



Fillipe Guerra <fillipe.backup@gmail.com>

IA_REVIEW_PARTE1

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com>

27 de outubro de 2025 às 18:27

Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

Etapa 1 – Arquitetura Base Multimodal 2.0.

Aqui vai um *drop* completo, coeso e “mastigado”— com **código integral**, **matemática**, **arquivos prontos**, **nomenclatura padronizada**, **Single-Tenant**, **independência de IA externa**, **ANN HNSW incremental com checkpoint**, e **suporte multimodal (texto + imagem)** já desde a fundação.

Conteúdo desta etapa:

1. Estrutura de pastas + .env
2. Schemas Drizzle (texto + imagem + ANN + políticas + interações)
3. Embeddings locais ONNX (texto + imagem) com tokenização WordPiece e pooling
4. ANN HNSW **incremental** (add em tempo real) + checkpoint automático
5. Vector Store unificado (texto + imagem)
6. Scorer matemático híbrido base (foco em arquitetura; $\gamma/\delta/\zeta$ ativaremos na Etapa 2 ao ligar grafo/parsers)
7. Ingest multimodal (texto bruto, URL/arquivo, imagem)
8. Rotas HTTP e bootstrap
9. Passo a passo de verificação rápida

0) Estrutura recomendada de pastas

Coloque estes arquivos (copiar/colar):

```
/shared/  
  schema.ai.core.ts  
/server/  
  db.ts                # (o seu já existente)  
  ai/  
    embeddings.text.ts  
    embeddings.image.ts  
    ann.ts  
    vector-store.ts  
    hybrid-score.ts  
    ingest.ts  
    parser.ts          # (HTML/PDF/DOCX/PPTX/CSV/OCR virá na Etapa 2; aqui deixo stub mínimo)  
    routes.ts  
  routes.ts           # registra rotas  
/models/  
  minilm.onnx  
  vocab.txt  
  mobilenet.onnx      # (ou vit-small.onnx; escolha 1)  
  imagenet_labels.json # labels da rede de visão  
/data/ann/            # será criado em runtime (checkpoint ANN)
```

.env (exemplo):

```

PRIMARY_TENANT_ID=tenant-main
AI_EMB_MODEL_PATH=./models/minilm.onnx
AI_EMB_VOCAB_PATH=./models/vocab.txt
AI_EMB_MAX_LEN=128
AI_EMB_DIM=384

AI_VISION_MODEL_PATH=./models/mobilenet.onnx
AI_VISION_DIM=1024
AI_VISION_LABELS=./models/imagenet_labels.json

AI_ANN_M=16
AI_ANN_EF_CONS=200
AI_ANN_EF=64

```

package.json (dependências necessárias nesta etapa):

```

{
  "dependencies": {
    "onnxruntime-node": "^1.19.2",
    "hnswlib-node": "^2.1.0",
    "multer": "^1.4.5-lts.1"
  }
}

```

1) Schemas Drizzle — /shared/schema.ai.core.ts

Tabelas **multimodais** (texto e imagem), **ANN meta** (para auditoria/checkpoint), **políticas, interações**.
Tudo **Single-Tenant** com tenantId.

```

// shared/schema.ai.core.ts
import {
  pgTable, uuid, varchar, text, integer, boolean, timestamp, real,
  primaryKey, index, jsonb
} from "drizzle-orm/pg-core";

/** Documentos e chunks (conteúdo canônico para RAG) */
export const aiDocuments = pgTable("ai_documents", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  source: varchar("source", { length: 64 }).notNull(), // "url"|"file"|"crm"|"erp"|"brand-
scanner"|"manual"|"image"
  uri: text("uri"), // URL, caminho, id externo
  title: text("title"),
  metaJson: text("meta_json"),
  createdAt: timestamp("created_at").defaultNow().notNull(),
  updatedAt: timestamp("updated_at").defaultNow().notNull()
}, t => ({
  idxTenant: index("ai_documents_tenant_idx").on(t.tenantId),
  idxUri: index("ai_documents_uri_idx").on(t.uri),
}));

export const aiChunks = pgTable("ai_chunks", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  documentId: uuid("document_id").notNull(),
  modality: varchar("modality", { length: 16 }).notNull(), // "text" | "image"
  pos: integer("pos").notNull(), // ordem no documento
  text: text("text"), // para "text" (opcionalmente legenda de imagem)
  imageUri: text("image_uri"), // para "image"
  tokens: integer("tokens").default(0).notNull(),
  metaJson: text("meta_json"),
  createdAt: timestamp("created_at").defaultNow().notNull()
}, t => ({
  idxDoc: index("ai_chunks_doc_idx").on(t.documentId),
  idxTenant: index("ai_chunks_tenant_idx").on(t.tenantId),
  idxModality: index("ai_chunks_modality_idx").on(t.tenantId, t.modality),
}));

