Gmail **Fillipe Guerra <fillipe.backup@gmail.com>**

IA_Autonoma_Parte5

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com> 28 de outubro de 2025 às 09:43
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

APÊNDICE X — Parser Multimodal com OCR+CLIP (local)

X.0 Estrutura de arquivos

```
/server/ai/
  ├─ ocr.ts
  ├─ clip.embed.ts
  ├─ parser.multimodal.ts
  └─ routes.ingest.ext.ts        # (já tem do round anterior; manter)
/models/
  ├─ clip_vit32.onnx
  ├─ clip_bpe_vocab.json
  └─ clip_merges.txt
```

X.1 `.env` (acréscimos)

```
AION_OCR_LANG=eng+por
AION_CLIP_MODEL_PATH=./models/clip_vit32.onnx
AION_CLIP_MAXLEN=77
```

X.2 `ocr.ts` (OCR local com tesseract.js)

```
// /server/ai/ocr.ts
import Tesseract from "tesseract.js";
const OCR_LANG = process.env.AION_OCR_LANG || "eng+por";

export async function ocrBufferToText(buf: Buffer): Promise<string> {
  const r = await Tesseract.recognize(buf, OCR_LANG, { logger: () => {} });
  const txt = (r.data?.text || "").replace(/\s+/g, " ").trim();
  return txt;
}
```

X.3 `clip.embed.ts` (CLIP-ViT em ONNX local)

```
// /server/ai/clip.embed.ts
import { InferenceSession, Tensor } from "onnxruntime-node";
import sharp from "sharp";
import fs from "fs";

const MODEL_PATH = process.env.AION_CLIP_MODEL_PATH || "./models/clip_vit32.onnx";
const MAXLEN = Number(process.env.AION_CLIP_MAXLEN || 77);
let session: InferenceSession | null = null;

async function load() {
  if (!session) session = await InferenceSession.create(MODEL_PATH, { executionProviders: ["cpu"] });
  return session!;
}

// Normalização padrão CLIP (ViT-B/32)
```

```
function norm(x: number) { return x/255; }
const MEAN = [0.48145466, 0.4578275, 0.40821073];
const STD  = [0.26862954, 0.26130258, 0.27577711];

export async function embedImageCLIP(buf: Buffer): Promise<number[]> {
  const sess = await load();
  const img = sharp(buf).resize(224,224).toFormat("png"); // ViT-B/32
  const { data, info } = await img.raw().toBuffer({ resolveWithObject: true });
  const H = info.height, W = info.width, C = info.channels; // RGBA
  // Converter para RGB float32 normalizado
  const out = new Float32Array(1*3*H*W);
  for (let y=0; y<H; y++){
    for (let x=0; x<W; x++){
      const i = (y*W + x)*C;
      const r = norm(data[i+0]), g = norm(data[i+1]), b = norm(data[i+2]);
      const idx = y*W + x;
      out[0*H*W + idx] = (r - MEAN[0]) / STD[0];
      out[1*H*W + idx] = (g - MEAN[1]) / STD[1];
      out[2*H*W + idx] = (b - MEAN[2]) / STD[2];
    }
  }
  const input = new Tensor("float32", out, [1,3,H,W]);
  const res = await sess.run({ "image": input });
  // nome da saída pode variar; padronize conforme o modelo exportado
  const feat = (res["emb"] || res["pooled_output"] || res[Object.keys(res)[0]]) as Tensor;
  const v = Array.from(feat.data as Float32Array);
  // normaliza L2
  const n = Math.sqrt(v.reduce((a,b)=>a+b*b,0)) || 1;
  return v.map(x=>x/n);
}
```

Observação: CLIP local fornece embedding visual consistente com o espaço textual e fundamenta a fusão multimodal (tokenização/encoders) discutida na tua Parte III-C-1.

