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

IA_Autonoma_Parte6

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com> 28 de outubro de 2025 às 09:46
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

A AION continua **100% autônoma** (busca/entende/responde localmente), **e** você pode deixar o **ChatGPT apenas como fallback automático** — ele só entra quando:

- a confiança do resultado local ficar **abaixo de um limiar** (ex.: `τ = 0.62`),
- **e** houver **orçamento** disponível,
- **e** o *rate limit* de fallbacks não estourou.

1) `.env` (liga/desliga e controla o fallback)

```
# Geração local vs. fallback
USE_OPENAI=true                    # true = permite fallback; false = nunca chama
OPENAI_API_KEY=sk-...              # Project API key (não use Admin key)
OPENAI_PROJECT=YYD-Prod            # opcional

# Orçamento e limites (guarda-chuva adicional ao pré-pago do billing)
OPENAI_MAX_DAILY_USD=1.50          # teto diário (estimado por requisição)
OPENAI_MAX_TOKENS_PER_REPLY=400    # por resposta do fallback
AION_MAX_FALLBACKS_PER_HOUR=15     # rate-limit de fallbacks/hora

# Política de confiança (quando cair abaixo, podemos acionar fallback)
AION_CONFIDENCE_THRESHOLD=0.62
```

---

2) Driver híbrido (local + fallback)

Se já colou `server/ai/llm.ts` e `server/ai/budget.ts` dos drops anteriores, **mantenha-os**.
Senão, aqui vai tudo num arquivo só.

`server/ai/llm.ts`

```
import fetch from "node-fetch";
import { canFallback, registerSpend, rateLimitOk } from "./budget";

type GenOpts = { maxTokens?: number; temperature?: number; top_p?: number; stop?: string[] };
type GenOut = { text: string };

const USE_OPENAI = process.env.USE_OPENAI === "true";
const LOCAL_LLM_ENDPOINT = process.env.LOCAL_LLM_ENDPOINT; // ex: http://localhost:8080/generate
const MAXTOK = Number(process.env.OPENAI_MAX_TOKENS_PER_REPLY || 400);

// —— Local
export async function generateLocal(prompt: string, opts: GenOpts = {}): Promise<GenOut> {
  if (!LOCAL_LLM_ENDPOINT) throw new Error("LOCAL_LLM_ENDPOINT ausente");
  const r = await fetch(LOCAL_LLM_ENDPOINT, {
    method: "POST", headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      prompt,
      max_tokens: opts.maxTokens ?? 512,
      temperature: opts.temperature ?? 0.7,
      top_p: opts.top_p ?? 0.9,
```

```
      stop: opts.stop ?? []
    })
  });
  if (!r.ok) throw new Error(`Local LLM HTTP ${r.status}`);
  const j = await r.json();
  return { text: j.text || j.choices?.[0]?.text || "" };
}

// ——— Fallback (OpenAI) sob budget
export async function generateFallback(prompt: string, opts: GenOpts = {}): Promise<GenOut> {
  if (!USE_OPENAI) throw new Error("Fallback desativado (USE_OPENAI=false)");
  if (!rateLimitOk()) throw new Error("Rate limit de fallbacks/hora atingido");
  // Estimativa simples de custo (conservadora). Ajuste conforme seu uso real.
  const estUsd = ( (prompt.length/4 + (opts.maxTokens ?? MAXTOK)) / 1000 ) * 0.002; // ex.: gpt-4o-
mini ~ $0.002/1k
  if (!canFallback(estUsd)) throw new Error("Teto diário de custo atingido");

  const key = process.env.OPENAI_API_KEY!;
  const project = process.env.OPENAI_PROJECT;
  const r = await fetch("https://api.openai.com/v1/chat/completions", {
    method: "POST",
    headers: {
      "Content-Type":"application/json",
      "Authorization":`Bearer ${key}`,
      ...(project ? {"OpenAI-Project": project} : {})
    },
    body: JSON.stringify({
      model: "gpt-4o-mini",
      messages: [{ role: "system", content: "Você é a AION. Responda SOMENTE com base no contexto. Se
faltar evidência, diga: 'Não tenho dados suficientes'." },
                 { role: "user", content: prompt }],
      max_tokens: Math.min(opts.maxTokens ?? MAXTOK, MAXTOK),
      temperature: opts.temperature ?? 0.7,
      top_p: opts.top_p ?? 0.9,
      stop: opts.stop ?? []
    })
  });
  if (!r.ok) throw new Error(`OpenAI HTTP ${r.status}`);
  const j = await r.json();
  registerSpend(estUsd);
  return { text: j.choices?.[0]?.message?.content || "" };
}
```

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; }
  fallbackCount++;
  return fallbackCount <= MAXH;
}
```

---

3) Orquestrador de resposta com **confiança + fallback**

`server/ai/answer.ts`

Este arquivo une:
(1) **retrieval híbrido** local,
(2) **medidor de confiança**,
(3) decisão **usar local** vs **fallback automático**.

