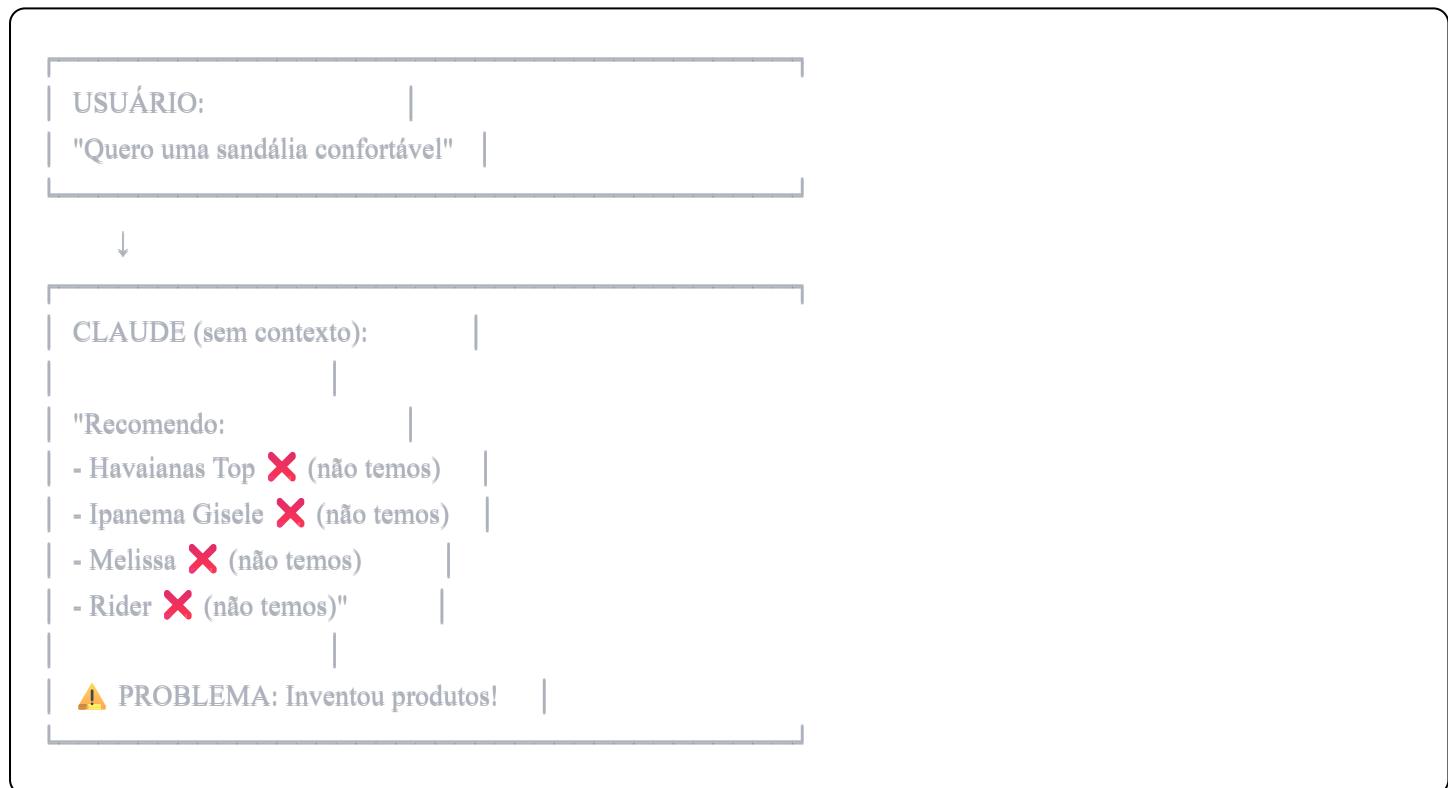
Análise da Resposta

O que o Claude fez bem:

1. Focou na Sandália Feminina (maior score)
2. Destacou a avaliação positiva (4.5/5)
3. Mencionou quantidade de avaliações (120)
4. Comparou com a segunda opção
5. Fez recomendação fundamentada
6. Não inventou nenhum produto
7. Usou apenas dados do contexto