IA_Autonoma_Parte2

X.4 `parser.multimodal.ts` (integra imagem+OCR ao pipeline de ingest)

```
// /server/ai/parser.multimodal.ts
import fs from "fs";
import sharp from "sharp";
import { embedImageCLIP } from "./clip.embed";
import { ocrBufferToText } from "./ocr";

export type ImgParseOut = {
  rgba?: Uint8ClampedArray;
  width?: number;
  height?: number;
  ocrText?: string;
  clipVec?: number[];
  meta?: any;
};

export async function parseImage(buf: Buffer, doOCR=true): Promise<ImgParseOut> {
  const s = sharp(buf).ensureAlpha();
  const { data, info } = await s.raw().toBuffer({ resolveWithObject: true });
  const rgba = new Uint8ClampedArray(data.buffer, data.byteOffset, data.byteLength);
  const [clipVec, ocrText] = await Promise.all([
    embedImageCLIP(buf),
    doOCR ? ocrBufferToText(buf) : Promise.resolve("")
  ]);
  return {
    rgba, width: info.width, height: info.height,
    ocrText: (ocrText||"").trim(),
    clipVec, meta: { type: "image", w: info.width, h: info.height, sourceRank: 0.6 }
  };
}
```

Isso “fecha” a ingest multimodal: além de texto (HTML/PDF/DOCX/PPTX/CSV) você terá **imagem→embedding CLIP** e **imagem→texto (OCR)** locais. A atenção e custo ficam coerentes com o que você formalizou (FlashAttention/kernel/Nyström), escolhendo “rota” conforme regime de T e hardware.

IA_Autonoma_Parte4

APÊNDICE Y — Treinador incremental (Embeddings + LoRA do gerador local)

Objetivo: a AION **aprende sozinha** com:

- (a) **SFT/LoRA** periódico no gerador **local** (Rota A com runner externo),
- (b) **Refino incremental** de **embeddings** (média móvel/EMA) sobre novos dados e feedbacks.

Y.0 Estrutura

```
/trainer/
  ├─ sft/
  │   ├─ build_jsonl.ts
  │   ├─ train_lora.py
  │   └─ promote_adapter.ts
  └─ emb/
      └─ update_ema.ts
/server/ai/
  ├─ routes.train.ts
  └─ llm.ts              # driver híbrido (já te passei)
```

Y.1 Matemática (SFT + LoRA)

Y.1.1 Objetivo SFT (LM causal)

LLM(θ)=−N1t=1∑Nlogpθ(wt|w<t),

com **LoRA**: W≈W0+ΔW, ΔW=BA, onde A∈Rr×k,B∈Rd×r e r≪min(d,k). Gradiente restringe-se a A,B; os pesos base W0 ficam **congelados**. (vide Parte I/III-B)

IA_Autonoma_Parte1

IA_Autonoma_Parte3_4

Y.1.2 PPO (opcional)

Mantemos seus termos KL e *clip ratio* para refino comportamental, mas no apêndice enviamos SFT como **primeiro passo** (simples, barato), compatível com tua Parte III-B.

IA_Autonoma_Parte3_4

Y.1.3 Embeddings EMA incremental

Para vetor ei(t), com alvo e¯i (média de novos exemplos):

ei(t+1)=(1−ηt)ei(t)+ηte¯i,ηt=min{1,ni+tc}.

Conforme tua formalização de atualização estável (memória vetorial), garantimos convergência por passo decrescente.

IA_Autonoma_Parte2

Y.2 `build_jsonl.ts` (extrai dataset do DB)

```
// /trainer/sft/build_jsonl.ts
import fs from "fs";
import path from "path";
import { db } from "../../server/db";
import { aiInteractions } from "../../shared/schema.ai.core";

// Gera dataset SFT no formato OpenAI/Alpaca-like: {"instruction","input","output"} ou chat.
async function run(outPath = "./data/sft/aion_sft.jsonl") {
  const rows = await db.select().from(aiInteractions).limit(50000);
  const lines: string[] = [];
  for (const r of rows) {
    // r: { user_input, final_answer, context_citations, rating, ... }
    const obj = {
      messages: [
        { role: "system", content: "Você é a AION, IA autônoma da plataforma." },
        { role: "user", content: r.user_input },
        { role: "assistant", content: r.final_answer }
      ],
      meta: { rating: r.rating || null, citations: r.context_citations || [] }
    };
    lines.push(JSON.stringify(obj));
  }
  fs.mkdirSync(path.dirname(outPath), { recursive: true });
  fs.writeFileSync(outPath, lines.join("\n"), "utf8");
  console.log("SFT JSONL salvo em", outPath, "(", lines.length, "exemplos )");
}

run().catch(e=>{ console.error(e); process.exit(1); });
```

