Construindo um Sistema Multimodal RAG

Integrando texto, áudio, imagem em um único espaço vetorial

Alex Salgado Developer Advocate @ Elastic

elastic | The Search AI Company

TDC FLORIPA

Three solutions powered by one stack

3 solutions

Enterprise Search

Observability

Security

Powered by the Elastic Stack

Kibana

Elasticsearch

Agent

Beats

Logstash

Deployed anywhere

Elastic Cloud

Elastic Cloud Enterprise

Elastic Cloud on Kubernetes

Saas

Orchestration

elastic

Introdução

O que é um Sistema RAG Multimodal?

- Um sistema de Geração Augmentada por Recuperação (RAG) combina diferentes modalidades de dados, como texto, imagem, áudio e vídeo, para fornecer respostas mais ricas e contextualizadas.

Por que isso é importante?

- No mundo atual orientado por dados, a capacidade de buscar em múltiplos formatos de dados é essencial para uma recuperação mais precisa e significativa.

Preocupações em torno da IA Generativa.

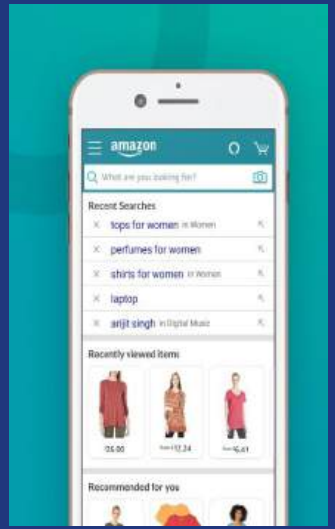
80%

Dados mundiais são não-estruturados

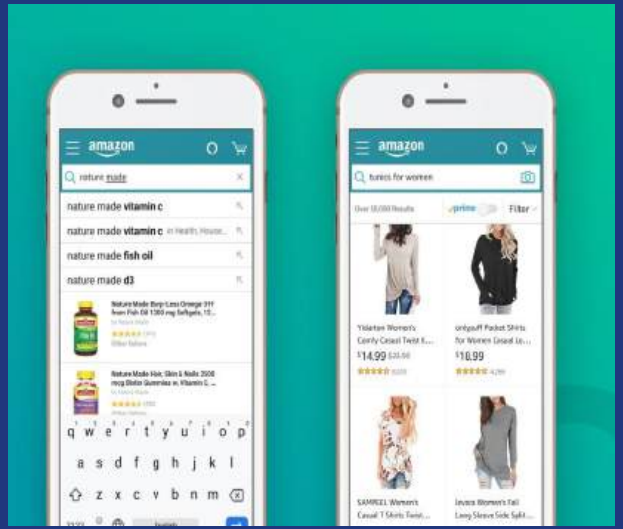
KPMG Generative AI Survey
The Prompt: Generative AI survey | Google Cloud Blog

elastic

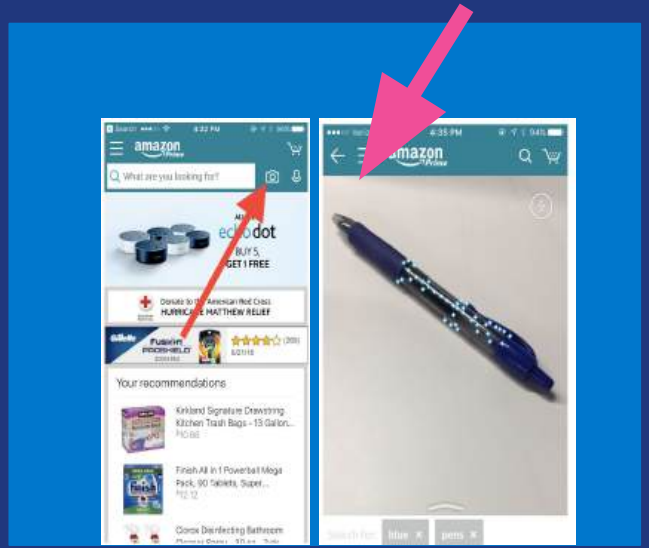
As pessoas estão mudando o jeito de pesquisar



