Inteligência Artificial 2

Aula 12: Ingestão de Dados (O Pré-RAG)

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Inteligência Artificial 2

Sumário

1. O Desafio do Text-Mining em IA
2. A Engenharia do Chunking (Fatiamento)
3. A Fronteira do Sentido: Overlap
4. Desmistificando a Extração em PDF
5. Laboratório Prático e Integração

Onde Paramos?

Na aula anterior, dominamos a **matemática dos Embeddings** e os bancos vetoriais:

- Representação densa: palavras como vetores em $\mathbb{R}^d$.
- Similaridade de Cosseno e busca por vizinhos mais próximos.
- ChromaDB: armazenamento e consulta de vetores semânticos.

A Pergunta de Hoje

O banco vetorial está pronto, mas **de onde vêm** os textos?
Como transformar PDFs selvagens em chunks vetorizáveis?

Objetivos da Sessão

Ao final desta aula, você será capaz de:

1. **Extrair** texto de PDFs e documentos corporativos usando bibliotecas especializadas.
2. **Fragmentar** documentos longos em chunks com overlap controlado.
3. **Explicar** por que enviar um PDF inteiro para o LLM é uma má ideia (custo + diluição semântica).
4. **Implementar** um pipeline ETL completo: Extract $\rightarrow$ Transform $\rightarrow$ Load.
5. **Popular** um ChromaDB com chunks vetorizados prontos para RAG.

Conexão

Vocês já viram ETL em Engenharia de Dados — aqui ele ganha uma dimensão semântica.

O Abismo da Vida Real

- Até agora assumimos que possuíamos os textos em formato de *Strings* bonitinhas carregadas na memória RAM do Python.

O Abismo da Vida Real

- Até agora assumimos que possuíamos os textos em formato de *Strings* bonitinhas carregadas na memória RAM do Python.
- Mas a **Inteligência Corporativa** está trancada em formatos selvagens: relatórios financeiros em PDF de 500 páginas, manuais de RH no Word, planilhas monstruosas e caixas de e-mail.

O Abismo da Vida Real

- Até agora assumimos que possuíamos os textos em formato de *Strings* bonitinhas carregadas na memória RAM do Python.
- Mas a **Inteligência Corporativa** está trancada em formatos selvagens: relatórios financeiros em PDF de 500 páginas, manuais de RH no Word, planilhas monstruosas e caixas de e-mail.
- Como conectamos o universo não-estruturado dos escritórios ao cérebro dos Modelos de Linguagem? Através de um pipeline de ETL (*Extract*, *Transform*, *Load*) focado em **pré-processamento semântico**.

O Perigo do Documento Inteiro

Por que não calculamos o vetor (*Embedding*) de um PDF inteiro de 200 páginas e salvamos no banco? Duas restrições fundamentais:

O Perigo do Documento Inteiro

Por que não calculamos o vetor (*Embedding*) de um PDF inteiro de 200 páginas e salvamos no banco? Duas restrições fundamentais:

1. **Janela de Contexto (Tokens):** Quando o RAG encontrar o documento mais próximo, ele enviará o documento inteiro para o ChatGPT ler. A API negará a requisição por estourar o limite de tokens da conversa, ou cobrará dezenas de dólares por uma única busca.

O Perigo do Documento Inteiro

Por que não calculamos o vetor (*Embedding*) de um PDF inteiro de 200 páginas e salvamos no banco? Duas restrições fundamentais:

1. **Janela de Contexto (Tokens):** Quando o RAG encontrar o documento mais próximo, ele enviará o documento inteiro para o ChatGPT ler. A API negará a requisição por estourar o limite de tokens da conversa, ou cobrará dezenas de dólares por uma única busca.
2. **A Perda de Resolução Semântica:** Um vetor de 1536 dimensões tem um limite de complexidade. Se o livro trata de culinária no capítulo 1, assassinato no capítulo 2 e direito civil no capítulo 3, o vetor resultante será uma "média de cores" cinzenta e sem sentido. **A direção do vetor deixará de apontar para fatos específicos.**

O Fatiamento Obrigatório

- A etapa primária para construir um bom RAG chama-se **Chunking** (Fatiamento).