Y.3 `train_lora.py` (treino LoRA local; usa `transformers`+`peft`)

```
# /trainer/sft/train_lora.py
import os, json
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForCausalLM, TrainingArguments, Trainer,
DataCollatorForLanguageModeling
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training

MODEL_NAME = os.environ.get("AION_BASE_CKPT", "./models/base-llm")  # checkpoint local
DATA_JSONL = os.environ.get("AION_SFT_JSONL", "./data/sft/aion_sft.jsonl")
OUT_DIR    = os.environ.get("AION_LORA_OUT", "./models/adapters/aion-lora")
RANK       = int(os.environ.get("AION_LORA_RANK", "16"))
ALPHA      = int(os.environ.get("AION_LORA_ALPHA", "32"))
DROPOUT    = float(os.environ.get("AION_LORA_DROPOUT", "0.05"))

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, use_fast=True)
def format_example(ex):
    # dataset no formato {"messages":[...]}
    msgs = ex["messages"]
    text = ""
    for m in msgs:
        text += f"<|{m['role']}|>: {m['content']}\n"
    text += "<|end|>\n"
    return {"text": text}

ds = load_dataset("json", data_files=DATA_JSONL, split="train")
ds = ds.map(format_example, remove_columns=ds.column_names)

model = AutoModelForCausalLM.from_pretrained(MODEL_NAME, trust_remote_code=True)
model = prepare_model_for_kbit_training(model)  # se quantizado
peft_cfg = LoraConfig(r=RANK, lora_alpha=ALPHA, lora_dropout=DROPOUT,
                      target_modules=["q_proj","v_proj","k_proj","o_proj","gate_
proj","up_proj","down_proj"])
model = get_peft_model(model, peft_cfg)

collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)

args = TrainingArguments(
    output_dir=OUT_DIR, per_device_train_batch_size=2, gradient_accumulation_steps=16,
```

```
    learning_rate=2e-4, weight_decay=0.0, num_train_epochs=1, fp16=True,
    logging_steps=50, save_steps=500, save_total_limit=2
)

def tokenize(batch):
    return tokenizer(batch["text"], truncation=True, max_length=2048)

tok = ds.map(tokenize, batched=True, remove_columns=["text"])
trainer = Trainer(model=model, args=args, train_dataset=tok, data_collator=collator)
trainer.train()
model.save_pretrained(OUT_DIR)
tokenizer.save_pretrained(OUT_DIR)
print("Adapter salvo em", OUT_DIR)
```

Y.4 `promote_adapter.ts` (ativa novo adapter no runner local)

```
// /trainer/sft/promote_adapter.ts
import fs from "fs";
const NEW = process.argv[2] || "./models/adapters/aion-lora";
const CUR = "./models/adapters/aion-current";

fs.rmSync(CUR, { recursive: true, force: true });
fs.cpSync(NEW, CUR, { recursive: true });
console.log("Adapter promovido:", NEW, "→", CUR);
```

Y.5 Atualizações no servidor (rotas de treino)

```
// /server/ai/routes.train.ts
import type { Express } from "express";
import { exec } from "child_process";

export function registerTrainRoutes(app: Express) {
  app.post("/api/ai/train/sft.start", (req,res)=>{
    // 1) exporta dataset
    exec("pnpm ts-node ./trainer/sft/build_jsonl.ts", (e,o,er)=>{
      if (e) return res.status(500).json({ error: er || e.message });
      // 2) roda treino (ideal: job async/queue; aqui síncrono simples)
      exec("python3 ./trainer/sft/train_lora.py", (e2,o2,er2)=>{
        if (e2) return res.status(500).json({ error: er2 || e2.message });
        return res.json({ ok:true, log:o2 });
      });
    });
  });
  app.post("/api/ai/train/sft.promote", (req,res)=>{
    exec("pnpm ts-node ./trainer/sft/promote_adapter.ts", (e,o,er)=>{
      if (e) return res.status(500).json({ error: er || e.message });
      res.json({ ok:true });
    });
  });
}
```

Observação: isto segue a linha do teu **deploy modular** (Model Server/Agent/RAG) e é coerente com a tua arquitetura de execução (III-D).

