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

# IA_Autonoma_Parte13

1 mensagem

**Fillipe Guerra** <fillipe.backup@gmail.com> 28 de outubro de 2025 às 11:26
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

Aqui vai mais um **pacote** para deixar a AION **100% autônoma** com **LLM local + LoRA** e **fallback realmente excepcional** (só se você permitir).

> Se algum arquivo já existir dos meus envios anteriores, **substitua** por esta versão. Mantém single-tenant, telemetria e o painel que já entreguei.

---

# Microserviço de Inferência Local (FastAPI + PEFT)

## 1) Estrutura

```
trainer/
  lora/
    run_lora.py            # (já enviado)
    data_builder.py        # (já enviado)
  inference/
    app.py                 # FastAPI de inferência local
    serve.sh               # script de boot
    requirements.txt
    settings.example.env
```

## 2) `trainer/inference/requirements.txt`

```
fastapi==0.115.0
uvicorn==0.30.6
pydantic==2.9.2
torch==2.3.1
transformers==4.43.3
accelerate==0.34.2
bitsandbytes==0.43.1
peft==0.12.0
safetensors==0.4.4
```

> Se não tiver GPU, dá para rodar com CPU em modelos menores (ex.: `Qwen2-1.5B-Instruct`, `Phi-3-mini-4k-instruct`). Para 7B, recomendo GPU (A100/T4/L4).

## 3) `trainer/inference/settings.example.env`

```
AION_BASE_MODEL=mistralai/Mistral-7B-Instruct-v0.3
AION_ADAPTER_DIR=./trainer/lora/out/adapter     # pasta criada pelo run_lora.py
AION_HOST=0.0.0.0
AION_PORT=8008
AION_LOAD_IN_4BIT=true
AION_MAX_NEW_TOKENS=512
AION_TEMPERATURE=0.6
AION_TOP_P=0.9
AION_SYSTEM_PROMPT=Você é a AION, assistente especialista da plataforma. Responda em PT-BR, cite fontes internas quando pertinente e mantenha objetividade.
```

> Copie para `.env` e ajuste.