```

```

/** Embeddings por modalidade */
export const aiEmbeddingsText = pgTable("ai_embeddings_text", {
  chunkId: uuid("chunk_id").primaryKey().notNull(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  vectorJson: text("vector_json").notNull(),
  dim: integer("dim").notNull(),
  model: varchar("model", { length: 64 }).notNull(), // ex "minilm-384-onnx"
  createdAt: timestamp("created_at").defaultNow().notNull()
}, t => ({
  idxTenant: index("ai_emb_text_tenant_idx").on(t.tenantId),
  idxModel: index("ai_emb_text_model_idx").on(t.model),
}));

export const aiEmbeddingsImage = pgTable("ai_embeddings_image", {
  chunkId: uuid("chunk_id").primaryKey().notNull(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  vectorJson: text("vector_json").notNull(),
  dim: integer("dim").notNull(),
  model: varchar("model", { length: 64 }).notNull(), // ex "mobilenet-onnx"
  createdAt: timestamp("created_at").defaultNow().notNull()
}, t => ({
  idxTenant: index("ai_emb_img_tenant_idx").on(t.tenantId),
  idxModel: index("ai_emb_img_model_idx").on(t.model),
}));

/** Metadados do índice ANN (auditoria/checkpoints) */
export const aiAnnMeta = pgTable("ai_ann_meta", {
  tenantId: varchar("tenant_id", { length: 64 }).primaryKey(),
  textDim: integer("text_dim").default(0).notNull(),
  imageDim: integer("image_dim").default(0).notNull(),
  sizeText: integer("size_text").default(0).notNull(),
  sizeImage: integer("size_image").default(0).notNull(),
  lastCheckpointAt: timestamp("last_checkpoint_at"),
  updatedAt: timestamp("updated_at").defaultNow().notNull()
});

/** Políticas e Interações (aproveitaremos mais adiante) */
export const aiPolicies = pgTable("ai_policies", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  name: varchar("name", { length: 160 }).notNull(), // "retrieval_weights" | "persuasion_bandit" | ...
  dataJson: text("data_json").notNull(),
  updatedAt: timestamp("updated_at").defaultNow().notNull()
}, t => ({
  idxTenant: index("ai_policies_tenant_idx").on(t.tenantId),
  idxName: index("ai_policies_name_idx").on(t.name),
}));

export const aiInteractions = pgTable("ai_interactions", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  conversationId: uuid("conversation_id"),
  userId: varchar("user_id", { length: 64 }),
  policyId: uuid("policy_id"),
  input: text("input").notNull(),
  output: text("output").notNull(),
  featuresJson: text("features_json"),
  reward: real("reward").default(0).notNull(),
  eventJson: text("event_json"),
  createdAt: timestamp("created_at").defaultNow().notNull()
}, t => ({
  idxTenant: index("ai_interactions_tenant_idx").on(t.tenantId),
  idxConv: index("ai_interactions_conv_idx").on(t.conversationId),
}));

