# Desafio Técnico: Implementação de Busca Semântica com Python e PostgreSQL (pgvector)

---

**Objetivo**

Criar um sistema de busca semântica (vetorial) para produtos de uma farmácia, utilizando PostgreSQL com extensão pgvector para armazenar e buscar vetores de similaridade. O sistema deve permitir consultas semânticas para encontrar os produtos mais relevantes e incluir um método UPSERT que atualize os dados da tabela assim como os vetores de embedding.

**Descrição:**

Criar um arquivo Python no formato .ipynb que seja capaz de:

1. **Gerar Embeddings**:

   Criar embeddings para os dados de produtos fornecidos, com o objetivo de proporcionar uma busca semântica flexível e eficiente.

2. **Consultas Semânticas**:

   Realizar buscas semânticas flexíveis que retornem os 5 produtos (resultado em tabela com todos os campos) mais semelhantes para as consultas:

   - *"amoxicilina"*
   - *"amoxicilina 500mg"*
   - *"amoxicilina 500mg generico"*
   - *"genérico do FLUIMUCIL"*
   - *"similar do FLUIMUCIL"*

3. **Método UPSERT:**

   Criar um método que insira ou atualize produtos no banco de dados, garantindo que os embeddings sejam atualizados em caso de alterações.

---

**Instruções**

1. **Configuração do Banco de Dados**:

- Configure ou utilize um banco de dados PostgreSQL com a extensão pgvector habilitada.
- Crie uma tabela products com as seguintes colunas:
  - id_produto: Identificador único do produto.
  - nome_produto: Nome do produto.
  - laboratorio: Laboratório fabricante.
  - substancia: Substância ativa.
  - apresentacao: Apresentação do produto.
  - classe_terapeutica: Classe terapêutica.
  - tipo_produto: Tipo de produto.
  - tarja: Tarja.
  - embedding VECTOR(1536): Campo para armazenar os embeddings.

2. **Embeddings para Melhor Busca**:
   - **Estratégia**: Explique a estratégia adotada para gerar embedding de forma a obter os melhores resultados possíveis na busca.
3. **Método UPSERT**:
   - Implemente um método que:
     - Insira um produto no banco de dados caso o id_produto ainda não exista.
     - Atualize os dados e os embeddings caso o id_produto já esteja presente.
4. **Busca Semântica**:
   - Para cada consulta:
     - Gere o embedding da consulta usando o mesmo modelo de embeddings.
     - Utilize a função cosine_similarity() do pgvector para buscar no banco de dados.
     - Retorne os 5 produtos mais semelhantes.

---

**Critérios de Avaliação**

- **Busca Flexível**:
    - A solução deve ser capaz de lidar com consultas variadas e retornar resultados relevantes.
- **Eficácia dos Embeddings**:
    - Avaliaremos a estratégia utilizada para a geração e uso dos embeddings, com foco na precisão e flexibilidade da busca.
- **Clareza e Organização do Código**:
    - O código deve ser comentado para explicar o funcionamento de cada parte.
- **Criatividade para lidar com os desafios encontrados.**

---

**Entrega**

- Um arquivo .ipynb contendo:
    - A configuração do banco de dados e da tabela.
    - O código para geração e armazenamento de embeddings.
    - O método UPSERT.
    - A busca semântica para as consultas fornecidas.
- Instruções para execução no próprio notebook.

---

**Observações**

- Se julgar necessário utilizar outro banco de dados que não seja o PostgreSQL, leve em consideração que deve conter a parte relacional sincronizada.
- Como sugestão de modelos LLM, utilize os da OpenAI para simplificar o desenvolvimento e provisionamento. Se julgar impactar positivamente o resultado utilizar outro modelo, fique à vontade.
- Os custos com API da OpenAI, Banco de Dados ou outras plataformas serão reembolsados com limite de R$150,00. Caso necessite de um valor maior informe o time de recrutamento antes de consumir. Ao final, favor enviar consumos para reembolso.
- Se houver alguma dúvida busque o time de recrutamento.