IA_Autonoma_Parte3_2

Y.6 Incremental de Embeddings (EMA)

```
// /trainer/emb/update_ema.ts
import { db } from "../../server/db";
import { aiChunks } from "../../shared/schema.ai.core";
const C = 0.5; // constante da taxa

// exemplo: recalcula vetor médio de um doc a partir dos chunks recém-aprovados
export async function updateEMA(docId: string) {
  const rows = await db.select().from(aiChunks).where((aiChunks.documentId as any).eq(docId));
  const news = rows.filter(r => r.is_new);
  if (!rows.length || !news.length) return;
```

```
  const d = rows[0].vector.length;
  const e_old = rows[0].doc_vector || new Array(d).fill(0);
  const e_bar = new Array(d).fill(0);
  for (const r of news) for (let i=0;i<d;i++) e_bar[i]+=r.vector[i];
  for (let i=0;i<d;i++) e_bar[i]/=news.length;

  const n_i = Number(rows[0].doc_updates || 0);
  const eta = Math.min(1, C/(n_i+1));
  const e_new = e_old.map((x,i)=> (1-eta)*x + eta*e_bar[i]);

  // salvar de volta
  // (ajuste para teu schema real)
  await db.execute(`update ai_documents set doc_vector=$1, doc_updates=$2 where id=$3`,
[JSON.stringify(e_new), n_i+1, docId]);
}
```

APÊNDICE Z — Curadoria Web & Auto-alimentação (busca, pré-visualização, aprovação)

> Complementa o painel que já te entreguei: agora com **descoberta assistida**, **filtros éticos/legais** e **fila de revisão** — coerente com tua separação de núcleo vs. políticas (Teorema de Separação).
>
> IA_Autonoma_Parte3_1

Z.0 Estrutura

```
/server/ai/
  ├─ curator/
  │   ├─ crawler.ts
  │   ├─ filters.policy.ts
  │   └─ curator.routes.ts
/ui/pages/admin/knowledge.tsx   # (vamos ampliar)
```

Z.1 `.env` (novas flags)

```
AION_CRAWL_MAX_PAGES=50
AION_ALLOW_ROBOTS=true
AION_BLOCKLIST_DOMAINS=facebook.com,twitter.com
```

Z.2 `crawler.ts` (descoberta básica com respeito a robots)

```
// /server/ai/curator/crawler.ts
import fetch from "node-fetch";
import { JSDOM } from "jsdom";

const MAXP = Number(process.env.AION_CRAWL_MAX_PAGES || 50);
const ALLOW_ROBOTS = process.env.AION_ALLOW_ROBOTS !== "false";
const BLOCK = new Set((process.env.AION_BLOCKLIST_DOMAINS||"").split(",").filter(Boolean));

function sameHost(a:string,b:string){ try{ return new URL(a).host===new URL(b).host; }catch{ return
false; } }

export async function crawlSeed(seed: string){
  const seen = new Set<string>(); const queue = [seed]; const out: string[] = [];
  while (queue.length && out.length < MAXP) {
    const u = queue.shift()!; if (seen.has(u)) continue; seen.add(u);
    const host = new URL(u).host;
    if ([...BLOCK].some(d=>host.endsWith(d))) continue;

    // robots.txt (simplificado)
    if (ALLOW_ROBOTS) {
      try {
```

```
        const r = await fetch(`${new URL(u).origin}/robots.txt`);
        const txt = r.ok ? await r.text() : "";
        if (/Disallow:\s*\/\s*$/i.test(txt)) continue;
      } catch {}
    }

    const r = await fetch(u); if (!r.ok) continue;
    const html = await r.text();
    out.push(u);

    // extrai links internos
    try {
      const dom = new JSDOM(html);
      const links = [...dom.window.document.querySelectorAll("a[href]")].map(a => (a as
HTMLAnchorElement).href);
      for (const l of links) {
        try {
          const abs = new URL(l, u).toString();
          if (sameHost(abs, seed) && !seen.has(abs)) queue.push(abs);
        } catch {}
      }
    } catch {}
  }
  return out;
}
```