O Fatiamento Obrigatório

- A etapa primária para construir um bom RAG chama-se **Chunking** (Fatiamento).
- Nós abrimos o PDF gigante e o cortamos em pedaços (*chunks*) menores. Cada pedaço torna-se uma entidade isolada no Vector Database.

O Fatiamento Obrigatório

- A etapa primária para construir um bom RAG chama-se **Chunking** (Fatiamento).
- Nós abrimos o PDF gigante e o cortamos em pedaços (*chunks*) menores. Cada pedaço torna-se uma entidade isolada no Vector Database.
- Quando a IA for ler para o usuário, ela não lerá o "Livro inteiro", mas lerá os "3 Chunks" do livro que mais se parecem com a pergunta.

O Fatiamento Obrigatório

- A etapa primária para construir um bom RAG chama-se **Chunking** (Fatiamento).
- Nós abrimos o PDF gigante e o cortamos em pedaços (*chunks*) menores. Cada pedaço torna-se uma entidade isolada no Vector Database.
- Quando a IA for ler para o usuário, ela não lerá o "Livro inteiro", mas lerá os "3 Chunks" do livro que mais se parecem com a pergunta.
- Mas o **Como Cortar** dita o fracasso ou sucesso do seu produto. Um mau chunking destrói as capacidades analíticas do RAG mais caro do mundo.

Estratégia 1: Fatiamento de Tamanho Fixo

- A técnica mais rudimentar é o *Fixed-size Character Chunking*.
- O programador decide: "A cada 1000 caracteres lidos, feche a string e comece um novo vetor".

Estratégia 1: Fatiamento de Tamanho Fixo

- A técnica mais rudimentar é o *Fixed-size Character Chunking*.
- O programador decide: "A cada 1000 caracteres lidos, feche a string e comece um novo vetor".
- **Desvantagem Óbvia:** O corte é cego à sintaxe. O caractere de número 1000 pode cair no meio de uma palavra, ou pior, separar o sujeito do verbo.
- "O paciente sofre de infarto agudo do [FIM DO CHUNK 1]"
  "[INÍCIO DO CHUNK 2] miocárdio grave e requer..."
- O Vector DB não saberá responder perguntas clínicas precisas porque nenhum dos dois vetores contém o diagnóstico semântico completo.

Estratégia 2: Fatiamento Sintático e Recursivo

- **NLTK / spaCy Splitters:** Utiliza bibliotecas linguísticas para rastrear pontos finais, pontos de interrogação e quebras gramaticais, e só fatia o texto quando uma sentença finaliza.

Estratégia 2: Fatiamento Sintático e Recursivo

- **NLTK / spaCy Splitters:** Utiliza bibliotecas linguísticas para rastrear pontos finais, pontos de interrogação e quebras gramaticais, e só fatia o texto quando uma sentença finaliza.
- **Recursive Character Splitter:** O padrão Ouro da indústria (usado no LangChain). Ele tenta fatiar o texto quebrando por duplo-enter (parágrafos: `\n\n`).

Estratégia 2: Fatiamento Sintático e Recursivo

- **NLTK / spaCy Splitters:** Utiliza bibliotecas linguísticas para rastrear pontos finais, pontos de interrogação e quebras gramaticais, e só fatia o texto quando uma sentença finaliza.
- **Recursive Character Splitter:** O padrão Ouro da indústria (usado no LangChain). Ele tenta fatiar o texto quebrando por duplo-enter (parágrafos: `\n\n`).
- Se o pedaço ainda for muito grande (excedeu 1000 letras), ele entra em recursão e tenta quebrar por ponto final (`.`), depois por vírgula (`,`) e por último por espaço ( ), preservando ao máximo as caixas lógicas de sentido original.

Estratégia 3: Semantic Chunking (Vanguarda)

- Uma evolução recente é o fatiamento guiado por Inteligência. O *Semantic Chunking* abandona os caracteres fixos.