💡 Vantagens do RAG vs Apenas LLM

✗ Sem RAG (LLM Puro)



Problemas:

- ✗ Recomenda produtos que não existem no catálogo
- ✗ Preços inventados
- ✗ Informações desatualizadas
- ✗ Não considera estoque real

- X Cliente fica frustrado

Com RAG (Nosso Sistema)

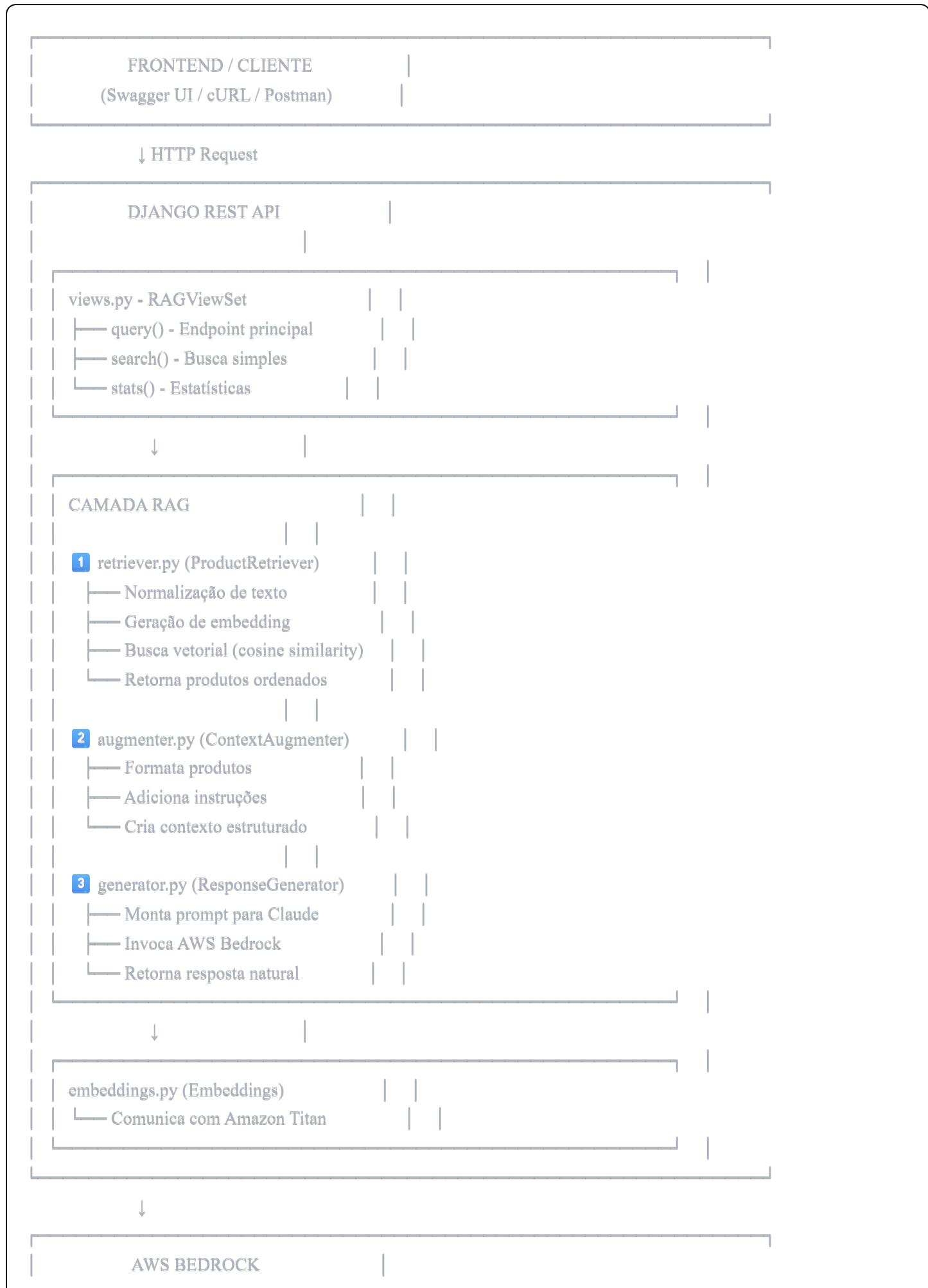


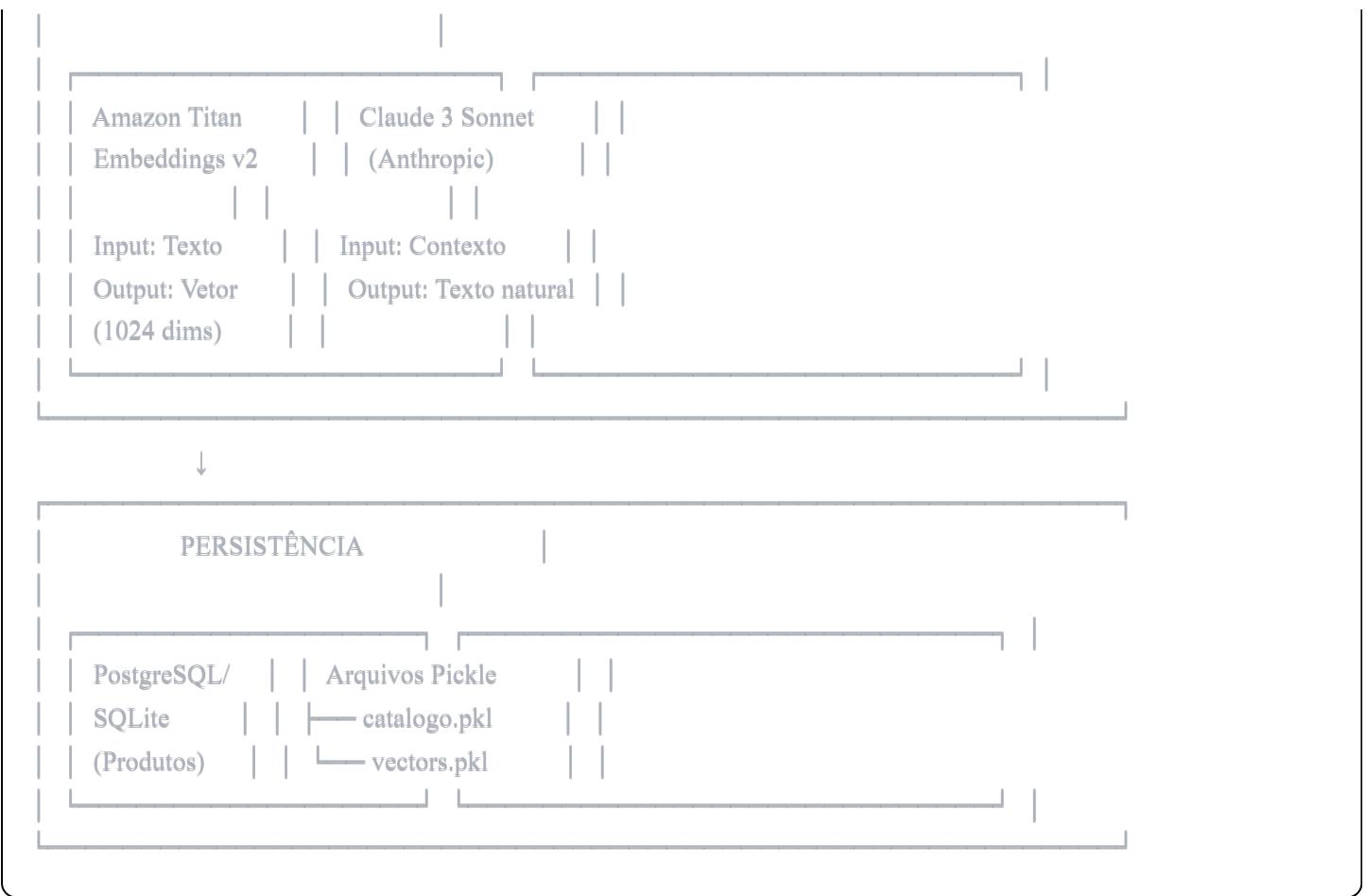
Vantagens:

- Recomenda APENAS produtos existentes
- Preços precisos e atualizados
- Informações de estoque em tempo real
- Avaliações reais de clientes
- Cliente encontra o que procura

Arquitetura Técnica

Stack Completo





Tecnologias Utilizadas

Componente	Tecnologia	Função
Framework Web	Django 5.2.8	Gerenciamento da aplicação
API REST	Django REST Framework	Endpoints e serialização
Documentação	drf-spectacular	Swagger/OpenAPI
Embeddings	Amazon Titan v2	Vetorização de texto (1024D)
LLM	Claude 3 Sonnet	Geração de respostas
Busca Vetorial	NumPy	Cálculo de similaridade
Normalização	unidecode	Remoção de acentos
AWS SDK	boto3	Comunicação com Bedrock
Orquestração LLM	LangChain AWS	Interface com Claude
Banco de Dados	SQLite/PostgreSQL	Armazenamento de produtos
Cache	Pickle	Serialização de embeddings

💡 Conceitos-Chave

1. Embedding

O que é?

- Representação vetorial (numérica) do significado de um texto

- Texto → Vetor de números (ex: 1024 dimensões)

Exemplo:

python

```
"sandália confortável" → [0.234, -0.127, 0.891, ..., -0.321]
"sapato macio"      → [0.221, -0.130, 0.885, ..., -0.318]
"carro rápido"     → [-0.450, 0.890, -0.234, ..., 0.567]
```

Propriedade importante:

- Textos com significados similares têm vetores próximos
- Textos com significados diferentes têm vetores distantes

2. Similaridade do Cosseno

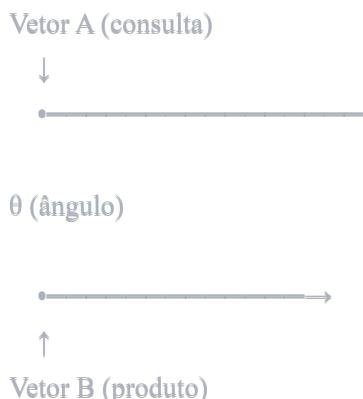
O que é?

- Medida de quão "parecidos" são dois vetores
- Resultado: número entre -1 e 1
 - 1.0 = idênticos
 - 0.0 = sem relação
 - -1.0 = opostos

Fórmula:

$$\cos(\theta) = (\mathbf{A} \cdot \mathbf{B}) / (\|\mathbf{A}\| \times \|\mathbf{B}\|)$$

Visualização:



Se θ é pequeno → $\cos(\theta)$ próximo de 1 → ALTA similaridade
 Se θ é grande → $\cos(\theta)$ próximo de 0 → BAIXA similaridade

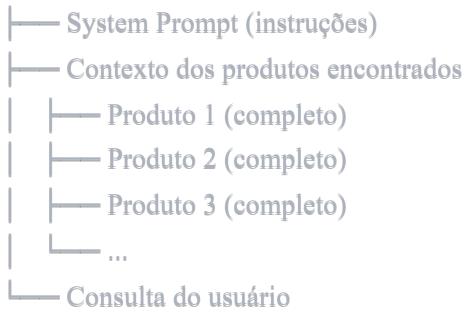
3. Context Window

O que é?

- Quantidade de informação que o LLM pode processar de uma vez
- Para Claude 3 Sonnet: ~200.000 tokens

No nosso sistema:

Context Window do Claude:



Total: ~2.000 tokens (bem dentro do limite)

4. Grounding

O que é?

- Técnica para "ancorar" respostas do LLM em dados reais
- LLM só usa informações fornecidas no contexto
- Evita "alucinações" (inventar informações)

Como implementamos:

python

```
system_prompt = f"""
Você é um assistente que responde EXCLUSIVAMENTE
com base nos produtos fornecidos.
```

REGRAS:

1. NÃO invente produtos
2. NÃO use conhecimento externo
3. SÓ responda usando o catálogo abaixo

CATÁLOGO:

```
{produtos_encontrados}
```

""""

5. Retrieval-Augmented Generation (RAG)

Definição:

- Técnica que combina busca de informações (Retrieval) com geração de texto (Generation)
- LLM recebe contexto relevante antes de gerar resposta

Componentes:

1. **Retrieval:** Busca informações relevantes em uma base de conhecimento
2. **Augmentation:** Enriquece e formata o contexto
3. **Generation:** LLM gera resposta baseada no contexto

Benefícios:

- Respostas baseadas em dados atualizados
- Reduz alucinações
- Permite especialização sem retreinar modelo
- Mantém controle sobre a fonte de verdade

💡 Experimentos Sugeridos

Experimento 1: Busca por Categoria

```
json
{
  "query": "Quero ver produtos da categoria calçados",
  "limit": 5
}
```

O que observar:

- Todos os calçados terão scores altos
- Claude listará as opções disponíveis

Experimento 2: Busca por Preço

```
json
{
  "query": "Produtos até 100 reais",
  "limit": 5
}
```

O que observar:

- Produtos com preço <= 100 terão destaque
 - Claude mencionará os preços
-

Experimento 3: Busca por Promoção

```
json

{
  "query": "Quais produtos estão em promoção?",
  "limit": 5
}
```

O que observar:

- Tênis com preço promocional terá destaque
 - Claude calculará o desconto
-

Experimento 4: Comparação

```
json

{
  "query": "Compare todas as sandálias disponíveis",
  "limit": 5
}
```

O que observar:

- Claude fará análise comparativa
 - Destacará diferenças de preço, avaliação, etc.
-

Experimento 5: Produto Específico

```
json

{
  "query": "Me fale sobre o Tênis Corrida Pro Run",
  "limit": 5
}
```

O que observar:

- Busca vetorial encontrará o tênis
 - Claude dará detalhes completos
-

Experimento 6: Produto que Não Existe

json

```
{  
  "query": "Quero uma bota de couro marrom",  
  "limit": 5  
}
```

O que observar:

- Busca retornará produtos de calçados (similar)
- Claude dirá que não tem bota específica
- Pode sugerir alternativas

Métricas e Performance

Tempo de Processamento

Componente	Tempo Médio
Normalização	~0.001s
Embedding (Titan)	~0.3-0.5s
Busca Vetorial	~0.01s
Formatação	~0.01s
Claude (Bedrock)	~3-6s
TOTAL	~4-7s

Custos AWS (estimativa)

Serviço	Custo/1000 requests
Titan Embeddings	~\$0.10
Claude 3 Sonnet	~\$3.00
TOTAL	~\$3.10

* Valores aproximados, consulte pricing AWS

Troubleshooting

Problema: Scores muito baixos para todos os produtos

Causa: Embeddings podem estar desatualizados

Solução:

```
bash
```

```
python manage.py popularize_embeddings --force
```

Problema: Claude inventa produtos

Causa: System prompt pode estar mal formatado

Solução: Verificar `(generator.py)` - método `(generate())`

Problema: Busca retorna produtos irrelevantes

Causa: Normalização do texto pode estar incorreta

Solução: Verificar `(retriever.py)` - método `(normalize())`

Problema: Erro de throttling no Bedrock

Causa: Muitas requisições simultâneas

Solução: Implementar rate limiting ou retry logic

Referências

- [Amazon Bedrock Documentation](#)
 - [Titan Embeddings](#)
 - [Claude 3 Models](#)
 - [RAG Pattern](#)
 - [Django REST Framework](#)
-

Resumo Final

O que você aprendeu:

1. RAG combina busca + contexto + LLM
2. Embeddings transformam texto em vetores matemáticos
3. Similaridade do cosseno mede relevância

4. Grounding evita alucinações do LLM
5. System prompt controla comportamento do Claude
6. Busca vetorial é mais poderosa que busca por palavras-chave

Arquitetura do sistema:

Consulta → Embedding → Busca Vetorial → Contexto → Claude → Resposta

Tecnologias principais:

- Django + DRF (API)
- Amazon Titan (Embeddings)
- Claude 3 Sonnet (LLM)
- NumPy (Busca vetorial)

 Parabéns! Você tem um sistema RAG de produção funcionando!

Gerado em: 2025-11-16 Versão: 1.0.0