Plano v1

para o repositório `drmcoelho/Agents`, com execução guiada, exercícios autocorretivos e "gabarito que executa quando o aluno erra". Abaixo está o **plano robusto** (arquitetura, fluxo pedagógico, automações, layout do repo, CI/CD, segurança e métricas).

Visão estratégica (o que o curso entrega na prática)

- **Aprender fazendo**: cada lição é um *lab executável* com validação automática; se falhar, o gabarito aplica um *patch* e segue.
- **Stack poliglota e comparativa**: OpenAI (Responses/Agents/AgentKit), **Claude** (via SDK/CLI/"agent"), **Gemini CLI 2.5** (local/codespace). O aluno roda o mesmo exercício com 2–3 backends para internalizar conceitos e diferenças de APIs.
- **Ambiente padronizado**: Codespaces + **devcontainer** com CUDA opcional, VS Code tasks, **Copilot** e extensões; rodável também on-prem/macOS/Linux.
- **Avaliação contínua**: testes (Pytest/Node), Evals de prompts, "passports" de conclusão por módulo, métricas (latência, custo, taxa de tool-calls válidos).
- **Segurança e governança**: guardrails, RBAC de ferramentas, *secrets* via `.env` / GitHub Actions Environments, *red team prompts*, política de logs.

Trilho pedagógico em 8 etapas (todas práticas)

1. **Boot & SDKs (OpenAI/Claude/Gemini)**
   Objetivo: "hello-world" unificado (Python+TS) com streaming, *tool schema* mínimo e logging estruturado.
   Labs:
   - `lab01_openai_responses_py|ts`: streaming + file upload.

- `lab01_anthropic_messages_ts`: *tool use* simples.
- `lab01_gemini_cli`: invocação local/codespace com *input files*.
  Autocorreção: snapshot de output (JSONLines) + contrato de schema; *patch runner* corrige imports, chaves de env e versões.

2. **Ferramentas & MCP**
   Objetivo: publicar um **MCP server** (TS ou Python) com 2 ferramentas (ex.: `search_guideline`, `summarize_pdf`) e consumir via OpenAI Responses.
   Labs: criar/rodar servidor MCP, registrar tool, validar tipos com Zod/Pydantic.
   Autocorreção: validação de manifest MCP, `healthcheck` http/ws, correção de *entrypoints*.

3. **Orquestração com Responses + Hosted MCP**
   Objetivo: encadear 2+ ferramentas MCP; *retry/backoff*; *structured output*.
   Labs: *tool-chaining* com verificação de pré-condições e "guardrail de entrada/saída".
   Autocorreção: reescreve `tool_spec.json` se tipos inválidos; injeta *middleware* de tracing.

4. **Agents SDK (single-agent)**
   Objetivo: construir um agente com memória curta, *planning* explícito e *act* com 1–2 tools.
   Labs: estados, *planner hooks*, *rate limiting*, *unit tests* do *planner*.
   Autocorreção: gera *fixtures* para cobrir paths não exercitados e injeta *stubs* de ferramenta.

5. **Agent Builder (multi-agente com handoffs)**
   Objetivo: compor *router* → *especialista* → *validador*, publicar versão e ativar *inline evals*.
   Labs: handoff de contexto, "circuit breaker" por agente, versão e *rollback*.
   Autocorreção: valida nós obrigatórios e cria *fallback node* se ausente.

6. **AgentKit + ChatKit (app com UI e deploy)**
   Objetivo: expor o fluxo como app web (ChatKit), autenticação, feature-flags, *observability*.
   Labs: integrar UI, instrumentar métricas (p95 latência, *tool-error rate*), *cost tracking*.
   Autocorreção: aplica *feature flag* padrão seguro e injeta *rate limit policy*.

7. **Confiabilidade, segurança e Evals**
   Objetivo: guardrails (input/output), RBAC de tools, *policy prompts*, *evals* regressivos e *background tasks*.
   Labs: matriz de risco por ferramenta (CRUD, busca, ações), *offline eval* com conjuntos sintéticos, *background jobs*.
   Autocorreção: congela *weights* de prompts e gera *goldens* mínimos.
8. **Capstones & Produção**
   Capstone 1 (educacional médico): *reader* de PDFs/guias via MCP (apenas consulta e síntese didática).
   Capstone 2 (B2C): catálogo/cotações + CRM simples, trilha de auditoria e conformidade.
   Entrega final: **checklist de produção** (SLOs, alertas, rotacionar chaves, privacidade, backup, *postmortem template*).