Estratégia 3: Semantic Chunking (Vanguarda)

- Uma evolução recente é o fatiamento guiado por Inteligência. O *Semantic Chunking* abandona os caracteres fixos.
- Ele divide o texto pelas sentenças e calcula o *Embedding* de cada sentença separadamente.

Estratégia 3: Semantic Chunking (Vanguarda)

- Uma evolução recente é o fatiamento guiado por Inteligência. O *Semantic Chunking* abandona os caracteres fixos.
- Ele divide o texto pelas sentenças e calcula o *Embedding* de cada sentença separadamente.
- Em seguida, ele mede a similaridade do cosseno sequencial (da Frase 1 com a Frase 2, da 2 com a 3).

Estratégia 3: Semantic Chunking (Vanguarda)

- Uma evolução recente é o fatiamento guiado por Inteligência. O *Semantic Chunking* abandona os caracteres fixos.
- Ele divide o texto pelas sentenças e calcula o *Embedding* de cada sentença separadamente.
- Em seguida, ele mede a similaridade do cosseno sequencial (da Frase 1 com a Frase 2, da 2 com a 3).
- Quando a Similaridade despenca abruptamente (um *valley* no gráfico), o algoritmo entende que **O Assunto Mudou**. Ali ele passa a tesoura e encerra o chunk, gerando vetores altamente puros em sentido.

Estratégia 4: Agentic/LLM Chunking

- Outra fronteira de arquitetura é usar Modelos Menores (ex: Llama-3-8B) para coordenar a extração **antes** de vetorizar.

Estratégia 4: Agentic/LLM Chunking

- Outra fronteira de arquitetura é usar Modelos Menores (ex: Llama-3-8B) para coordenar a extração **antes** de vetorizar.
- O LLM lê a página crua e extrai *Insights* ou "Proposições" fechadas (ex: "A empresa faturou X em 2023").

Estratégia 4: Agentic/LLM Chunking

- Outra fronteira de arquitetura é usar Modelos Menores (ex: Llama-3-8B) para coordenar a extração **antes** de vetorizar.
- O LLM lê a página crua e extrai *Insights* ou "Proposições" fechadas (ex: "A empresa faturou X em 2023").
- Em vez de salvar blocos de texto confuso no ChromaDB, você salva pequenas declarações perfeitamente afirmativas. A precisão do RAG no ambiente produtivo dispara para níveis quase humanos, embora seja incrivelmente custoso montar esse pipeline (*Compute-Intensive*).

Trade-offs no Tamanho do Chunk

O tamanho máximo do *Chunk* dita como seu modelo operará:

Trade-offs no Tamanho do Chunk

O tamanho máximo do *Chunk* dita como seu modelo operará:

- **Chunks Pequenos (ex: 200 tokens)**: Geram vetores extremamente exatos. Acham fatos muito precisos rapidamente. Mas entregam pouco contexto ao LLM.

Trade-offs no Tamanho do Chunk

O tamanho máximo do *Chunk* dita como seu modelo operará:

- **Chunks Pequenos (ex: 200 tokens)**: Geram vetores extremamente exatos. Acham fatos muito precisos rapidamente. Mas entregam pouco contexto ao LLM.
- **Chunks Grandes (ex: 1500 tokens)**: Geram vetores mais diluídos (pode não achar a agulha no palheiro na busca via cosseno), mas se for encontrado, fornece ao LLM uma página inteira de raciocínio profundo.

O Perigo do Corte Seco

- Mesmo com cortadores recursivos espertos, às vezes uma premissa teórica se estende por dois parágrafos longos sucessivos.

O Perigo do Corte Seco

- Mesmo com cortadores recursivos espertos, às vezes uma premissa teórica se estende por dois parágrafos longos sucessivos.
- Se o limite numérico for atingido, a divisão separará a premissa 1 da conclusão 2 em dois vetores independentes que nunca conversarão entre si de forma isolada.

