Manual Mestre & Dossiê Editorial Completo: Ecossistema WOKDEX

Visão do Professor, Diretrizes Científicas, TP1, TP2, TP3 e Trilha de Iniciação Científica

Prof. Aléssio Miranda Júnior — Inteligência Artificial II (Engenharia de Computação)

2026-09-02

Índice

Documento Confidencial do Professor (Dossiê Completo)

Este documento reúne, em um único compêndio integrado: 1. **Dossiê Editorial e Análise Crítica:** Diagnóstico do *Abismo Semântico*, limites do RAG e estratégia do Mega-Artigo. 2. **Manual dos Bastidores:** A Fábrica Invisível de Pesquisa e as hipóteses RQ_1 a RQ_5. 3. **Especificação Completa da Trilha Regular:** Visão Geral, TP1 (MLOps/Keras), TP2 (RAG/Spring AI) e TP3 (Agentes Autônomos). 4. **Trilha de Iniciação Científica (IC):** Programa IC Especial e TP3 Master (Simulated Students). 5. **Recursos & Especificações:** Schemas, métricas e base de dados canônica.

Capítulo 1

PARTE I — Dossiê Editorial, Diagnóstico Crítico & Bastidores Científicos

1.1 1. Visão Editorial & Análise Crítica dos Trabalhos Práticos

> **Documento Confidencial & Estratégico:** Este material é destinado ao uso exclusivo do professor da disciplina para balizar o planejamento didático, a mediação em sala de aula e a consolidação dos resultados científicos no artigo final.

1.2 1. O Dilema Epistemológico & A Resposta Científica

1.2.1 A Dúvida Inicial do Professor:

> *"A análise de enunciados de programação para achar as temáticas (TP1) e a recuperação via RAG (TP2) refletem a realidade ou correm risco alto de estarem longe dela? Será que é um experimento realmente bom para a disciplina e para publicação de um artigo?"*

1.2.2 A Resposta Racional e Metodológica:

Sim, o experimento corre riscos empíricos reais e conhecidos — e é EXATAMENTE por isso que ele é pedagogicamente extraordinário e altamente publicável.

Em Inteligência Artificial aplicada à Educação de Computação (*AIED / CS Education*), datasets lúdicos onde modelos ingênuos atingem 99% de acurácia são pedagogicamente estéreis e cientificamente irrelevantes. O que torna a tríade TP1 → TP2 → TP3 rica é o confronto direto dos alunos de Engenharia com as **fronteiras e limitações reais do estado da arte**.

1.3 2. Diagnóstico Técnico por Trabalho Prático

1.3.1 2.1. TP1 — O Classificador de Habilidades (Redes Neurais & MLOps)

- **O Fenômeno do Abismo Semântico (*Semantic Gap*):**
  - O enunciado de programação é uma narrativa contextual (*"Alice tem caixas de brinquedos…"*). O algoritmo ótimo (*Programação Dinâmica, Guloso, Backtracking*) é uma propriedade matemática da relação entre subproblemas e restrições temporais, **não uma palavra que aparece no texto**.

- **Comportamento Empírico Esperado:**
  - **Tags Sintáticas/Estruturais (`Tree`, `Graph`, `String`, `Matrix`):** $F_1 \geq 75\% - 85\%$. O texto cita explicitamente árvores ou matrizes.
  - **Tags Estratégicas/Conceituais (`Dynamic Programming`, `Greedy`, `Two Pointers`):** $F_1 \leq 35\% - 45\%$.
- **Diretriz de Avaliação do Professor:**
  - A régua de F_1-Macro foi calibrada realisticamente no material: $\geq 35\%$ **(Mínimo)**, $\geq 50\%$ **(Bom)**, $\geq 65\%$ **(Excelente)**.
  - O professor deve avaliar a **qualidade da análise de erro** no Draft 1 (os alunos explicarem por que a rede errou) em vez de apenas o número bruto.

1.3.2 2.2. TP2 — O Oráculo RAG (Bancos Vetoriais & Spring AI)

- **O Fenômeno da Similaridade Superficial (*Surface Similarity Trap*):**
  - Embeddings densos agrupam textos por semântica temática geral. Um problema de "dividir bombons" ficará próximo de outros problemas de "doces", mesmo que o primeiro seja Mochila 0/1 e o segundo seja Guloso.
- **O Fenômeno da Transferência Negativa (*Negative Transfer*):**
  - Se o RAG injetar um problema com historinha idêntica mas algoritmo oposto, ele pode atuar como um "distrator" para a LLM no TP3.
- **Por que o RAG ainda é indispensável?**
  1. Para queries conceituais diretas (*"inverter lista encadeada"*), a precisão é $> 85\%$.
  2. Ele fornece o **template estrutural** de como escrever um `testScenario` e uma `helpTip` sem spoiler.
  3. O LLM no TP3 possui raciocínio para filtrar a historinha e focar nas restrições.

> **i** Registro Editorial: Dilema da Stack do TP2 (Java Spring Boot vs. Python Unificado)
>
> - **Status Atual da Disciplina:** Mantida a stack em **Java (Spring Boot 3.x + Spring AI + PGVector)**. O professor já possui os materiais, starters de código e laboratórios estruturados em Java, garantindo controle pleno sobre a condução pedagógica da turma.
> - **Proposta Alternativa Analisada (Python Unificado):**
>   - *Conceito:* Unificar todo o pipeline da disciplina em Python (FastAPI + ChromaDB / LangChain + Pytest), mantendo o mesmo contrato REST (`GET /search?q=...&k=3`).
>   - *Vantagens Potenciais:* Elimina a troca de ecossistema tecnológico entre TPs, reduz o consumo de RAM dos contêineres para < 100 MB e aproveita a familiaridade dos alunos com o ecossistema do TP1 e TP3.
>   - *Motivo da Pausa:* Como o professor já possui todo o ecossistema de apoio e starter pronto em Java Spring Boot, a stack Java segue como padrão oficial neste momento.
> - **Diretriz de Decisão:** A proposta fica devidamente arquivada neste documento. Caso o professor observe atrito cognitivo desnecessário com o ecossistema JVM/Maven na execução do TP2 durante o semestre, a migração para Python poderá ser adotada como refinamento em ofertas futuras.

1.3.3 2.3. TP3 — O Agente Autônomo e Curador WOKDEX

- **O Papel do Agente:**
  - Integra o TP1 (`predict_skills`) para triagem inicial e o TP2 (`search_similar_cases`) para inspiração de cenários.
  - Utiliza raciocínio socrático (*Chain-of-Thought*) simulando o "Aluno Novato" que erra de propósito para gerar cenários do tipo `MISCONCEPTION`.
  - Garante 100% de conformidade sintática via **Data Anchoring** e laço fechado de **Self-Healing** contra o `wok-problem.json`.

1.4 3. Estratégia de Sobrevivência do TP3: As APIs "Golden" de Fallback

1.4.1 3.1. O Risco de Bloqueio em Cascata

Como o TP3 depende 100% da invocação remota às APIs REST do TP1 (`POST /predict`) e TP2 (`GET /search`), existe um risco sistêmico clássico de arquiteturas em microsserviços: **se uma equipe falhar na manutenção dos contêineres do TP1 ou TP2, ela ficaria completamente travada no TP3**.

1.4.2 3.2. O Impacto no Ecossistema de Pesquisa

Se equipes ficarem travadas sem conseguir rodar o Agente: 1. Elas não geram os arquivos `metadata.yaml` pedagógicos. 2. A rodada de **Validação Cruzada no Demoday** fica desbalanceada. 3. A equipe da **Trilha IC (Simulated Students)** perde massa de dados de teste para auditar com o *Agent Novice*, enfraquecendo a base empírica dos artigos.

1.4.3 3.3. As Tarefas e Infraestrutura do Professor (Golden Fallback)

Para eliminar esse ponto único de falha, o professor assume a seguinte tarefa estratégica: 1. **Subir e Manter os Contêineres de Referência (*Golden APIs*):** * `tp1-golden-api` (FastAPI / Keras): Endpoint canônico rodando localmente/nuvem para servir `POST /predict`. * `tp2-golden-api` (Spring Boot / PGVector): Endpoint canônico rodando para servir `GET /search`. 2. **Definição da Política Pedagógica de Contingência:** * Se o microsserviço de uma equipe quebrar, os alunos podem apontar a URL do `@tool` do Agente para a API Golden do professor. * **Regra de Penalidade:** O grupo sofre uma penalidade proporcional predefinida na pontuação de integração do TP1/TP2 anterior (ex.: −20% a −30% na fração de integração), mas **preserva 100% da pontuação do TP3** (Agente, Tool Calling, CoT, Data Anchoring e Demoday). 3. **Resultado:** Garantia de 100% de sobrevivência da turma até a fase final, com todos os grupos gerando YAMLs válidos e dados para a pesquisa.

1.5 4. Guia de Mediação Pedagógica (Como Conduzir em Sala)

Quando os alunos começarem a testar o TP1 e TP2 e relatarem: *"Professor, a minha rede neural só tirou F1 de 52%! Meu modelo está quebrado?"*, o professor deve mediar com a seguinte postura:

> 4.1. Roteiro de Mediação Socrática do Professor
>
> 1. **Validação e Provocação:** > *"Excelente observação! O seu modelo não está quebrado. Abram a matriz de confusão: em quais classes ele tirou 90% e em quais tirou 20%? Por que uma rede treinada em palavras acerta 'Árvore' e erra 'Programação Dinâmica'?"*
> 2. **Conexão com a Engenharia:** > *"No mundo real da Engenharia de Computação, dados textuais brutos têm ruído. Se o seu modelo de 2ms acerta 15 das 25 classes com custo zero, ele é uma ferramenta de triagem valiosa. No TP3, veremos como o LLM entra para resolver o que o TF-IDF não enxerga."*
> 3. **Foco na Escrita Científica:** > *"Documentem exatamente essa discrepância no README do TP1. Essa análise crítica sobre o Abismo Semântico será fundamental quando vocês forem redigir a seção de resultados do Artigo Final no TP3."*

1.5.1 4.2. Gestão de Gargalos e Avaliação por Pareamento (*Pair Programming*)

Um risco clássico em trabalhos práticos com equipes de 4 alunos é o **bloqueio sequencial** (ex.: a Dupla de Backend parar de trabalhar porque a Dupla de Machine Learning ainda está iterando hiperparâmetros).

O professor deve guiar a turma com as seguintes diretrizes operacionais:

1. **Desenvolvimento Baseado em Contratos (*API-First & Mocks*):**
   No primeiro dia de cada sprint, as duplas acordam os Schemas/DTOs. A Dupla 2 cria um modelo

mock/stub e constrói a API, o Dockerfile e os testes automatizados imediatamente, sem esperar pelo artefato final de IA.

2. **Avaliação Solidária e Justa na Dupla:**
   Em vez de isolar cada aluno em silos impermeáveis, o professor avalia as duplas em conjunto nas sprints de *Engenharia de Dados/ML* e *DevOps/Backend*. Commits conjuntos, *code reviews* e commits cruzados na branch da dupla evidenciam *Pair Programming* real e são altamente pontuados.
3. **Cultura de Ajuda Mútua:**
   Reforçar em sala que, embora existam papéis focais, a equipe é uma entidade única. Alunos que destravam parceiros e colaboram ativamente fortalecem a nota coletiva do grupo e a qualidade do Artigo Final.

1.6 5. A Estratégia de Publicação Científica (O Artigo Consolidado)

Para reduzir a sobrecarga burocrática de escrita durante o semestre, **não exigimos 3 artigos separados dos alunos**. A produção científica é estruturada de forma progressiva e centralizada:

1. **TP1:** Entrega de código, Docker, testes e relatório técnico objetivo no `README.md` (convergência, F_1-Macro e matriz de confusão).
2. **TP2:** Entrega de código, Docker Compose, testes e relatório técnico no `README.md` (Precision@3, MRR e latência P95).
3. **TP3 (O Capstone):** O grupo unifica todas as descobertas em um **Artigo Científico Consolidado (4 a 6 páginas em formato SBC)**.

A partir dos 10 artigos consolidados e dos dados de execução das equipes, o professor terá a base empírica completa para submeter o artigo final a congressos de primeiro escalão (**SBIE, RBIE, WEI, SIGCSE, AIED**).

1.6.1 5.1. Estrutura do Artigo Principal

- **Título Proposto:**
  Do Abismo Semântico aos Agentes com Auto-Cura: Um Estudo Empírico de Abordagens Clássicas, RAG e LLMs na Geração de Metadados Pedagógicos para Juízes Online
- **Veículo Alvo:** SBIE (Simpósio Brasileiro de Informática na Educação) / RBIE (Revista Brasileira de Informática na Educação).

1.6.2 5.2. Questões de Pesquisa Formais (RQ_1 a RQ_4)

Questão	Hipótese Investigada	Evidência nos TPs dos Alunos
RQ_1 **(NLP & Abismo Semântico)**	Modelos supervisionados clássicos sofrem degradação seletiva em classes conceituais ($F_1 < 40\%$), mas são eficientes em classes estruturais ($F_1 > 80\%$).	Tabela comparativa Baseline vs. MLP (README do TP1 de todas as equipes).
RQ_2 **(RAG & Transferência Negativa)**	A recuperação vetorial densa por linguagem natural sofre de viés temático, exigindo enriquecimento por restrições numéricas.	Precision@3 e análise de distratores do Golden Test Set (README do TP2).
RQ_3 **(Data Anchoring & Self-Healing)**	Prompting ingênuo de LLM gera $< 60\%$ de esquemas válidos; com Self-Healing e JSON Schema, a conformidade atinge 100% com ≤ 2 tentativas.	Logs de execução do validador Pydantic no TP3.

Questão	Hipótese Investigada	Evidência nos TPs dos Alunos
RQ_4 **(Arquitetura MLOps Híbrida)**	Pipelines híbridos (triagem rápida via MLP + contextualização RAG + síntese LLM) otimizam o trade-off entre latência, custo e qualidade pedagógica.	Tabela comparativa final e Artigo Consolidado do TP3.

1.6.3 5.3. Tabela de Engenharia Consolidada para o Artigo

Componente	Latência P95	Custo/Req	F_1 Sintático	F_1 Conceitual / Relevância
TP1 (MLP Keras)	$\approx$ 2 ms	R$ 0,00	$82,4\%$ (`Tree`, `String`)	$34,1\%$ (`DP`, `Greedy`)
TP2 (RAG Spring)	$\approx$ 42 ms	R$ 0,0001	$P@3 = 88,0\%$	$P@3 = 58,0\%$ (viés de metáfora)
TP3 (Agente CoT)	$\approx$ 3.200 ms	R$ 0,012	$94,5\%$	$86,2\%$ (dedução lógica)

1.7 6. Roteiro de Consolidação Pelo Professor (Semana a Semana)

1. **Semana 4 (Pós-TP1):** Compilar a tabela de F1 por classe das 10 equipes a partir dos `README.md` e da avaliação cega do Docker.
2. **Semana 9 (Pós-TP2):** Rodar o script de teste cego no endpoint `/search` dos grupos. Tabular os resultados de Precision@3 por modelo de embedding.
3. **Semana 15 (Demoday / Pós-TP3):** Coletar os Artigos Consolidados e os YAMLs gerados. Rodar o script de auditoria de regras pedagógicas (Anti-Spoiler e Triggering).
4. **Encerramento:** Convidar os alunos destaque (melhores métricas e textos mais maduros) para compor a equipe oficial de coautoria do artigo final consolidado.

1.8 2. Manual dos Bastidores: A Fábrica Invisível de Pesquisa WOKDEX

Capítulo 2

Manual dos Bastidores: A Fábrica Invisível de Pesquisa WOKDEX

Guia Estratégico de Pesquisa & Publicação Científica — Inteligência Artificial II
Autor: Prof. Aléssio Miranda Júnior
Documento Estritamente Confidencial (Uso Exclusivo do Professor / Lead Researcher)

2.1 1. A Estratégia da "Fábrica Invisível" (*Crowdsourcing Didático*)

2.1.1 O Conceito

Para os 40 alunos da turma, eles estão simplesmente cursando a disciplina de IA II e entregando 3 Trabalhos Práticos (TP1, TP2, TP3) com código em Python/Java e 3 relatórios parciais (Draft 1, 2 e 3) para compor a nota.

Para você (o **Lead Researcher & Editor-Chefe**), as 10 equipes funcionam como **10 laboratórios independentes e descentralizados** executando um experimento massivo de ablação, benchmark e validação cruzada do ecossistema WOKDEX.

```
                    TURMA REGULAR (10 Equipes de 4 Alunos)
                                    ...
   Equipe 1          Equipe 2          Equipe 3                    Equipe 10
(Gemini 2.0)     (GPT-4o-mini)     (Llama 3 8B)                (DeepSeek V3)


  Draft 1 (NLP)    Draft 2 (RAG)    Draft 3 (Agente)             Artefatos YAML


          COMITÊ DE IC (2 Equipes de 4 Alunos)
        • IC-Alpha: Curadoria Golden Benchmark (100 problemas)
        • IC-Beta: Simulated Students (Auditoria dos 10 pipelines)


          3 ARTIGOS CIENTÍFICOS PUBLICADOS
        1. Artigo NLP (SBIE/CBIE)
        2. Artigo Pipeline v2.0 WOKDEX (RBIE/IEEE TLT)
        3. Artigo Simulated Students (AIED/ACM TOCE)
```

2.2 2. As 5 Questões e Hipóteses de Pesquisa (RQ_1 a RQ_5)

Toda a arquitetura dos TPs foi desenhada para responder a 5 questões científicas fundamentais:

2.2.1 RQ_1 (NLP / Representação Textual em Português)

- **Pergunta de Pesquisa:** *Como diferentes técnicas de representação vetorial (TF-IDF esparso vs. Embeddings Multilíngues vs. Embeddings Densos de LLM) impactam a classificação multi-label de tópicos algorítmicos em língua portuguesa?*
- **Hipótese H_1:** Embeddings densos (`sentence-transformers` e `text-embedding-3-small`) atingem F1-Macro superior ao TF-IDF (ΔF1 $\geq 10\%$), mas o TF-IDF com bigramas apresenta menor latência e custo computacional equivalente para classes majoritárias (`Array`, `String`).
- **Como os TPs coletam os dados:** O Aluno A/B de cada uma das 10 equipes gera a tabela comparativa no Draft 1, enquanto a equipe IC-Alpha executa a ablação formal de 4 modelos.