## 4) `trainer/inference/app.py`

```
import os, time, json
from typing import List, Optional
from fastapi import FastAPI
from pydantic import BaseModel
from transformers import AutoModelForCausalLM, AutoTokenizer, TextIteratorStreamer, BitsAndBytesConfig
from threading import Thread
from peft import PeftModel, PeftConfig

BASE_MODEL = os.getenv("AION_BASE_MODEL", "mistralai/Mistral-7B-Instruct-v0.3")
ADAPTER_DIR = os.getenv("AION_ADAPTER_DIR", "./trainer/lora/out/adapter")
HOST        = os.getenv("AION_HOST","0.0.0.0")
PORT        = int(os.getenv("AION_PORT","8008"))
LOAD_4BIT   = os.getenv("AION_LOAD_IN_4BIT","true").lower()=="true"
SYS_PROMPT  = os.getenv("AION_SYSTEM_PROMPT","Você é a AION. Responda em PT-BR de forma objetiva e
cite as evidências do contexto quando possível.")
MAX_NEW     = int(os.getenv("AION_MAX_NEW_TOKENS","512"))
DEFAULT_T   = float(os.getenv("AION_TEMPERATURE","0.6"))
DEFAULT_TOPP= float(os.getenv("AION_TOP_P","0.9"))

app = FastAPI(title="AION Local LLM")

tokenizer = None
model     = None
adapter_loaded = False

def load_model():
    global tokenizer, model, adapter_loaded
    quant = BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_quant_type="nf4", bnb_4bit_compute_dtype="
bfloat16") if LOAD_4BIT else None
    tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL, use_fast=True)
    if tokenizer.pad_token is None: tokenizer.pad_token = tokenizer.eos_token
    model = AutoModelForCausalLM.from_pretrained(BASE_MODEL, device_map="auto", torch_dtype="auto",
quantization_config=quant)
    adapter_loaded = False
    if ADAPTER_DIR and os.path.isdir(ADAPTER_DIR):
        try:
            # carrega LoRA sem fundir (runtime)
            model.load_adapter(ADAPTER_DIR, "lora", adapter_name="aion")
            model.set_adapter("aion")
            adapter_loaded = True
        except Exception as e:
            print("[AION] Aviso: não foi possível carregar LoRA:", e)

def merge_and_unload():
    """Opcional: mescla LoRA no base e libera PEFT (mais rápido em inferência, custo de RAM maior)."""
    global model, adapter_loaded
    try:
        model = PeftModel.from_pretrained(model, ADAPTER_DIR)
        model = model.merge_and_unload()
        adapter_loaded = False
        return True
    except Exception as e:
        print("[AION] Merge falhou:", e)
        return False

class GenReq(BaseModel):
    system: Optional[str] = None
    prompt: str
    context: Optional[str] = ""
    temperature: Optional[float] = None
    top_p: Optional[float] = None
    max_new_tokens: Optional[int] = None
    stop: Optional[List[str]] = None

class Health(BaseModel):
    base_model: str
    adapter_loaded: bool

@app.on_event("startup")
```

```
def boot():
    load_model()

@app.get("/health", response_model=Health)
def health():
    return {"base_model": BASE_MODEL, "adapter_loaded": adapter_loaded}

@app.post("/adapter/reload")
def reload_adapter():
    global adapter_loaded
    try:
        model.load_adapter(ADAPTER_DIR, "lora", adapter_name="aion", exists_ok=True)
        model.set_adapter("aion")
        adapter_loaded = True
        return {"ok": True}
    except Exception as e:
        return {"ok": False, "error": str(e)}

@app.post("/adapter/merge")
def merge_adapter():
    ok = merge_and_unload()
    return {"ok": ok}

def build_chat(system_prompt:str, user_prompt:str, context:str):
    # Formato estilo Instruct simples
    ctx = f"\n\n[CONTEXT]\n{context.strip()}\n\n" if context else ""
    return f"{system_prompt}\n\n### Instrução:\n{user_prompt}{ctx}\n\n### Resposta:\n"

@app.post("/generate")
def generate(r: GenReq):
    t0 = time.time()
    system = (r.system or SYS_PROMPT).strip()
    text = build_chat(system, r.prompt, r.context or "")
    inputs = tokenizer([text], return_tensors="pt").to(model.device)

    temp = r.temperature if r.temperature is not None else DEFAULT_T
    topp = r.top_p if r.top_p is not None else DEFAULT_TOPP
    mx   = r.max_new_tokens if r.max_new_tokens is not None else MAX_NEW

    streamer = TextIteratorStreamer(tokenizer, skip_prompt=True, skip_special_tokens=True)
    gen_kwargs = dict(
        **inputs,
        do_sample=True,
        temperature=float(temp),
        top_p=float(topp),
        max_new_tokens=int(mx),
        repetition_penalty=1.05,
        streamer=streamer,
        eos_token_id=tokenizer.eos_token_id
    )
    thread = Thread(target=model.generate, kwargs=gen_kwargs)
    thread.start()

    chunks = []
    for tok in streamer:
        chunks.append(tok)
    out = "".join(chunks).strip()
    dt = time.time()-t0
    return {"ok": True, "ms": int(dt*1000), "text": out}
```

## 5) `trainer/inference/serve.sh`

```
#!/usr/bin/env bash
set -e
export PYTHONUNBUFFERED=1
if [ -f .env ]; then
  export $(grep -v '^#' .env | xargs)
fi
```

```
uvicorn trainer.inference.app:app --host "${AION_HOST:-0.0.0.0}" --port "${AION_PORT:-8008}" --log-level info
```

**Subir localmente**:

```
cd trainer/inference
python3 -m venv .venv && source .venv/bin/activate
pip3 install -r requirements.txt
cp settings.example.env .env   # ajuste se quiser
bash serve.sh
# => http://localhost:8008/health
```