O Perigo do Corte Seco

- Mesmo com cortadores recursivos espertos, às vezes uma premissa teórica se estende por dois parágrafos longos sucessivos.
- Se o limite numérico for atingido, a divisão separará a premissa 1 da conclusão 2 em dois vetores independentes que nunca conversarão entre si de forma isolada.
- A genialidade adotada pela comunidade de IA para estabilizar isso é o **Chunk Overlap** (Sobreposição Fatiada).

Chunk Overlap: Entendendo a Ponte Semântica

- Com o Overlap, nós configuramos (ex: 20%).
- Quando um chunk atinge 1000 caracteres e fecha, o próximo chunk não inicia no caractere 1001. Ele **retrocede** e inicia a captura a partir do caractere 800.

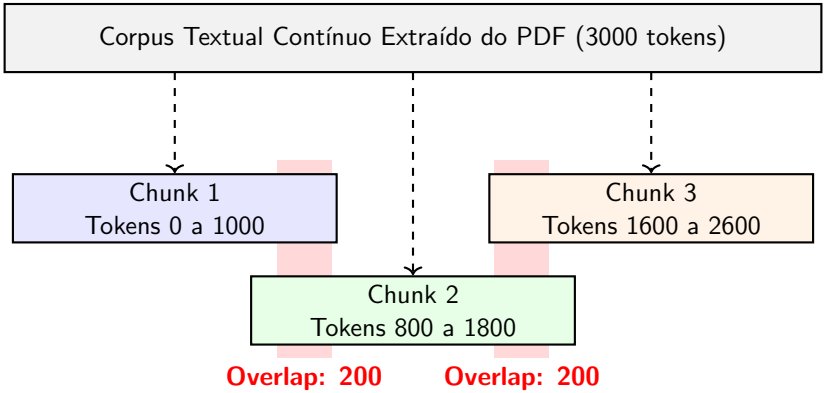
Chunk Overlap: Entendendo a Ponte Semântica

- Com o Overlap, nós configuramos (ex: 20%).
- Quando um chunk atinge 1000 caracteres e fecha, o próximo chunk não inicia no caractere 1001. Ele **retrocede** e inicia a captura a partir do caractere 800.
- **Resultado:** Os últimos 200 caracteres do bloco anterior são artificialmente injetados no início do bloco seguinte.

Chunk Overlap: Entendendo a Ponte Semântica

- Com o Overlap, nós configuramos (ex: 20%).
- Quando um chunk atinge 1000 caracteres e fecha, o próximo chunk não inicia no caractere 1001. Ele **retrocede** e inicia a captura a partir do caractere 800.
- **Resultado:** Os últimos 200 caracteres do bloco anterior são artificialmente injetados no início do bloco seguinte.
- Isso garante que haja sempre uma "memória transiente" ligando blocos próximos e garantindo que o contexto vital seja herdado pelo vetor subsequente.

Diagrama Avançado: Chunking e Overlapping



Implementação Lógica de um Chunk com Overlap

- Na ausência de frameworks gigantescos (Langchain/Llamaindex), a rotina de Overlap em Python é um simples deslocamento de índices de *String*.

Implementação Lógica de um Chunk com Overlap

- Na ausência de frameworks gigantescos (Langchain/Llamaindex), a rotina de Overlap em Python é um simples deslocamento de índices de *String*.

```python
def chunk_text_overlap(text, chunk_size=1000, overlap=200):
    chunks = []
    start = 0

    while start < len(text):
        # Captura o limite de 1000 letras a partir de Start
        end = min(start + chunk_size, len(text))
        chunks.append(text[start:end])

        # A magia do overlap: O proximo bloco comeca recuado!
        # Avancamos apenas o tamanho LIMIDO do overlap.
        start += (chunk_size - overlap)

    return chunks
```

A Natureza Enganosa dos PDFs

- Formatos web (`.html`) ou dados em disco (`.json`) guardam texto legível codificado em UTF-8 ou ASCII.
- O **PDF** (*Portable Document Format*), por design na década de 90 (Adobe), é uma coleção de instruções PostScript.