1-Busca textual



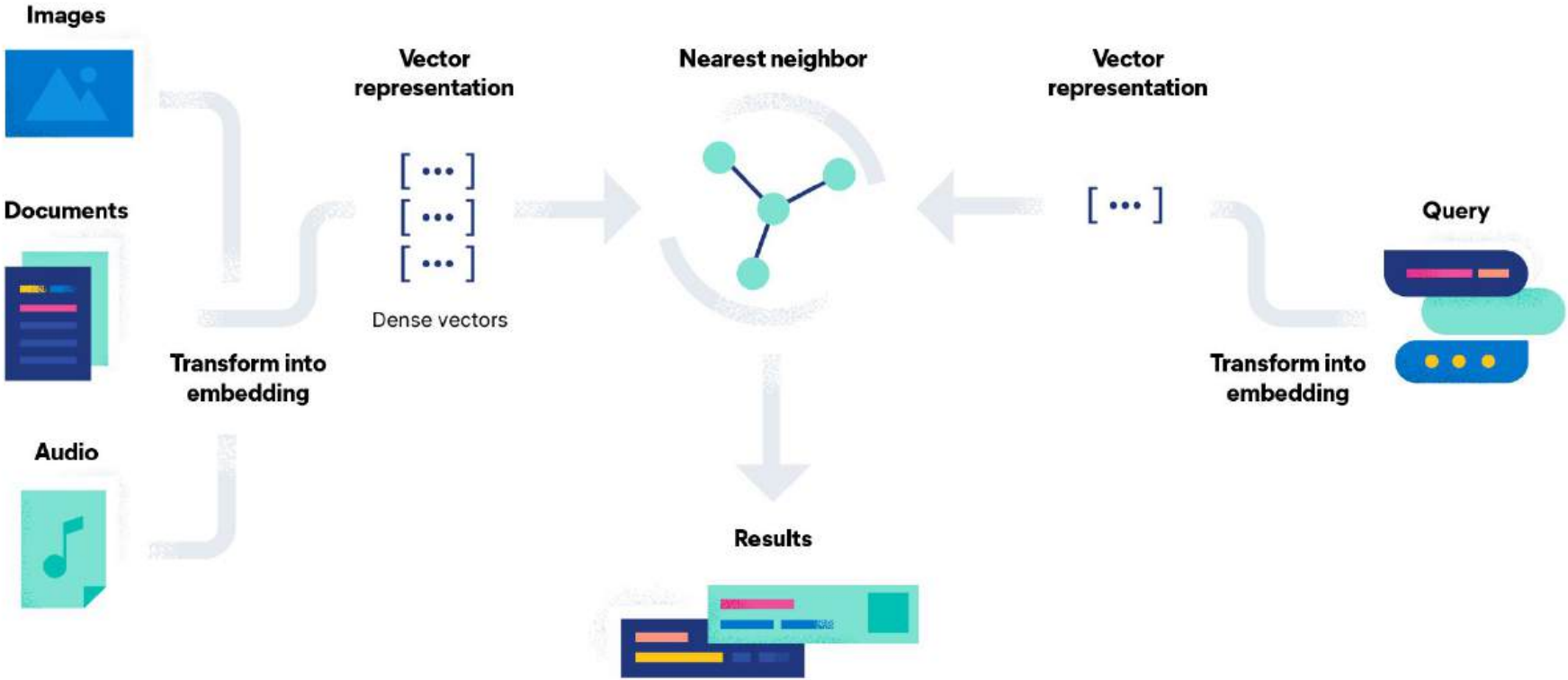
2-Busca textual + Semantica



3-Busca por Imagem e Som

elastic

Queries are also vectorized



O Desafio da Multimodalidade

Limitações Tradicionais

- Silos de dados isolados
- Busca limitada por tipo
- Perda de contexto
- Análise fragmentada

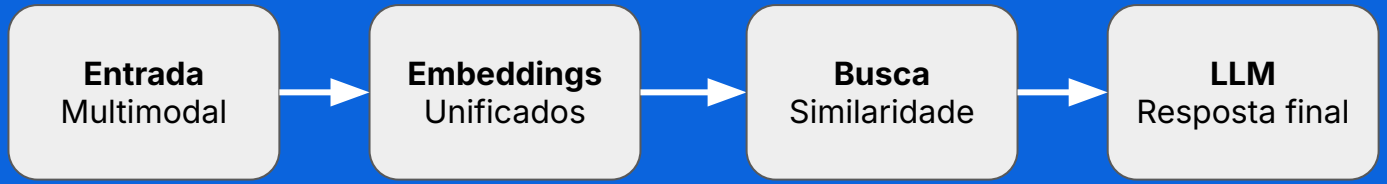
Dados Heterogêneos

- Texto: documentos, notas
- Imagens: fotos, diagramas
- Áudio: gravações, música
- Vídeo: filmagens, streams

RAG Multimodal

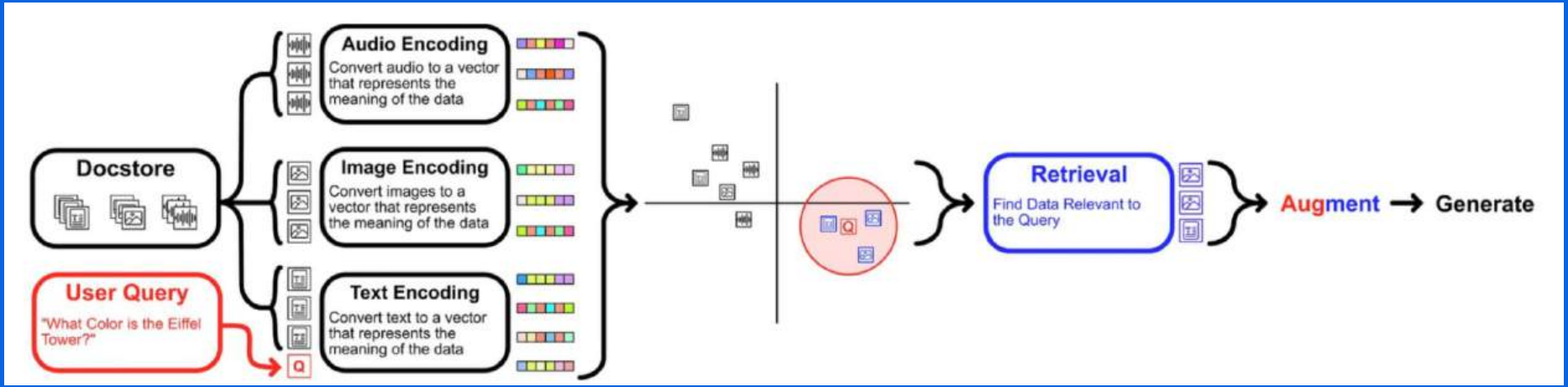
Retrieval-Augmented Generation

Sistema que combina diferentes modalidades de dados em um único espaço vetorial, permitindo buscas cruzadas e geração de respostas contextualizadas.