```

2) Embeddings locais ONNX

2.1 Texto — /server/ai/embeddings.text.ts

- Tokenização **WordPiece** (vocab local)
- Entrada: input_ids, attention_mask
- Saída: last_hidden_state → **mean pooling** + **L2** (ou pooled_output)

```
// server/ai/embeddings.text.ts
import * as ort from "onnxruntime-node";
import fs from "fs";
import path from "path";

const MODEL_PATH = process.env.AI_EMB_MODEL_PATH || path.resolve(process.cwd(), "models", "minilm.onnx");
const VOCAB_PATH = process.env.AI_EMB_VOCAB_PATH || path.resolve(process.cwd(), "models", "vocab.txt");
const MAX_LEN = Number(process.env.AI_EMB_MAX_LEN || 128);

let session: ort.InferenceSession | null = null;
let vocab: Map<string, number> | null = null;

const CLS = "[CLS]", SEP = "[SEP]", PAD = "[PAD]", UNK = "[UNK]";

async function loadVocab() {
  if (vocab) return vocab;
  const lines = fs.readFileSync(VOCAB_PATH, "utf8").split(/\r?\n/).filter(Boolean);
  const m = new Map<string, number>();
  lines.forEach((tok, idx) => m.set(tok, idx));
  for (const t of [CLS, SEP, PAD, UNK]) if (!m.has(t)) throw new Error(`vocab.txt inválido - faltou ${t}`);
  vocab = m;
  return m;
}

function basicTokenize(text: string): string[] {
  return text
    .normalize("NFKC")
    .toLowerCase()
    .replace(/[\^\p{L}\p{N}\s\.\,\-\_\]/gu, " ")
    .split(/\s+/)
    .filter(Boolean);
}

function wordpiece(tokens: string[], vocab: Map<string, number>): number[] {
  const ids: number[] = [];
  const unkId = vocab.get(UNK)!;
  for (const token of tokens) {
    if (vocab.has(token)) { ids.push(vocab.get(token)!); continue; }
    const chars = Array.from(token);
    let start = 0;
    const sub: number[] = [];
    while (start < chars.length) {
      let end = chars.length, found: number | null = null;
      while (start < end) {
        let piece = chars.slice(start, end).join("");
        if (start > 0) piece = "##" + piece;
        const id = vocab.get(piece);
        if (id !== undefined) { found = id; break; }
        end--;
      }
      if (found === null) { sub.length = 0; break; }
      sub.push(found); start = end;
    }
    ids.push(...(sub.length ? sub : [unkId]));
  }
  return ids;
}
```

```

function padTrunc(arr: number[], max: number, padId: number) {
  if (arr.length > max) return arr.slice(0, max);
  if (arr.length < max) return arr.concat(Array(max - arr.length).fill(padId));
  return arr;
}

export async function ensureTextSession() {
  if (session) return session;
  session = await ort.InferenceSession.create(MODEL_PATH, { executionProviders: ["cpu"] });
  await loadVocab();
  return session;
}

function meanPool(lastHidden: Float32Array, T: number, D: number, attn: number[]) {
  const out = new Float32Array(D);
  let denom = 0;
  for (let t = 0; t < T; t++) {
    if (!attn[t]) continue;
    denom++;
    const base = t * D;
    for (let d = 0; d < D; d++) out[d] += lastHidden[base + d];
  }
  if (denom === 0) denom = 1;
  for (let d = 0; d < D; d++) out[d] /= denom;
  // L2
  let n = 0; for (let d = 0; d < D; d++) n += out[d] * out[d];
  n = Math.sqrt(n) || 1;
  for (let d = 0; d < D; d++) out[d] /= n;
  return Array.from(out);
}

export async function embedTexts(texts: string[]): Promise<number[][]> {
  const sess = await ensureTextSession();
  const v = await loadVocab();

  const padId = v.get(PAD)!; const clsId = v.get(CLS)!; const sepId = v.get(SEP)!;
  const B = texts.length;
  const inputIds = new BigInt64Array(B * MAX_LEN);
  const attnMask = new BigInt64Array(B * MAX_LEN);

  for (let b = 0; b < B; b++) {
    const toks = wordpiece(basicTokenize(texts[b] || ""), v);
    const seq = [clsId, ...toks, sepId];
    const arr = padTrunc(seq, MAX_LEN, padId);
    for (let i = 0; i < MAX_LEN; i++) {
      const id = arr[i];
      inputIds[b * MAX_LEN + i] = BigInt(id);
      attnMask[b * MAX_LEN + i] = BigInt(id === padId ? 0 : 1);
    }
  }

  const out = await sess.run({
    "input_ids": new ort.Tensor("int64", inputIds, [B, MAX_LEN]),
    "attention_mask": new ort.Tensor("int64", attnMask, [B, MAX_LEN]),
  });

  if (out["last_hidden_state"]) {
    const t = out["last_hidden_state"] as ort.Tensor;
    const [BB, T, D] = t.dims;
    const data = t.data as Float32Array;
    const vecs: number[][] = [];
    for (let b = 0; b < BB; b++) {
      const base = b * T * D;
      const attn = Array.from(attnMask.slice(b * MAX_LEN, b * MAX_LEN + MAX_LEN)).map(Number);
      vecs.push(meanPool(data.subarray(base, base + T * D), T, D, attn));
    }
    return vecs;
  }
  if (out["pooled_output"]) {