```
import { hybridRetrieveFull } from "./hybrid-score";
import { generateLocal, generateFallback } from "./llm";

const TAU = Number(process.env.AION_CONFIDENCE_THRESHOLD || 0.62);

type QAOptions = { k?: number };

function confidenceFromParts(parts: { text:number; image:number; graph:number; fresh:number;
auth:number }) {
  // Heurística calibrável (0..1). Pese mais o texto e o grafo.
  const w = { t:0.55, i:0.05, g:0.25, f:0.10, a:0.05 };
  const s = w.t*parts.text + w.i*parts.image + w.g*parts.graph + w.f*parts.fresh + w.a*parts.auth;
  return Math.max(0, Math.min(1, s));
}

function mkPrompt(contexts: {text?:string; meta?:any}[], question: string) {
  const ctx = contexts.map((c,i)=>`[${i+1}] ${ (c.text||"").slice(0,1200) }`).join("\n\n");
  return `Contexto (usar APENAS isto, cite indices [n] quando possível):\n${ctx}\n\nPergunta:
${question}\nResposta (objetiva, fundamentada, sem inventar; se faltar evidência, diga isso
claramente):`;
}

export async function answerQuestion(tenantId: string, query: string, opts: QAOptions = {}) {
  const k = opts.k ?? 8;

  // 1) Busca local (sempre)
  const hits = await hybridRetrieveFull(tenantId, query, undefined, Math.max(k, 12));
  const contexts = hits.map(h => ({
    text: h.chunk?.text || "",
    meta: h.chunk?.meta_json ? JSON.parse(h.chunk.meta_json) : {}
  }));

  // 2) Confiança agregada (máximo entre top-k)
  const conf = Math.max(...hits.slice(0,k).map(h => confidenceFromParts(h.parts)));

  // 3) Geração local primeiro (barata/offline)
  const localText = await generateLocal(mkPrompt(contexts, query), { maxTokens: 420 });

  // 4) Se a confiança estiver alta, fica com local; senão tenta fallback sob budget
  if (conf >= TAU) {
    return { mode: "local", confidence: conf, text: localText.text, citations:
hits.slice(0,k).map((h,i)=>({rank:i+1, doc:h.chunk?.document_id, score:h.score})) };
  }

  try {
    const fb = await generateFallback(mkPrompt(contexts, query), { maxTokens: 420 });
    return { mode: "fallback", confidence: conf, text: fb.text, citations: hits.slice(0,k).map((h,i)=>
({rank:i+1, doc:h.chunk?.document_id, score:h.score})) };
  } catch (e:any) {
    // Sem budget / erro de rede → retorna local mesmo (degradado, mas consistente)
    return { mode: "local-degraded", confidence: conf, text: localText.text, note: e?.message,
citations: hits.slice(0,k).map((h,i)=>({rank:i+1, doc:h.chunk?.document_id, score:h.score})) };
  }
}
```

Rota HTTP para responder perguntas

server/ai/routes.answer.ts

```
import type { Express, Request } from "express";
import { answerQuestion } from "./answer";

function ctx(req: Request){
  return (req as any).tenantId || process.env.PRIMARY_TENANT_ID!;
}

export function registerAnswerRoute(app: Express){
  app.post("/api/ai/answer", async (req,res)=>{
    try{
      const tenantId = ctx(req);
      const { query, k } = req.body || {};
      if (!query) return res.status(400).json({ error:"query required" });
      const r = await answerQuestion(tenantId, query, { k });
      res.json(r);
    } catch(e:any){ res.status(400).json({ error: e?.message }); }
  });
}
```

No bootstrap (`server/routes.ts`), adicione:

```
import { registerAnswerRoute } from "./ai/routes.answer";
// ...
registerAnswerRoute(app);
```

---

4) Como isso funciona (resumo operacional)

1. **AION faz tudo localmente**: parsers, OCR/CLIP, embeddings, HNSW, grafo, scorer α/β/γ/δ/ζ → **recupera** os trechos certos.
2. **Orquestrador** estima uma **confiança** do resultado (pondera texto/grafo/frescor/autoridade).
3. **Geração local** formata a resposta com base **apenas** no contexto recuperado.
4. Se **confiança ≥ τ**, devolve a resposta local.
5. Se **confiança < τ** e **há orçamento e limite**, **chama ChatGPT** (fallback) para **melhorar apenas a redação**, **sem inventar**, usando **o mesmo contexto**.
6. Se o fallback falhar (sem saldo/erro), devolve a versão local (“local-degraded”).

Você pode **ajustar τ** no `.env` (`AION_CONFIDENCE_THRESHOLD`) para tornar o sistema **mais ou menos “gastão”**. Com o tempo — conforme a KB cresce e o SFT/LoRA local melhora — a confiança média sobe e o fallback **quase não é usado**.

---

5) Teste rápido (curl)