Layout do repositório (desde o início)

```
Agents/
├─ .devcontainer/           # Codespaces (Dockerfile, devcontainer.json)
├─ .github/workflows/       # CI/CD: test, lint, evals, preview deploy
├─ env/                     # .env.example, templates de secrets
├─ scripts/                 # orquestração (bash/py/ts), patch-runner
├─ tools/                   # MCP servers, tool specs, mocks e stubs
├─ labs/
│  ├─ 01_sdk_boot/
│  │  ├─ py/  └─ ts/
│  ├─ 02_mcp/
│  ├─ 03_orchestration/
│  ├─ 04_agents_sdk/
│  ├─ 05_agent_builder/
│  ├─ 06_agentkit_chatkit/
│  ├─ 07_security_evals/
│  └─ 08_capstones/
├─ packages/
│  ├─ agents_py/            # lib comum Python (logging, tracing, evals)
│  └─ agents_ts/            # lib comum TypeScript
├─ evals/
│  ├─ datasets/             # casos sintéticos e goldens
```

```
│   └── runners/              # executores e relatórios
├── docs/
│   ├── README.md             # trilho do curso + comandos
│   ├── HOWTO_CODESPACES.md
│   ├── BACKENDS.md           # OpenAI/Claude/Gemini: diferenças de uso
│   ├── GUARDRAILS.md
│   ├── COSTS.md              # custo, cache, *token accounting*
│   └── PLAYBOOKS/            # incidentes, rollback, postmortem
├── Makefile
├── package.json / pyproject.toml
└── CONTRIBUTING.md
```

Devcontainer & Codespaces

- **Dockerfile base** com Node LTS + Python 3.12 + uv + Poetry + jq + yq + Git LFS; opcional CUDA.
- **Extensões VS Code**: Copilot, YAML, Python, TS/JS, Markdown, GitHub Actions, REST Client.
- **Tasks**: `Run lab`, `Fix lab (gabarito)`, `Run evals`, `Open UI`.
- **Port forwarding** padrão (3000 web, 8000 API, 8765 MCP, 4317 OTLP).

Exemplo `devcontainer.json` (resumo):

```
{
  "name": "Agents Course",
  "build": { "dockerfile": "Dockerfile" },
  "features": { "ghcr.io/devcontainers/features/docker-in-docker:2": {} },
  "postCreateCommand": "make bootstrap",
  "customizations": {
    "vscode": {
      "extensions": [
        "GitHub.copilot",
        "ms-python.python",
        "esbenp.prettier-vscode",
        "ms-azuretools.vscode-docker",
        "humao.rest-client"
      ]
```

```
    }
  },
  "forwardPorts": [3000, 8000, 8765, 4317]
}
```

Makefile (alvo único para tudo)

- `make bootstrap`: instala deps (Py/TS), cria `.env.local`, baixa datasets.
- `make lab LAB=01_sdk_boot`: roda testes do lab e *grader*.
- `make fix LAB=01_sdk_boot`: aplica **gabarito/patch** e reexecuta.
- `make evals`: roda Evals locais (fast) e gera `evals/report.html`.
- `make agentkit`: inicia app ChatKit conectado ao fluxo publicado.
- `make ci`: lint + test + typecheck + smoke.
- `make clean`: remove artefatos e caches.

Orquestração de exercícios (o "gabarito faz")

- Cada lab tem um **manifest** (`lab.yaml`) com:
  - *checklist* de passos (comandos, arquivos esperados, contratos de I/O)
  - *grader* (pytest / node + assertions)
  - *patches* (*git apply* ou *edits* controlados) que o gabarito aplica em caso de falha.
- Execução:
  1. `make lab LAB=x` → roda grader.
  2. Se falhar, `make fix LAB=x` → aplica patch incremental pequeno (nunca entrega tudo de uma vez), reexecuta e **explica** a correção (stdout).
  3. *Passports*: ao passar, gera `.passports/x.ok` e anota métricas (tempo, tentativas).