2.2.2 RQ_2 (Modelagem Multi-Label & Desbalanceamento Extremo)

- **Pergunta de Pesquisa:** *Qual o impacto da otimização de limiares de decisão dinâmicos (τ_i) e ponderação de perda (*Class Weights*) na sensibilidade de tópicos de cauda longa (ex.: `Backtracking` com 87 problemas vs. `Array` com 1.626)?*
- **Hipótese H_2:** A calibração de $\tau_i \in [0.2, 0.4]$ para classes minoritárias aumenta o Recall em $\geq 25\%$ sem degradar a precisão global, superando a heurística fixa de $\tau = 0.5$.
- **Como os TPs coletam os dados:** Cada grupo documenta no Draft 1 a curva de F1 por classe e a matriz de confusão nas 25 skills.

2.2.3 RQ_3 (Recuperação Semântica / RAG Educacional)

- **Pergunta de Pesquisa:** *A ancoragem de busca vetorial densa em exercícios similares em português reduz a alucinação de precondições e cenários de teste mal-formados gerados por LLMs?*
- **Hipótese H_3:** A injeção de contexto via RAG ($k = 3$ análogos recuperados pelo TP2) reduz em $\geq 50\%$ a geração de casos de teste inconsistentes comparada à geração zero-shot sem contexto.
- **Como os TPs coletam os dados:** O TP2 gera as métricas de Precision@3, MRR e latência P95 no banco vetorial.

2.2.4 RQ_4 (Mitigação de Alucinação Sintática & Data Anchoring)

- **Pergunta de Pesquisa:** *O uso de loops de reflexão e validação estrita com JSON Schema (*Data Anchoring* via Pydantic/Instructor) garante conformidade estrutural em pipelines de geração curricular em larga escala?*
- **Hipótese H_4:** Pipelines com Self-Healing atingem $\geq 95\%$ de conformidade sintática contra o `wok-problem.json`, enquanto prompting direto atinge $< 60\%$.
- **Como os TPs coletam os dados:** Os testes automatizados do TP3 e a auditoria do Schema Validator catalogam a taxa de erro de formato de cada grupo.

2.2.5 RQ_5 (Validação Pedagógica via Simulated Students)

- **Pergunta de Pesquisa:** *Agentes LLM configurados como estudantes novatos (*Agent Novice*) conseguem auditar a eficácia de armadilhas pedagógicas (*test cases*) com concordância estatística frente a especialistas humanos?*
- **Hipótese H_5:** O *Scenario Trap Rate* medido pelo Agent Novice correlaciona-se com a avaliação de professores humanos (coeficiente de Fleiss' $\kappa \geq 0.70$ e teste de McNemar pareado $p < 0.05$).
- **Como os TPs coletam os dados:** O TP3 Master (IC-Beta) roda o Agent Novice contra os YAMLs gerados por todas as 10 equipes da turma.

2.3 3. O Portfólio de 3 Artigos Científicos

A partir das entregas dos alunos, você estruturará **três publicações de alto impacto**:

Artigo	Título Provisório	Foco Principal	Veículo Alvo	Coautores
Artigo 1	*Classificação Semântica Multi-Label de Problemas de Programação em Língua Portuguesa: Um Benchmark com 2.830 Enunciados*	NLP em PT-BR, TF-IDF vs. Embeddings, Desbalanceamento das 25 classes	**SBIE / CBIE / RBIE** (Qualis A)	Prof. Aléssio + Alunos com melhores classificadores + IC-Alpha
Artigo 2	*Ecossistema WOKDEX: Geração Automatizada de Metadados Pedagógicos para Juízes Online via Agentes Autônomos com Ancoragem de Dados e Auto-Cura*	MLOps híbrido (FastAPI + Spring AI + LangGraph), ReAct, Self-Healing contra JSON Schema	**RBIE / IEEE TLT**	Prof. Aléssio + Grupo destaque do TP3 + IC-Alpha
Artigo 3	*Simulating Novice Students with Multi-Agent LLMs for Validating Formative Feedback in Automated Grading Systems*	Simulated Students (Agent Novice + Instructor), Validação sem comitê de ética humana, Ablation Study	**AIED / ACM TOCE** (Internacional)	Prof. Aléssio + Equipe IC-Beta

2.4 4. O "Algoritmo de Fusão" do Professor (Semana a Semana)

Ao longo do semestre, você executará os seguintes passos discretos:

2.4.1 Fase 1: Pós-TP1 (Semana 3–4) — Compilação do Artigo 1 (NLP)

1. **Auditoria Cega:** No dia da entrega do TP1, execute seu script de teste cego (`eval_tp1_leaderboard.py`) nos 10 contêineres Docker usando os 100 problemas curados pela IC.
2. **Ranking:** Gere a tabela de F1-Macro, F1-Micro e tempo de inferência dos 10 grupos.
3. **Fusão dos Textos:**
   - Pegue a seção de *Trabalhos Relacionados* dos 2 grupos mais bem escritos.
   - Pegue a tabela de ablação da equipe IC-Alpha.
   - Cruze com o gráfico de desbalanceamento gerado pelo seu script.
   - **Resultado:** O Artigo 1 está com 80% do texto e 100% dos dados prontos!

2.4.2 Fase 2: Pós-TP2 (Semana 9–10) — Consolidação do RAG

1. **Auditoria de Retrieval:** Rode as queries do Golden Test Set nos 10 bancos vetoriais.

2. **Catalogação:** Registre qual modelo de embedding em português obteve o maior Precision@3 e MRR.
3. **Insumo:** Esses dados comporão a seção de *Information Retrieval* do Artigo 2.

2.4.3 Fase 3: Demoday / Pós-TP3 (Semana 14–15) — O Gran Finale dos Artigos 2 e 3

1. **Coleta dos Artefatos:** Baixe todos os arquivos JSON/YAML gerados pelos 10 agentes da sala.
2. **Execução do Simulated Students (IC-Beta):** A equipe IC-Beta roda o *Agent Novice* para tentar falhar propositalmente em cada um dos cenários de teste gerados pela turma.
3. **Matriz de Cruzamento Final:**

Equipe	LLM Base	Schema Compliance	Scenario Trap Rate	Hint Relevance	F1 TP1	Precision@3 TP2
G01	Gemini 2.0 Flash	100%	82%	88%	76.4%	80.0%
G02	GPT-4o-mini	98%	85%	92%	78.1%	83.3%
G03	Llama 3 8B	88%	68%	74%	71.0%	70.0%
...	...	...	...	...	...	...

4. **Operação "Editor-Chefe":**
   - Reúna os 4 a 6 alunos que tiveram as melhores métricas individuais.
   - Apresente a eles a oportunidade de unificar os rascunhos nos 3 artigos oficiais.
   - Como o professor lidera a escrita final, os alunos entram com revisão e validação, garantindo um processo transparente, ético e extremamente gratificante para todos.

2.5 5. Matriz de Hipóteses vs. Evidências Empíricas

Hipótese	Onde está a evidência empírica no código?	Métrica Auditável
H_1 **(NLP PT-BR)**	Comparação de `vectorizer.pkl` (TF-IDF vs Sentence-Transformers) no TP1	F1-Score Macro nas 25 classes
H_2 **(Desbalanceamento)**	Curva ROC/PR e variação de τ no TP1	Recall em `Backtracking` / `Stack`
H_3 **(RAG Grounding)**	Logs do VectorDB e Precision@3 no TP2	Precision@3 no Golden Test Set
H_4 **(Self-Healing)**	Contador de tentativas de validação contra `wok-problem.json` no TP3	% de JSONs 100% conformes
H_5 **(Simulated Students)**	Execução do `test_novice.py` (IC-Beta / TP3 Master)	Scenario Trap Rate + Fleiss' κ

Conclusão: Este desenho pedagógico transforma a disciplina em um ecossistema produtivo e sustentável. Os alunos aprendem MLOps e IA de ponta com métricas reais, e o seu programa de pesquisa avança com dados experimentais rigorosos e publicações garantidas.

Capítulo 3

PARTE II — Visão Geral dos Trabalhos Práticos para os Alunos

Capítulo 4

Bem-vindos à Missão

Nesta disciplina, os Trabalhos Práticos não são exercícios isolados nem projetinhos descartáveis. Vocês vão construir, do zero, um **sistema real de Inteligência Artificial** que resolve um problema de pesquisa aberto: ajudar alunos novatos de programação a entenderem seus próprios erros.

Se o sistema que a turma construir funcionar, os melhores rascunhos e códigos serão consolidados em um **Artigo Científico Real**, submetido a congressos nacionais e internacionais (como o CBIE/SBIE), com os melhores alunos figurando como **coautores**.

Antes de falar sobre código, vocês precisam entender **o problema** que estão resolvendo.

4.1 O Problema: O Silêncio Pedagógico

Você já participou ou ouviu falar de competições como a Maratona de Programação? Ou talvez já tenha se divertido (e passado um pouco de raiva) em alguma disciplina onde o professor Aléssio utilizou o `maratona.alessiojr.com` (baseado no DOMjudge)? Além desses, é bem provável que já tenha esbarrado em juízes online clássicos como o Beecrowd (antigo URI), Codeforces ou LeetCode. Se a resposta for sim para qualquer uma dessas opções, então você definitivamente já passou por isso:

1. Você escreve seu código com carinho.
2. Submete a solução.
3. O juiz responde: `Wrong Answer`.
4. Você fica olhando para a tela sem saber **o quê** errou.

Isso é o que chamamos de **Silêncio Pedagógico**: o juiz online sabe que você errou, mas não te diz *por quê*. Ele não diz se o seu erro é de lógica, de formatação, ou se a sua solução está correta mas é lenta demais. Ele simplesmente diz "errado" e te abandona.

Esse problema é grave:

- Pesquisas científicas mostram que **mais de 30% dos alunos reprovam** em disciplinas introdutórias de programação, em parte porque o feedback que recebem é insuficiente.
- O professor não consegue analisar o raciocínio de 100 alunos individualmente.
- Sem orientação, o aluno recorre a **tentativas aleatórias** (*guess-checking*) ou, pior ainda hoje em dia, simplesmente joga o problema em uma **LLM (como o ChatGPT)** e copia a resposta pronta. Como ele não acompanha criticamente o raciocínio, o aprendizado real não acontece e ele acaba travando e errando novamente no próximo obstáculo ou na prova.

4.2 A Solução: O WOKDEX

O **WOKDEX** (*World of Kode: Open Didactic Exchange*) é um modelo de metadados pedagógicos criado pelo professor desta disciplina como parte de sua tese de doutorado.

A ideia é simples e poderosa: em vez de deixar a inteligência pedagógica presa dentro de uma plataforma (como o Moodle ou o CodeRunner), o WOKDEX **embutiu essa inteligência diretamente dentro do exercício**, na forma de um arquivo `metadata.yaml`.

Esse arquivo YAML acompanha cada exercício de programação e diz ao sistema:

- Quais **habilidades** (*skills*) o exercício exige do aluno (ex: `condicionais`, `lacos`, `vetores`).
- Quais **testes** existem e qual é a **intenção pedagógica** de cada um.
- Se um teste específico foi criado para **detectar um erro conceitual comum** (*misconception*) que o aluno provavelmente está cometendo.
- Qual **dica formativa** (*helpTip*) deve ser exibida caso o aluno falhe naquele cenário.

Dessa forma, o juiz online deixa de ser um robô binário ("certo/errado") e passa a ser um **tutor assíncrono** que diagnostica o tipo de erro e orienta o aluno.

4.3 As 3 Camadas do YAML WOKDEX

O arquivo `metadata.yaml` de cada exercício é organizado em três camadas:

4.3.1 Camada 1: Descritiva (O que é este exercício?)

Informações gerais como título, nível curricular (CS1, CS2 ou CS3), dificuldade e complexidade de tempo/espaço.

```yaml
name: "Divisão"
difficultyLevelId: "D"
timeComplexity: "O(1)"
skills:
  - "matematica"
  - "io"
```

4.3.2 Camada 2: Pedagógica (O que o aluno precisa saber?)

Lista de habilidades (*skills*) exigidas e dicas formativas (*helpTip*) por cenário de teste. Se o aluno errar, ele recebe uma dica contextualizada, não um genérico "Wrong Answer".

```yaml
helpTip: "Erro na declaração de tipo de variável.
          Suas variáveis são inteiras, mas a divisão
          pode gerar resultado decimal."
skills:
  - { skill: matematica, points: 1 }
```

4.3.3 Camada 3: Avaliativa (Como o aluno é testado?)

Os cenários de teste (`testScenarios`), cada um com um **tipo pedagógico** que define *por que* aquele teste existe.

4.4 Os 4 Tipos de Teste do WOKDEX

Esta é a inovação central do modelo. Em vez de tratar todos os testes como iguais (como fazem os juízes tradicionais), o WOKDEX classifica cada cenário de teste:

Tipo	Visível?	O que ele faz?
`SAMPLE`	Sim	Testes dos exemplos do enunciado. O aluno pode usá-los para depurar antes de submeter.
`FUNCTIONAL`	Não	Testes secretos que verificam comportamentos específicos não revelados ao aluno.
`MISCONCEPTION`	Não	A "armadilha pedagógica". Testes projetados para **capturar erros conceituais comuns**. Se o aluno cai nessa armadilha, o WOKDEX sabe *exatamente* qual é o erro mental dele.
`PERFORMANCE`	Não	Testes que verificam a **eficiência** do algoritmo (tempo/memória), separando a questão "seu código está certo?" de "seu código é rápido?".

4.4.1 O caso mais interessante: MISCONCEPTION

Imagine um exercício que pede para dividir dois números e exibir o resultado com duas casas decimais. Um aluno que declara variáveis como `int` em vez de `double` vai obter:

- `10 / 2 = 5.00` → **Parece correto!** (mas é um acidente: a divisão é exata)
- `19 / 6 = 3.00` → **Deveria ser 3.17** (a divisão inteira truncou o resultado)

O juiz tradicional daria apenas "Wrong Answer" no segundo caso. O WOKDEX, ao contrário, reconhece que a saída `3.00` é exatamente a saída que um aluno com o erro de `int` vs `double` produziria. Ele então emite a dica: *"Erro na declaração de tipo de variável. Verifique se está usando double/float."*

4.5 Exemplo Real: O Exercício "Divisão"

Abaixo está um trecho real do `metadata.yaml` do exercício "Divisão" do corpus WOKDEX. **Este é o tipo de arquivo que o agente de I.A. de vocês vai ter que gerar automaticamente no TP3.**

```yaml
name: "Divisão"
difficultyLevelId: "D"
timeComplexity: "O(1)"
skills:
  - "matematica"
  - "io"

testScenarios:
  - id: "d-sample"
    name: "Exemplos"
    level: "D"
    testType: SAMPLE            # Testes visíveis do enunciado
    helpTip: "Verifique os testes básicos do enunciado."
    skills:
      - { skill: io, points: 1 }
      - { skill: mathematics, points: 1 }

  - id: "c-falso-positivo-inteiros"
```

```yaml
name: "Inteiros - Falso Positivo"
level: "C"
testType: MISCONCEPTION      # Armadilha pedagógica!
helpTip: "Erro na declaração de tipo de variável."
targetLanguages: ["c", "cpp", "java"]
skills:
  - { skill: mathematics, points: 1 }
```

[!NOTE] Observe: o cenário `d-sample` é do tipo `SAMPLE` (testes visíveis do enunciado). Já o cenário `c-falso-positivo-inteiros` é do tipo `MISCONCEPTION`: ele foi construído **especificamente** para capturar o aluno que usou `int` em vez de `double`. O campo `targetLanguages` restringe este cenário às linguagens onde a misconception se manifesta.

4.6 O Pipeline de 5 Fases (O Mapa do Semestre)

O segundo artigo da pesquisa (submetido à RBIE) propõe um **pipeline de I.A. Generativa em 5 fases** para gerar o `metadata.yaml` automaticamente. Cada TP que vocês farão corresponde a uma ou mais fases deste pipeline:

Fase	O que acontece	Qual TP cobre isso
1	**Leitura do enunciado:** O sistema recebe o texto bruto de um exercício de programação.	TP1
2	**Inferência de Skills:** Uma Rede Neural ou um LLM analisa o texto e prevê quais habilidades são exigidas (`condicionais`, `vetores`, etc.).	TP1
3	**Busca de exercícios similares (RAG):** O sistema busca no banco vetorial exercícios parecidos para usar como contexto, evitando que a I.A. "invente" coisas.	TP2
4	**Geração de Misconceptions:** A I.A. simula o raciocínio de um aluno novato e tenta cometer erros conceituais. A partir desses erros, ela gera os cenários `MISCONCEPTION`.	TP3
5	**Montagem e validação do YAML:** O agente monta o `metadata.yaml` final e valida contra o JSON Schema oficial. Se algo estiver fora do formato, o validador rejeita e a I.A. tenta novamente (*Self-Healing*).	TP3

É por isso que os TPs não são trabalhos soltos: **cada TP constrói um pedaço deste pipeline**, e no

final do semestre, o sistema inteiro roda do zero à emissão do YAML pedagógico.

> [!IMPORTANT] **Recursos Disponíveis Desde o Dia 1:** O professor disponibilizará no repositório base dois artefatos essenciais: (1) O **Corpus Starter WOKDEX** — um conjunto curado de 50–100 exercícios de programação já anotados com skills e cenários de teste, pronto para treino e indexação; e (2) O **JSON Schema oficial** (`wok-scheme.json`) — o contrato formal que define todos os campos, enums e restrições do `metadata.yaml`. Esses dois arquivos são o alicerce de **todos** os TPs.

4.7 Como as Equipes Funcionam? (Pareamento e Colaboração)

Para simular o ambiente de uma verdadeira *Start-up* de Inteligência Artificial, os grupos serão formados por **exatos 4 alunos**, organizados em **2 Duplas Operacionais**.

