Inteligência Artificial 2

Aula 26: Hackathon do Projeto Final (Arquitetura e Kickoff)

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Inteligência Artificial 2

Sumário

Onde Paramos?

Nas aulas anteriores, vocês dominaram **todas as peças** do sistema de IA:

- **Módulo I:** Perceptron → MLP → Backprop → CNNs → RNNs.
- **Módulo II:** Embeddings → Transformers → Hugging Face.
- **Módulo III:** APIs → RAG → Spring AI → Full-Stack.
- **Módulo IV:** Fine-Tuning → Agentes → Segurança.

A Virada Final

Agora é hora de **construir**.
5 dias. 1 produto. 1 Demoday.

De Scripts para Produtos Escaláveis

Durante a disciplina, a zona de conforto foi o *Jupyter Notebook* ou pequenos *scripts* no terminal.

- **A Falácia do Protótipo:** Chamar a API da OpenAI e imprimir no console leva 5 linhas de código. Isso cria a falsa sensação de que "Construir IA é fácil".
- **A Realidade da Engenharia:** Colocar IA em produção exige latência controlada, tolerância a falhas, *streaming* assíncrono para o cliente (SSE), gestão de estado (histórico do usuário) e persistência em bancos de dados relacionais e vetoriais simultaneamente.
- **Sistemas Híbridos:** Produtos reais unem sistemas determinísticos (Regras de negócio, Bancos SQL, Permissões) com sistemas probabilísticos (LLMs). O segredo está na fronteira entre eles.

Microserviços vs Monolitos na Era da IA

O *design* arquitetural de um produto moderno geralmente bifurca em dois padrões:

1. **Monolito Potencializado (Ex: Java/Spring Boot + LangChain4j):** A regra de negócio clássica e o código de IA rodam sob o mesmo processo de servidor. Mais fácil de manter para equipes pequenas, mas pesado de escalar isoladamente.
2. **Microserviço Misto (Ex: Backend Java + Microserviço Python/FastAPI):** O sistema *core* continua em Java/C#, mas envia as requisições complexas de IA via rede para um servidor Python isolado. Traz o benefício de rodar a IA no ecossistema Python (onde 99% das inovações nascem) sem ter que reescrever todo o *backend* corporativo da empresa.

A Orquestração de Agentes

Além de chamar a IA, a arquitetura moderna embute *Agentes*:

- Se o projeto exige busca em tempo real na Web, o Backend não deve apenas chamar o LLM; deve invocar um Agente ReAct.
- Agentes exigem laços (*loops*) de observação e pensamento, que consomem **tempo** e multiplicam o gasto de **tokens**.
- Arquiteturalmente, isso exige um servidor tolerante a requisições longas (Timeout ¿ 30s) e um mecanismo de auditoria para saber "por que" o agente falhou se entrar em *loop* infinito.

Engenharia de Dados vs Inteligência Artificial

O *Kickoff* do projeto costuma expor uma dura realidade: faltam dados limpos.

- As Startups dedicam cerca de 80% do tempo na Engenharia de Dados (Coletar, Limpar, Estruturar) e apenas 20% na Modelagem de IA.
- Um banco vetorial alimentado com PDFs quebrados, tabelas corrompidas e parágrafos sem sentido gera uma IA burra.
- A qualidade da sua aplicação no *Demoday* será proporcional ao cuidado que vocês tiveram na etapa de extração de dados no Dia 1.

A Camada de Interface (Frontend)

A interação do usuário com IA muda o paradigma de *Request/Response* tradicional para um fluxo em tempo real.

- **Tempo de Resposta Perceptivo:** Um LLM pode demorar de 5 a 15 segundos para gerar uma resposta densa. Esperar em silêncio faz o usuário achar que a página travou.
- **Streaming de Tokens (SSE - Server-Sent Events):** A aplicação deve enviar a resposta *token* a *token* para a tela do usuário (o famoso efeito "digitando").
- **Gestão de Contexto na Interface:** A UI deve repassar, a cada nova mensagem do chat, o histórico visualizado, ou garantir que o *Backend* rastreie isso através do ID de sessão do usuário.

A Camada Lógica (Backend)

O servidor (Backend) atua como o regente da orquestra, nunca confiando cegamente no usuário ou na IA. O que um backend de IA deve fazer?

- Receber a requisição web (HTTP POST).
- Consultar o banco de dados relacional para validar se o usuário existe e possui cota de mensagens.
- Recuperar o histórico de mensagens da sessão no SQL.
- Chamar a biblioteca orquestradora (*LangChain* ou *LlamaIndex*) passando os *prompts* rigorosos do sistema injetados com as variáveis do usuário.
- Lidar com exceções e *Timeouts* da API da LLM, gerenciando falhas graciosamente.

Gateways de LLM e Roteamento

Em sistemas corporativos maiores, o Backend não chama a API da OpenAI diretamente, ele passa por um **LLM Gateway**.