Z.3 `filters.policy.ts` (checagens simples antes da ingest)

```
// /server/ai/curator/filters.policy.ts
export type PolicyCheck = { ok: boolean; reason?: string };

export function basicPolicy(url:string, textPreview:string): PolicyCheck {
  // Ex.: bloquear PII óbvia, pages de login, termos etc.
  if (/login|signin|cart|checkout/i.test(url)) return { ok:false, reason:"rota privada" };
  if (textPreview.length < 120) return { ok:false, reason:"conteúdo insuficiente" };
  return { ok:true };
}
```

Z.4 `curator.routes.ts` (descobrir → pré-visualizar → aprovar → ingerir)

```
// /server/ai/curator/curator.routes.ts
import type { Express } from "express";
import { crawlSeed } from "./crawler";
import { parseURL } from "../parser.full";
import { basicPolicy } from "./filters.policy";
import { ingestText } from "../ingest";

export function registerCuratorRoutes(app: Express) {
  app.post("/api/ai/curator/discover", async (req,res)=>{
    const { seed } = req.body||{};
    if (!seed) return res.status(400).json({ error:"seed required" });
    const urls = await crawlSeed(seed);
    res.json({ urls });
  });

  app.post("/api/ai/curator/preview", async (req,res)=>{
    const { urls=[] } = req.body||{};
    const out:any[] = [];
    for (const u of urls.slice(0,200)) {
      try{
        const { text, title, meta } = await parseURL(u);
        const pol = basicPolicy(u, text.slice(0,800));
        out.push({ url:u, ok:pol.ok, reason:pol.reason, title, excerpt:text.slice(0,800), meta });
      } catch(e:any){ out.push({ url:u, ok:false, error:e?.message }); }
    }
    res.json({ previews: out });
  });
```

```
  app.post("/api/ai/curator/ingest", async (req,res)=>{
    const { items=[] } = req.body||{};
    const results:any[] = [];
    for (const it of items) {
      if (!it.approved) continue;
      const { text, title, meta } = await parseURL(it.url);
      const r = await ingestText(process.env.PRIMARY_TENANT_ID!, { source:"url", uri:it.url, title,
text, meta });
      results.push({ url:it.url, ok:true, r });
    }
    res.json({ results });
  });
}
```

Z.5 UI — ampliar `/admin/knowledge` com **Discover**

```
// /ui/pages/admin/knowledge.tsx (ADICIONAR seção Discover)
import { useState } from "react";

export default function KnowledgePage(){
  // ... (parte anterior permanece)
  const [seed,setSeed] = useState("");
  const [found,setFound] = useState<string[]>([]);

  async function discover(){
    const r = await fetch("/api/ai/curator/discover", { method:"POST", headers:{ "Content-
Type":"application/json" }, body: JSON.stringify({ seed }) });
    const j = await r.json(); setFound(j.urls||[]);
  }
  async function previewFound(){
    const r = await fetch("/api/ai/curator/preview", { method:"POST", headers:{ "Content-
Type":"application/json" }, body: JSON.stringify({ urls: found }) });
    const j = await r.json(); setPreviews(j.previews||[]);
  }

  return (
    <div className="p-6 space-y-6">
      {/* bloco novo */}
      <div className="border p-4 rounded space-y-2">
        <h2 className="font-semibold">Discover (semente → crawl interno)</h2>
        <div className="flex gap-2">
          <input className="flex-1 border p-2" placeholder="https://exemplo.com" value={seed} onChange=
{e=>setSeed(e.target.value)} />
          <button className="px-3 py-2 bg-indigo-600 text-white rounded" onClick=
{discover}>Descobrir</button>
          <button className="px-3 py-2 bg-slate-700 text-white rounded" onClick={previewFound}>Pré-
visualizar</button>
        </div>
        {!!found.length && <p className="text-xs text-gray-500">Encontrados: {found.length} URLs</p>}
      </div>

      {/* resto da página (prévia/ingest) permanece */}
    </div>
  );
}
```

APÊNDICE W — **Budget e Fallback Controller** (opcional, mas recomendado)

W.1 `.env`

```
USE_OPENAI=false
OPENAI_API_KEY=
OPENAI_PROJECT=
```

```
OPENAI_MAX_DAILY_USD=1.50
OPENAI_MAX_TOKENS_PER_REPLY=400
AION_CONFIDENCE_THRESHOLD=0.62
AION_MAX_FALLBACKS_PER_HOUR=15
```