---

# 🧩 Conector Node → Microserviço Local

## 6) `server/ai/local-llm.client.ts`

```
import fetch from "node-fetch";

const LLM_URL = process.env.AION_LOCAL_LLM_URL || "http://localhost:8008";

export type GenOpts = {
  system?: string;
  prompt: string;
  context?: string;
  temperature?: number;
  top_p?: number;
  max_new_tokens?: number;
  stop?: string[];
  timeoutMs?: number;
};

export async function localGenerate(opts: GenOpts){
  const ctrl = new AbortController();
  const t = setTimeout(()=>ctrl.abort(), opts.timeoutMs ?? 60000);
  try{
    const r = await fetch(`${LLM_URL}/generate`, {
      method: "POST",
      headers: { "Content-Type":"application/json" },
      body: JSON.stringify({
        system: opts.system,
        prompt: opts.prompt,
        context: opts.context,
        temperature: opts.temperature,
        top_p: opts.top_p,
        max_new_tokens: opts.max_new_tokens,
        stop: opts.stop
      }),
      signal: ctrl.signal
    });
    if (!r.ok) throw new Error(`Local LLM HTTP ${r.status}`);
    const j = await r.json();
    return j?.text as string;
  } finally { clearTimeout(t); }
}
```

## 7) Prompt Builder (RAG → resposta)

```
server/ai/prompt.ts

export function buildRagPrompt(userMsg: string, passages: { text: string; source?: string }[]){
  // Usa os melhores k chunks; cite fontes
  const ctx = passages.map((p,i)=>`[${i+1}] ${p.text}`).join("\n\n");
  const refs = passages.map((p,i)=>`[${i+1}] ${p.source||"KB"}`).join("; ");
  const prompt =
`Usuário: ${userMsg}
```

```
Instruções:
1) Responda em PT-BR, de forma objetiva, didática e com passos claros.
2) Use APENAS as informações do CONTEXTO abaixo; não invente nada.
3) No final, liste as fontes entre colchetes como [1], [2], ... e depois "Fontes: ${refs}".

CONTEXTO:
${ctx}
`;
  return prompt;
}
```

# 8) Roteador de Respostas (Local → Fallback OpenAI)

> Já tínhamos o **score de confiança C**, o **ANN**, o **MMR**, o **budget** e o **log de fallback**. Aqui consolidamos num único **handler**:

server/ai/answer.router.ts

```
import { localGenerate } from "./local-llm.client";
import { openaiFallbackGenerate } from "./openai.fallback";  // mantém o que já fizemos p/ fallback
orçamentado
import { buildRagPrompt } from "./prompt";
import { searchANN } from "./vector";
import { mmrSelect } from "./math";
import { logRetrievalEvent } from "../aion/events";
import { db } from "../db";
import { aiEvents } from "@shared/schema.ai.metrics";

/** Gera resposta preferindo LOCAL; só cai para OpenAI se:
 *  - C < tau_confidence, OU
 *  - houve erro de microserviço, OU
 *  - usuário forçou "usar_fallback".
 */
export async function answerPreferLocal(tenantId:string, userMsg:string, tau=0.62,
forceFallback=false){
  // 1) Recuperação: topN brutos + MMR
  const raw = await searchANN(tenantId, userMsg, 24);
  const qVec = raw.queryVec!;
  const cand = raw.hits.map(h => ({ id:h.id, score:h.score, vec:h.vec ?? h.vector!, text:h.text,
doc:h.doc, source:h.doc }));
  const chosen = mmrSelect(cand as any, qVec, 8, 0.7);

  // 2) Confiança (já enviamos a fórmula; aqui simplificada com relevância média)
  const avg = chosen.reduce((s,c)=>s+c.score,0)/Math.max(1,chosen.length);
  const C = Math.max(0, Math.min(1, (avg+1)/2)); // map cos[-1,1] → [0,1]

  // 3) Build prompt & prefer local
  const prompt = buildRagPrompt(userMsg, chosen.map(c=>({ text:c.text, source:c.source })));
  let text: string | null = null;
  let used = "local";

  if (!forceFallback && C >= tau){
    try{
      text = await localGenerate({ prompt, max_new_tokens: 512, temperature: 0.5, top_p: 0.9 });
    } catch (e){
      text = null;
    }
  }

  // 4) Fallback se necessário (com orçamento)
  if (!text){
    used = "fallback";
    text = await openaiFallbackGenerate({ prompt, budgetTag:"rag-answer" });  // essa função já
registra custo
    await db.insert(aiEvents).values({
      tenantId, kind: "fallback", value: 0, meta: { reason: "local_failed_or_low_conf" }
    });
  }
```

```
  // 5) Loga retrieval (para heatmap e métricas)
  await logRetrievalEvent(tenantId, userMsg, chosen.map((c,i)=>({ text:c.text, rank:i+1, rel: c.score
})));

  // 6) Loga answer (para SFT futuro)
  await db.insert(aiEvents).values({
    tenantId, kind:"answer", value:1, meta:{ prompt: userMsg, reply: text, rank: 1, source: used }
  });

  return { text, used, C };
}
```