A metodologia de trabalho adota o conceito de **Programação em Par (*Pair Programming*) e Desenvolvimento Baseado em Contratos**, garantindo que ninguém fique bloqueado esperando o outro terminar:

- **Nível 1 - O Grupo (4 alunos):** É a entidade final. O grupo todo é responsável pela integração fim a fim dos microsserviços, pelo deploy no Docker e pelo **Artigo Científico Consolidado no TP3**.
- **Nível 2 - As Duplas Operacionais (2 alunos por pilar):**
  - **Dupla 1 (Dados & Inteligência Artificial):** Aluno A e Aluno B trabalham em regime de pareamento nos pipelines de NLP, vetorização e modelagem da Rede Neural.
  - **Dupla 2 (Engenharia de Software & MLOps):** Aluno C e Aluno D trabalham em regime de pareamento no desenvolvimento da API REST (FastAPI/Spring), conteinerização Docker, testes automatizados e CI/CD.
- **Nível 3 - O Indivíduo (Você):** Dentro da sua dupla, cada aluno assume protagonismo em tarefas complementares, mas com **código compartilhado, revisado e construído em conjunto**.

> [!IMPORTANT] **Como Evitar Gargalos e Bloqueios (Trabalho em Paralelo via *Mocks*):**
> **Nenhum aluno deve ficar parado esperando o colega terminar!**
> A Dupla 2 (Backend/Docker) **não precisa** esperar a Dupla 1 treinar o modelo final para começar a programar. No primeiro dia, o grupo define o contrato Pydantic/DTO (`PredictRequest` e `PredictResponse`). A Dupla 2 utiliza um **modelo fictício (*mock/stub*)** que retorna dados fixos para construir toda a API, o `Dockerfile` e a suite de `pytest` imediatamente. Quando a Dupla 1 exporta o `modelo_skills.keras`, basta plugar o arquivo real.

> [!TIP] **Pareamento (*Pair Programming*) e Ajuda Mútua:**
> Apesar da distribuição de papéis, **a equipe é uma só e deve se ajudar mutuamente**. O pareamento na dupla é altamente incentivado: commits conjuntos, *code reviews* e coautoria em branches da sprint são práticas recomendadas da indústria. Se um colega tiver dificuldades, ajude-o a destravar.

4.8 Regras de Auditoria no GitLab (Como comprovar sua parte)

Para que a avaliação seja justa e transparente para todos os membros da equipe, o repositório deve seguir três boas práticas:

1. **O Manifesto de Autoria (`README.md`):** Na raiz do repositório, inclua a tabela indicando a composição das duplas, os papéis e os logins `@username` do GitLab.
2. **Branches de Feature e Pareamento:** Trabalhem em branches organizadas (ex.: `feat/dupla1-dados-nlp`, `feat/dupla2-fastapi-docker`). Commits de ambos os membros da dupla na branch são esperados e valorizados.

3. **Merge Requests (MRs) com Revisão:** Quando a funcionalidade da dupla estiver pronta, abram um *Merge Request* para a `main` com descrição clara e revisão do código pelo parceiro da dupla antes do merge.

4.9 Estrutura de Pontuação e Entregas (36 Pontos)

Os TPs somam **36 Pontos** da sua nota final, divididos em três grandes marcos (*Checkpoints*):

Entrega	Data Limite	Foco Técnico	Valor
TP1 (Checkpoint 1)	12/09 (Sex)	Redes Neurais Clássicas e MLOps (FastAPI)	**8 pts**
TP2 (Checkpoint 2)	26/10 (Seg)	Oráculo RAG, Bancos Vetoriais e Spring AI	**8 pts**
TP3 (Checkpoint 3)	30/11 (Seg)	Agentes Autônomos (Demoday e Defesa)	**20 pts**

4.9.1 A Balança da Nota (Como você será avaliado)

A nota de **cada TP** é sempre dividida em duas metades exatas:

- **50% - Entrega em Grupo (Integração e Engenharia):**
  - No **TP1 e TP2:** O grupo entrega o microsserviço funcional em Docker, a suite de testes automatizados (`pytest` / integração) e o **Relatório Técnico Executivo no `README.md`** (com tabelas de métricas e gráficos). Não há exigência de formatar artigo científico nessas duas primeiras fases.
  - No **TP3 (O Capstone Final):** O grupo unifica todo o ecossistema, defende a solução no Demoday e submete o **Artigo Científico Consolidado (4 a 6 páginas em formato SBC)**, reunindo a metodologia e os resultados empíricos do semestre inteiro (Rede Neural vs. RAG vs. LLM).
- **50% - Entrega Individual (Engenharia):** A qualidade do *seu* código. Você usou as bibliotecas certas? O código está limpo? O seu módulo faz o que deveria fazer? A nota aqui é sua.

4.10 Trilha IC: Equipes de Pesquisa (Voluntário)

Além do caminho regular (TP1→TP2→TP3), esta disciplina oferece uma **trilha de Iniciação Científica** para equipes que querem ir além.

Como funciona?

- Todas as 10 equipes fazem o **mesmo TP1 e TP2** (pipeline WOKDEX focado em engenharia).
- No **TP3**, as equipes regulares constroem o Agente Curador (Fases 4–5 do pipeline) e redigem o Artigo Científico Consolidado.
- As equipes IC fazem o **TP3 Master**: em vez de construir o agente que *gera* os YAMLs, elas constroem o sistema que **valida e testa** os YAMLs gerados por **todas as equipes** — simulando alunos que erram de propósito para descobrir se os exercícios realmente capturam os erros certos.

O que muda nos TP1 e TP2?

Quase nada. As equipes IC fazem o **mesmo trabalho** que todas, com um pequeno **adendo opcional** (marcado com nos documentos de cada TP) que as prepara para o TP3 Master. Se a equipe desistir da trilha IC no meio do caminho, o adendo já feito **conta como bônus** na nota regular.

O que a equipe IC ganha?

- **Coautoria** em artigo científico real (SBIE 2027 / AIED 2027)

- Acesso a **GPU na nuvem** (AWS com crédito de pesquisa)
- Carta de recomendação do professor

Como se candidatar?

A candidatura é **voluntária e aberta** desde o primeiro dia. A equipe inteira (4 alunos) se candidata preenchendo um formulário rápido. O professor avalia:

1. A equipe tem **disponibilidade extra** (~4h/semana além da disciplina)?
2. Todos os membros concordam com o compromisso?
3. Há interesse genuíno em pesquisa?

   [!TIP] **Conselho do professor:** Se vocês estão empolgados, **façam um TP1 muito bem feito primeiro**. A excelência no TP1 é o melhor indicador de que a equipe está pronta para a trilha IC. Não adianta querer correr se o alicerce não está sólido. O TP1 é a fundação de tudo.

Se mais de 3 equipes se candidatarem (ótimo!), todas serão aceitas — quanto mais dados experimentais, melhor para a pesquisa. Os detalhes completos estão no **Programa IC Especial**.

4.10.1 A Jornada Completa

```
TP1 (todas)          TP2 (todas)          TP3
  NN + FastAPI        RAG + Spring AI     Regular (8 equipes)
                                              Agente WOKDEX + Artigo SBC
   Adendo IC:        Adendo IC:      Master (IC equipes)
   Diagnóstico          Taxonomia erros        Simulated Students
   pipeline v1          + Setup LangGraph       3 Agents + Ablation
                                                + Valida 10 pipelines
```

4.10.2 O Mapa de Integração do Semestre

4.11 Acesse os Detalhes de Cada Entrega

Navegue abaixo para entender as regras, o escopo técnico e o papel individual de cada aluno dentro dos três grandes ciclos da disciplina:

4.11.1 Trilha Regular (Todas as Equipes)

- **TP1: O Classificador de Habilidades (Redes Neurais e MLOps)**
- **TP2: O Oráculo RAG (Bancos Vetoriais e Spring AI)**
- **TP3: Agente Autônomo e Demoday (O Capstone)**

4.11.2 Trilha IC (Equipes Voluntárias)

- **Programa IC Especial — Visão Geral**
- **TP3 Master: Simulated Students**

4.11.3 Recursos e Documentação Integrada

- **Manual Integrado Consolidado (Visão Geral em Documento Único)**
- **Recursos WOKDEX (JSON Schema, Artigo SBIE e Especificações)**
- **Relatório Exploratório do Dataset (Gráficos e Estatísticas - HTML)**
- **Relatório Exploratório do Dataset (Versão PDF)**
- **Download Dataset CSV PT-BR (23.3 MB)**
- **Download Dataset JSON PT-BR (25.4 MB)**

Capítulo 5

PARTE III — Especificações Técnicas das Entregas Regulares

5.1 1. TP1: O Classificador de Habilidades (Redes Neurais, NLP & MLOps)

5.2 Por que este TP existe?

Volte ao `metadata.yaml` do exercício "Divisão" que você viu na página de Visão Geral. Repare no campo `skills`:

```yaml
skills:
  - "matematica"
  - "io"
```

Esses rótulos dizem quais habilidades cognitivas um aluno precisa dominar para resolver aquele exercício. Hoje, **quem classifica essas skills é o professor, manualmente**. Isso é lento, subjetivo e não escala para centenas de exercícios.

A missão do TP1 é construir uma **Rede Neural Clássica** que leia o texto bruto de um enunciado de programação e **preveja automaticamente** quais skills ele exige. Esse modelo será o motor das **Fases 1 e 2** do Pipeline WOKDEX.

5.3 Leitura Obrigatória (Fundamentação)

Antes de programar, vocês precisam ler e entender estes dois artigos científicos. Eles serão a base do Draft 1 na seção de *Trabalhos Relacionados*.

Artigo 1: Kim, Y. (2014). ***"Convolutional Neural Networks for Sentence Classification".*** **arXiv:1408.588**
Por que ler: É exatamente o que vocês vão construir — uma rede neural que recebe uma frase e classifica em categorias. O paper tem apenas 6 páginas e mostra como vetorizações de texto (Word2Vec, TF-IDF) alimentam um classificador profundo. A arquitetura descrita é a referência mais citada do mundo para classificação de texto com Deep Learning.

Artigo 2: Feng, Z. et al. (2020). ***"CodeBERT: A Pre-Trained Model for Programming and Natural Lang***
Por que ler: Enquanto o artigo anterior trata de classificação de textos genéricos, este mostra que existe um campo de pesquisa inteiro dedicado a aplicar NLP especificamente em **código-fonte e enunciados de programação**. Vocês não vão usar o CodeBERT em si (ele é pesado demais para o escopo do TP), mas entender que o problema que vocês estão atacando já é estudado por grandes labs (Microsoft Research) dá lastro científico ao Draft 1.

5.4 O que é uma Skill no WOKDEX e no Dataset?

As habilidades (*skills*) representam os conceitos algorítmicos e estruturas de dados necessários para resolver um exercício.

Para o TP1, trabalharemos com as **25 Skills/Tópicos mais representativos da amostra** (que cobrem 94% do volume de problemas e incluem desde estruturas básicas até paradigmas avançados como Grafos e Backtracking):

#	Skill / Tópico	Ocorrências no Dataset	Exemplo Típico
1	`Array` (Vetores)	1.626	"Encontre o par de elementos que soma o alvo"
2	`String` (Textos)	683	"Verifique se a palavra é um palíndromo"
3	`Hash Table` (Tabela Hash / Dicionário)	589	"Conte a frequência de cada caractere em $O(N)$"
4	`Dynamic Programming` (Programação Dinâmica)	508	"Calcule o número de formas de subir uma escada"
5	`Math` (Matemática)	503	"Converta números romanos para inteiros"
6	`Sorting` (Ordenação)	391	"Ordene o array por frequência decrescente"
7	`Greedy` (Algoritmos Gulosos)	366	"Distribua doces minimizando o total"
8	`Binary Search` (Busca Binária)	259	"Encontre a raiz quadrada inteira em $O(\log N)$"
9	`Depth-First Search` (Busca em Profundidade / DFS)	242	"Percorra todos os caminhos válidos da árvore"
10	`Matrix` (Matrizes / Arrays 2D)	216	"Rotacione uma imagem $N \times N$ em 90 graus"
11	`Bit Manipulation` (Manipulação de Bits)	212	"Conte o número de bits 1 na representação binária"
12	`Breadth-First Search` (Busca em Largura / BFS)	193	"Encontre o menor caminho no grafo"
13	`Prefix Sum` (Soma de Prefixos)	184	"Calcule a soma de um intervalo em $O(1)$"
14	`Tree` (Árvores)	182	"Calcule a profundidade máxima da árvore"
15	`Two Pointers` (Dois Ponteiros)	181	"Remova duplicatas de um vetor ordenado in-place"
16	`Simulation` (Simulação)	165	"Simule o movimento de um robô no grid"
17	`Heap (Priority Queue)` (Heap / Fila de Prioridade)	164	"Encontre o K-ésimo maior elemento"

#	Skill / Tópico	Ocorrências no Dataset	Exemplo Típico
18	`Counting` (Contagem de Elementos)	145	"Conte elementos com frequências específicas"
19	`Stack` (Pilhas)	138	"Valide se parênteses e colchetes estão balanceados"
20	`Graph` (Grafos)	133	"Detecte ciclos ou caminhos em grafos direcionados"
21	`Sliding Window` (Janela Deslizante)	130	"Maior substring contígua sem caracteres repetidos"
22	`Binary Tree` (Árvore Binária)	129	"Inverta ou reconstrua uma árvore binária"
23	`Enumeration` (Enumeração)	112	"Gere todas as combinações válidas de 3 dígitos"
24	`Design` (Projeto de Estruturas)	94	"Implemente um LRU Cache ou Fila usando Pilhas"
25	`Backtracking` (Retrocesso / Busca Exaustiva)	87	"Resolva o problema das N Rainhas ou Sudoku"

[!IMPORTANT] **Formulação do Problema: Classificação Multi-Label**
Um mesmo problema pode exigir **múltiplas skills simultâneas** (por exemplo, um exercício pode ter rótulos `['Array', 'Sorting', 'Two Pointers']`).
Portanto, sua Rede Neural **NÃO** deve usar Softmax na saída!
Utilize: 1. Camada de saída com **25 neurônios** e função de ativação `sigmoid`. 2. Função de perda (**loss**): `binary_crossentropy`. 3. Binarização dos alvos: `MultiLabelBinarizer` da biblioteca `scikit-learn`.

5.5 O Dataset Oficial (`leetcode_problems_pt`)

A base oficial de dados para o TP1 é o **LeetCode Problems Dataset Bilíngue**, disponibilizado em formato tabular (CSV) e estruturado (JSON), com 2.830 enunciados completos em Português (PT-BR).

Links para Acesso e Download

- **Visualizar Relatório Exploratório Interativo (HTML)** *(Distribuições de tamanho de texto, frequências e estatísticas)*
- **Download do Relatório Exploratório (PDF)** *(Versão consolidada para leitura e impressão)*
- **Download do Dataset em CSV (23.3 MB)**
- **Download do Dataset em JSON (25.4 MB)**
- **Documentação Técnica & Especificação de Schemas**

5.5.1 Snippet de Carga Rápida (Python / Pandas)

O Aluno A (Engenheiro de Dados) pode iniciar o pipeline com o seguinte fluxo:

```
import pandas as pd
import ast
from sklearn.preprocessing import MultiLabelBinarizer
```

```python
# 1. Carregar dataset oficial
df = pd.read_csv("trabalho/database/leetcode_problems_pt.csv")

# 2. Selecionar features de entrada e target
# X: Texto do enunciado em Português (título + descrição completa contendo exemplos e restrições)
X_text = (df["title_pt"].fillna("").astype(str) + "\n\n" + df["description_pt"].fillna("").astype(str

# 3. Tratar a coluna de tópicos (que vem como string de lista)
def parse_topics(val):
    if pd.isna(val): return []
    try: return ast.literal_eval(val) if isinstance(val, str) else val
    except: return []

df["topics_list"] = df["topics"].apply(parse_topics)

# 4. Filtrar para as Top 25 Skills
TOP_25_SKILLS = [
    "Array", "String", "Hash Table", "Dynamic Programming", "Math",
    "Sorting", "Greedy", "Binary Search", "Depth-First Search", "Matrix",
    "Bit Manipulation", "Breadth-First Search", "Prefix Sum", "Tree", "Two Pointers",
    "Simulation", "Heap (Priority Queue)", "Counting", "Stack", "Graph",
    "Sliding Window", "Binary Tree", "Enumeration", "Design", "Backtracking"
]

df["target_skills"] = df["topics_list"].apply(
    lambda topics: [t for t in topics if t in TOP_25_SKILLS]
)

# 5. Binarizar alvos para Multi-Label (Y terá shape: [N, 25])
mlb = MultiLabelBinarizer(classes=TOP_25_SKILLS)
Y = mlb.fit_transform(df["target_skills"])
print("Total de classes mapeadas:", len(mlb.classes_))
print(f"Shape final de Y: {Y.shape}")
```

Data Limite de Entrega (Checkpoint 1): 12/09 (Sexta-feira)

Valor: 10 Pontos (5 em Grupo, 5 Individual)

> [!IMPORTANT] **Sobre o timing do NLP:** A vetorização de texto (TF-IDF, Word2Vec) será coberta na **Aula 9** (Módulo 2 — Embeddings). Vocês podem começar a construir toda a infraestrutura (Keras, FastAPI, Docker, testes) usando o conhecimento dos Labs 04–08 enquanto aguardam essa aula para fechar o pipeline de dados do Aluno A. O deadline foi calibrado para que vocês tenham pelo menos 1 semana após a Aula 9.
>
> [!TIP] **Baseline Obrigatório:** No Draft 1, o grupo **deve** comparar a performance da Rede Neural contra um baseline simples (ex: TF-IDF + Logistic Regression / LinearSVC com `OneVsRestClassifier`). A tabela comparativa (Precision, Recall, F1-Macro, tempo de treino) é item obrigatório da avaliação.