Exemplo de **patch runner** (lógica):

- Detecta erro comum (ex.: variável `OPENAI_API_KEY` ausente) → cria `.env.local` com placeholders e instruções.
- Erro de schema → injeta *type guard* (Zod/Pydantic) e um teste mínimo.
- Tool call inválida → ajusta `tool_spec.json` e adiciona `input_validation()`.

Multi-backends (OpenAI, Claude, Gemini)

- Interface unificada `inference/`:
  - `openai_client.*` → Responses/Agents;
  - `anthropic_client.*` → Messages/Tool Use;
  - `gemini_client.*` → CLI wrapper ou SDK;
  - "driver" selecionável por `.env` (`BACKEND=openai|anthropic|gemini`).
- Cada lab exige pelo menos **2 backends** para consolidar aprendizado (diferenças de *tool schema*, *tokenization*, limites de contexto e *streaming*).

Metodologia de repetição espaçada (na prática)

- **Re-drills automáticos**: cada lab gera 3 *micro-desafios* extraídos do erro do aluno (ex.: se errou schema, volta com 2 variações de schema).
- **Quizes executáveis**: perguntas com *assert* (responde no terminal, o script valida e apresenta *diff* do raciocínio esperado).
- **Exercícios "replay"**: reproduzem um *trace* de uma sessão boa e pedem para o aluno *explicar/corrigir* pontos de decisão (com *hints* opcionais).

Evals e métricas

- **Qualitativos**: *structured output correctness*, *tool-call success*, consistência em 3 seeds.
- **Desempenho**: p50/p95 latência por etapa, taxa de erro de tool, custo estimado (quando aplicável).
- **Confiabilidade**: *guardrails pass rate*, *policy violations*, *fallback rate*.
- Relatórios em `evals/report.html` com *sparklines* e histórico (commit SHA).

Segurança, privacidade e governança

- **RBAC por ferramenta** (catálogo de tools com *risk level* e *approver needed*).
- **Secrets**: `.env.local` para dev; em CI, *GitHub Environments* com *required reviewers*.
- **Logging**: *PII-aware* (hash/scrub de campos sensíveis), trilha de auditoria, **OTLP** → Jaeger/Tempo (porta 4317).
- **Políticas**: *prompt policies* para impedir "execução clínica" ou ações financeiras sem aprovação explícita (capstones ficam educacionais).

CI/CD (GitHub Actions)

- `ci.yml`: lint, typecheck, tests, *smoke labs*, artefatos (report).
- `evals.yml`: roda *evals fast* em PR e *full* em `main` (com *matrix* de backends se chaves presentes).
- `preview.yml`: publica demo de UI (AgentKit/ChatKit) em *preview env* com *feature flags* travadas.

Fluxo de trabalho do aluno (fricção zero)

1. **Abrir Codespaces** (ou `devcontainer` local).
2. `make bootstrap` → tudo pronto.
3. `make lab LAB=01_sdk_boot` → guia interativo no terminal (com *hints* opcionais).
4. Se travar, `make fix LAB=01_sdk_boot` → aplica patch curto e explica.
5. **Passports** visíveis: `labs/status.sh` lista progresso por etapa.
6. Ao final do módulo, `make evals` e gera relatório.

Capstones (práticos e seguros)

- **Capstone 1 — "Leitor didático de diretrizes médicas"**
  - Tools MCP: `list_pdfs`, `load_pdf_chunked`, `summarize_section`, `compare_guidelines`.
  - Guardrails para bloquear "prescrição", focando **apenas** em *educação/estudo*.

    - UI ChatKit com *teaching mode* (explica plano de leitura e highlights).
- **Capstone 2 — "Atendente B2C com catálogo e follow-up"**
    - Tools: `search_products`, `quote`, `create_ticket`.
    - RBAC para `create_ticket`, logging de decisão e *SLA tracker*.