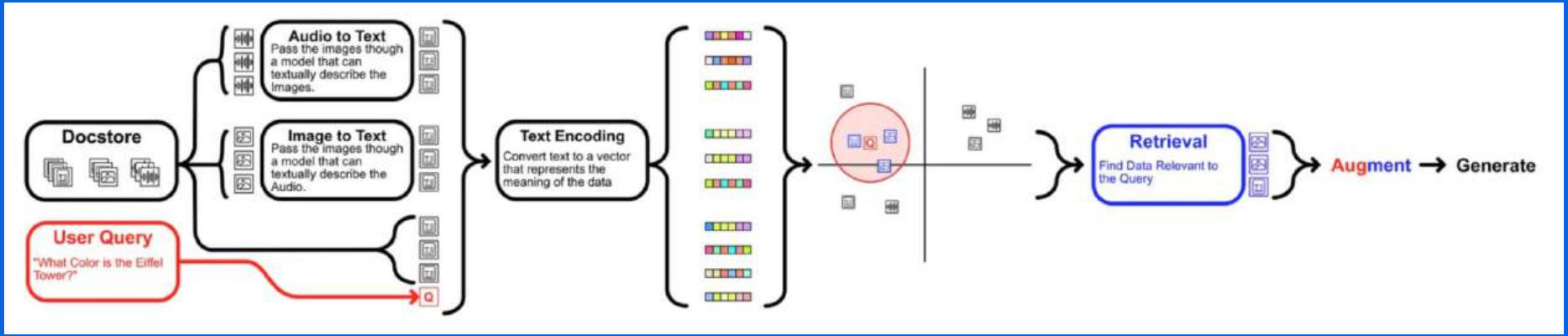
Abordagem 1: Espaço Vetorial Compartilhado

Uma abordagem para RAG multimodal é usar um embedding que funciona com múltiplas modalidades. Basicamente, você passa seus dados por vários codificadores que são projetados para interagir bem entre si, e então recupera os dados mais semelhantes em todas as modalidades à consulta do usuário.



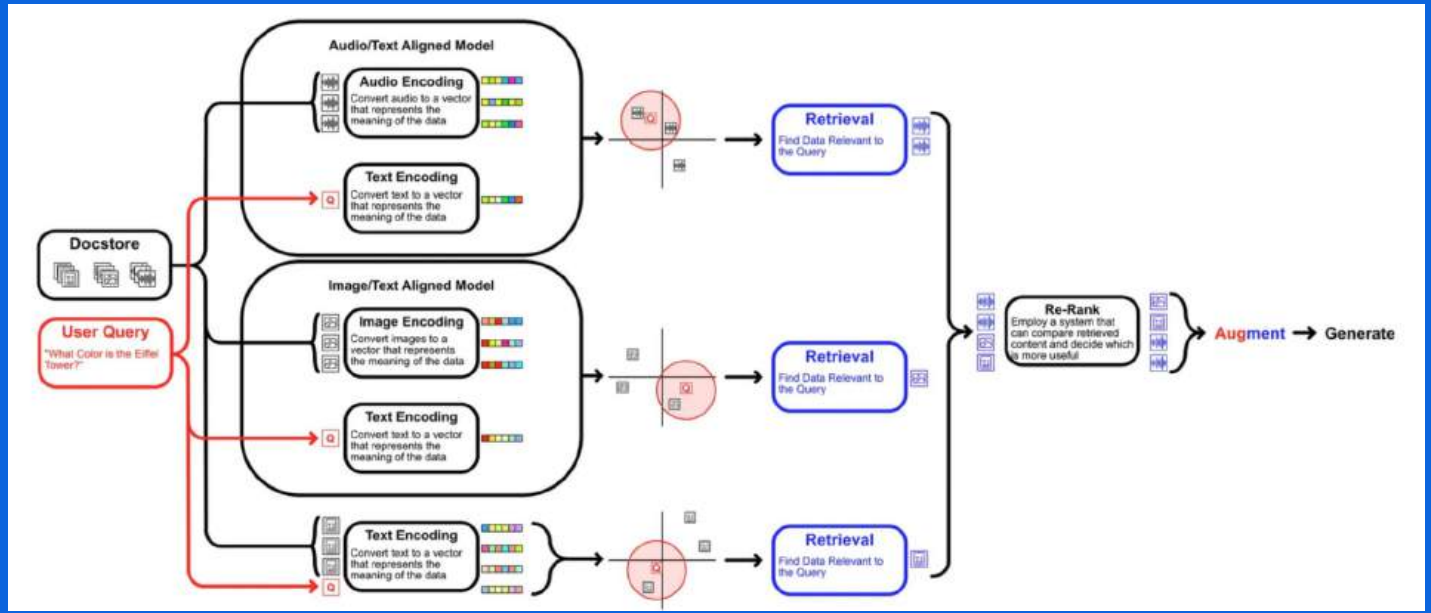
Abordagem 2: Modalidade Única Fundamentada

Outra abordagem para RAG multimodal é converter todas as modalidades de dados em uma única modalidade, geralmente texto. Todas as modalidades são convertidas para uma única modalidade antes de serem passadas para um único codificador.



Abordagem 3: Recuperação Separada

A terceira abordagem é usar uma coleção de modelos projetados para trabalhar com diferentes modalidades. Nesse contexto, você faria a recuperação várias vezes em diferentes modelos, e em seguida, combinaria seus resultados.