```
# Local-only (com fallback habilitado por env, mas só aciona se precisar)
curl -s -X POST http://localhost:3000/api/ai/answer \
  -H "Content-Type: application/json" \
  -d '{"query":"Explique nossos passeios em Sintra e diferenças entre full-day e half-day."}' | jq
```

Retorno típico:

```
{
  "mode": "local" | "fallback" | "local-degraded",
  "confidence": 0.78,
  "text": "…resposta…",
  "citations": [{ "rank":1, "doc":"…", "score":0.91 }, ...]
}
```

Fechou!

- **Sim**, sua IA **pode** usar o ChatGPT **mesmo já tendo a knowledge base**, mas **somente como fallback automático**, orquestrado por **confiança + orçamento + rate limit**.
- O padrão recomendado: **USE_OPENAI=true**, **τ=0.62**, **budget pequeno** (ex.: US$1.50/dia).
- Se quiser zero dependência, basta **`USE_OPENAI=false`** — todo o fluxo continua funcionando **100% local**.

🛠️ Dashboard: tela para configurar fallback e limites

Abaixo vai um **drop completo** (curto) para habilitar no admin:

1) Tabela de configurações (Single-Tenant)

```
/shared/schema.ai.settings.ts

import { pgTable, varchar, boolean, real, integer, timestamp } from "drizzle-orm/pg-core";

export const aiSettings = pgTable("ai_settings", {
  tenantId: varchar("tenant_id", { length: 64 }).primaryKey(),
  useOpenAI: boolean("use_openai").default(false).notNull(),
  confidenceTau: real("confidence_tau").default(0.62).notNull(),
  maxDailyUSD: real("max_daily_usd").default(1.50).notNull(),
  maxTokensPerReply: integer("max_tokens_per_reply").default(400).notNull(),
  maxFallbacksPerHour: integer("max_fallbacks_per_hour").default(15).notNull(),
  updatedAt: timestamp("updated_at").defaultNow().notNull()
});
```

Migração: crie a tabela e insira o registro do `PRIMARY_TENANT_ID`.

---

2) Rotas de leitura/alteração

```
/server/ai/routes.settings.ts

import type { Express, Request } from "express";
import { db } from "../db";
import { aiSettings } from "@shared/schema.ai.settings";
import { eq } from "drizzle-orm";

function tenant(req: Request){ return (req as any).tenantId || process.env.PRIMARY_TENANT_ID!; }

export function registerAiSettingsRoutes(app: Express){
  app.get("/api/ai/settings", async (req, res)=>{
    const t = tenant(req);
    const rows = await db.select().from(aiSettings).where(eq(aiSettings.tenantId, t));
    res.json(rows[0] || null);
  });

  app.post("/api/ai/settings", async (req, res)=>{
    const t = tenant(req);
    const body = req.body || {};
    // saneamento simples
    const s = {
      useOpenAI: !!body.useOpenAI,
      confidenceTau: Math.max(0, Math.min(1, Number(body.confidenceTau ?? 0.62))),
      maxDailyUSD: Math.max(0, Number(body.maxDailyUSD ?? 1.5)),
      maxTokensPerReply: Math.max(1, Number(body.maxTokensPerReply ?? 400)),
      maxFallbacksPerHour: Math.max(0, Number(body.maxFallbacksPerHour ?? 15)),
      updatedAt: new Date()
    };
    await db.insert(aiSettings).values({ tenantId: t, ...s })
      .onConflictDoUpdate({ target: aiSettings.tenantId, set: s });
    res.json({ ok:true });
```

```
  });
}
```

Registre no bootstrap:

```
import { registerAiSettingsRoutes } from "./ai/routes.settings";
registerAiSettingsRoutes(app);
```

---

3) O driver passa a ler do DB (em vez de só .env)

`/server/ai/settings.ts`

