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

IA_Autonoma_Parte14

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com> 28 de outubro de 2025 às 11:31
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

Segue o **pacote extra** — **Docker (GPU/CPU)**, **docker-compose** orquestrando tudo, **chat em streaming (SSE)** ponta-a-ponta, e **exportação em JSONL** dos eventos para treinos LoRA automáticos. É só **colar os arquivos** nos respectivos caminhos. Se algum já existir, **substitua** por estes (são supersets compatíveis com o que entreguei).

🐳 Docker do Microserviço de Inferência (GPU e CPU)

1) `trainer/inference/Dockerfile.gpu`

```
# Base NVIDIA para GPUs (CUDA 12.x). Requer runtime nvidia-docker.
FROM nvidia/cuda:12.1.0-runtime-ubuntu22.04

ENV DEBIAN_FRONTEND=noninteractive \
    PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1

RUN apt-get update && apt-get install -y --no-install-recommends \
    python3 python3-venv python3-pip git curl && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /app

# Copia requisitos e app
COPY trainer/inference/requirements.txt ./requirements.txt
RUN python3 -m pip install --upgrade pip && \
    pip3 install --no-cache-dir -r requirements.txt

COPY trainer/inference ./trainer/inference

# Modelos/adapter via variáveis de ambiente
ENV AION_HOST=0.0.0.0 \
    AION_PORT=8008 \
    AION_LOAD_IN_4BIT=true

EXPOSE 8008
CMD ["bash", "trainer/inference/serve.sh"]
```

2) `trainer/inference/Dockerfile.cpu`

```
# CPU-only (sem GPU). Usa torch CPU.
FROM python:3.11-slim

ENV DEBIAN_FRONTEND=noninteractive \
    PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1

RUN apt-get update && apt-get install -y --no-install-recommends git curl && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /app

# Requisitos: substitui torch + bitsandbytes por CPU
COPY trainer/inference/requirements.txt ./requirements.txt
```

```
# Remove bitsandbytes (GPU) e instala torch CPU
RUN sed -i '/bitsandbytes/d' requirements.txt && \
    python3 -m pip install --upgrade pip && \
    pip3 install --no-cache-dir -r requirements.txt && \
    pip3 install --no-cache-dir torch==2.3.1 --index-url https://download.pytorch.org/whl/cpu

COPY trainer/inference ./trainer/inference

ENV AION_HOST=0.0.0.0 \
    AION_PORT=8008 \
    AION_LOAD_IN_4BIT=false

EXPOSE 8008
CMD ["bash", "trainer/inference/serve.sh"]
```

---

docker-compose (App + DB + Inferência)

3) `docker-compose.yml` (na raiz do projeto)

```
version: "3.9"
services:
  db:
    image: postgres:16
    environment:
      POSTGRES_USER: aion
      POSTGRES_PASSWORD: aion
      POSTGRES_DB: aion
    ports: ["5432:5432"]
    volumes:
      - pgdata:/var/lib/postgresql/data

  app:
    build:
      context: .
      dockerfile: Dockerfile # seu Dockerfile do Node/Express
    depends_on: [db, inference]
    env_file:
      - .env
    environment:
      DATABASE_URL: postgres://aion:aion@db:5432/aion
      AION_LOCAL_LLM_URL: http://inference:8008
    ports: ["3000:3000"]
    volumes:
      - ./:/usr/src/app
    command: ["pnpm","run","start:prod"]

  inference:
    build:
      context: .
      dockerfile: trainer/inference/Dockerfile.gpu # troque para .cpu se for CPU
    deploy:
      resources:
        reservations:
          devices:
            - capabilities: ["gpu"]   # Remova este bloco caso use CPU
    env_file:
      - trainer/inference/.env
    ports: ["8008:8008"]
    volumes:
      - ./trainer/lora/out:/app/trainer/lora/out  # persiste adapters

  pgadmin:
    image: dpage/pgadmin4
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@aion.local
      PGADMIN_DEFAULT_PASSWORD: admin
```

```
    ports: ["5050:80"]
    depends_on: [db]

volumes:
  pgdata:
```

GPU: instale `nvidia-container-toolkit` e rode `docker compose --profile gpu up` se necessário; no compose acima já está preparado (se houver GPU). Para CPU, mude a linha do Dockerfile para `Dockerfile.cpu` e remova o bloco `devices`.