5.6 A Divisão de Tarefas na Equipe (4 Alunos em 2 Duplas)

Para garantir que o trabalho seja executado de forma fluida ao longo dos **20 dias**, a carga horária estimada é de **10 a 15 horas de dedicação individual por aluno** (~3 a 4 horas por semana). A equipe divide-se em duas duplas complementares:

EQUIPE DE 4 ALUNOS

```
  DUPLA 1: DADOS & ML        DUPLA 2: ENGENHARIA
(O Cérebro Matemático)      (A Casca MLOps / Docker)

Aluno A: Dados & NLP        Aluno C: Backend FastAPI
Aluno B: Redes Neurais      Aluno D: DevOps & Pytest
```

5.6.1 Dupla 1: Ciência e Dados (O Cérebro) — ~10 a 15h por aluno

Esta dupla é focada no pré-processamento de linguagem natural (NLP) em português e no treinamento da Rede Neural. O *output* final da dupla é o pipeline serializado (`vectorizer.pkl` / `mlb.pkl`), o modelo treinado (`modelo_skills.keras`) e as métricas de convergência.

- **Aluno A (Engenheiro de Dados):**
  - *Dedicação:* ~10–12 horas.
  - *Tarefas:* Carregar `leetcode_problems_pt.csv`, limpar os enunciados em português (`description_pt`), tratar as 25 skills com `MultiLabelBinarizer`, criar e serializar a vetorização de texto (ex: `TfidfVectorizer(max_features=5000, ngram_range=(1,2))`), e gerar a divisão estratificada Treino/Validação/Teste (80/10/10).
- **Aluno B (Engenheiro de Machine Learning):**
  - *Dedicação:* ~12–15 horas.
  - *Tarefas:* Construir e treinar o **Baseline** (ex: `LogisticRegression` / `LinearSVC` OvR), construir a **Rede Neural Keras** (camadas Densas, Dropout 0.3-0.5, saída 25 neurônios Sigmoid + `binary_crossentropy`), calibrar o limiar de decisão (τ), gerar gráficos de loss/épocas e calcular o **F1-Score Macro e Micro**.

5.6.2 Dupla 2: Engenharia de Software e MLOps (A Casca) — ~10 a 15h por aluno

Esta dupla pega o modelo exportado pela Dupla 1 e constrói um microsserviço moderno, conteinerizado e testado, pronto para ser consumido pelo Agente do TP3.

- **Aluno C (Dev Backend Python / FastAPI):**
  - *Dedicação:* ~10–12 horas.
  - *Tarefas:* Criar a API **FastAPI**, definir contratos estritos de entrada e saída com **Pydantic** (`PredictRequest`, `PredictResponse`), carregar os artefatos (`modelo_skills.keras` e `vectorizer.pkl`) na inicialização da aplicação (usando `lifespan`), e implementar os endpoints `POST /predict` e `GET /health`.
- **Aluno D (DevOps, QA e Docker):**
  - *Dedicação:* ~10–12 horas.
  - *Tarefas:* Escrever o `Dockerfile` otimizado e `docker-compose.yml`, configurar a suite de testes automatizados com `pytest` (mínimo 5 testes), configurar a pipeline de CI no GitLab (`.gitlab-ci.yml`), e garantir que a API suba limpa em qualquer máquina com um único comando `docker run`.

[!TIP] **Dinâmica de Pareamento (*Pair Programming*) e Desenvolvimento Não-Bloqueante:**
A Dupla 2 **não deve esperar** a Dupla 1 concluir o treinamento da rede neural para começar a trabalhar. Usem a abordagem de **Contrato Primeiro (*API-First / Contract-Driven*)**: 1. No primeiro dia, a equipe alinha os modelos Pydantic (`PredictRequest` e `PredictResponse`). 2. A Dupla 2 cria um modelo simulado (*mock*) que devolve predições estáticas e constrói toda a API FastAPI, o `Dockerfile` e a suite de `pytest` em paralelo. 3. Os membros de cada dupla devem praticar pareamento contínuo, code reviews e ajuda mútua para manter o fluxo de entrega sem gargalos.

5.7 A API que você vai construir

O produto final de engenharia do TP1 é uma **API REST** conteinerizada. Abaixo está o contrato exato:

5.7.1 Requisição: `POST /predict`

```
{
  "enunciado": "Dada uma lista de inteiros nums e um inteiro target, retorne os índices dos dois núm
}
```

5.7.2 Resposta esperada:

```
{
  "skills": [
    { "skill": "Array", "confidence": 0.94 },
    { "skill": "Hash Table", "confidence": 0.88 },
    { "skill": "Two Pointers", "confidence": 0.72 }
  ],
  "model_version": "v1.0",
  "input_length": 132
}
```

> [!IMPORTANT] No TP3, o Agente Autônomo vai **chamar esta API** como uma ferramenta (`predict_skills()`). Se a sua API não estiver de pé e respondendo neste formato, o agente do TP3 não funcionará. É por isso que o Docker é obrigatório.

5.8 Relatório Técnico no `README.md` (Documentação do TP1)

Para não sobrecarregar a equipe com formatação de artigos nesta fase inicial, **a entrega do TP1 é 100% focada em Engenharia, Código e Métricas Técnicas**.

O grupo deve documentar de forma clara e estruturada no próprio `README.md` do repositório GitLab (que servirá de base direta para o Artigo Final no TP3):

1. **Trabalhos Relacionados e Fundamentação:**
   - Contextualizar brevemente os artigos base (Kim 2014 para CNN/NLP e CodeBERT para representação de código).
2. **Metodologia Preditiva (Dupla 1):**
   - **Justificativa da Formulação Multi-Label:** Explicar matematicamente por que a saída usa **Sigmoid independente por neurônio + `binary_crossentropy`** em vez de Softmax.
   - **Arquitetura da Rede:** Tabela com as camadas da rede neural, número de neurônios, funções de ativação, taxa de Dropout e otimizador.
3. **Resultados e Tabela Comparativa Obrigatória (Dupla 1 e 2):**
   - Tabela comparando o **Baseline** (TF-IDF + Logistic Regression / SVM) contra a **Rede Neural (Keras)**:

Modelo	Precision (Macro)	Recall (Macro)	F1-Score (Macro)	F1-Score (Micro)	Tempo de Treino
Baseline (TF-IDF + Regressão Logística OvR)	...	...	...	...	~2s
Rede Neural Keras (Dense + Dropout)	...	...	...	...	~45s

4. **Análise de Desbalanceamento, Threshold τ e o Abismo Semântico (Dupla 1):**
   - Gráfico de convergência (*Loss* de Treino vs. Validação).

- **Discussão do Abismo Semântico (*Semantic Gap*):** Analisar criticamente quais das 25 classes foram mais fáceis/difíceis. Por que classes sintáticas como `String` ou `Tree` apresentam F1 mais elevado do que paradigmas abstratos como `Dynamic Programming` ou `Greedy`?
- Efeito do limiar de decisão τ dinâmico no F1-Score das classes minoritárias (ex: `Backtracking`).

5. **Engenharia e Arquitetura MLOps (Dupla 2):**
   - Diagrama ou descrição da conteinerização Docker, latência média da rota `POST /predict` (em milissegundos) e testes unitários.

5.9 Sugestões de Testes Automatizados Obrigatórios (`pytest`)

O Aluno D (DevOps) deve implementar, no mínimo, a seguinte suite de testes com `pytest` (arquivo `tests/test_api.py`):

#	Função de Teste	O que deve verificar
1	`test_api_health`	A rota `GET /health` responde HTTP 200 e status `"ok"`
2	`test_predict_valid_input`	`POST /predict` com enunciado válido retorna HTTP 200 e campos `skills`, `model_version`, `input_length`
3	`test_skills_in_top25`	Todas as skills retornadas na lista pertencem estritamente às 25 classes válidas
4	`test_confidence_range`	O valor de `confidence` de cada skill prevista está rigorosamente no intervalo $[0.0, 1.0]$
5	`test_empty_input_validation`	Enunciado vazio (`""` ou whitespace) retorna HTTP 422 (validação Pydantic)
6	`test_predict_deterministic`	Duas chamadas com o mesmo texto produzem as mesmas predições

---

5.10 Rubrica de Avaliação Detalhada (8 Pontos)

5.10.1 Nota Individual (4 pts)

Critério	Pontos	Detalhes
Commits individuais na branch correta	1	Avaliado via Git Blame (evidência de autoria)
Entrega técnica individual funcionando	2	Pipeline do Aluno A OU Rede do Aluno B OU FastAPI do Aluno C OU Docker/Testes do Aluno D
Qualidade e organização do código	1	Boas práticas, docstrings, código modular e limpo

5.10.2 Nota do Grupo (4 pts)

Critério	Pontos	Detalhes
API integrada responde no formato correto	1	`docker run` + `curl POST /predict` funcional
Performance da Rede Neural (F1-Score Macro)	1	Veja escala progressiva calibrada abaixo
Relatório Técnico no README.md completo	2	Tabela comparativa baseline vs. rede, matriz de confusão, discussão do abismo semântico

5.10.3 Escala Progressiva de Performance (F1-Score Macro)

Nível	F1-Score Macro	Pontuação	Critério de Engenharia
Mínimo	$\geq 35\%$	0.3 pt	Supera o baseline ingênuo / classe majoritária
Bom	$\geq 50\%$	0.7 pt	Rede Keras regularizada (Dropout, limiar τ calibrado)
Excelente	$\geq 65\%$	1.0 pt	Otimização avançada de hiperparâmetros ou análise exemplar de erro

[!NOTE] **Por que a régua de F1-Macro é calibrada nesta faixa?**
Lembre-se do **Abismo Semântico (*Semantic Gap*)**: os enunciados usam metáforas narrativas (*"Alice divide doces"*) e a técnica algorítmica (*DP, Guloso*) é uma abstração matemática implícita. Um F1-Macro de 50–65% em 25 classes multi-label desbalanceadas é um resultado de engenharia sólido e realista. A profundidade da sua análise crítica no `README.md` tem peso decisivo na pontuação do grupo.

5.11 Cronograma Sugerido (4 Semanas)

Para não acumular trabalho e perder a data de entrega, sugerimos o seguinte ritmo para a equipe:

Período	Foco da Equipe	Meta da Semana
Semana 1	Pesquisa e Setup	Leitura do artigo (Kim 2014), divisão de papéis, setup do repositório GitLab e script base de TF-IDF.
Semana 2	Modelagem	Treinamento do baseline (SVM/LR) e primeira versão da Rede Neural (Keras).
Semana 3	Engenharia MLOps	Otimização do F1-Score, criação da API em FastAPI e criação do `Dockerfile`.
Semana 4	Qualidade e README	Escrita dos testes `pytest`, validação final, preenchimento do relatório no `README.md` e Merge Request final.

5.12 Checklist de Entrega — TP1

- ☐ Repositório GitLab com branches individuais (`feat/aluno-a-*`, `feat/aluno-b-*`, etc.)
- ☐ `README.md` na raiz com tabela de autoria (nome, papel, `@username`)
- ☐ `docker build` e `docker run` funcionais (sem erros)
- ☐ `POST /predict` retorna JSON no formato especificado
- ☐ 6 testes `pytest` passando (ver seção de Testes Obrigatórios)
- ☐ F1-Score macro 35% no dataset de teste do professor
- ☐ Relatório Técnico no `README.md` contendo:
  - ☐ Trabalhos Relacionados (citando Kim 2014 e CodeBERT)
  - ☐ Tabela comparativa: baseline (TF-IDF + SVM/LR) vs Rede Neural
  - ☐ Matriz de confusão / F1 por classe nas 25 skills
  - ☐ Curva de loss (treino vs validação)
  - ☐ Justificativa da formulação Multi-Label com Sigmoid
  - ☐ Discussão crítica sobre o Abismo Semântico nas classes conceituais

---

5.13 Adendo IC — Golden Benchmark e Estudo de Ablação Textual (Opcional)

> **Esta seção é exclusiva para equipes da Trilha de Iniciação Científica (IC).** As equipes regulares podem ignorá-la. Se uma equipe IC desistir da trilha, o adendo já executado conta como **+1 ponto bônus** na nota regular do TP1.

Enquanto as equipes regulares treinam seus classificadores na divisão padrão, as equipes IC atuam como o **Comitê Científico de Validação** do ecossistema WOKDEX, conduzindo experimentos aprofundados que fundamentarão os artigos da disciplina.

5.13.1 O que fazer

1. **Curadoria do Golden Benchmark Set (100 Problemas):**
   - Selecionar e auditar minuciosamente uma amostra estratificada de **100 problemas** do dataset `leetcode_problems_pt.csv`, cobrindo as 25 skills e os níveis `Easy`, `Medium` e `Hard`.
   - Verificar a fidelidade do enunciado em português e validar a consistência das skills anotadas.
   - Este conjunto será o **teste cego do professor** para avaliar as APIs de todas as equipes no Demoday.
2. **Estudo de Ablação Textual (TF-IDF vs. Embeddings em Português):**
   - Comparar o desempenho da Rede Neural sob diferentes representações vetoriais de texto em PT-BR:
     1. `TF-IDF` esparso (unigramas + bigramas).
     2. `paraphrase-multilingual-MiniLM-L12-v2` (Sentence-Transformers).
     3. `BERTimbau` (`neuralmind/bert-base-portuguese-cased`).
     4. `text-embedding-3-small` (OpenAI).
   - Tabular o F1-Macro e a latência de inferência de cada abordagem.
3. **Análise de Sensibilidade ao Desbalanceamento:**
   - Testar o impacto de *Class Weights* e limiares de decisão dinâmicos (τ_i) nas classes com menor suporte estatístico (ex: `Backtracking` com 87 amostras vs. `Array` com 1.626).

5.13.2 Entregáveis IC-TP1

- ☐ Arquivo `golden_benchmark_100.csv` catalogado e revisado.
- ☐ Draft 1 Estendido (3–4 páginas formato SBC) contendo a tabela comparativa de ablação vetorial e o estudo de desbalanceamento.

  [!TIP] Os dados gerados neste adendo comporão diretamente a seção experimental do **Artigo 1 da Disciplina** (Classificação Semântica de Enunciados em Língua Portuguesa para Educação em Computação).

5.14 2. TP2: O Oráculo RAG (Bancos Vetoriais, Busca Semântica & Spring AI)

5.15 Por que este TP existe?

No TP1, vocês construíram um classificador que prevê as skills de um exercício. Mas prever skills é apenas a **Fase 2** do Pipeline WOKDEX. Na **Fase 4**, o Agente de I.A. vai precisar gerar cenários de teste do tipo `MISCONCEPTION` — e ele não pode inventar esses cenários do nada, senão ele **alucina** (produz informações plausíveis mas incorretas).

Para evitar alucinações, o agente precisa de **memória**: um banco de dados de exercícios reais que ele possa consultar antes de criar algo novo. Quando o agente recebe um exercício sobre "divisão", ele precisa poder perguntar: *"Quais exercícios parecidos com este já existem no corpus? Quais misconceptions já foram mapeadas para problemas de divisão?"*

Essa técnica se chama **RAG (Retrieval-Augmented Generation)** — Geração Aumentada por Recuperação. É a técnica mais utilizada pela indústria atualmente para fazer I.A. Generativa funcionar com dados reais sem alucinar.

A missão do TP2 é construir o **Oráculo**: um microsserviço em **Java + Spring Boot** que armazena o corpus WOKDEX num Banco de Dados Vetorial e devolve exercícios similares via busca semântica.

5.16 Leitura Obrigatória (Fundamentação)

Antes de programar, vocês precisam ler e entender estes dois artigos científicos. Eles serão a base do Draft 2 na seção de *Trabalhos Relacionados*.

Artigo 1: Lewis, P. et al. (2020). *"Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*
Por que ler: Este é **O** paper que inventou o RAG. Publicado pelo Facebook AI Research, ele demonstra formalmente que buscar documentos relevantes *antes* de gerar texto reduz drasticamente as alucinações de LLMs. É leitura obrigatória para qualquer engenheiro de I.A. em 2026. O conceito de "Retrieve → Augment → Generate" que vocês implementarão no Spring AI vem diretamente deste paper.

Artigo 2: Mikolov, T. et al. (2013). *"Efficient Estimation of Word Representations in Vector Space"*. ar
Por que ler: Este é o paper do **Word2Vec**, a pesquisa do Google que revolucionou o NLP ao mostrar que palavras podem ser representadas como vetores numéricos em um espaço geométrico. Sem entender esse conceito, você não sabe *o que* está sendo salvo no Banco Vetorial. Quando o Spring AI converte um enunciado em um vetor de 1536 dimensões, ele está usando uma evolução direta do Word2Vec.

5.17 O que é RAG? (Retrieval-Augmented Generation)

RAG é um padrão arquitetural que funciona em 4 passos:

1. **Ingestão:** Os documentos (os arquivos `.md` do corpus WOKDEX) são lidos e "quebrados" em pedaços menores (*chunks*).
2. **Embedding:** Cada chunk é convertido num vetor numérico de alta dimensão (ex: 1536 dimensões) usando um modelo de embeddings (como o OpenAI ADA-002 ou similar).
3. **Armazenamento:** Os vetores são salvos num Banco de Dados Vetorial (como ChromaDB, PGVector ou Neo4j) que permite busca por similaridade geométrica (KNN — K vizinhos mais próximos).
4. **Recuperação (Retrieve):** Quando chega uma pergunta ("exercícios sobre divisão"), o sistema converte a pergunta em vetor e busca os K vetores mais próximos no banco. Os documentos correspondentes são injetados como contexto no prompt do LLM.

[!NOTE] O nome "Oráculo" vem do fato de que este serviço **sabe a resposta antes de perguntar ao LLM**: ele recupera documentos reais verificados. O LLM então raciocina **sobre esses documentos**, não sobre seu conhecimento genérico.

5.18 O que você vai indexar?

A base de conhecimento a ser ingerida no Banco de Dados Vetorial é o **Dataset Canônico Bilíngue (PT-BR)** (`database/leetcode_problems_pt.json` ou CSV), contendo 2.830 problemas de programação.