Três Abordagens para RAG Multimodal

1. Espaço Vetorial Compartilhado

Todas as modalidades em um único espaço

✓ Busca cruzada direta
✓ Sem perda de contexto
✕ Requer modelo robusto

2. Modalidade Única

Tudo convertido para texto

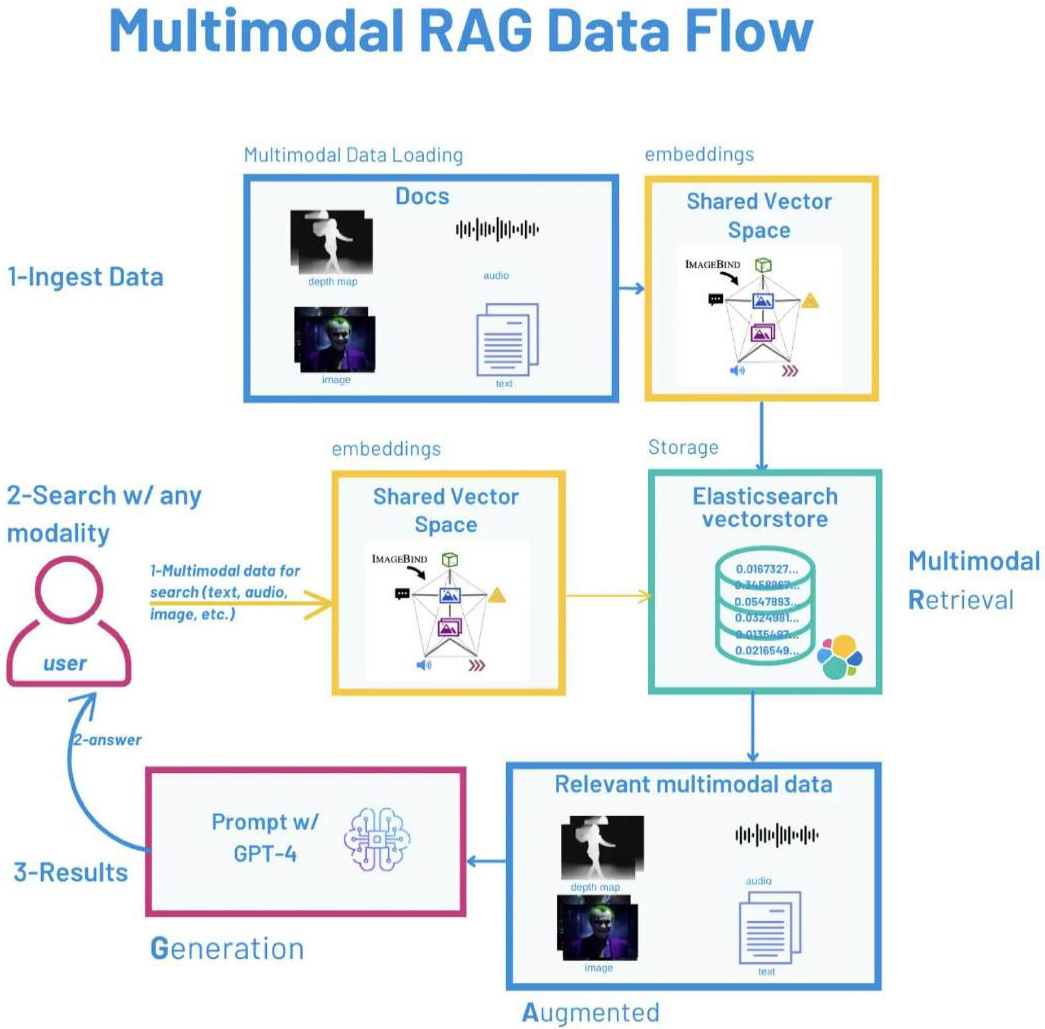
✓ Simples de implementar
✓ Usa infra existente
✕ Perde detalhes visuais/sonoros

3. Recuperação Separada

Modelo específico por modalidade

✓ Otimização customizada
✓ Controle fino
✕ Fusão complexa de resultados

Nossa escolha: Espaço Vetorial Compartilhado com ImageBind



Arquitetura escolhida

Espaço Vetorial Compartilhado

elastic

Stack Tecnológico

🎯 ImageBind

Modelo da Meta AI que gera embeddings unificados para múltiplas modalidades

- • Vetores de 1024 dimensões
- • Suporta 6 modalidades
- • Espaço vetorial compartilhado

🔍 Elasticsearch

Armazena e busca vetores com alta performance

- • Dense vectors
- • k-NN search
- • Escalabilidade

🤖 LLM (GPT-4)

Analisa evidências e gera relatórios conclusivos

Como Funciona o Espaço de Embeddings?

ImageBind mapeia diferentes modalidades em um espaço vetorial comum:

📝 Texto

Sentenças → Vetores 1024D

🌅 Imagens

Pixels → Vetores 1024D

🎵 Áudio

Ondas sonoras → Vetores 1024D

🗺️ Profundidade

Mapas 3D → Vetores 1024D

💡 Resultado: Qualquer modalidade pode ser comparada com qualquer outra usando similaridade de cosseno!