```

```

const t = out["pooled_output"] as ort.Tensor;
const [BB, D] = t.dims; const data = t.data as Float32Array;
const vecs: number[][] = [];
for (let b = 0; b < BB; b++) {
  const v = Array.from(data.subarray(b * D, (b + 1) * D));
  let n = Math.sqrt(v.reduce((s, x) => s + x * x, 0)) || 1;
  vecs.push(v.map(x => x / n));
}
return vecs;
}
throw new Error("Saídas do encoder ONNX não reconhecidas.");
}

```

2.2 Imagem — /server/ai/embeddings.image.ts

- Suporta **MobileNet** (ou ViT) ONNX
- **Global Average Pooling** do penúltimo mapa ou usa pool/embedding se existir
- Normalização **L2**
- *Opcional*: mapeamento de labels (útil para meta/QA)

```

// server/ai/embeddings.image.ts
import * as ort from "onnxruntime-node";
import fs from "fs";
import path from "path";

const MODEL_PATH = process.env.AI_VISION_MODEL_PATH || path.resolve(process.cwd(), "models", "mobilenet.onnx");
const LABELS_PATH = process.env.AI_VISION_LABELS || path.resolve(process.cwd(), "models", "imagenet_labels.json");
const VISION_DIM = Number(process.env.AI_VISION_DIM || 1024);

let session: ort.InferenceSession | null = null;
let labels: string[] | null = null;

export async function ensureVisionSession() {
  if (session) return session;
  session = await ort.InferenceSession.create(MODEL_PATH, { executionProviders: ["cpu"] });
  try {
    labels = JSON.parse(fs.readFileSync(LABELS_PATH, "utf8"));
  } catch { labels = null; }
  return session;
}

/** Pré-processa imagem RGB uint8 → float32 (NCHW, normalização [0,1]) */
export function preprocessImageRGB(imageData: Uint8ClampedArray, width: number, height: number) {
  // Você pode redimensionar/pad aqui caso seu ONNX exija dimensões fixas (por ex. 224x224)
  // Para simplificar, assumimos que já vem no tamanho esperado.
  const C = 3, W = width, H = height;
  const out = new Float32Array(1 * C * H * W);
  // NCHW
  let idx = 0, o = 0;
  // canal R
  for (let y=0;y<H;y++) for (let x=0;x<W;x++) { out[o++] = imageData[idx] / 255; idx+=4; }
  // voltar para G (ajusta índice)
  idx = 1;
  for (let y=0;y<H;y++) for (let x=0;x<W;x++) { out[o++] = imageData[(y*W + x)*4 + 1] / 255; }
  // B
  for (let y=0;y<H;y++) for (let x=0;x<W;x++) { out[o++] = imageData[(y*W + x)*4 + 2] / 255; }
  return new ort.Tensor("float32", out, [1, C, H, W]);
}

function l2(v: number[]) {
  let n = Math.sqrt(v.reduce((s, x) => s + x * x, 0)) || 1;
  return v.map(x => x / n);
}