A Natureza Enganosa dos PDFs

- Formatos web (`.html`) ou dados em disco (`.json`) guardam texto legível codificado em UTF-8 ou ASCII.
- O **PDF** (*Portable Document Format*), por design na década de 90 (Adobe), é uma coleção de instruções PostScript.
- Ele não entende "A palavra Computador está no parágrafo 1". O arquivo interno do PDF diz: "Desenhe a letra C nas coordenadas X=45 Y=900; Desenhe a letra o na X=51...".

Os Riscos ao Ler PDFs em IA

- **Textos em Múltiplas Colunas:** O software extrator pode ler horizontalmente cruzando as duas colunas, criando frases no estilo Frankenstein.
- **Tabelas sem Grades:** O PDF desenha números espalhados, e a IA lê uma nuvem de matriz de números aleatórios e misturados.
- **Marcas d'água e Cabeçalhos:** Repetem-se em todas as páginas, intoxicando a base do Vector DB.

O Ecossistema Python

Usamos bibliotecas especialistas para remediar isso:

- `PyPDF2 / pypdf`: Básica, rápida, texto puro estruturado.
- `pdfplumber`: Extrai geometria. Capaz de reconstruir visualmente e raspar tabelas com perfeição.
- `pytesseract`: Quando o PDF não é texto nativo, mas sim o scan de uma imagem de um scanner antigo (Requer OCR óptico).

Lidando com Tabelas em PDFs

- Tabelas são o cemitério dos pipelines de RAG comuns. O extrator tradicional (`pypdf`) junta as células lado a lado, e a IA lê os saldos bancários misturados com as datas.

Lidando com Tabelas em PDFs

- Tabelas são o cemitério dos pipelines de RAG comuns. O extrator tradicional (`pypdf`) junta as células lado a lado, e a IA lê os saldos bancários misturados com as datas.
- Para resolver isso, utilizamos algoritmos como `Camelot` ou modelos de Visão (*Vision-Language Models*).

Lidando com Tabelas em PDFs

- Tabelas são o cemitério dos pipelines de RAG comuns. O extrator tradicional (`pypdf`) junta as células lado a lado, e a IA lê os saldos bancários misturados com as datas.
- Para resolver isso, utilizamos algoritmos como `Camelot` ou modelos de Visão (*Vision-Language Models*).
- A melhor prática atual: O software extrai a tabela, converte para sintaxe **Markdown** (ou **HTML**), e o Vector DB salva o Markdown cru. Quando o ChatGPT (LLM) lê o Markdown pelo RAG, ele reconstrói perfeitamente o raciocínio matemático tabular na sua mente.

O Poder Supremo dos Metadados

- O Chunk nunca viaja sozinho. Se fatiarmos 10.000 livros para o Vector DB, e um usuário perguntar sobre biologia, as fatias de romances sci-fi podem causar ruído no vetor (falsos positivos).

O Poder Supremo dos Metadados

- O Chunk nunca viaja sozinho. Se fatiarmos 10.000 livros para o Vector DB, e um usuário perguntar sobre biologia, as fatias de romances sci-fi podem causar ruído no vetor (falsos positivos).
- Todo Chunk **obrigatóriamente** recebe um dicionário de Metadados (*Tags*):
  - `source: "relatorio2023.pdf"`
  - `page: 45`
  - `author: "Setor Financeiro"`
  - `category: "Balanço"`

O Poder Supremo dos Metadados

- O Chunk nunca viaja sozinho. Se fatiarmos 10.000 livros para o Vector DB, e um usuário perguntar sobre biologia, as fatias de romances sci-fi podem causar ruído no vetor (falsos positivos).
- Todo Chunk **obrigatóriamente** recebe um dicionário de Metadados (*Tags*):
  - `source: "relatorio2023.pdf"`
  - `page: 45`
  - `author: "Setor Financeiro"`
  - `category: "Balanço"`
- O VectorDB primeiro aplica um filtro estrito WHERE category='Balanço', e **só depois** calcula o Cosseno sobre a coleção filtrada.