```
import { db } from "../db";
import { aiSettings } from "@shared/schema.ai.settings";
import { eq } from "drizzle-orm";

export type AionCfg = {
  useOpenAI: boolean;
  tau: number;
  maxUSD: number;
  maxTok: number;
  maxFbh: number;
};

export async function loadAionCfg(tenantId: string): Promise<AionCfg> {
  const r = await db.select().from(aiSettings).where(eq(aiSettings.tenantId, tenantId));
  const row = r[0];
  return {
    useOpenAI: row?.useOpenAI ?? (process.env.USE_OPENAI === "true"),
    tau: row?.confidenceTau ?? Number(process.env.AION_CONFIDENCE_THRESHOLD || 0.62),
    maxUSD: row?.maxDailyUSD ?? Number(process.env.OPENAI_MAX_DAILY_USD || 0),
    maxTok: row?.maxTokensPerReply ?? Number(process.env.OPENAI_MAX_TOKENS_PER_REPLY || 400),
    maxFbh: row?.maxFallbacksPerHour ?? Number(process.env.AION_MAX_FALLBACKS_PER_HOUR || 15)
  };
}
```

Aplique no `answer.ts` e `llm.ts`:

- em `answer.ts`, troque `const TAU = ...` por `const { tau } = await loadAionCfg(tenantId);`
- em `llm.ts`, antes do fallback: `const cfg = await loadAionCfg(tenantId);` e use `cfg.useOpenAI`, `cfg.maxTok` etc.
- no `budget.ts`, receba limites por parâmetro (ou chame `loadAionCfg`).

---

4) Página no Admin

`/ui/pages/admin/ai-settings.tsx`

```
import { useEffect, useState } from "react";

type S = { useOpenAI:boolean; confidenceTau:number; maxDailyUSD:number; maxTokensPerReply:number;
maxFallbacksPerHour:number };

export default function AiSettingsPage(){
  const [s,setS] = useState<S>({ useOpenAI:false, confidenceTau:0.62, maxDailyUSD:1.5,
maxTokensPerReply:400, maxFallbacksPerHour:15 });
  const [ok,setOk] = useState<string>("");

  useEffect(()=>{ (async ()=>{
    const r = await fetch("/api/ai/settings"); const j = await r.json();
    if (j) setS(j);
  })(); },[]);

  async function save(){
    const r = await fetch("/api/ai/settings", { method:"POST", headers:{ "Content-Type":"application/
json" }, body: JSON.stringify(s) });
    setOk(r.ok ? "Salvo!" : "Erro");
    setTimeout(()=>setOk(""), 1500);
```

```
  }

  return (
    <div className="p-6 space-y-6 max-w-xl">
      <h1 className="text-2xl font-bold">IA · Fallback & Orçamento</h1>

      <label className="flex items-center gap-2">
        <input type="checkbox" checked={s.useOpenAI} onChange={e=>setS({...s,
useOpenAI:e.target.checked})}/>
        <span>Permitir fallback (ChatGPT)</span>
      </label>

      <div>
        <label className="block text-sm font-medium">Confiança mínima (τ)</label>
        <input type="range" min={0} max={1} step={0.01} value={s.confidenceTau}
               onChange={e=>setS({...s,confidenceTau:Number(e.target.value)})}/>
        <div className="text-xs text-gray-600">Atual: {s.confidenceTau.toFixed(2)}</div>
      </div>

      <div className="grid grid-cols-2 gap-4">
        <div>
          <label className="block text-sm font-medium">Teto diário (US$)</label>
          <input className="border p-2 w-full" type="number" step="0.01"
                 value={s.maxDailyUSD} onChange={e=>setS({...s,maxDailyUSD:Number(e.target.value)})}/>
        </div>
        <div>
          <label className="block text-sm font-medium">Tokens por resposta</label>
          <input className="border p-2 w-full" type="number"
                 value={s.maxTokensPerReply} onChange={e=>setS({...s,maxTokensPerReply:Number(e.
target.value)})}/>
        </div>
        <div>
          <label className="block text-sm font-medium">Fallbacks por hora</label>
          <input className="border p-2 w-full" type="number"
                 value={s.maxFallbacksPerHour} onChange={e=>setS({...s,maxFallbacksPerHour:Number(e.
target.value)})}/>
        </div>
      </div>

      <button className="px-4 py-2 bg-emerald-600 text-white rounded" onClick={save}>Salvar</button>
      {!!ok && <span className="ml-3 text-sm text-gray-600">{ok}</span>}
    </div>
  );
}
```

No seu menu admin, adicione um link para **“IA → Fallback & Orçamento”**.

🧩 Mais dois toques para ficar “nível perfeição”

1. **Auditoria & Telemetria do Fallback**
   - Tabela `ai_fallback_logs` com: `id, when, reason(low_confidence|rate_limit|budget), cost_est, tokens_est, query_hash`.
   - Gráficos no painel: *fallback rate* por dia e **histograma de confiança** (pra calibrar τ).
2. **“Dry-run” do fallback**
   - No admin, um botão **“Testar com dry-run”**: simula custo e decisão sem realmente chamar o LLM externo.
   - Útil para ajustar τ e tetos antes de abrir em produção.