```

```

export async function embedImages(batchTensors: ort.Tensor[]): Promise<number[][]> {
  const sess = await ensureVisionSession();
  const vecs: number[][] = [];
  for (const tensor of batchTensors) {
    const out = await sess.run({ "input": tensor }); // ajuste o nome da input de acordo com o seu
ONNX
    // Estratégia: procurar tensores candidatos
    // - "embedding" ou "pooled_output"
    // - ou "features" 4D → GAP (global average pooling)
    const key = Object.keys(out)[0];
    const t = out[key];
    if (!t) throw new Error("Saída da rede de visão não encontrada.");
    const data = t.data as Float32Array;
    if (t.dims.length === 2) {
      // (1, D)
      vecs.push(l2(Array.from(data)));
    } else if (t.dims.length === 4) {
      // (1, C, H, W) → GAP em HxW
      const [N, C, H, W] = t.dims;
      const pooled = new Array(C).fill(0);
      for (let c=0; c<C; c++) {
        let sum=0; for (let i=0; i<H*W; i++) sum += data[c*H*W + i];
        pooled[c] = sum / (H*W);
      }
      vecs.push(l2(pooled));
    } else {
      throw new Error("Formato de saída de visão não suportado.");
    }
  }
  return vecs;
}

export function topLabels(logits: Float32Array, k=5) {
  if (!labels) return [];
  const idx = Array.from(logits).map((v,i)=>({v,i})).sort((a,b)=>b.v-a.v).slice(0,k).map(o=>labels!
[o.i]);
  return idx;
}

```

Observação: Ajuste o nome da entrada/saída conforme o seu modelo ONNX (alguns usam images, input_1, pixel_values, etc.). Mantive "input" como default.

3) ANN HNSW incremental + checkpoint — /server/ai/ann.ts

- Um índice por modalidade e por tenant
- add() a cada novo embedding
- save() **automático** com throttling + **hook de shutdown** para checkpoint final

```

// server/ai/ann.ts
import { HierarchicalNSW } from "hnswlib-node";
import fs from "fs";
import path from "path";

type Space = "cosine" | "l2";
type Modality = "text" | "image";

const DIR = path.resolve(process.cwd(), "data", "ann");
function ensureDir() { if (!fs.existsSync(DIR)) fs.mkdirSync(DIR, { recursive: true }); }

function idxPath(tenantId: string, modality: Modality, dim: number, space: Space) {
  const base = path.join(DIR, `${tenantId}.${modality}.${space}.${dim}`);
  return { meta: base + ".meta.json", bin: base + ".bin" };
}