🦇 Demo: O Crime em Gotham City

Evidências Multimodais

📸 **Imagens**: Cena do crime
🎵 **Áudio:** Risada sinistra
📝 **Texto:** Notas enigmáticas
🗺️ **Profundidade:** Mapas 3D

Objetivo:
Usar IA multimodal para identificar o criminoso analisando todas as evidências em conjunto

Os 4 Estágios da Implementação

Estágio 1: Coleta

Organizar evidências em diferentes formatos

Estágio 2: Embeddings

Gerar vetores com ImageBind

Estágio 3: Armazenamento

Indexar no Elasticsearch

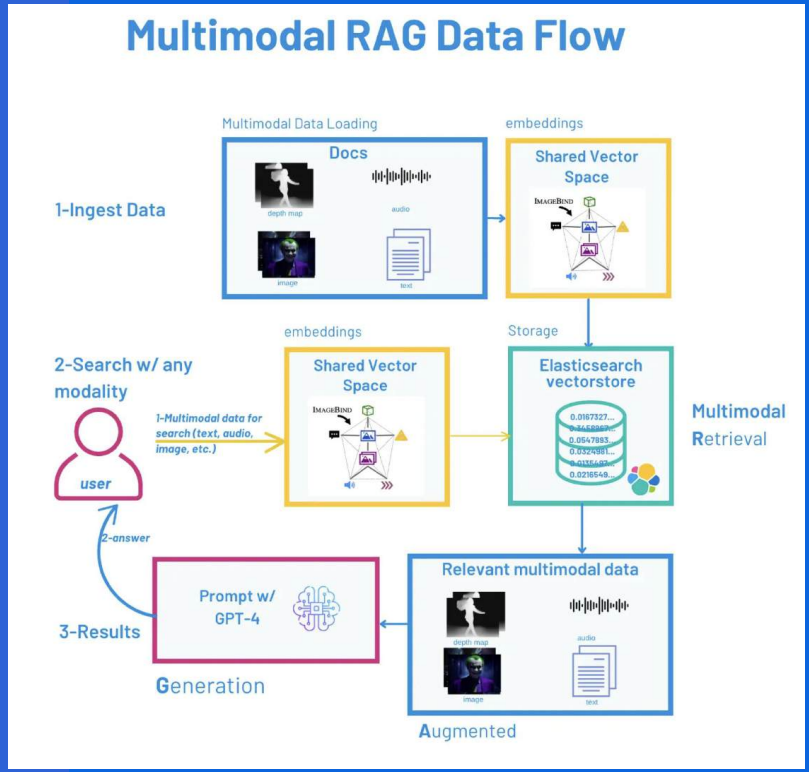
Estágio 4: Análise

LLM conecta os pontos

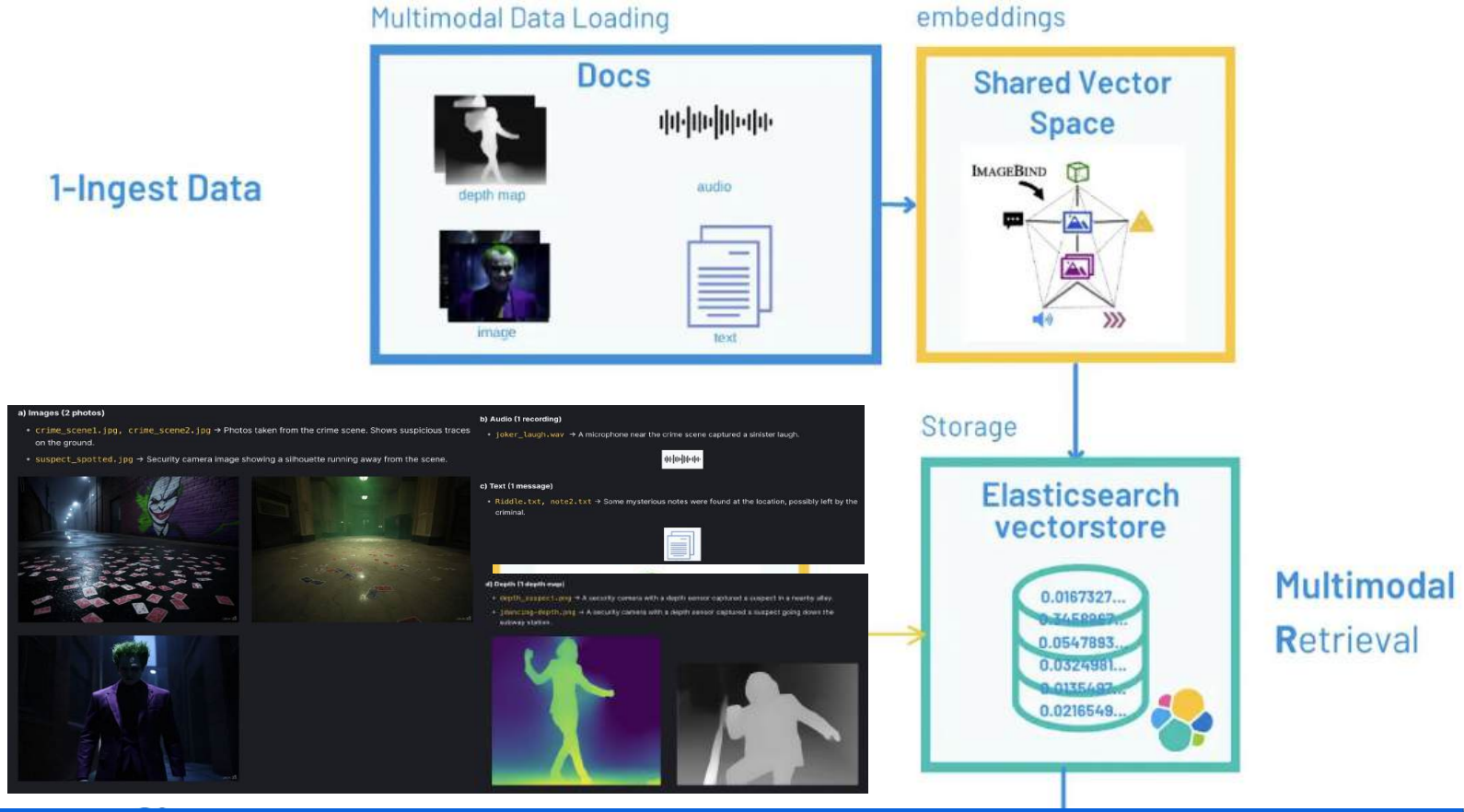
Pipeline de Implementação

pipeline.py

```python
# 1. Gerar embeddings com ImageBind
generator = EmbeddingGenerator()
embedding = generator.generate_embedding(input_data, modality)

# 2. Indexar no Elasticsearch
es_manager.index_content(
    embedding=embedding,
    modality="vision",
    description="Crime scene photo"
)

# 3. Buscar evidências similares
results = es_manager.search_similar(query_embedding, k=5)

# 4. Analisar com LLM
report = llm.analyze_evidence(results)
```



Etapa 1 - Coletando as evidências

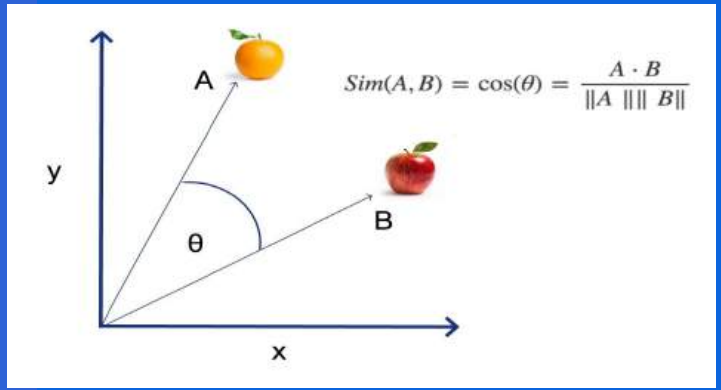


Configuração do Elasticsearch

elasticsearch_mapping.json

```
# Mapeamento para vetores multimodais
{
  "mappings": {
    "properties": {
      "embedding": {
        "type": "dense_vector",
        "dims": 1024,
        "index": true,
        "similarity": "cosine"
      },
      "modality": {
        "type": "keyword"
      },
      "description": {
        "type": "text"
      }
    }
  }
}
```