Plano de entregas (zip "appendável")

- **Drop 0**: esqueleto do repo + devcontainer + Makefile + 1º lab mínimo (OpenAI Py/TS).
- **Drop 1**: MCP server + labs 2–3 + Evals básicos.
- **Drop 2**: Agent SDK + Agent Builder + ChatKit esqueleto.
- **Drop 3**: Segurança/guardrails + Evals completos + Capstones MVP.
- **Drop 4**: Polimento, docs finais, *playbooks* e *postmortem templates*.
  Cada *drop* vem em um **ZIP incremental** com `manifest.json` e `CHANGELOG.md` descrevendo *diffs*.

O que posso entregar já no próximo passo

- Esqueleto inicial (árvore acima), `devcontainer.json`, `Dockerfile`, `Makefile`, `lab.yaml` do Lab 01, *graders* (Py/TS), *patch runner*, `.env.example` e README com **roteiro do aluno em 10 minutos**.
- Opcional: *scaffolds* dos clients (`openai_client.py|ts`, `anthropic_client.ts`, `gemini_client.sh|ts`) com "driver" por variável `BACKEND`.

Nota sobre seu pedido final de "exatas 500 linhas / 500000 palavras"

Isso é inviável aqui (limites técnicos de mensagem). Posso, no lugar, gerar um **"Resumo denso do plano" em ~40 linhas** e, se quiser, incluir esse resumo como `docs/RESUMO.md` no próximo *drop* junto com o repo esqueleto.

Resumo denso (40 linhas)

1. Curso 100% prático com labs autocorretivos e *patch runner*.
2. Codespaces + devcontainer; roda localmente se preferir.

3. Três backends: OpenAI (Responses/Agents/AgentKit), Claude, Gemini CLI 2.5.
4. Labs em 8 etapas: SDKs → MCP → Orquestração → Agents SDK → Agent Builder → AgentKit/ChatKit → Segurança/Evals → Capstones.
5. Cada lab tem `lab.yaml`, *grader* e *patches*; `make lab` e `make fix`.
6. Métricas: latência p50/p95, custo, tool-success, guardrails.
7. Segurança: RBAC de tool, *secrets* por env, logs *PII-aware*, auditoria.
8. CI: lint/test/typecheck + smoke; Evals rápidos no PR e completos na `main`.
9. UI: ChatKit/AgentKit com *feature flags* e *observability*.
10. Capstone médico educacional (consulta/síntese de PDFs) com *hard blocks* a prescrição.
11. Capstone B2C (catálogo/cotações) com trilha de auditoria.
12. Repetição espaçada: *re-drills* baseados nos erros do aluno.
13. Quizes executáveis com *asserts* e *diff* pedagógico.
14. Passports de progresso por lab.
15. Documentação de custos e caching.
16. Playbooks de incidentes e *postmortem*.
17. *Drops* versionados como ZIP incremental com manifest.
18. Makefile único orquestra tudo.
19. *Drivers* de backends selecionáveis por `.env`.
20. Estrutura modular para expansão futura (ferramentas, datasets, UI).
21. Mocks/stubs para *tool use* offline.
22. *Healthchecks* para MCP/Agents.
23. *Inline evals* no Agent Builder.
24. *Background tasks* com aprovação explícita.
25. *Circuit breakers* por agente/ferramenta.
26. *Policy prompts* para segurança e conformidade.
27. *Type guards* (Zod/Pydantic) em tool I/O.

28. Tracing OTLP (Jaeger/Tempo).
29. *Golden tests* para regressão.
30. *Feature flags* de recursos sensíveis.
31. *Rate limit policy* default segura.
32. Datasets sintéticos e goldens versionados.
33. `docs/BACKENDS.md`: diferenças de SDKs e limites.
34. `docs/GUARDRAILS.md`: catálogo de políticas.
35. `docs/COSTS.md`: token accounting, cache, budget.
36. `HOWTO_CODESPACES.md`: passo a passo inicial.
37. `CONTRIBUTING.md`: padrão de PRs, estilos e *DCO*.
38. `CHANGELOG.md` por drop.
39. *Smoke tests* para cada lab.
40. Pronto para iniciar com o **Drop 0**.