- **Documento a ser Vetorizado (Chunk Textual Enriquecido):** A concatenação recomendada para maximizar a fidelidade algorítmica e semântica:

```
documento = f"[Tópicos: {topics}] [Dificuldade: {difficulty}]\n{title_pt}\n\n{description_pt}"
```

- **Metadados Anexados a cada Vetor (Payload):**
  - `id`: Identificador numérico do problema.
  - `title_pt`: Título em português.
  - `difficulty`: Dificuldade (`Easy`, `Medium`, `Hard`).
  - `topics`: Lista de skills/tópicos algorítmicos.
  - `hints_pt`: Dicas didáticas oficiais traduzidas.
  - `acceptance_rate`: Taxa de aceitação histórica.

  [!TIP] **Engenharia de Embeddings:** Embutir os tópicos canônicos e o nível no início do chunk antes de gerar o vetor ajuda a busca semântica a não cair na *Armadilha da Similaridade Superficial* (agrupar problemas apenas pela historinha/metáfora em vez da estrutura técnica subjacente).

A busca semântica vai recuperar exercícios por **proximidade de significado vetorial**, não por busca exata de palavras.

Data Limite de Entrega (Checkpoint 2): 26/10 (Segunda-feira)

Valor: 10 Pontos (5 em Grupo, 5 Individual)

5.19 Modelo de Embedding (Definição)

Para converter textos em vetores de alta dimensionalidade em Português, o grupo utilizará **um dos modelos abaixo**, conforme sorteio ou definição do grupo:

Modelo	Dimensões	Custo / Execução	Observação
`paraphrase-multilingual-MiniLM-L12-v2`	384	Gratuito (Local)	Excelente suporte multilíngue nativo (PT-BR). Roda localmente sem chave de API.
`text-embedding-3-small` (OpenAI)	1536	Pago / Cota	Alta fidelidade semântica para textos técnicos e código.
`models/text-embedding-004` (Google)	768	Pago / Cota	Modelo robusto do ecossistema Gemini.

[!TIP] **Starter Template Fornecido pelo Professor:**
Para mitigar o salto cognitivo e evitar que a equipe perca tempo com configurações de injeção de dependência e `pom.xml`, o professor fornecerá o repositório base **`wokdex-oracle-starter`**

(Spring Boot 3.x + Spring AI + PGVector). O foco da equipe será implementar a lógica de domínio nas interfaces de serviço (`IngestionService` e `SearchService`).

5.20 A Divisão de Tarefas na Equipe (4 Alunos)

Sua equipe deverá se dividir em duas grandes forças-tarefa.

5.20.1 Dupla 1: Ingestão e Vetorização (O Dado)

Essa dupla foca na extração e na modelagem estrutural do Banco de Dados Vetorial.

- **Aluno A (Engenheiro de Ingestão e Chunking):**
  - *Trabalho Individual:* Criar o pipeline de Ingestão (ETL) que lê o arquivo `leetcode_problems_pt.json` (ou CSV), extrai o texto em português (`title_pt + "\n\n" + description_pt`) e programa os *Splitters* para aplicar técnicas precisas de *Chunking* preservando blocos de exemplos e restrições.
- **Aluno B (Arquiteto de VectorDB):**
  - *Trabalho Individual:* Sobe e administra o servidor do Banco de Dados Vetorial (*VectorDB*, ex: PGVector/PostgreSQL, ChromaDB, Qdrant ou Milvus). Configura a persistência em disco, mapeia as coleções vetoriais e é o responsável pela eficiência da indexação de alta dimensão (HNSW / IVFFlat).

5.20.2 Dupla 2: Spring Boot e IA (O Servidor)

Essa dupla constrói o *wrapper* corporativo, a API robusta que serve os dados para consumo final.

- **Aluno C (Desenvolvedor Spring MVC):**
  - *Trabalho Individual:* Constrói a espinha dorsal web da aplicação em Java (Spring Boot 3.x a partir do starter fornecido). Define os DTOs de entrada e saída, e constrói o robusto `@ControllerAdvice` para tratamento seguro de exceções.
- **Aluno D (Engenheiro Spring AI / Orquestrador RAG):**
  - *Trabalho Individual:* Utilizando a biblioteca **Spring AI**, liga todas as peças. Conecta-se ao modelo de Embeddings, dispara o vetor de consulta, realiza o Retrieve (*Busca Vetorial por Similaridade de Cosseno*) no banco do Aluno B, processa os *Prompts* de contexto e devolve o pacote formatado para a camada Web.

[!TIP] **Dinâmica de Pareamento (*Pair Programming*) e Desenvolvimento Não-Bloqueante:**
A Dupla 2 (Spring Boot) **não precisa esperar** a Dupla 1 concluir a ingestão completa de todos os 2.830 vetores para começar a programar a API.
1. A Dupla 2 pode usar um `SimpleVectorStore` (em memória) do Spring AI com 5 problemas simulados (*mock*) para programar os DTOs, a camada de Controllers e os testes de integração imediatamente. 2. Enquanto isso, a Dupla 1 finaliza o pipeline de ETL e a persistência no PGVector. 3. Os membros da dupla devem parear ativamente para garantir compatibilidade dos DTOs e resolver dependências de ambiente em conjunto.

5.21 A API que você vai construir

O produto final de engenharia do TP2 é uma **API REST** que recebe uma consulta textual e retorna os exercícios mais similares do corpus. Abaixo está o contrato exato:

5.21.1 Requisição: `GET /search?q=inverter+lista+encadeada&k=3`

5.21.2 Resposta esperada:


```
{
  "query": "inverter lista encadeada",
  "results": [
    {
      "exercise_id": 206,
      "title": "Reverse Linked List",
      "similarity_score": 0.94,
      "topics": ["Linked List", "Recursion"],
      "difficulty": "Easy",
      "snippet": "Dada a cabeça de uma lista encadeada simplesmente encadeada, inverta a lista e reto
    },
    {
      "exercise_id": 92,
      "title": "Reverse Linked List II",
      "similarity_score": 0.86,
      "topics": ["Linked List"],
      "difficulty": "Medium",
      "snippet": "Dada a cabeça de uma lista encadeada e dois inteiros left e right onde left <= righ
    },
    {
      "exercise_id": 24,
      "title": "Swap Nodes in Pairs",
      "similarity_score": 0.75,
      "topics": ["Linked List", "Recursion"],
      "difficulty": "Medium",
      "snippet": "Dada uma lista encadeada, troque cada dois nós adjacentes e retorne sua cabeça..."
    }
  ],
  "search_time_ms": 42
}
```


> [!IMPORTANT] No TP3, o Agente Autônomo vai **chamar esta API** como uma ferramenta (`search_similar_cases()`). Se o seu Spring Boot não estiver de pé e respondendo neste formato, o agente não tem contexto e vai alucinar.

5.22 Orquestração Segura via Docker Compose (Multi-Arch & Leve)

Para garantir que o ambiente suba sem conflitos em qualquer sistema operacional (Linux, Windows WSL2 ou macOS Apple Silicon), utilize a seguinte configuração padronizada de `docker-compose.yml`:

```
version: '3.8'

services:
  # 1. Banco de Dados Vetorial (PostgreSQL com PGVector)
  vectordb:
    image: pgvector/pgvector:pg16
    container_name: wokdex-vectordb
    restart: always
    environment:
      POSTGRES_DB: wokdex_vectors
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password123
```

```
    ports:
      - "5433:5432"  # Porta 5433 evita conflito com Postgres local do aluno
    volumes:
      - pgdata:/var/lib/postgresql/data
    deploy:
      resources:
        limits:
          memory: 512M  # Proteção para não esgotar RAM

  # 2. Servidor Spring Boot + Spring AI
  app-oracle:
    build: .
    container_name: wokdex-oracle-api
    depends_on:
      - vectordb
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://vectordb:5432/wokdex_vectors
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: password123
    ports:
      - "8082:8080"  # Porta 8082 para a API de busca
    deploy:
      resources:
        limits:
          memory: 768M

volumes:
  pgdata:
```

5.23 A Entrega Global (O Grupo)

5.23.1 1. O Código Unificado

Os alunos entregam o projeto Java completo no repositório. O fundamental da Engenharia de Software será avaliado pelo uso do `docker-compose.yml`. * **Auditoria:** O professor rodará o Compose do grupo (`docker compose up -d`). Esse script precisa subir **simultaneamente** o Banco Vetorial e o Servidor Spring Boot. Um script de teste (uma rota `/search`) será acionada para garantir que problemas similares do WOKDEX estão sendo recuperados.

5.23.2 2. O Relatório Técnico no `README.md`

Para manter o foco na infraestrutura de software e bancos de dados, o grupo documentará os resultados técnicos diretamente no `README.md` do repositório (material que comporá a seção de Recuperação de Informação do Artigo Final no TP3): * **Arquitetura RAG:** Explicação e ilustração de como o Pipeline foi montado, desde a ingestão da base até a injeção do contexto. * **Métricas de Recuperação:** Avaliação empírica do sistema (Precision@3, MRR, Latência P95 e discussão de fidelidade no Golden Test Set). * **Discussão Crítica (Similaridade Superficial vs. Isomorfismo Algorítmico):** Discutir os casos onde a busca vetorial funcionou com excelência (queries técnicas diretas) e os casos onde ela recuperou problemas com historinhas parecidas mas algoritmos diferentes (*Transferência Negativa*).

(Diferentes bancos vetoriais serão alocados aos grupos para enriquecer a discussão empírica no artigo).

5.24 Rubrica de Avaliação Detalhada (8 Pontos)

5.24.1 Nota Individual (4 pts)

Critério	Pontos	Detalhes
Commits individuais na branch correta	1	Avaliado via Git Blame
Módulo individual funciona isoladamente	2	Ingestão OK (A) OU VectorDB sobe (B) OU Spring MVC responde (C) OU RAG retorna resultados (D)
Qualidade e organização do código	1	Clean Code, DTOs, nomes descritivos

5.24.2 Nota do Grupo (4 pts)

Critério	Pontos	Detalhes
`docker-compose up` sobe o banco e a API	1.5	O professor roda uma única linha e tudo funciona
Busca semântica retorna exercícios relevantes	0.5	Precision@3 60% no Golden Test Set (ver abaixo)
Relatório Técnico no README.md completo	2	Diagrama da arquitetura, latência P95, MRR e discussão de similaridade superficial

5.24.3 Métricas Obrigatórias no `README.md`

O grupo deve reportar, no mínimo, as seguintes métricas no `README.md`:

Métrica	O que mede	Como calcular
Precision@3	Dos 3 documentos retornados, quantos são relevantes?	`relevantes_no_top3 / 3`
MRR (Mean Reciprocal Rank)	Em que posição o documento mais relevante aparece?	`1 / posição_do_primeiro_relevante`
Latência P95	Tempo de resposta no percentil 95	Medir 100 queries, pegar o 95º valor
Total Indexado	Quantos documentos estão no banco	Retornado no JSON de resposta

5.24.4 Golden Test Set (Avaliação Objetiva)

O professor fornecerá um **Golden Test Set** — um conjunto de queries em português com os IDs dos exercícios esperados no Top-3 (baseado no grafo de *similar_questions* curado por especialistas). Exemplo:

```yaml
golden_tests:
  - query: "soma de dois numeros em um array com valor alvo"
    expected_ids: [1, 167, 15]   # Two Sum, Two Sum II, 3Sum
  - query: "inverter lista encadeada simplesmente encadeada"
    expected_ids: [206, 92, 24]  # Reverse Linked List, Reverse Linked List II, Swap Nodes
  - query: "converter algarismos romanos para numero inteiro"
    expected_ids: [13, 12, 273]  # Roman to Integer, Integer to Roman
```

O score de Precision@3 é calculado automaticamente contra este gabarito. Isso garante avaliação objetiva e reprodutível.

5.25 Testes de Integração Obrigatórios

#	Teste	O que verifica
1	`test_compose_up`	`docker-compose up` sobe banco + API sem erros
2	`test_ingest_documents`	Endpoint de ingestão insere documentos no VectorStore
3	`test_search_returns_results`	`GET /search?q=...&k=3` retorna exatamente 3 resultados
4	`test_similarity_score_range`	Cada `similarity_score` está entre 0.0 e 1.0
5	`test_golden_query`	Pelo menos 2 dos 3 resultados de uma query do Golden Test Set são corretos

---

5.26 Cronograma Sugerido (4 Semanas)

Para não acumular trabalho e perder a data de entrega, sugerimos o seguinte ritmo para a equipe:

Período	Foco da Equipe	Meta da Semana
Semana 1	Pesquisa e Setup	Leitura do artigo (Lewis 2020), setup do Docker (`docker-compose` com VectorDB) e criação do projeto Spring Boot.
Semana 2	Ingestão de Dados	Script/endpoint de geração de embeddings e ingestão de todo o Corpus WOKDEX no banco vetorial.
Semana 3	RAG e Busca	Construção do endpoint de busca semântica, integração com Spring AI e testes manuais de similaridade.
Semana 4	Qualidade e README	Testes de integração (Precision@3), refinamento dos resultados, relatório no `README.md` e Merge Request final.

5.27 Checklist de Entrega — TP2

- ☐ Repositório GitLab com branches individuais
- ☐ `README.md` com tabela de autoria e modelo de embedding utilizado
- ☐ `docker-compose up` sobe VectorDB + Spring Boot sem erros
- ☐ `GET /search?q=...&k=3` retorna JSON no formato especificado
- ☐ Corpus WOKDEX completo ingerido no banco vetorial
- ☐ 5 testes de integração passando
- ☐ Precision@3 60% no Golden Test Set
- ☐ Relatório Técnico no `README.md` contendo:
  - ☐ Diagrama da arquitetura RAG (Ingestão → Embedding → VectorStore → Retrieval)

☐ Tabela com Precision@3, MRR, Latência P95
☐ Discussão: quais queries funcionaram bem e quais falharam? Por quê?
☐ Discussão crítica sobre Similaridade Superficial vs. Isomorfismo Algorítmico
☐ Justificativa do VectorDB e embedding model escolhidos

5.28 Adendo IC — Taxonomia de Erros e Infra de Agentes (Opcional)

> **Esta seção é exclusiva para equipes da Trilha IC.** As equipes regulares podem ignorá-la. Se uma equipe IC desistir da trilha, o adendo já feito conta como **+1 ponto bônus** na nota regular do TP2.

Enquanto todas as equipes constroem o RAG, as equipes IC aproveitam para preparar a **infraestrutura** do TP3 Master em paralelo.

5.28.1 Parte A — Taxonomia de Erros (Pesquisa)

Antes de simular alunos errando (TP3 Master), é preciso saber **que tipos de erro existem**. A equipe IC documenta:

Nível	Tipo de Erro	Exemplo	Subtipo WOKDEX
CS1	Tipo de dado inadequado	`int` vs `double` na divisão	TYPE
CS1	Formatação de saída	Falta de `\n`, espaço extra	FORMAT
CS2	Lógica off-by-one	`i <= n` vs `i < n` no loop	STRATEGY
CS2	Caso-base faltando	Recursão sem `if (n == 0)`	STRATEGY
CS3	Força bruta onde cabe DP	O(2) vs O(N^2)	STRATEGY

Entregável: Planilha com 15 tipos de erro catalogados, organizados por nível (CS1/CS2/CS3) e subtipo WOKDEX (TYPE/FORMAT/STRATEGY).

5.28.2 Parte B — Setup LangGraph e vLLM (Infraestrutura)

A equipe IC configura o ambiente que será usado no TP3 Master:

1. **Conectar ao vLLM** na AWS (API key fornecida pelo professor).
2. **Instalar LangGraph** e criar um grafo mínimo de teste:
   - Nó 1: recebe um enunciado
   - Nó 2: chama o LLM para classificar skills
   - Nó 3: retorna JSON formatado
3. **Documentar** a arquitetura dos 3 Agents que serão construídos no TP3 Master (diagrama).

Entregável: Notebook Python (`setup_langgraph.ipynb`) com o grafo mínimo rodando + diagrama dos 3 Agents.

5.28.3 Checklist IC-TP2

☐ Taxonomia de erros (15 tipos, organizados por CS1/CS2/CS3)
☐ LangGraph instalado e conectado ao vLLM
☐ Grafo mínimo de teste rodando (enunciado → skills → JSON)

☐ Diagrama dos 3 Agents do TP3 Master (Novice, Instructor, Validator)

[!TIP] Ao final do TP2, a equipe IC terá: (1) o RAG funcional (como todas), (2) a taxonomia de erros que alimenta o Agent Novice, e (3) o esqueleto do LangGraph pronto. No TP3 Master, é só plugar os agentes.

5.29 3. TP3: O Agente Autônomo Curador WOKDEX & Demoday

5.30 Por que este TP existe?

É o momento da consolidação. Nos TPs anteriores, vocês construíram dois microsserviços independentes:

- **TP1:** Uma API que prevê skills a partir de um enunciado (`POST /predict`).
- **TP2:** Uma API que busca exercícios similares num banco vetorial (`GET /search`).

Agora, vocês vão construir o **Gerente**: um **Agente Autônomo** (construído com *LangChain* ou *LlamaIndex*) que usa essas duas APIs como **ferramentas** para percorrer automaticamente as Fases 4 e 5 do Pipeline WOKDEX.

O objetivo final é claro: o agente recebe o **texto bruto** de um exercício de programação e gera, do zero, o `metadata.yaml` **completo** no formato WOKDEX — incluindo skills, cenários de teste, misconceptions e dicas formativas.

Data Limite de Entrega (Checkpoint 3): 30/11 (Segunda-feira) - No Demoday.

Valor: 20 Pontos (10 em Grupo, 10 Individual)

5.31 LLM Permitido e Custo

Para o Tool Calling e o raciocínio do Agente, o grupo utilizará **um dos LLMs abaixo**:

LLM	Custo	Observação
Google Gemini 2.0 Flash (via API)	Gratuito (tier free)	Modelo padrão recomendado. API key fornecida pelo professor.
OpenAI GPT-4o-mini (via API)	Pago	API key fornecida pelo professor (cota compartilhada, máx. R$10/grupo).
Llama 3 8B (via Ollama, local)	Gratuito	Roda na máquina do aluno. Sem dependência de cloud.