Chat por Streaming (SSE) – ponta-a-ponta

Vamos expor **/generate/stream** no microserviço e **/api/ai/chat/stream** no Node, repassando os chunks em **Server-Sent Events**.

4) Atualize o microserviço FastAPI: `trainer/inference/app.py`

(adicione os endpoints abaixo no mesmo arquivo já enviado)

```
from fastapi.responses import StreamingResponse

def sse_format(event: str, data: str) -> str:
    return f"event: {event}\ndata: {data}\n\n"

@app.post("/generate/stream")
def generate_stream(r: GenReq):
    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
    )

    def token_stream():
        th = Thread(target=model.generate, kwargs=gen_kwargs); th.start()
        for tok in streamer:
          yield sse_format("token", json.dumps({"text": tok}))
        yield sse_format("done", "{}")

    return StreamingResponse(token_stream(), media_type="text/event-stream")
```

5) Cliente Node para streaming local: `server/ai/local-llm.client.ts`

(adicione esta função ao arquivo que já criei antes)

```
export async function localGenerateStream(opts: GenOpts, onToken: (t:string)=>void){
  const url = `${process.env.AION_LOCAL_LLM_URL || "http://localhost:8008"}/generate/stream`;
  const r = await fetch(url, {
    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
```

```
      })
    });
    if (!r.ok || !r.body) throw new Error(`Local LLM stream HTTP ${r.status}`);

    const reader = r.body.getReader();
    const decoder = new TextDecoder();
    let buffer = "";
    while (true){
      const { value, done } = await reader.read();
      if (done) break;
      buffer += decoder.decode(value, { stream: true });
      let idx;
      while ((idx = buffer.indexOf("\n\n")) >= 0){
        const chunk = buffer.slice(0, idx); buffer = buffer.slice(idx+2);
        if (chunk.startsWith("event:")){
          const evLine = chunk.split("\n").find(l=>l.startsWith("event:"))!;
          const dataLine = chunk.split("\n").find(l=>l.startsWith("data:")) || "data:{}";
          const ev = evLine.slice(6).trim();
          const data = JSON.parse(dataLine.slice(5) || "{}");
          if (ev === "token" && data.text) onToken(data.text);
        }
      }
    }
  }
```

6) Rota SSE no Node (Express): `server/ai/routes.chat.ts`

(adicione este endpoint; mantém o `/api/ai/chat` normal)

```
import type { Express, Request, Response } from "express";
import { answerPreferLocal } from "./answer.router";
import { localGenerateStream } from "./local-llm.client";
import { buildRagPrompt } from "./prompt";
import { searchANN } from "./vector";
import { mmrSelect } from "./math";

export function registerChatRoutes(app: Express){
  // já existia:
  app.post("/api/ai/chat", async (req, res)=>{ /* ... (como já mandei) ... */ });

  // NOVO: SSE streaming
  app.get("/api/ai/chat/stream", async (req: Request, res: Response)=>{
    const tenantId = (req as any).tenantId || process.env.PRIMARY_TENANT_ID!;
    const q = String(req.query.q || "");
    const tau = Number(req.query.tau || "0.62");
    res.setHeader("Content-Type", "text/event-stream");
    res.setHeader("Cache-Control", "no-cache");
    res.setHeader("Connection", "keep-alive");

    // Recupera contexto (igual ao router tradicional, mas monta prompt e streama)
    const raw = await searchANN(tenantId, q, 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);
    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));

    if (C < tau){
      // se confiança baixa, pode encerrar SSE e sugerir fallback (ou já fazer fallback em modo não-
SSE)
      res.write(`event: warn\ndata: ${JSON.stringify({ reason:"low_confidence" })}\n\n`);
    }

    const prompt = buildRagPrompt(q, chosen.map(c=>({ text:c.text, source:c.source })));

    await localGenerateStream({ prompt, max_new_tokens: 512, temperature: 0.5, top_p: 0.9 }, (tok)=>{
      res.write(`event: token\ndata: ${JSON.stringify({ text: tok })}\n\n`);
    });
```

```
    res.write(`event: done\ndata: {}\n\n`);
    res.end();
  });
}
```

No frontend, consuma via `EventSource('/api/ai/chat/stream?q=...')`.