Código: Lendo PDFs e Preparando o Banco

```python
from pypdf import PdfReader

# Abrindo o PDF
reader = PdfReader("relatorio_financeiro.pdf")
texto_completo = ""

# Loop sobre as paginas extraindo strings limpas
for page in reader.pages:
    pagina_texto = page.extract_text()
    if pagina_texto:
        texto_completo += pagina_texto + "\n"

# Após gerar a grande String, passamos pela nossa funcao de Chunking
chunks_finais = chunk_text_overlap(texto_completo, chunk_size=800,
    overlap=100)

print(f"O documento rendeu {len(chunks_finais)} documentos para o
    banco vetorial!")
```

RecursiveCharacterTextSplitter: A Ordem Importa

- O segredo do fatiamento recursivo está na lista de **separadores**.
- Por padrão: `["\n\n", "\n", ". ", " ", ""]`
- **Como funciona?**
  1. Tenta quebrar por parágrafo (`\n\n`).
  2. Se um parágrafo sozinho for maior que o Chunk Size, ele tenta quebrar por linha (`\n`).
  3. Se uma linha for muito longa, tenta quebrar por sentenças (`. `).
  4. Em último caso, quebra por palavras.
- Isso preserva a máxima integridade lógica em 99% dos casos corporativos.

Tokenizers: Palavras vs. Subwords

- O Chunk Size raramente é medido em caracteres na vida real, mas sim em **Tokens**.
- Modelos como GPT-4 usam **BPE (Byte-Pair Encoding)**.
- "Inconstitucionalissimamente" não é 1 token, mas sim fragmentos: "In", "constitucional", "issima", "mente".
- **Regra de ouro:** 1 token $\approx$ 4 letras (em inglês). Em português, idiomas não-nativos, o modelo consome **mais tokens** para representar a mesma palavra.

Multimodalidade: Vendo o Documento

- Bibliotecas puramente textuais (pypdf) ignoram gráficos de barra e pizza.
- O Estado da Arte atual usa **Vision-Language Models (VLMs)**.
- Soluções como `LlamaParse` ou `GPT-4o` transformam a página do PDF em **imagem** e pedem: "Descreva toda a informação financeira deste gráfico em Markdown".
- O Markdown gerado é então enviado ao pipeline de Chunking!

O Custo Oculto da Ingestão

- Extrair texto de PDFs escaneados via OCR (Tesseract) é **Lento**.
- Passar imagens por VLMs é **Caro** (chamadas de API).
- Fatiar milhões de chunks e pedir Embeddings para a OpenAI custa dólares a cada atualização da base.

Solução Híbrida

Empresas usam `pypdf` (grátis/rápido) para as primeiras 10.000 páginas nativas, e só roteiam para OCR/Vision os documentos que disparam erro de leitura (ex: "texto vazio" ou "caracteres quebrados").

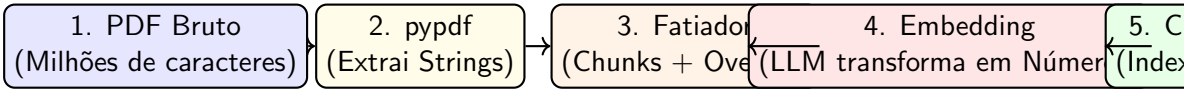
Metadados Essenciais no Mundo Real

- Para além do nome do arquivo e da página, a **segurança** e **governança** exigem:
  - `access_level: "CONFIDENTIAL"` (Garante que um estagiário não faça uma busca que retorne salários da diretoria).
  - `created_at: 1690000000` (Garante que o RAG possa filtrar "Apenas relatórios do último mês").
  - `chunk_index: 42` (Para reconstruir a ordem se precisarmos de um contexto maior).

Recapitulação — O que Levar Desta Aula

1. **Text-Mining**: O mundo corporativo vive em PDFs e não estruturados.
2. **Chunking**: Necessário para contornar limite de tokens e manter precisão semântica.
3. **Chunk Overlap**: A ponte que preserva o contexto (10% a 20%).
4. **Recursive Splitter**: A tática de fatiar graciosamente por parágrafos e frases.
5. **Limitações do PDF**: OCR e Vision são o futuro para lidar com tabelas e gráficos.
6. **Metadados**: A chave para a segurança e filtragem correta no Vector Store.