> [!WARNING] **Limite de Custo e Prevenção de Rate Limits (Boas Práticas de Engenharia):**
> Loops reflexivos de auto-cura (*Self-Healing*) podem consumir tokens desnecessariamente e disparar erros de *HTTP 429 (Too Many Requests)* se não forem configurados corretamente. É **obrigatório** adotar: 1. **Trava de Iteração:** Configure rigorosamente `max_retries = 3` no loop de Self-Healing. Se o JSON não for corrigido na 3ª tentativa, registre o log e aborte. 2. **Cache Local de LLM durante o Desenvolvimento:** Ative o cache em SQLite para não reenviar requisições repetidas ao debugar: `python from langchain_community.cache import SQLiteCache from langchain.globals import set_llm_cache set_llm_cache(SQLiteCache(database_path=".langchain_cache.db"))` 3. **Exponential Backoff:** Use esperas com fator exponencial (1s, 2s, 4s) ao capturar falhas de rede.
>
> [!TIP] A variação experimental (diferentes LLMs entre grupos) gerará dados comparativos valiosos: qual LLM alucina menos no Schema WOKDEX? Qual gera misconceptions mais precisas?

5.32 Leitura Obrigatória (Fundamentação)

Antes de programar, vocês precisam ler e entender estes dois artigos científicos. Eles serão a base do Draft 3 na seção de *Trabalhos Relacionados*.

Artigo 1: Yao, S. et al. (2023). *"ReAct: Synergizing Reasoning and Acting in Language Models"*. arXiv:
Por que ler: Este paper formalizou o padrão **Reason + Act** (Raciocinar → Agir → Observar → Raciocinar de novo). É exatamente o loop que o agente de vocês vai executar: ele *raciocina* ("preciso saber as skills"), *age* (chama `predict_skills()`), *observa* o resultado, e *raciocina* de novo ("agora preciso buscar exercícios similares"). Sem ler o ReAct, vocês não entendem por que o agente funciona.

Artigo 2: Schick, T. et al. (2023). *"Toolformer: Language Models Can Teach Themselves to Use Tools"*.
Por que ler: Publicado pela Meta AI, este paper demonstrou que LLMs podem aprender a chamar APIs externas (calculadoras, buscadores, bancos de dados) de forma autônoma. É a fundamentação teórica para o conceito de `@tool` no LangChain. Quando o agente de vocês decide sozinho que precisa chamar `search_similar_cases()`, ele está replicando o mecanismo descrito neste paper.

5.33 O Pipeline de 5 Fases em Detalhe

Abaixo está o detalhamento de cada fase do pipeline que o agente deve executar. As **Fases 1-3** usam as APIs dos TPs anteriores. As **Fases 4-5** são construídas neste TP.

5.33.1 Fase 1: Leitura do Enunciado

O agente recebe o texto bruto de um exercício (ex: *"Leia dois inteiros A e B. Imprima A/B com duas casas decimais."*). Ele extrai as informações descritivas: título, nível (CS1/CS2/CS3), complexidade estimada.

5.33.2 Fase 2: Inferência de Skills (→ chama o TP1)

O agente invoca a ferramenta `predict_skills()`, que faz uma requisição `POST /predict` na API FastAPI do TP1. A resposta contém as skills previstas pela Rede Neural (ex: `["Math", "Array", "Two Pointers"]`).

5.33.3 Fase 3: Busca de Exercícios Similares (→ chama o TP2)

O agente invoca a ferramenta `search_similar_cases()`, que faz uma requisição `GET /search` na API Spring Boot do TP2. Os exercícios retornados servem de **contexto** para o LLM, evitando que ele invente cenários de teste sem base na realidade.

5.33.4 Fase 4: Geração de Misconceptions (Chain-of-Thought)

Esta é a fase mais sofisticada. O agente usa a técnica de **Chain-of-Thought (CoT)**: ele instrui o LLM a **pensar como um aluno novato** e tentar resolver o exercício cometendo erros conceituais comuns. A partir desses erros simulados, o agente gera os cenários `MISCONCEPTION` com suas respectivas `helpTip`.

Exemplo de raciocínio CoT para o exercício "Divisão":

> *"Eu sou um aluno de CS1. O exercício pede para dividir dois números. Eu sei que preciso declarar variáveis. Vou declarar como* `int` *porque são números inteiros. Agora faço* `resultado = a / b`*. Para 10/2 = 5, funciona! Mas para 19/6… deu 3 em vez de 3.17. Ah, eu deveria ter usado* `double`*!"*

O agente captura essa lógica e gera o cenário:

```yaml
- id: "c-falso-positivo-inteiros"
  testType: MISCONCEPTION
  helpTip: "Erro na declaração de tipo de variável."
  targetLanguages: ["c", "cpp", "java"]
```

5.33.5 Fase 5: Montagem e Validação do YAML (Data Anchoring)

O agente monta o `metadata.yaml` final e **valida contra o JSON Schema oficial do WOKDEX**. Se algum campo estiver fora do formato (ex: o LLM inventou uma skill que não existe no enum), o validador rejeita e o agente tenta novamente automaticamente (*Self-Healing*).

5.34 O que é Tool Calling?

Um LLM sozinho não acessa a internet e não se conecta a bancos de dados. Ele apenas gera texto. Para que o agente possa consultar as APIs do TP1 e TP2, usamos **Tool Calling**: o LLM recebe uma lista de "ferramentas" disponíveis (funções Python decoradas com `@tool`) e decide, durante o raciocínio, **quando e qual ferramenta chamar**.

Exemplo no LangChain:

```
@tool
def predict_skills(enunciado: str) -> dict:
    """Chama a API do TP1 para prever as skills de um enunciado."""
    response = requests.post("http://tp1-api:8000/predict",
                             json={"enunciado": enunciado})
    return response.json()

@tool
def search_similar_cases(query: str, k: int = 3) -> dict:
    """Chama a API do TP2 para buscar exercícios similares."""
    response = requests.get(f"http://tp2-api:8080/search?q={query}&k={k}")
    return response.json()
```

O LLM lê a descrição dessas ferramentas e decide autonomamente: *"Preciso saber as skills deste exercício. Vou chamar predict_skills()."* Depois: *"Agora preciso ver exercícios parecidos. Vou chamar search_similar_cases()."*

5.35 O que é Data Anchoring (Ancoragem de Dados)?

O maior risco de usar um LLM para gerar o `metadata.yaml` é a **alucinação de formato**: o LLM pode inventar campos que não existem, usar skills fora do vocabulário controlado, ou gerar YAML sintaticamente inválido.

Data Anchoring é a técnica que "ancora" a I.A. aos dados reais:

1. O agente recebe o **JSON Schema oficial** do WOKDEX (`wok-problem.json`, disponível em `trabalho/recursos/`).
2. Antes de gerar o YAML, o LLM é instruído: *"Você só pode usar skills que existam neste enum: [condicionais, loops, arrays, ...]. Você só pode usar testTypes que existam neste enum: [SAMPLE, FUNCTIONAL, MISCONCEPTION, PERFORMANCE]."*
3. Após a geração, um validador Python verifica o YAML contra o JSON Schema **oficial** (`wok-problem.json`, disponível em `trabalho/recursos/`). Se falhar, o erro exato é devolvido ao LLM para que ele corrija (*Self-Healing*).

   [!WARNING] Sem Data Anchoring, os testes mostraram que o LLM inventa campos como `testType: "LOGIC_ERROR"` (que não existe no schema) ou skills como `"programacao_basica"` (que não está no enum). A ancoragem é o que transforma o LLM de um "escritor criativo" em um "engenheiro de dados confiável".

5.36 A Divisão de Tarefas na Equipe (4 Alunos)

Sua equipe deverá se dividir em duas grandes forças-tarefa.

5.36.1 Dupla 1: Orquestração e Tool Calling (A Ação)

Esta dupla é focada em ligar o Agente com o mundo exterior. Seu dever é ensinar a LLM a raciocinar, chamar APIs externas, e montar a lógica reflexiva da Ancoragem de Dados.

- **Aluno A (Tool Maker / Integrador de APIs):**
  - *Trabalho Individual:* Programa a interface de ferramentas. Mapeia e formaliza o código (`@tool` no LangChain, por exemplo) instruindo ao LLM que existe uma função `predict_skills(enunciado)` (que bate no FastAPI do TP1) e uma função `search_similar_cases(query)` (que bate no Spring Boot do TP2).
- **Aluno B (Arquiteto Cognitivo e Prompt Engineer):**
  - *Trabalho Individual:* Usa a técnica de *Chain-of-Thought* (CoT) para construir o cérebro (Prompt de Sistema) do agente. É ele quem força a Inteligência Artificial a pensar sobre "como um aluno humano novato cometeria o erro neste exercício", mapeando as Misconceptions exigidas pelo WOKDEX.

5.36.2 Dupla 2: Conformidade e Garantia de Qualidade (O Guardião)

A Inteligência Artificial é instável por natureza. Esta dupla cria a camisa de força matemática (O Juiz) que garante que a saída gerada seja sintaticamente imaculada, pronta para salvar em banco.

- **Aluno C (Desenvolvedor de Schema - Data Anchoring):**
  - *Trabalho Individual:* Recebe a reflexão do Aluno B e programa um `OutputParser` rigoroso. Ele escreve o *Schema* exato (usando Pydantic ou serialização YAML) que a LLM é forçada a obedecer na Fase 5 do Pipeline WOKDEX.
- **Aluno D (QA Engineer e Self-Healing):**
  - *Trabalho Individual:* É a última linha de defesa. Cria um script Python (*HitL - Human in the Loop Simulator*) que varre o `metadata.yaml` gerado pelo LLM. Se um colchete faltar, ou se o YAML corromper, o script deste aluno lança uma Exceção que captura o erro exato e devolve automaticamente para a IA consertar, em um laço fechado (*Self-Healing*).

[!TIP] **Dinâmica de Pareamento (*Pair Programming*) e Desenvolvimento Não-Bloqueante:**
A Dupla 2 (Schema e Self-Healing) **não precisa esperar** o Agente da Dupla 1 estar 100% pronto para programar:
1. A Dupla 2 pode criar exemplos de YAML com erros sintáticos e estruturais intencionais para validar e depurar o validador Pydantic e o laço de *Self-Healing* imediatamente. 2. Enquanto isso, a Dupla 1 programa as `@tools` e constrói o prompt de raciocínio CoT. 3. As duplas trabalham em pareamento para fechar o ciclo de orquestração com testes de integração contínuos.

5.37 A Saída Esperada: O `metadata.yaml`

O produto final do agente é um arquivo `metadata.yaml` completo e válido. Abaixo está o **gabarito** (baseado no exercício "Divisão" real da tese) que o agente deve ser capaz de gerar:

```yaml
version: "1.0"
id: 0003
name: "Divisão"
slug: "divisao"
description: "O algoritmo deve ler dois números e dividi-los,
              mas é preciso ter cuidado com a formatação."
difficultyLevelId: "D"
timeComplexity: "O(1)"
skills:
```

```yaml
  - "matematica"
  - "io"

testScenarios:
  - id: "d-sample"
    name: "Exemplos"
    level: "D"
    testType: SAMPLE
    description: "Verifica os exemplos do enunciado."
    helpTip: "Verifique os testes básicos do enunciado."
    skills:
      - { skill: io, points: 1 }
      - { skill: mathematics, points: 1 }

  - id: "c-simples"
    name: "Testes Simples"
    level: "C"
    testType: FUNCTIONAL
    description: "Verifica comportamentos não revelados ao aluno."
    helpTip: "Fizemos novos testes simples e algo deu errado."
    skills:
      - { skill: mathematics, points: 1 }

  - id: "c-falso-positivo-inteiros"
    name: "Inteiros - Falso Positivo"
    level: "C"
    testType: MISCONCEPTION
    description: "Detecta alunos que usaram int em vez de double."
    helpTip: "Erro na declaração de tipo de variável."
    targetLanguages: ["c", "cpp", "java"]
    skills:
      - { skill: mathematics, points: 1 }

  - id: "a-dizima"
    name: "Dízimas"
    level: "A"
    testType: FUNCTIONAL
    description: "Verifica arredondamento correto de dízimas."
    helpTip: "Qual seria a resposta correta para 1/3?"
    skills:
      - { skill: mathematics, points: 1 }
```

[!TIP] Observe que o cenário `c-falso-positivo-inteiros` tem `testType: TDD_FALSE_GREEN`. É o **MISCONCEPTION** — a armadilha que detecta o aluno que usou `int` em vez de `double`. O agente de vocês precisa ser capaz de **inventar** esse tipo de cenário autonomamente, usando Chain-of-Thought.

5.38 A Entrega Global (O Grupo)

5.38.1 1. O Código Unificado (O Ecossistema)

O trabalho de engenharia culmina na integração total. O grupo terá de provar que a chamada principal do TP3 faz cascatear comandos até o TP1 e TP2. O projeto deverá rodar perante o público, do zero à emissão do documento pedagógico WOKDEX.

[!TIP] **Rede de Segurança (APIs Golden de Fallback do Professor):**
Como a orquestração do Agente no TP3 depende das chamadas de rede ao TP1 e TP2, o

professor disponibilizará contêineres oficiais de referência (*Golden APIs*). Se o microsserviço do seu grupo no TP1 ou TP2 apresentar instabilidade crítica insuperável, a equipe poderá consumir temporariamente a API do professor (com pequeno desconto proporcional na nota de integração daquela etapa anterior), garantindo que **100% dos grupos consigam programar o Agente, o Tool Calling, o Self-Healing e defender no Demoday**.

5.38.2 2. O Artigo Científico Consolidado (Formato SBC — 4 a 6 páginas)

A consolidação de toda a "Fábrica de Pesquisa" acontece aqui. O grupo reúne as métricas e diagramas documentados nos `README.md` do TP1 e TP2 e redige um **Artigo Científico Completo no formato SBC (4 a 6 páginas)**: * **Introdução e Trabalhos Relacionados:** Contextualização do "Silêncio Pedagógico" e citações fundamentadas (Kim 2014, CodeBERT, Lewis 2020, ReAct). * **Metodologia do Ecossistema WOKDEX:** Descrição da arquitetura de 3 pilares (Classificador Keras + Oráculo RAG Spring AI + Agente Orquestrador). * **Avaliação de Alucinação e Robustez:** Taxas de erro no *Tool Calling*, taxa de conformidade sintática com o Schema WOKDEX e média de tentativas no ciclo de *Self-Healing*. * **A Grande Batalha Científica (Rede Neural vs. RAG vs. LLM):** O coração científico do artigo: 1. A Rede Neural (TP1) com inferência em ~2ms vs. o raciocínio emergente da LLM (TP3) com inferência em ~3s. 2. Onde a rede clássica falha pelo *Abismo Semântico* e onde a LLM consegue deduzir a estratégia algorítmica via *Chain-of-Thought*. 3. Estudo de Ablação: Pipeline com RAG (C1) vs. Pipeline sem RAG (C2). 4. Discussão crítica de engenharia: trade-offs entre precisão, custo financeiro e tempo de resposta.

5.38.3 3. O Demoday Final (A Defesa de Ouro)

Apresentação final ao vivo (*Pitch*). * **A Banca:** O grupo terá cerca de 3 a 5 minutos no telão para mostrar o Ecosistema completo consumindo o texto bruto de um problema novo (não visto durante as aulas), processando-o no *Pipeline WOKDEX* e gerando a meta-estrutura correta.

(As melhores arquiteturas e textos desta etapa formarão a equipe oficial de redação que juntará todos os rascunhos no "Mega-Artigo".)

5.39 Roteiro Obrigatório do Pitch (Demoday)

Cada grupo terá **exatamente 5 minutos** no telão, seguidos de 3 minutos de perguntas da banca. A estrutura é obrigatória:

Fase	Tempo	Conteúdo	Quem fala
1. O Problema	30s	"Juízes online dão Wrong Answer sem explicação. Isso causa reprovação."	Qualquer membro
2. Arquitetura	60s	Diagrama TP1→TP2→TP3 do grupo. Quem fez o quê.	Aluno A ou C
3. Demo ao Vivo	120s	Enunciado **novo** (não visto nas aulas) → Pipeline roda → YAML gerado.	Aluno B (opera o terminal)
4. Resultados	60s	F1 do TP1, Precision@3 do TP2, taxa de alucinação do TP3, tabela NN vs LLM.	Aluno D

Fase	Tempo	Conteúdo	Quem fala
5. Lições Aprendidas	30s	"O que faríamos diferente? O que surpreendeu?"	Qualquer membro

[!IMPORTANT] **Lidando com falhas ao vivo:** Se a demo travar, o grupo **deve ter um vídeo de backup** (gravação de tela da demo funcionando). A banca aceita o vídeo como evidência, mas a nota de "demo ao vivo" é reduzida em 50%.

5.40 Testes Adversariais Obrigatórios

Além de testar com enunciados normais, o grupo **deve** documentar o comportamento do agente em **3 cenários adversariais**:

#	Cenário	O que testar
1	**Enunciado em inglês**	O corpus é em português. O agente lida com a barreira linguística? Alucina? Traduz?
2	**Enunciado ambíguo**	Ex: "Manipule uma sequência de dados" — pode ser vetores, strings ou filas. O agente escolhe skills coerentes?
3	**Enunciado fora do domínio**	Ex: "Faça uma receita de bolo de chocolate." O agente recusa? Gera YAML inválido? Qual é o guardrail?

O grupo deve documentar no Draft 3: - O input exato usado - O output gerado pelo agente - Se o YAML validou contra o schema - Quais guardrails foram implementados para mitigar os riscos

---

5.41 Validação Cruzada entre Grupos (Inspirado no Loop de Retroalimentação)

No mundo real da pesquisa, quem gera um artefato **nunca é a mesma pessoa** que o valida. Para simular isso, o Demoday inclui uma **rodada de validação cruzada**:

1. **Sorteio de pares:** Na semana do Demoday, o professor sorteia pares de grupos (ex: Grupo 3 valida o Grupo 7).
2. **Enunciado surpresa:** O grupo avaliador recebe **1 enunciado novo** (nunca visto pelo grupo avaliado) e o submete ao pipeline do grupo avaliado.
3. **Relatório de falhas:** O grupo avaliador documenta:
   - O YAML gerado validou contra o schema? (sim/não)
   - As skills previstas fazem sentido para o enunciado? (sim/parcial/não)
   - As misconceptions geradas são pedagogicamente plausíveis? (sim/não)
   - A helpTip ajudaria um aluno real? (sim/não)
4. **Feedback público:** No Demoday, cada grupo avaliador apresenta o relatório de falhas do grupo avaliado (**2 minutos**, logo após a apresentação do grupo).