```

```

export class AnnIndex {
  private idx: HierarchicalNSW | null = null;
  private dim = 0;
  private space: Space = "cosine";
  private modality: Modality = "text";
  private tenantId: string;
  private lastSave = 0;
  private dirty = false;

  constructor(tenantId: string, modality: Modality, dim: number, space: Space = "cosine") {
    this.tenantId = tenantId; this.modality = modality; this.dim = dim; this.space = space;
    ensureDir();
  }

  loadOrCreate(capacity: number) {
    const p = idxPath(this.tenantId, this.modality, this.dim, this.space);
    const idx = new HierarchicalNSW(this.space, this.dim);
    if (fs.existsSync(p.meta) && fs.existsSync(p.bin)) {
      idx.readIndex(p.bin);
      this.idx = idx;
    } else {
      idx.initIndex(capacity, Number(process.env.AI_ANN_M || 16), Number(process.env.AI_ANN_EF_CONS ||
200));
      this.idx = idx;
      this.saveMeta({ size: 0 });
      this.saveNow(); // cria .bin inicial
    }
    process.on("SIGINT", () => { try { this.saveNow(); } catch {} process.exit(0); });
    process.on("SIGTERM", () => { try { this.saveNow(); } catch {} process.exit(0); });
  }

  setEf(ef: number) { this.idx?.setEf(ef); }

  add(items: number[][], ids: number[]) {
    if (!this.idx) throw new Error("ANN não inicializado");
    this.idx.addItems(items, ids);
    this.dirty = true;
    this.maybeSave();
  }

  search(query: number[], k: number) {
    if (!this.idx) throw new Error("ANN não inicializado");
    this.idx.setEf(Number(process.env.AI_ANN_EF || 64));
    const { neighbors, distances } = this.idx.searchKNN(query, k);
    // Para cosine, a lib retorna distância – convertemos para score de afinidade (1 - dist)
    return neighbors.map((hid, i) => ({ hid, score: 1 - distances[i] }));
  }

  private maybeSave() {
    const now = Date.now();
    // salva no máx. a cada 2 min se sujo
    if (this.dirty && now - this.lastSave > 120000) this.saveNow();
  }

  private saveNow() {
    if (!this.idx) return;
    const p = idxPath(this.tenantId, this.modality, this.dim, this.space);
    this.idx.writeIndex(p.bin);
    this.saveMeta({ size: this.idx.getCurrentCount() });
    this.lastSave = Date.now();
    this.dirty = false;
  }

  private saveMeta(meta: { size: number }) {
    const p = idxPath(this.tenantId, this.modality, this.dim, this.space);
    fs.writeFileSync(p.meta, JSON.stringify({
      tenantId: this.tenantId, modality: this.modality, space: this.space, dim: this.dim,
      size: meta.size, savedAt: new Date().toISOString()
    }));
  }
}

```



```

    }));
  }
}

```

4) Vector Store unificado — /server/ai/vector-store.ts

- Upsert em **tabelas separadas** por modalidade
- Índices ANN **incrementais** por modalidade
- Busca k-NN via ANN (sem fallback exaustivo)
- Map de hid → chunkId determinístico

```

// server/ai/vector-store.ts
import { db } from "../db";
import { aiEmbeddingsText, aiEmbeddingsImage, aiChunks, aiAnnMeta } from "@shared/schema.ai.core";
import { eq } from "drizzle-orm";
import { AnnIndex } from "../ann";

type Modality = "text" | "image";
function idHash(id: string) { let h=0; for (let i=0;i<id.length;i++){ h=(h*31 + id.charCodeAt(i))|0; }
return Math.abs(h); }

async function upsertAnnMeta(tenantId: string, modality: Modality, dim: number, sizeDelta: number) {
  const rows = await db.select().from(aiAnnMeta).where(eq(aiAnnMeta.tenantId, tenantId));
  if (!rows.length) {
    await db.insert(aiAnnMeta).values({
      tenantId, textDim: modality==="text"?dim:0, imageDim: modality==="image"?dim:0,
      sizeText: modality==="text"?sizeDelta:0, sizeImage: modality==="image"?sizeDelta:0
    });
  } else {
    const cur = rows[0];
    await db.update(aiAnnMeta).set({
      textDim: modality==="text" ? dim : cur.textDim,
      imageDim: modality==="image" ? dim : cur.imageDim,
      sizeText: modality==="text" ? cur.sizeText + sizeDelta : cur.sizeText,
      sizeImage: modality==="image" ? cur.sizeImage + sizeDelta : cur.sizeImage,
      updatedAt: new Date()
    }).where(eq(aiAnnMeta.tenantId, tenantId));
  }
}

/** TEXT */
export async function upsertTextEmbedding(tenantId: string, chunkId: string, vector: number[], model: string) {
  await db.insert(aiEmbeddingsText).values({
    chunkId, tenantId, vectorJson: JSON.stringify(vector), dim: vector.length, model
  }).onConflictDoUpdate({
    target: aiEmbeddingsText.chunkId,
    set: { vectorJson: JSON.stringify(vector), dim: vector.length, model }
  });
  // ANN incremental
  const ann = new AnnIndex(tenantId, "text", vector.length, "cosine");
  ann.loadOrCreate(1000);
  ann.add([vector], [idHash(chunkId)]);
  await upsertAnnMeta(tenantId, "text", vector.length, +1);
}

export async function knnText(tenantId: string, queryVec: number[], k = 50) {
  const ann = new AnnIndex(tenantId, "text", queryVec.length, "cosine");
  ann.loadOrCreate(1000);
  const out = ann.search(queryVec, k);
  // mapear HID->chunkId
  const rows = await db.select().from(aiEmbeddingsText).where(eq(aiEmbeddingsText.tenantId,