- **Balanceamento de Carga:** Distribui chamadas entre diferentes contas ou provedores para evitar limites de taxa (*Rate Limits*).
- **Roteamento Dinâmico:** Se a pergunta for simples ("Olá"), o roteador envia para o Llama-3-8B local (barato). Se for análise de contrato, envia para o GPT-4o (caro).
- **Fallbacks:** Se a API oficial cair (*Downtime*), o roteador muda a chamada para um provedor alternativo automaticamente.

A Camada de Dados Dupla

Projetos avançados possuem memória particionada em dois modelos arquiteturais operando simultaneamente:

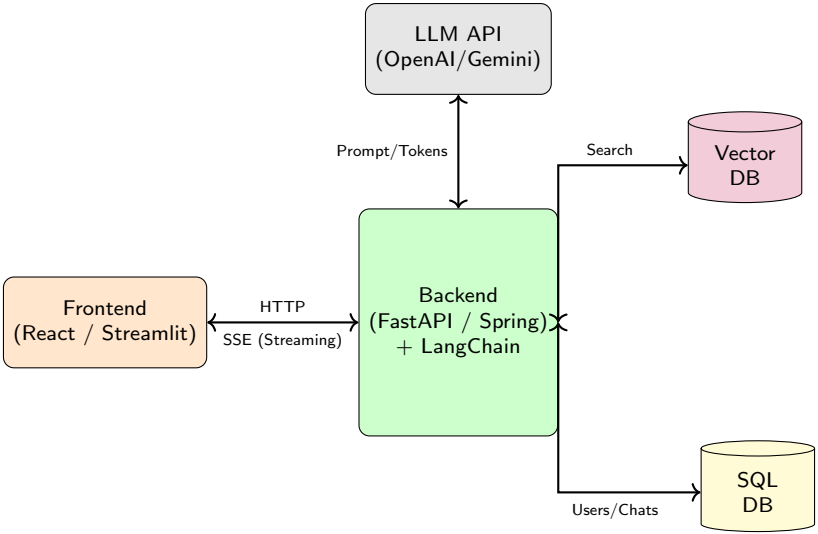
Banco Relacional/NoSQL (PostgreSQL / MongoDB)

- Armazena o "Quem", "Quando" e "O Quê".
- *Tabelas típicas:* `User`, `Session`, `MessageHistory`, `Settings`.
- Consulta exata por ID ou chave primária.

Banco Vetorial (ChromaDB / Pinecone)

- Armazena o Conhecimento (*Knowledge Base*).
- *Tabelas típicas:* `Coleção_Arquivos`, contendo texto original e Vetores Densos ($d = 768$, $d = 1536$).
- Consulta semântica por similaridade de Cosseno ou L2.

Arquitetura de Sistema Completa (System Design)



Revisão: A Engenharia de Ingestão (ETL)

Antes do usuário fazer a primeira pergunta, seus dados precisam existir no banco vetorial. O pipeline de *Ingestão* (ETL - Extract, Transform, Load) corporativo é complexo.

1. **Extração:** Ler PDFs brutos, raspar páginas HTML da empresa, consultar *logs*. O texto extraído costuma vir sujo.
2. **Transformação (Chunking Semântico):** Dividir 10.000 páginas de manual em blocos (ex: 500 *tokens*). Se quebrado abruptamente no meio de uma frase, o sentido (e o vetor) se perde. Estratégias como *Overlap* (sobreposição) garantem que o fim de um bloco tenha o começo do próximo.
3. **Carga (Embedding + Storage):** Cada *chunk* é transformado num vetor usando um modelo de Embeddings e inserido no Banco, junto a Metadados estruturados (ex: `autor`, `data_publicacao`).

Estratégias Avançadas de Busca (Hybrid Search)

A busca apenas por vetor (Dense Search) sofre falhas. Se o usuário busca "Contrato XYZ-998", a semântica importa menos do que a equivalência exata da *string*. **Busca Híbrida (Hybrid Search):**

- Executa-se simultaneamente a busca Densa (Cosseno no VectorDB) e a busca Esparsa / Lexical clássica (BM25, ElasticSearch).
- Os resultados de ambas são unidos.
- Um algoritmo final, como o **Reciprocal Rank Fusion (RRF)**, reordena (Re-rank) a lista para entregar os Top-5 documentos perfeitos ao contexto do LLM.

Metadados e Filtros Rígidos (Pre-filtering)

Muitas vezes, a busca vetorial retorna documentos similares, mas que pertencem a outro usuário ou departamento. **A solução é o Pre-filtering (Filtro Rígido de Metadados):**

- Durante a Ingestão (ETL), anexa-se metadados ao vetor: {‘‘ano’’: 2024, ‘‘depto’’: ‘‘RH’’}.
- Na busca, a API não procura similaridade nos dados antigos. Ela passa um comando: `WHERE ano = 2024 AND depto = 'RH'`.
- Isso aumenta a precisão para quase 100% em contextos corporativos restritos.