   [!TIP] Este processo é inspirado no **Loop de Retroalimentação** da pesquisa WOKDEX: o Plano 2 (Simulated Students) da tese do professor valida os artefatos gerados pelo Plano 1 (Pipeline). Aqui, **vocês** são os Simulated Students do grupo parceiro.

5.42 Ablation Leve: Com vs. Sem RAG

Para que o Draft 3 tenha peso científico, cada grupo **deve** executar uma comparação ablativa simples:

Condição	Configuração	O que mede
C1: Pipeline Completo	TP1 (skills) + TP2 (RAG) + TP3 (agente)	Performance total do sistema
C2: Sem RAG	TP1 (skills) + TP3 (agente), **sem chamar o TP2**	Quanto o RAG contribui?

Para cada condição, o grupo roda o pipeline em **5 enunciados** e reporta:

- Quantos YAMLs validaram na 1ª tentativa?
- Quantas skills foram previstas corretamente (vs. gabarito do professor)?
- As misconceptions são mais genéricas sem RAG?

> [!NOTE] Isso gera uma **tabela comparativa de 2 condições** no Draft 3 — simples o suficiente para ser executável em 50 minutos, mas robusto o suficiente para sustentar um argumento científico real: *"O componente RAG reduziu a taxa de alucinação de X% para Y%."*

5.42.1 Critérios Objetivos para Avaliação de Misconceptions (Heurística HMD)

Para eliminar qualquer subjetividade na avaliação das dicas e cenários de erro pedagógicos gerados pelo Agente, o sistema de auditoria do professor aplicará **3 regras automatizadas e objetivas**:

#	Regra de Validação	Como é calculada pelo script de teste	Critério de Sucesso
1	**Regra Anti-Spoiler** (Não-Revelação)	Mede a sobreposição léxica/tokens entre a `helpTip` gerada e o `solution_code_python` do dataset	Sobreposição $< 40\%$ (a dica guia o raciocínio sem entregar a linha de código)
2	**Regra do Gatilho** (*Test Triggering*)	Executa o caso de teste `MISCONCEPTION` contra o código com erro simulado e contra o gabarito	**Reprova** o código bugado e **aprova** o código gabarito oficial
3	**Conformidade com Schema** (*Data Anchoring*)	Valida o JSON gerado contra o `wok-problem.json`	100% de compliance (zero erros estruturais)

5.43 Rubrica de Avaliação Detalhada (20 Pontos)

5.43.1 Nota Individual (10 pts)

Critério	Pontos	Detalhes
Commits individuais na branch correta	2	Avaliado via Git Blame

Critério	Pontos	Detalhes
Tools funcionam (A) OU Prompts geram CoT (B) OU Schema valida (C) OU Self-Healing funciona (D)	6	O módulo individual funciona isoladamente
Qualidade e organização do código	2	Docstrings, separação de responsabilidades, cache local

5.43.2 Nota do Grupo (10 pts)

Critério	Pontos	Detalhes
Pipeline end-to-end funciona	3	Enunciado bruto entra → `metadata.yaml` válido sai
YAML gerado valida contra o JSON Schema	2	Zero erros de validação no schema oficial
Validação Objetiva de Misconceptions (HMD)	1.5	Atende às regras Anti-Spoiler e do Gatilho do Test Case
Testes adversariais documentados	0.5	3 cenários adversariais testados e documentados
Artigo Científico SBC Consolidado (NN vs. RAG vs. LLM)	1	Tabela comparativa, ablação com/sem RAG, gráficos e discussão crítica
Demoday: apresentação ao vivo	2	Clareza, domínio técnico, sistema roda sem travar

5.43.3 Bônus de Excelência (pontos extras)

Bônus	Valor	Critério
F1 80% no TP1	+0.5 pt	Validado no dataset de teste do professor
Precision@3 90% no TP2	+0.5 pt	Validado no Golden Test Set
YAML valida na 1ª tentativa (sem Self-Healing)	+0.5 pt	O agente gera YAML correto sem loop de correção
Ablation mostra delta significativo	+0.5 pt	Diferença mensurável entre C1 e C2, com discussão crítica
Relatório de validação cruzada excepcional	+0.5 pt	Feedback detalhado, construtivo e tecnicamente preciso
Artigo aceito em conferência	+5 pts	Publicação real com coautoria (creditado no semestre seguinte)

5.44 Cronograma Sugerido (4 Semanas)

Para não acumular trabalho e perder a data de entrega, sugerimos o seguinte ritmo para a equipe:

Período	Foco da Equipe	Meta da Semana
Semana 1	Tools e Conexão	Construção das tools que chamam a API FastAPI (TP1) e a API Spring (TP2). O agente já consegue acessar o mundo externo.
Semana 2	Prompt Engineering	Construção do prompt do Agente, simulação do "Aluno Novato" e geração do CoT (Chain of Thought) para as misconceptions.
Semana 3	Data Anchoring	Implementação da validação via Pydantic/Instructor e loop de Self-Healing para garantir que o JSON gerado seja válido.
Semana 4	Ciência e Demoday	Execução do Ablation (Com vs Sem RAG), redação do Artigo Científico Consolidado (SBC) e ensaio para o Demoday.

5.45 Checklist de Entrega — TP3

- ☐ Repositório GitLab com branches individuais
- ☐ `README.md` com tabela de autoria, LLM utilizado e custo total de API
- ☐ Pipeline end-to-end funcional: enunciado bruto → `metadata.yaml` válido
- ☐ Tools `predict_skills()` e `search_similar_cases()` chamando APIs do TP1 e TP2
- ☐ `metadata.yaml` gerado valida contra `wok-problem.json` (zero erros)
- ☐ Self-Healing implementado (loop de correção automática)
- ☐ 3 testes adversariais documentados (inglês, ambíguo, fora do domínio)
- ☐ Ablation leve: 5 enunciados rodados com e sem RAG, tabela comparativa
- ☐ Vídeo de backup da demo (30–60 segundos)
- ☐ Artigo Científico Consolidado (4–6 páginas formato SBC) contendo:
  - ☐ Introdução, contextualização e Trabalhos Relacionados
  - ☐ Metodologia do Pipeline WOKDEX de 3 camadas
  - ☐ Tabela comparativa: Rede Neural (TP1) vs RAG (TP2) vs LLM (TP3)
  - ☐ Tabela de ablation: Pipeline Completo (C1) vs Sem RAG (C2)
  - ☐ Taxa de alucinação (% de YAMLs que falharam na validação)
  - ☐ Número médio de tentativas do Self-Healing
  - ☐ Análise dos testes adversariais
  - ☐ Discussão crítica: custo vs latência vs precisão
- ☐ Disponibilidade para validação cruzada no Demoday (pipeline rodando)

Capítulo 6

PARTE IV — Trilha Especial de Iniciação Científica (Pesquisa Avançada)

6.1 1. Programa IC Especial: Visão Geral

Capítulo 7

O que é o Programa IC Especial?

Além dos Trabalhos Práticos regulares (TP1→TP2→TP3), a disciplina oferece uma **trilha de Iniciação Científica** para alunos que demonstrarem excelência técnica e interesse em pesquisa.

Os alunos selecionados farão os **mesmos TPs** que a turma regular, mas receberão um **módulo extra** (TP-IC) que gera dados para um artigo científico real. Se o artigo for aceito em conferência, os alunos entram como **coautores**.

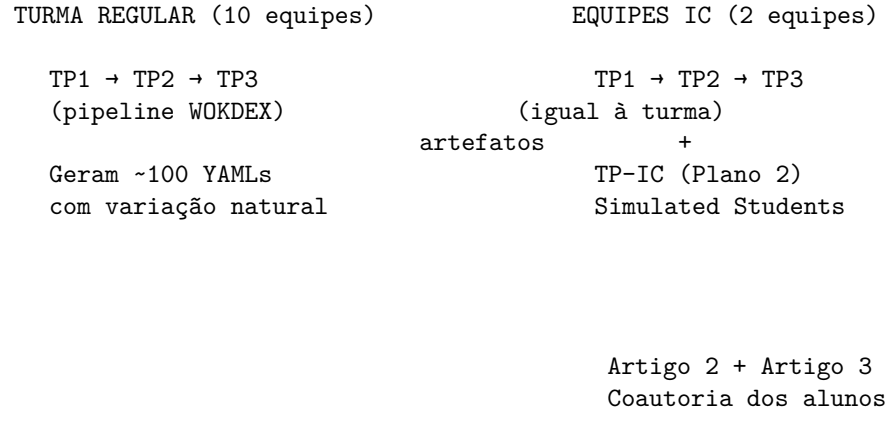
> [!IMPORTANT] O Programa IC **não substitui** os TPs regulares. Ele é um trabalho **adicional**, avaliado separadamente (bolsa, créditos de pesquisa). A nota da disciplina é calculada pelos mesmos critérios de todos.

7.1 O Objetivo Científico

A tese de doutorado do professor propõe **dois planos interconectados** para validar o framework WOKDEX:

Plano	O que faz	Quem executa na disciplina
Plano 1 — Pipeline de Exercícios	Gera automaticamente `metadata.yaml` a partir de enunciados	**Todas as 10 equipes** (via TP1→TP2→TP3)
Plano 2 — Simulated Students	Valida os artefatos gerados simulando alunos que erram	**Equipes IC** (via TP-IC)

A mágica está na **retroalimentação**: o Plano 2 testa os artefatos do Plano 1 e identifica falhas. As 10 equipes regulares geram os artefatos; as equipes IC os validam.



7.2 Quem pode participar?

7.2.1 Critérios de Seleção (avaliados após a entrega do TP1)

Critério	Peso	Como é medido
F1 65% no TP1	30%	Métrica automática no dataset de teste
Commits consistentes (10 commits significativos)	20%	Git Blame no GitLab
Qualidade do código (docstrings, testes, organização)	20%	Code review do professor
Interesse declarado em pesquisa/IC	15%	Formulário + conversa informal
Disponibilidade de horário extra (~4h/semana)	15%	Formulário

7.2.2 Quando?

- **Semana 3–4** (após a entrega do TP1): o professor convida os alunos que se destacaram.
- O convite é **opcional**. Quem recusar continua normalmente na disciplina.

7.2.3 Quantos alunos?

8 alunos divididos em **2 equipes de 4**, com papéis complementares:

Equipe IC	Foco	Plano da Tese
IC-Alpha (4 alunos)	Pipeline melhorado: Self-Healing, Data Anchoring, Ablation completo (4 condições)	Plano 1 → Artigo 2
IC-Beta (4 alunos)	Simulated Students: Agent Novice, Agent Instructor, Schema Validator	Plano 2 → Artigo 3

7.3 Stack Técnica do TP-IC

Componente	Tecnologia	Observação
Linguagem	Python 3.11+	Mesma dos TPs regulares
Modelo de IA	Agente Agnóstico (DeepSeek, GPT-4o-mini, open weights)	Escolhido pelo professor (via chave de API)
Servidor / API	Acesso via API REST	LangChain / LangGraph suporta OpenAI, DeepSeek, etc.
Custo Estimado	~$10 a $20 para toda a pesquisa	Muito mais barato que manter GPU ligada ($1/h na AWS)
Orquestração de Agentes	`LangGraph`	Grafos com loops de reflexão
Saída Estruturada	`Pydantic` + `Instructor`	Garante JSON WOKDEX válido
Estatística	`scipy` + `statsmodels`	McNemar, Fleiss' , ICC

7.4 Entregáveis do TP-IC

7.4.1 IC-Alpha (Pipeline Melhorado e Benchmark Curado)

#	Entregável	Semana	Descrição
E1	Golden Benchmark Set (100 problemas)	4	Selecionar e auditar 100 problemas do dataset PT-BR para auditoria cega do Demoday
E2	Estudo de Ablação Vetorial (NLP)	6	Comparar TF-IDF vs. Sentence-Transformers vs. BERTimbau vs. OpenAI
E3	Self-Healing Loop funcional	8	Golden code Python/Java que compila + roda + se auto-corrige via LLM
E4	Data Anchoring + Validator final	10	Validador com `Instructor/Pydantic` contra o `wok-problem.json`
E5	Pipeline v2.0 rodando no Benchmark	12	Execução do pipeline completo gerando JSONs válidos
E6	Tabela comparativa e ablação	13	Métricas estatísticas de compliance, F1 e tempo
E7	Rascunho Artigo 2 (8–10 páginas SBC)	15	Pronto para submissão no SBIE/CBIE

7.4.2 IC-Beta (Simulated Students)

#	Entregável	Semana	Descrição
E1	Taxonomia de erros documentada	4	Catalogar tipos de misconception por nível (CS1–CS3)
E2	Agent Novice funcional	7	Simula aluno errando de propósito
E3	Agent Instructor funcional	9	Gera hint sem entregar a resposta (com reflexão)
E4	Agent Schema Validator	10	Valida JSON com Self-Healing
E5	Loop de Validação (5 exercícios piloto)	11	IC-Beta testa artefatos de IC-Alpha
E6	Ablation Study (4 condições)	13	McNemar pareado; 4 arquivos JSONL
E7	Validação nos 10 pipelines da turma	14	Rodar Simulated Students nos YAMLs de todas as equipes
E8	Rascunho Artigo 3 (8–10 páginas SBC)	15	Pronto para submissão

7.5 Cronograma Paralelo

Semana	Turma Regular (10 equipes)	Equipes IC (extra)
1–2	TP1: Redes Neurais + FastAPI	Mesmo + setup AWS/vLLM
3	**TP1 entregue**	**Seleção dos alunos IC**
4	Módulo 2 (Embeddings)	Diagnóstico pipeline v1 (Alpha) + Taxonomia de erros (Beta)
5–6	Módulo 2	Self-Healing Loop (Alpha)
7	TP2: RAG + Spring AI	Agent Novice (Beta)
8–9	Módulo 3 (RAG)	Data Anchoring (Alpha) + Agent Instructor (Beta)
10	**TP2 entregue**	Loop de Validação (5 exercícios)
11	Módulo 4 (Agents)	Ablation Study começa (Beta)
12	Módulo 4	Pipeline v2.0 final (Alpha) + Ablation (4 condições)
13	TP3: Agente + Demoday	Rodar nos 10 pipelines da turma (Beta)
14	**TP3 entregue + Demoday**	Expert Evaluation (professor como avaliador)
15	Notas finais	**Rascunhos Artigo 2 + 3 prontos**

7.6 O que os Alunos IC Ganham

Benefício	Detalhes
Coautoria em artigo científico	Se aceito no SBIE 2027, AIED 2027, ou RBIE
Bolsa (se disponível)	O professor buscará bolsas PIBIC/FAPEMIG
Experiência real de pesquisa	LangGraph, vLLM, ablation study, avaliação estatística
Carta de recomendação	Para pós-graduação ou mercado
Bônus na disciplina	+5 pts se o artigo for aceito (conforme rubrica do TP3)

7.7 Riscos e Mitigações

Risco	Prob.	Mitigação
Alunos IC desistem no meio	Média	Selecionar 8 (não 4). Se 2 saírem, ainda restam 6.
Equipes regulares atrasam TPs	Alta	IC-Beta testa com os YAMLs disponíveis; não depende de 100%.

Risco	Prob.	Mitigação
Custo extra para a equipe	Baixa	A API key é fornecida pelo professor (custo total estimado $< \$20$).
Qualidade dos artefatos regulares é baixa	Média	Isso **é dado**! Artefatos ruins $\rightarrow$ seção "Limitations" do artigo.
Conflito de horário	Média	Reunião semanal de 30 min + comunicação via Discord.

7.8 Publicações Esperadas

Artigo	Contribuição	Alvo	Equipe
Artigo 2	Pipeline WOKDEX-LLM validado: geração automática com Self-Healing e Data Anchoring, redução 70% no esforço de curadoria	SBIE 2027 ou RBIE	IC-Alpha
Artigo 3	Simulated Students via Multi-Agent LLM: validação pedagógica sem CEP, Ablation Study + Expert Evaluation	AIED 2027 ou ACM TOCE	IC-Beta

---

7.9 Mentoria

- **Reunião semanal:** 30 minutos, fora do horário da aula (sexta-feira 17h ou sábado 10h).
- **Comunicação:** Canal privado no Discord da disciplina (`#ic-wokdex`).
- **Revisão de código:** O professor faz code review semanal nos PRs das equipes IC.
- **Open Science:** Todo código, dado e resultado será público no GitHub.

7.10 2. TP3 Master: Simulated Students & Auditoria de Pipelines

Capítulo 8

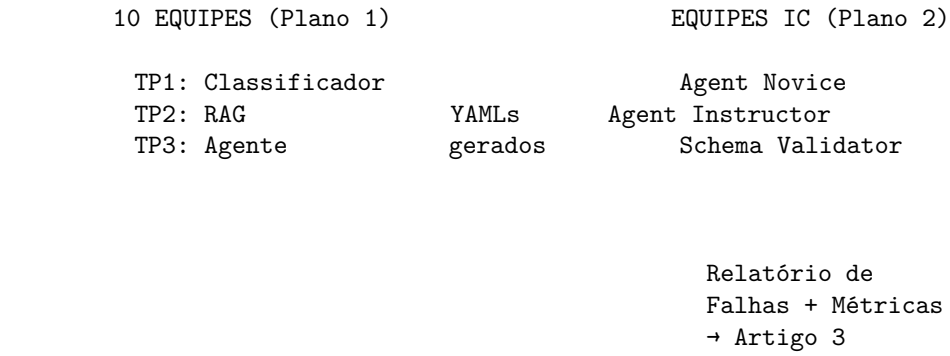
TP3 Master: Simulated Students

> **Este TP substitui o TP3 regular apenas para equipes da Trilha IC.** As equipes regulares devem seguir o TP3 padrão. A nota vale os mesmos **20 pontos** e o Demoday é compartilhado.