W.2 Middleware simples de orçamento

```
// /server/ai/budget.ts
let spentUSD = 0, lastDay = new Date().toDateString();
let fallbackCount = 0, windowStart = Date.now();

export function canFallback(costEstUSD: number) {
  const today = new Date().toDateString();
  if (today !== lastDay) { spentUSD = 0; lastDay = today; }
  const MAX = Number(process.env.OPENAI_MAX_DAILY_USD || 0);
  if (MAX <= 0) return false;
  return spentUSD + costEstUSD <= MAX;
}
export function registerSpend(cost: number) { spentUSD += cost; }
export function rateLimitOk() {
  const MAXH = Number(process.env.AION_MAX_FALLBACKS_PER_HOUR || 0);
  if (!MAXH) return true;
  const now = Date.now();
  if (now - windowStart > 3600_000) { fallbackCount = 0; windowStart = now; }
  return (++fallbackCount) <= MAXH;
}
```

Use no teu `llm.ts` para só cair no modelo externo quando **confiar** e **caber no budget** (compatível com teu plano de "autonomia crescente").

IA_Autonoma_Parte3_3

IA_Autonoma_Parte2

INTEGRAÇÃO FINAL

1. **Instalar deps** (alem das anteriores):

```
pnpm add onnxruntime-node sharp tesseract.js jsdom
pnpm add -D ts-node
pip3 install transformers peft datasets accelerate --upgrade
```

2. **Modelos locais**

- `models/clip_vit32.onnx` (ou outro CLIP-ViT ONNX).
- Gerador local (checkpoint em `./models/base-llm`) que o teu runner carrega com **adapter atual** `./models/adapters/aion-current/`.

3. **Registrar rotas** no bootstrap:

```
import { registerCuratorRoutes } from "./ai/curator/curator.routes";
import { registerTrainRoutes } from "./ai/routes.train";
// ...
registerCuratorRoutes(app);
registerTrainRoutes(app);
```

4. **Fluxo de autonomia**:

- **Discover** → **Preview** → **Approve** → **Ingest** (texto+imagem+OCR),
- **EMA embeddings** periódica,
- **SFT/LoRA** noturna (`POST /api/ai/train/sft.start` → `.../sft.promote`),
- **Driver híbrido** usa **gerador local**; **fallback** só se `score<τ` **e** `budget ok`.

Testes rápidos

Ingest OCR/CLIP (imagem):

```
curl -F "file=@./exemplo.png" http://localhost:3000/api/ai/ingest.file
```

Discover + preview + ingest:

```
curl -X POST -H "Content-Type: application/json" \
  -d '{"seed":"https://www.yesyoudeserve.tours"}' \
  http://localhost:3000/api/ai/curator/discover

# use as URLs retornadas:
curl -X POST -H "Content-Type: application/json" \
  -d '{"urls":["https://www.yesyoudeserve.tours/..."]}' \
  http://localhost:3000/api/ai/curator/preview

# aprove e ingerir:
curl -X POST -H "Content-Type: application/json" \
  -d '{"items":[{"url":"https://...","approved":true}]}' \
  http://localhost:3000/api/ai/curator/ingest
```

Treino SFT/LoRA (local):

```
pnpm ts-node ./trainer/sft/build_jsonl.ts
AION_BASE_CKPT=./models/base-llm \
AION_SFT_JSONL=./data/sft/aion_sft.jsonl \
AION_LORA_OUT=./models/adapters/aion-lora \
python3 ./trainer/sft/train_lora.py

pnpm ts-node ./trainer/sft/promote_adapter.ts
```

Fechamento

- Com estes 3 módulos (OCR+CLIP local, **treinador incremental LoRA/EMA**, e **curadoria com crawler controlado**), a AION **passa a operar 100% offline-first** e **autoaprende** com os teus dados e interações — alinhado à tua teoria de **atenção eficiente/FlashAttn, kernelização e MoE estável** para escalar sem custo explosivo.

  IA_Autonoma_Parte4

  IA_Autonoma_Parte3_4

- O **fallback OpenAI** fica **opcional** e totalmente **orçamentado**.