**Em qualquer rota de chat**, troque a geração pelo `answerPreferLocal`.

---

# ⚙️ Rotas de Chat (usando o roteador)

`server/ai/routes.chat.ts`

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

export function registerChatRoutes(app: Express){
  app.post("/api/ai/chat", async (req, res)=>{
    const tenantId = (req as any).tenantId || process.env.PRIMARY_TENANT_ID!;
    const { message, forceFallback, tau } = req.body || {};
    const out = await answerPreferLocal(tenantId, message, Number(tau ?? 0.62), !!forceFallback);
    res.json(out);
  });
}
```

Adicione um **toggle no painel** para `tau` e “Forçar fallback” (dev-only).

---

# 🛡️ Segurança e Governança

- **Whitelist de domínios** (já incluída na Fase 4).
- **robots.txt** respeitado; **rate limit por host**.
- **Single-tenant**: todas as rotas usam `tenantId` explícito.
- **Logs**: nenhum PII sensível nos datasets de treino (builder filtra).
- **Cotas**: orçamentação de fallback ativa (Fase 5).
- **Botão “merge LoRA”** (microserviço) para **fundir** o adapter quando quiser performance máxima e estabilidade.

---

# 🧪 Testes rápidos

1. **Subir microserviço**:

```
cd trainer/inference
python3 -m venv .venv && source .venv/bin/activate
pip3 install -r requirements.txt
cp settings.example.env .env
bash serve.sh
# -> /health deve retornar {"adapter_loaded": true/false}
```

2. **Ping pelo Node**:

```
curl -X POST http://localhost:3000/api/ai/chat \
  -H "Content-Type: application/json" \
  -d '{"message":"Explique diferença entre tour full-day e half-day em Sintra"}'
```

3. **Forçar fallback** (para testar):

```
curl -X POST http://localhost:3000/api/ai/chat \
  -H "Content-Type: application/json" \
  -d '{"message":"...", "forceFallback": true}'
```

---

# 🧰 Variáveis .env (lado Node)

```
AION_LOCAL_LLM_URL=http://localhost:8008
AION_TAU_CONFIDENCE=0.62
AION_TAU_FALLBACK=0.18
AION_TAU_NDCG=0.82
AION_TRAIN_ROOT=./trainer
AION_LORA_BASE=mistralai/Mistral-7B-Instruct-v0.3
AION_EVENTS_JSONL=./trainer/events.jsonl
```

**Dica**: exporte eventos para JSONL 1x/dia (ou no botão “Exportar dataset”). Um job simples:

```
-- pseudo: selecione answers recentes e serialize em JSONL no servidor
```

---

# 📈 Painel: botões extras

No topo da Telemetria (onde já pusemos `<Actions/>`), adicione:

- **Carregar adapter** (`POST /adapter/reload`)
- **Mesclar adapter** (`POST /adapter/merge`)
- **Status microserviço** (`GET /health`)

(Chame via `fetch` direto para `AION_LOCAL_LLM_URL` ou crie rotas proxy no Node se preferir.)