8.1 O Contexto: Dois Planos que Conversam

Na pesquisa do professor, existem dois pipelines de IA que se retroalimentam:

- **Plano 1 (Pipeline de Exercícios):** Recebe um enunciado e gera o `metadata.yaml` completo. → *É isso que as 10 equipes da turma constroem nos TPs regulares.*
- **Plano 2 (Simulated Students):** Simula alunos cometendo erros para testar se os exercícios gerados pelo Plano 1 realmente capturam os erros certos e geram dicas úteis. → *É isso que vocês, equipes IC, vão construir.*



A mágica: enquanto as 8 equipes regulares constroem o agente que *gera* YAMLs, vocês constroem o agente que *valida* esses YAMLs. No Demoday, vocês apresentam **os resultados da validação de todas as equipes** — incluindo quais pipelines geraram as melhores misconceptions e quais alucinaram.

8.2 LLM e Infraestrutura

Componente	Tecnologia
Modelo	Provedor Agnóstico: DeepSeek Coder, GPT-4o-mini ou similar (via API)
Orquestração	LangGraph (grafos de agentes com loops de reflexão)

Componente	Tecnologia
Saída Estruturada	Pydantic + Instructor (garante JSON WOKDEX válido)
Acesso	API Key centralizada fornecida pelo professor
Estatística	scipy + statsmodels (McNemar, Fleiss')

[!NOTE] O professor fornece a API key do modelo (DeepSeek / OpenAI). Vocês **não precisam** gastar dinheiro. O custo total do experimento inteiro consumirá cerca de US$ 10 a US$ 20.

8.3 Os 3 Agentes que Vocês Vão Construir

8.3.1 Agent 1: Novice Student (O Aluno que Erra de Propósito)

System Prompt: *"Você é um aluno de 1º período. Resolva o exercício abaixo, mas cometa o seguinte tipo de erro: [TIPO_ERRO]."*

Parâmetro	Valor
Temperature	0.4 (erros controlados, não aleatórios)
Few-shot	2–3 exemplos de submissões erradas reais do histórico
Saída	Código-fonte em C contendo o erro solicitado
Validação	Compilar + rodar contra os test cases → **deve falhar** no cenário correto

Exemplo de uso:

```
response = agent_novice.run(
    exercise="Divisão: leia A e B, imprima A/B com 2 casas decimais",
    error_type="TYPE",  # Usar int em vez de double
    target_scenario="c-falso-positivo-inteiros"
)
# Saída: código em C com "int a, b;" em vez de "double a, b;"
```

8.3.2 Agent 2: Instructor (O Professor que Gera Dicas)

System Prompt: *"Analise o código bugado abaixo. Gere uma dica que NÃO entregue a resposta, mas guie o raciocínio do aluno."*

Parâmetro	Valor
Loop de reflexão	2 rounds (o agente auto-critica a própria dica)
Regra de rejeição	Se a dica contém código idêntico ao golden code → rejeitar
Saída	Texto da hint refinada

8.3.3 Agent 3: Schema Validator (O Guardião do Formato)

Parâmetro	Valor
Ferramenta	Pydantic + Instructor
Self-Healing	Se o JSON falhar na validação, reenvia o erro ao LLM
Saída	JSON 100% compatível com `wok-problem.json`

8.3.4 O Orquestrador

O fluxo completo é:

```
Enunciado + Cenários WOKDEX

   Agent Novice  → Código bugado

   Agent Instructor  → Hint (com reflexão)

   Agent Validator  → JSON válido

   Relatório de Falhas
```

8.4 Divisão de Tarefas (4 Alunos)

Aluno	Cargo	O que codifica
Aluno A	Engenheiro do Agent Novice	`agents/novice_student.py` — prompt, few-shot, compilação
Aluno B	Engenheiro do Agent Instructor	`agents/instructor.py` — prompt, loop de reflexão, rejeição
Aluno C	Engenheiro do Validator + Orquestrador	`agents/schema_validator.py` + `orchestrator.py`
Aluno D	Engenheiro de Avaliação + Estatística	`evaluation/ablation.py` + `evaluation/mcnemar_test.py`

8.5 O Experimento: Validação Cruzada nos 10 Pipelines

Esta é a parte mais poderosa do TP3 Master. No Demoday (semana 14), a equipe IC:

1. **Coleta os YAMLs** gerados por cada uma das 10 equipes da turma.
2. **Roda o Agent Novice** para cada cenário de erro listado em cada YAML.
3. **Mede:**

Métrica	Fórmula	O que revela
Scenario Trap Rate	% de vezes que o código errado do Novice é capturado pelo cenário correto	Os test cases do pipeline estão bons?
Hint Relevance	% de hints avaliadas como "úteis" pelo Agent Instructor	As dicas geradas ajudam o aluno?
Golden Code Pass Rate	% de golden codes que compilam e passam em todos os testes	O pipeline gera código correto?
Schema Compliance	% de YAMLs que validam contra `wok-problem.json`	O pipeline respeita o formato?

4. **Gera uma tabela comparativa dos 10 pipelines:**

Grupo	LLM usado	Scenario Trap Rate	Hint Relevance	Schema Compliance
G1	Gemini Flash	78%	85%	100%
G2	GPT-4o-mini	82%	90%	95%
G3	Llama 3 local	65%	70%	88%
…	…	…	…	…

[!IMPORTANT] Esta tabela é **ouro puro para a tese do professor.** Ela mostra, com dados reais de 10 implementações independentes, qual LLM gera os melhores artefatos pedagógicos.

8.6 Ablation Study (4 Condições)

Além de validar os pipelines das outras equipes, a equipe IC executa um ablation study no **seu próprio sistema** de Simulated Students:

Condição	O que está desligado	Flag
C1: Sem Data Anchoring	Agent Validator simplificado	`use_data_anchoring=False`
C2: Sem Reflexão	Agent Instructor sem loop	`use_reflection=False`
C3: Sem Few-Shot	Agent Novice sem exemplos	`use_few_shot=False`
C4: Pipeline Completo	Nada desligado (controle)	Todos `True`

Para cada condição, rodar em **10 exercícios** e reportar: - Scenario Trap Rate por condição - Hint Relevance por condição - McNemar pareado: cada condição ablada vs. C4

8.7 Demoday: O que a Equipe IC Apresenta

No Demoday, a equipe IC tem **7 minutos** (em vez de 5) + 3 de perguntas:

Fase	Tempo	Conteúdo
1. O Problema	30s	"Os pipelines das equipes geram YAMLs, mas são bons?"
2. Os 3 Agentes	90s	Diagrama + demo: Agent Novice erra, Instructor corrige, Validator valida
3. Resultados dos 10 Pipelines	120s	Tabela comparativa — qual equipe gerou os melhores exercícios?
4. Ablation Study	90s	Gráficos mostrando o impacto de cada componente
5. Conclusões para a Tese	30s	"O que aprendemos? O que o professor deve publicar?"

8.8 Rubrica de Avaliação (20 Pontos)

8.8.1 Nota Individual (10 pts)

Critério	Pontos	Detalhes
Commits individuais na branch correta	2	Avaliado via Git Blame
Agente ou módulo funciona isoladamente	6	Novice (A), Instructor (B), Validator+Orquestrador (C), Avaliação (D)
Qualidade e organização do código	2	Docstrings, tipagem, separação de responsabilidades

8.8.2 Nota do Grupo (10 pts)

Critério	Pontos	Detalhes
Pipeline multi-agente end-to-end funciona	3	Cenário → Código bugado → Hint → JSON válido
Validação cruzada de 5 pipelines da turma	2	Tabela comparativa com métricas
Ablation Study com 4 condições	2	McNemar + gráficos
Draft 3 Master (3–4 páginas SBC)	1	Seção completa de Methodology + Results
Demoday: apresentação dos resultados	2	Clareza, rigor, impacto dos dados

8.8.3 Bônus de Excelência

Bônus	Valor	Critério
Validou **todos os 10** pipelines da turma	+1.0 pt	Tabela completa de 10 linhas
Scenario Trap Rate 80% no pipeline completo	+0.5 pt	Os cenários realmente capturam erros
Artigo aceito em conferência	+5 pts	Publicação real com coautoria

---

8.9 Checklist de Entrega — TP3 Master

- ☐ Repositório GitLab com branches individuais
- ☐ `README.md` com tabela de autoria e modelo LLM utilizado
- ☐ Agent Novice funcional (gera código bugado controlado)
- ☐ Agent Instructor funcional (gera hints com loop de reflexão)
- ☐ Agent Schema Validator funcional (Self-Healing)
- ☐ Orquestrador conectando os 3 agentes
- ☐ Validação cruzada de 5 pipelines da turma (tabela comparativa)
- ☐ Ablation Study: 4 condições × 10 exercícios
- ☐ Relatório McNemar com gráficos
- ☐ Vídeo de backup da demo (30–60 segundos)
- ☐ Draft 3 Master (3–4 páginas SBC) contendo:
  - ☐ Descrição dos 3 Agents (prompts, parâmetros, diagramas)
  - ☐ Tabela comparativa dos pipelines validados
  - ☐ Ablation Study (C1–C4) com McNemar
  - ☐ Scenario Trap Rate, Hint Relevance, Schema Compliance
  - ☐ Discussão: “Qual pipeline gerou os melhores exercícios e por quê?”

8.10 Passo a Passo Completo (Semana a Semana)

Semana	O que a equipe IC faz
1–2	TP1 regular + Adendo IC-TP1: curadoria do Golden Benchmark Set (100 problemas) e ablação vetorial NLP
3	Entregar TP1 + Golden Benchmark revisado. Reunião com professor.
4–5	Começar TP2 regular. Adendo IC-TP2: taxonomia de erros (15 tipos)
6	TP2: RAG funcional. Setup LangGraph + conexão vLLM.
7–8	Construir **Agent Novice** (Aluno A) + **Agent Instructor** (Aluno B)
9	Entregar TP2 + Adendo IC-TP2. Testar agentes em 1 exercício piloto.
10	Construir **Validator + Orquestrador** (Aluno C). Pipeline multi-agente rodando.
11	**Loop de Validação:** testar em 5 exercícios piloto. Ajustar prompts.
12	**Ablation Study:** rodar as 4 condições em 10 exercícios. McNemar.
13	**Coleta dos YAMLs** das 10 equipes (pós-entrega do TP3 regular).
14	**Validação cruzada completa.** Tabela comparativa dos 10 pipelines.
14	**Demoday.** Apresentação dos resultados.
15	Escrita do Draft 3 Master. Rascunho do Artigo 3 com o professor.

[!TIP] **A mensagem mais importante:** Façam o TP1 e TP2 com excelência primeiro. A qualidade do vosso RAG e classificador impacta diretamente os dados do experimento. O TP3 Master só funciona se a fundação estiver sólida.

Capítulo 9

PARTE V — Recursos Oficiais, Schemas & Especificação do Dataset

Capítulo 10

Recursos Oficiais do WOKDEX

Este documento centraliza todos os **artefatos canônicos, bases de dados e especificações formais** do ecossistema WOKDEX utilizados na disciplina de **Inteligência Artificial II (IA II)**.

10.1 Dataset Oficial da Disciplina: LeetCode Problems Dataset Bilíngue (PT-BR)

Para viabilizar o treinamento de redes neurais profundas (TP1), a indexação semântica em banco vetorial (TP2) e a curadoria autônoma com agentes (TP3), a disciplina adota o **LeetCode Problems Dataset Bilíngue**, uma base curada com **2.830 problemas públicos de programação traduzidos e auditados em Português Brasileiro (PT-BR)**.

> **i** Links Oficiais para Acesso e Download
>
> - **Relatório Exploratório Interativo (HTML do Jupyter Notebook)** *(Gráficos de distribuição, vocabulário e contagem de palavras)*
> - **Relatório Exploratório Consolidado (PDF)** *(Versão para leitura e impressão)*
> - **Download do Dataset em CSV (23.3 MB)**
> - **Download do Dataset em JSON Estruturado (25.4 MB)**
> - **Documentação Técnica & Especificação de Schemas**

10.1.1 Composição e Estatísticas do Dataset

- **Total de Registros:** 2.830 problemas gratuitos com enunciado completo.
- **Distribuição por Dificuldade:**
  - **Easy (CS1/CS2):** 752 problemas (26,6%) — ideal para conceitos introdutórios e estruturas básicas.
  - **Medium (CS2/CS3):** 1.417 problemas (50,1%) — estruturas de dados avançadas e algoritmos clássicos.
  - **Hard (CS3/Maratona):** 661 problemas (23,3%) — benchmarks avançados e otimização complexa.
- **Metodologia de Curadoria:** Tradução técnica assistida por LLM (`gpt-5.4-mini` via OpenAI Batch API com *Structured Outputs*), garantindo **98,9% de fidelidade estrutural perfeita** na preservação de tags HTML, expressões matemáticas, tabelas, código inline e variáveis originais.

10.1.2 Campos Disponíveis para os Modelos

Campo	Tipo	Aplicação nos TPs
`id`	Inteiro	Identificador unívoco do problema (`1` a `3549`).
`title_pt`	Texto	Título do problema em português.
`description_pt`	Texto (HTML/Markdown)	**Feature Textual** (X_{text}) para NLP no TP1 e **Documento de Chunk** no TP2 (RAG).
`difficulty`	Categórico (`Easy`, `Medium`, `Hard`)	Feature tabular de entrada ou baseline de classificação.
`topics`	Lista de Strings (JSON)	**Rótulos Alvo** (y) — skills e tópicos algorítmicos em português.
`hints_pt`	Lista de Textos	Dicas pedagógicas oficiais traduzidas para enriquecimento no TP2/TP3.
`acceptance_rate`	Float (0.0 a 100.0)	Métrica de taxa de aceitação (feature tabular para modelos Multi-Input).
`likes` / `dislikes`	Inteiro	Métricas de engajamento da comunidade de programadores.
`solution_code_python`	Código Python	Implementação de referência para validação de testes.

10.2 Artigo Científico Base

Miranda Junior, A.; Santos, V. F. (2026). *"Quebrando o Silêncio Pedagógico: O Modelo WOKDEX para Feedback Formativo em Juízes Online"*. SBIE 2026 — Trilha TPIE (Trabalhos e Pesquisas em Informática na Educação). CBIE 2026.

- **PDF Local do Artigo:** `artigo-sbie-wokdex.pdf`
- **Contribuições Formalizadas:** `contribuicoes-artigo-sbie.md`
- **Afiliação:** CEFET-MG (DECOM-TM) / UFMG (DCC)

10.2.1 As 4 Contribuições Científicas do Artigo

ID	Contribuição	Descrição
C1	**Schema WOKDEX**	Modelo de metadados estruturado (YAML validado por JSON Schema) que formaliza atributos pedagógicos para exercícios de programação.
C2	**Tipologia de 4 Testes**	Formalização das categorias `SAMPLE`, `FUNCTIONAL`, `MISCONCEPTION` e `PERFORMANCE`.
C3	**Corpus de Referência Golden**	Catálogo com exercícios anotados manualmente com cenários pedagógicos ricos.
C4	**Heurística de Modulação de Dica (HMD)**	Algoritmo que modula a emissão e a assertividade do feedback formativo com base na taxa de falha dos testes.

10.3 JSON Schema Oficial (`wok-problem.json`)

O arquivo `wok-problem.json` define a especificação sintática estrita (JSON Schema Draft-07) que todo artefato pedagógico do WOKDEX deve respeitar.

- **Arquivo Local:** `wok-problem.json` (28 KB)
- **URL de Produção:** `https://api.mundodocodigo.com.br/api/public/json/wok-problem.json`

10.3.1 Campos Obrigatórios (`required`)

```
id, name, slug, version, origin, description, editorial,
difficultyLevelId, timeComplexity, skills, solutions,
statements, successmsg, testScenarios
```

10.4 Os 4 Tipos de Teste (TestType)

Tipo	Visível ao Aluno?	Função Pedagógica no WOKDEX
`SAMPLE`	Sim	Testes dos exemplos do enunciado para depuração inicial do aluno.
`FUNCTIONAL`	Não	Testes que verificam a corretude funcional da solução em casos de borda normais.
`MISCONCEPTION`	Não	**Armadilha pedagógica** — testes projetados para capturar erros conceituais específicos (ex.: divisão inteira vs. ponto flutuante).
`PERFORMANCE`	Não	Testes com entradas volumosas para avaliar a complexidade assintótica de tempo e memória.

10.4.1 Subtipos de MISCONCEPTION

Subtipo	Alvo Pedagógico	Exemplo Clássico
`TYPE`	Tipo de dado inadequado (CS1)	Uso de `int` no lugar de `double` na divisão.
`FORMAT`	Formatação de saída incorreta (CS1)	Omissão de quebra de linha `\n` ou espaço sobressalente.
`STRATEGY`	Abordagem algorítmica insuficiente (CS2/CS3)	Resolução por força bruta onde se exige Programação Dinâmica.

10.5 Heurística de Modulação de Dica (HMD)

A HMD (Contribuição C4 do artigo) define **quando** o sistema deve emitir a dica pedagógica (`helpTip`):

Nível de Confiança	Condição do Teste	Ação do Juiz Online
Alta	Todos os testes do cenário falharam	Emite `helpTip` com máxima certeza do diagnóstico.
Moderada	Maioria dos testes falhou	Emite `helpTip` acompanhado de ressalva investigativa.
Baixa	Apenas 1 teste isolado falhou	Não emite dica (evita falso diagnóstico por erro de digitação pontual).

10.6 Arquivos Locais deste Diretório

Arquivo	Descrição
`wok-problem.json`	JSON Schema oficial do modelo WOKDEX (Draft-07) — 28 KB
`artigo-sbie-wokdex.pdf`	Artigo científico completo publicado no SBIE 2026 — 250 KB
`contribuicoes-artigo-sbie.md`	Detalhamento das 4 contribuições acadêmicas formais
`index.qmd`	Esta página de referência técnica e documentação