O Pipeline Final de ETL (Extract, Transform, Load)



Com este motor montado, o servidor está eternamente pronto para responder dúvidas e cruzar inteligência contra essa base!

Orquestração: O Papel Oculto

- Na engenharia de IA, nós quase nunca escrevemos esse loop de extração crua e *overlaps* do zero.

Orquestração: O Papel Oculto

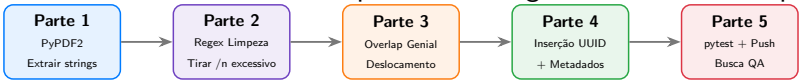
- Na engenharia de IA, nós quase nunca escrevemos esse loop de extração crua e *overlaps* do zero.
- Frameworks como **LangChain** ou **LlamaIndex** nos oferecem classes polidas: `PyPDFDirectoryLoader` ou `RecursiveCharacterTextSplitter`.

Orquestração: O Papel Oculto

- Na engenharia de IA, nós quase nunca escrevemos esse loop de extração crua e *overlaps* do zero.
- Frameworks como **LangChain** ou **LlamaIndex** nos oferecem classes polidas: `PyPDFDirectoryLoader` ou `RecursiveCharacterTextSplitter`.
- Ao usar frameworks, o que levasse 500 linhas de código limpo com regex torna-se um cano abstrato (*pipeline*) de 10 linhas. Eles padronizam a injeção corporativa no banco vetorial.

Roteiro do Laboratório 12

Missão: Extrair dados de um PDF não-estruturado, aplicar chunking manual com overlap, e popular o ChromaDB.



Construindo Ferramental

Vocês implementarão o `chunk_text_overlap` do zero. Sem LangChain. Precisam sentir a matemática do deslocamento de strings!

Lab Passo 1 e 2: Lendo o PDF

- Usaremos o `PyPDF2` ou `pypdf` para abrir o arquivo da atividade.
- O desafio inicial é aglutinar todas as páginas em uma *String* gigante limpa.

```
from pypdf import PdfReader
reader = PdfReader("dados/relatorio.pdf")

texto_cru = ""
for page in reader.pages:
    texto_cru += page.extract_text() + "\n"
```

Lab Passo 3: Implementar o Chunking Overlap

- Este é o coração do lab. Completar a função que avança pelo texto mas retrocede o tamanho do *Overlap*.

```
# Exemplo conceitual para chunks de 100 com overlap de 20
start = 0
while start < len(texto_cru):
    fim = start + 100
    chunk = texto_cru[start:fim]
    lista_de_chunks.append(chunk)

    # Ao inves de avancar 100, avanca apenas 80!
    start += (100 - 20)
```

Lab Passo 4: Povoamento com Metadados

- Subir cada chunk para a Collection do ChromaDB.
- O detalhe que fará o pytest passar: **ID único e página/índice no metadado**.

```
import uuid
# Para cada chunk gerado...
meu_id = str(uuid.uuid4())
collection.add(
    documents=[chunk_atual],
    metadatas=[{"fonte": "relatorio.pdf", "chunk_num": idx}],
    ids=[meu_id]
)
```

Lab Passo 5: Teste Final (Rumo ao RAG)

- Quando vocês derem o *Push*, o pipeline do GitLab fará perguntas ultra específicas cujo contexto foi fragmentado.
- Se o seu *overlap* estiver errado, a informação será dividida ao meio, e a busca via cosseno vai ignorar a metade crucial. O teste falha.
- Se o overlap salvou o contexto, a resposta virá intacta e vocês ganham o Selo Verde!

Dúvidas?

Email: alessio@cefetmg.br
Aula: Segunda e Quarta-Feira, 7h as 8:40h (Sala 39)
WhatsApp: Dúvidas rápidas

Lembrete

Confira sua Issue no GitLab!

Próxima aula: Arquitetura Transformer

100% da Aula 12 Concluída!