Similarity: Cosine

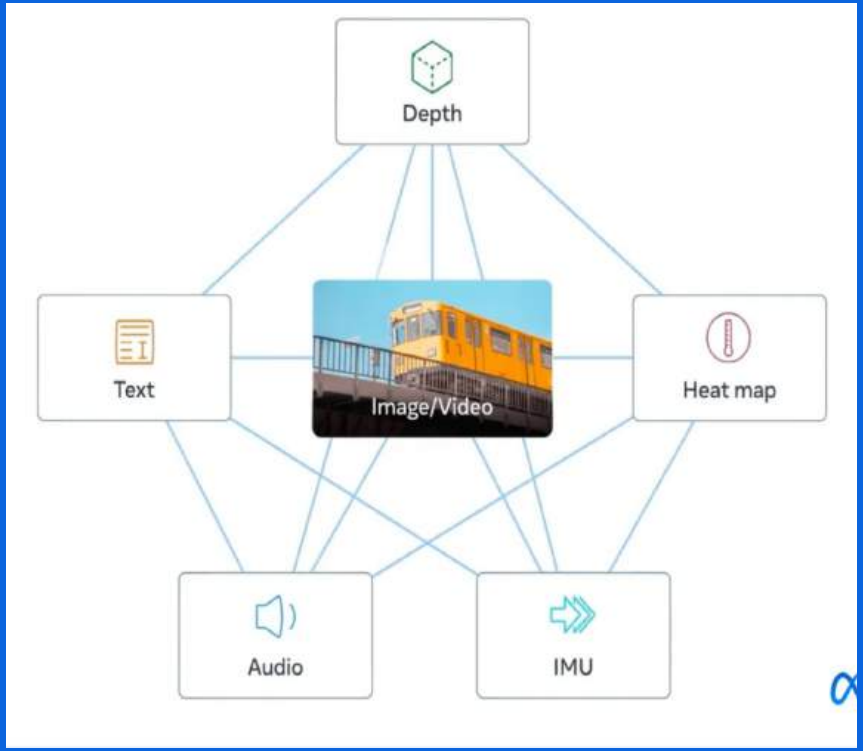


💡 k-NN search permite busca eficiente por similaridade

elastic

Etapa 2 - Organizando as evidências

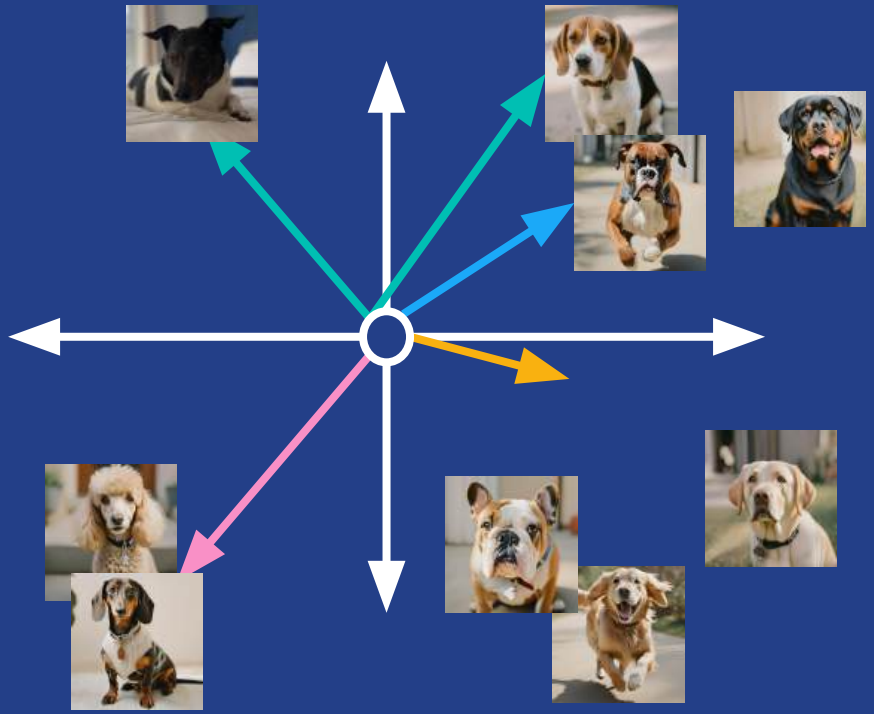
Gerando embeddings com ImageBind

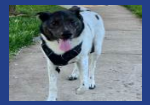
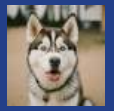



```python
class EmbeddingGenerator:
    """Class for generating multimodal embeddings using ImageBind."""

    def __init__(self):
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
        self.model = self._load_model()

    def _load_model(self):
        """Loads the ImageBind model and sets it to inference mode."""
        model = imagebind_model.imagebind_huge(pretrained=True)
        model.eval()
        model.to(self.device)
        return model

    def generate_embedding(self, input_data, modality):
        """Generates embedding for different modalities"""
        processors = {
            "vision": lambda x: data.load_and_transform_vision_data(x, self.device),
            "audio": lambda x: data.load_and_transform_audio_data(x, self.device),
            "text": lambda x: data.load_and_transform_text(x, self.device),
            "depth": self.process_depth
        }

        try:
            # Input type verification
            if not isinstance(input_data, list):
                raise ValueError(f"Input data must be a list. Received: {type(input_data)}")

            # Convert input data to a tensor format that the model can process
            # For images: [batch_size, channels, height, width]
            # For audio: [batch_size, channels, time]
            # For text: [batch_size, sequence_length]
            inputs = {modality: processors[modality](input_data)}
            with torch.no_grad():
                embedding = self.model(inputs)[modality]
            return embedding.squeeze(0).cpu().numpy()
        except Exception as e:
            logger.error(f"Error generating {modality} embedding: {str(e)}", exc_info=True)
            raise
```

elastic

Vector search ranks objects by similarity (relevance) to the query



Relevance	Result
Query	
1	
2	
3	
4	
5	

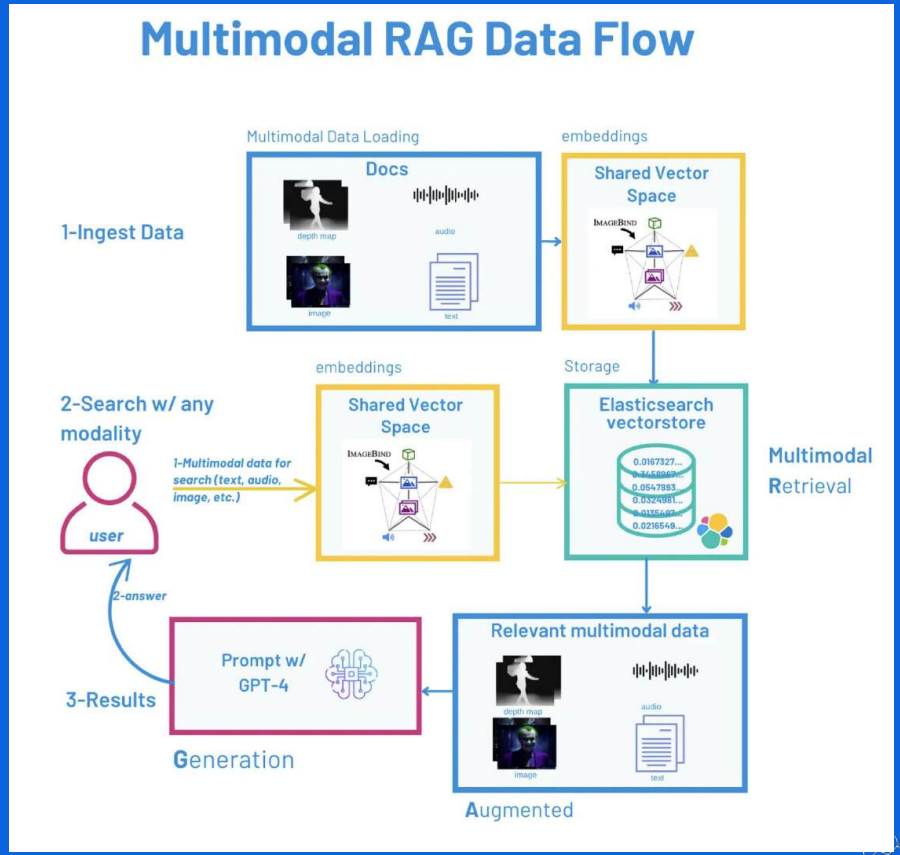
elastic

Etapa 3 - Armazenamento e busca no Elasticsearch

Agora que geramos os embeddings para as evidências, precisamos armazená-los em um banco de dados vetorial para permitir buscas eficientes. Para isso, usaremos o Elasticsearch, que suporta vetores densos (*dense_vector*) e permite buscas por similaridade.