Avaliação Sistemática de RAG (RAGAS)

Como saber se a nossa plataforma Hackathon ficou boa ou ruim? Lendo a olho nu? **RAGAS (RAG Assessment):** Um *framework* que utiliza uma LLM-Juíza para dar notas ao seu sistema:

1. **Answer Relevance:** A IA respondeu a pergunta ou fugiu do tema?
2. **Context Precision:** Os documentos trazidos do Banco Vetorial eram realmente relevantes?
3. **Faithfulness (Fidelidade):** A IA inventou fatos (alucinou) ou se manteve restrita aos PDFs fornecidos?

Como funcionará o Hackathon

Nas aulas 26 a 29, a sala atuará simulando o mundo corporativo.

- **Vocês (Startups):** Divididos em grupos (2 ou 3 pessoas). Tomarão as decisões tecnológicas e focarão na entrega de valor. O produto deve ser real.
- **Professor (Arquiteto Sênior / Tech Lead):** Não haverá exposição na lousa. O professor circulará resolvendo arquiteturas quebradas, dando pitacos sobre a escolha de bibliotecas, otimizando *prompts* em gargalos e ajudando no *debug* crítico de código.

Papéis nas Equipes de Dados

A divisão de tarefas reflete os papéis corporativos reais:

- **Data/ML Engineer:** Foca em capturar os dados, criar o *script* de RAG, configurar o banco ChromaDB/Pinecone e refinar os prompts.
- **Backend Engineer:** Cria as rotas do FastAPI/Spring, conecta os *endpoints* e lida com o banco de usuários.
- **Frontend Engineer:** Desenvolve as telas em React/Streamlit e gerencia os estados de carregamento (SSE, Loaders).

O Segredo da Velocidade: O Contrato de API (OpenAPI)

Como uma equipe pequena desenvolve um Frontend e um Backend em paralelo sem esperar o outro terminar? **Desacoplamento pelo Contrato.**

- **Passo 1:** No primeiro dia, defina o *JSON* exato que a API receberá e o que ela retornará. (Ex: Rota `POST /chat` recebe {``pergunta'': ``ola''} e devolve {``resposta'': ``xyz''}).
- **Passo 2:** O desenvolvedor do Frontend cria as telas *mockando* (falsificando) as requisições baseadas no contrato, sem precisar que a IA esteja pronta.
- **Passo 3:** O desenvolvedor do Backend faz a engenharia complexa da IA.
- **Passo 4:** Integração nos últimos dias.

A Gestão do Repositório (Git & Environments)

O caos surge pela má gestão da infraestrutura. Regras de Ouro para a Startup na Aula 1:

1. **Controle de Versão:** Crie um `GitHub` ou `GitLab` imediatamente. Adicione os membros.
2. **Isolamento (`.gitignore`):** NUNCA commite chaves de API (`.env`), bancos vetoriais em disco (`chroma_db/`), nem pastas virtuais (`venv/`, `node_modules/`).
3. **Reprodutibilidade:** O projeto deve rodar no computador do colega (e do Tech Lead) com 1 comando. Tenha um `requirements.txt` ou `pom.xml` atualizado diariamente.

Integração Contínua em Projetos de IA (CI)

Para times avançados, o Git Action é vital.

- A cada "Push" de código no GitLab, um robô na nuvem roda os testes automáticos.
- Testes como: "A API retorna erro 400 se faltar a pergunta do usuário?"
- Se o robô aprovar, a funcionalidade é incorporada ao projeto (*Merge*). Caso contrário, o *Pipeline* quebra e bloqueia a fusão, garantindo a estabilidade.

Conclusão: O Desafio de Integração

A Inteligência Artificial parou de ser uma novidade mágica e tornou-se um **componente de infraestrutura**.

- Dominar APIs ou arquitetura de Transformers é apenas metade da batalha.
- A outra metade é construir sistemas resilientes que encapsulam essas IAs e entregam experiência ao usuário.
- As decisões arquiteturais de hoje moldarão o sucesso ou o fracasso do *deploy* final na última aula do projeto.

Laboratório 26: Ideação e Configuração Base (Kickoff)

Objetivos de hoje para o Grupo:

1. Definir o domínio e o escopo do Projeto Final. O que a aplicação faz?
2. Definir a Stack Tecnológica (Ex: Frontend em HTML/JS, Backend em FastAPI Python).
3. Criar o repositório `git`, estruturar as pastas vazias.
4. Escrever o `README.md` definindo o "Contrato de API".
5. Dividir formalmente as tarefas (*Issues*) para o próximo encontro.

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: **Capstone Sprint 2**

100% da Aula 26 Concluída!