📤 Exportação de eventos → JSONL (treino LoRA)

7) Exportador: `server/metrics/export.ts`

```
import fs from "fs";
import path from "path";
import { db } from "../db";
import { aiEvents } from "@shared/schema.ai.metrics";
import { eq, and, gte, lte } from "drizzle-orm";

export async function exportEventsJSONL(tenantId:string, outPath:string, days=30){
  const since = new Date(Date.now() - days*24*60*60*1000);
  const rows = await db.select().from(aiEvents)
    .where(eq(aiEvents.tenantId, tenantId));
  const w = fs.createWriteStream(outPath, { flags: "w" });
  for (const r of rows){
    // só guarda answers com prompt/reply para SFT
    if (r.kind === "answer" && r.meta && (r.meta.prompt || r.meta.q) && (r.meta.reply || r.meta.a)){
      w.write(JSON.stringify({
        kind: r.kind,
        meta: { prompt: r.meta.prompt || r.meta.q, reply: r.meta.reply || r.meta.a }
      })+"\n");
    }
  }
  w.end();
  return rows.length;
}
```

8) Rotas para exportar: `server/ai/routes.export.ts`

```
import type { Express, Request } from "express";
import path from "path";
import { exportEventsJSONL } from "../metrics/export";

export function registerExportRoutes(app: Express){
  app.post("/api/ai/metrics/export.jsonl", async (req, res)=>{
    const tenantId = (req as any).tenantId || process.env.PRIMARY_TENANT_ID!;
    const out = process.env.AION_EVENTS_JSONL || "./trainer/events.jsonl";
    const n = await exportEventsJSONL(tenantId, out, Number(req.body?.days ?? 30));
    res.json({ ok:true, out, rows:n });
  });
}
```

9) Cron diário pra exportar (opcional): `server/metrics/export.cron.ts`

```
import cron from "node-cron";
import { exportEventsJSONL } from "./export";

export function startExportCron(){
  cron.schedule("5 0 * * *", async ()=>{
    const tenantId = process.env.PRIMARY_TENANT_ID!;
    const out = process.env.AION_EVENTS_JSONL || "./trainer/events.jsonl";
    try{ await exportEventsJSONL(tenantId, out, 30); }catch(e){ /* log */ }
  });
}
```

No bootstrap do servidor, acrescente:

```
import { registerExportRoutes } from "./ai/routes.export";
import { startExportCron } from "./metrics/export.cron";

registerExportRoutes(app);
startExportCron();
```

🧪 Smoke tests

1. **Subir compose** (CPU):

```
docker compose up -d --build
# se GPU:
# docker compose up -d --build
# (com nvidia runtime configurado)
```

2. **Health da inferência**
   `GET` http://localhost:8008/health → deve retornar `base_model` e `adapter_loaded`.

3. **Chat streaming** (no navegador):
   `new EventSource('/api/ai/chat/stream?q=Tour Sintra full-day vs half-day')`

4. **Export JSONL**:

```
curl -X POST http://localhost:3000/api/ai/metrics/export.jsonl \
  -H "Content-Type: application/json" -d '{"days": 30}'
# arquivo em ./trainer/events.jsonl
```

5. **Treino LoRA** (já integrado):

- Exporta JSONL (acima),
- `POST /api/ai/train/lora/start` → acompanha em `/api/ai/train/lora/status`.

✅ O que fechamos agora

- **Deploy padronizado** (Docker GPU/CPU + compose).
- **Chat com streaming (SSE)** direto do **LLM local**.
- **Exportar dataset** de treino em **JSONL** → **QLoRA** automático.
- Tudo **single-tenant**, com **telemetria**, **curadoria**, **ANN**, **fallback orçamentado** e **LoRA adaptativa**.