Esta etapa consiste em dois processos principais:

- **Indexação dos embeddings** → Armazena os vetores gerados no Elasticsearch.
- **Busca por similaridade** → Recupera os registros mais similares a uma nova evidência.



Busca Multimodal em Ação

search_multimodal.py

```python
# Buscar por áudio suspeito
audio_embedding = generator.generate_embedding(
    "data/audios/joker_laugh.wav",
    modality="audio"
)

# Encontrar evidências relacionadas
similar_evidences = es_manager.search_similar(
    query_embedding=audio_embedding,
    k=3
)

# Resultado:
# 1. joker_laugh.wav (audio) - 0.9985
# 2. joker_laughing.png (vision) - 0.6068
# 3. "Why so serious?" (text) - 0.5591
```

🎯 Áudio encontra imagem e texto relacionados!

a) Images (2 photos)

- `crime_scene1.jpg, crime_scene2.jpg` → Photos taken from the crime scene. Shows suspicious traces on the ground.
- `suspect_spotted.jpg` → Security camera image showing a silhouette running away from the scene.

b) Audio (1 recording)

- `joker_laugh.wav` → A microphone near the crime scene captured a sinister laugh.

c) Text (1 message)

- `Riddle.txt, note2.txt` → Some mysterious notes were found at the location, possibly left by the criminal.

Etapa 4 - Conectando os pontos com o LLM

Expected LLM output

```
**Prime Suspect:** The Joker

**Evidence Supporting Conclusion:**

- **Visual Evidence:**
  - The photo of the crime scene with playing cards scattered around and the graffiti of th
  - The image of the Joker with green hair, white face paint, and a sinister smile in an uri

- **Auditory Evidence:**
  - The captured sinister laugh with a similarity score of 1.00 perfectly matches known aud
  - Despite the lower similarity score of 0.61, the second audio piece further corroborates

- **Textual Evidence:**
  - The mysterious note found at the location, with a similarity score of 0.76, likely cont
  - The similarity score of 0.72 for the Joker's description in textual evidence reinforces

- **Depth Evidence:**
  - Depth sensor capture of the suspect with a similarity score of 0.77 suggests a physical
  - The lower similarity score of 0.53 in the second depth evidence still contributes to th

**Behavioral Patterns:**
- The Joker is known for his theatrical crimes, often leaving behind a signature trail of c
- His motives often include creating chaos, drawing attention to his acts, and challenging

**Confidence Level:** 95%

**Next Steps:** No further evidence required.

The combination of visual, auditory, textual, and depth evidence strongly points to the Jok
```

Aplicações no Mundo Real

🏥 Saúde

Análise de exames médicos combinando imagens, laudos e áudio de consultas

🏪 Varejo

Busca de produtos por foto, descrição ou comando de voz

🎓 Educação

Materiais didáticos multimodais com busca inteligente

🏭 Indústria

Manutenção preditiva com sensores, imagens e logs

Próximos Passos

🚀 Comece Hoje!

- Clone o repositório no GitHub
- Configure Elasticsearch (Cloud ou local)
- Instale as dependências Python
- Execute o notebook no Google Colab
- Experimente com seus próprios dados!

terminal

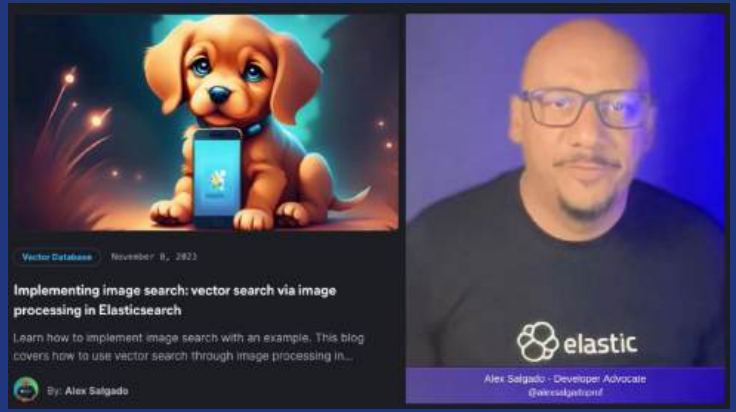
```
git clone https://github.com/elastic/elasticsearch-labs
cd supporting-blog-content/building-multimodal-rag-with-elasticsearch-gotham/
pip install -r requirements.txt
```

elastic

Conclusão

- **Multimodalidade = contexto mais rico**: Vai além do texto para integrar áudio, imagens e outros dados
- **Elasticsearch + ImageBind + LLMs(Bedrock)**: Pipeline escalável para busca e análise inteligente
- **Aplicações potenciais**: Desde segurança até recomendação de conteúdo

Recursos para desenvolvedores: Elasticsearch Labs



youtube.com/**@OfficialElasticCommunity**

elastic.co/**search-labs**

github.com/elastic/**elasticsearch-labs**

elastic

elastic

Gratidão

Alex Salgado
Developer Advocate @ Elastic

@alexsalgadoprof
@alexsalgadoprof
salgado
/in/alex-salgado/