```

```

tenantId));
  const map = new Map<number, string>();
  for (const r of rows) map.set(idHash(r.chunkId), r.chunkId);
  return out.map(o => ({ chunkId: map.get(o.hid)!, score: o.score })).filter(x => !!x.chunkId);
}

/** IMAGE */
export async function upsertImageEmbedding(tenantId: string, chunkId: string, vector: number[], model: string) {
  await db.insert(aiEmbeddingsImage).values({
    chunkId, tenantId, vectorJson: JSON.stringify(vector), dim: vector.length, model
  }).onConflictDoUpdate({
    target: aiEmbeddingsImage.chunkId,
    set: { vectorJson: JSON.stringify(vector), dim: vector.length, model }
  });
  const ann = new AnnIndex(tenantId, "image", vector.length, "cosine");
  ann.loadOrCreate(1000);
  ann.add([vector], [idHash(chunkId)]);
  await upsertAnnMeta(tenantId, "image", vector.length, +1);
}

export async function knnImage(tenantId: string, queryVec: number[], k = 50) {
  const ann = new AnnIndex(tenantId, "image", queryVec.length, "cosine");
  ann.loadOrCreate(1000);
  const out = ann.search(queryVec, k);
  const rows = await db.select().from(aiEmbeddingsImage).where(eq(aiEmbeddingsImage.tenantId, tenantId));
  const map = new Map<number, string>();
  for (const r of rows) map.set(idHash(r.chunkId), r.chunkId);
  return out.map(o => ({ chunkId: map.get(o.hid)!, score: o.score })).filter(x => !!x.chunkId);
}

/** util */
export async function getChunksByIds(ids: string[]) {
  if (!ids.length) return [];
  const placeholders = ids.map((_, i) => `${i+1}`).join(",");
  const rows = await db.execute(
    `select id, document_id, modality, pos, text, image_uri, meta_json from ai_chunks where id in (${placeholders})`,
    ids as any
  );
  return rows as any[];
}

```

5) Scorer híbrido base — /server/ai/hybrid-score.ts

Nesta etapa focamos no **alicerce**: combinamos *apenas* os canais vetoriais (texto e imagem). Na **Etapa 2**, plugaremos BM25, grafo semântico (γ), frescor (δ) e autoridade (ζ).
Matemática do score (nesta etapa):

$$S = \alpha \cdot S_{\text{text}} + \beta \cdot S_{\text{image}}, \alpha, \beta \geq 0, \alpha + \beta = 1$$

- S_{text} = similaridade do embedding textual (cosine via HNSW)
- S_{image} = similaridade do embedding visual (cosine via HNSW)

Os pesos α, β ficam em `ai_policies` (name="retrieval_weights_v2"). Se não existir, default $\alpha=0.8, \beta=0.2$.

```

// server/ai/hybrid-score.ts
import { embedTexts } from "../embeddings.text";
import { knnText, knnImage, getChunksByIds } from "../vector-store";
import { db } from "../db";
import { aiPolicies } from "@shared/schema.ai.core";
import { and, eq } from "drizzle-orm";

type Weights = { alpha:number; beta:number };

```

```

async function getWeights(tenantId: string): Promise<Weights> {
  const rows = await db.select().from(aiPolicies)
    .where(and(eq(aiPolicies.tenantId, tenantId), eq(aiPolicies.name, "retrieval_weights_v2"))).limit(1);
  if (!rows.length) return { alpha: 0.8, beta: 0.2 };
  try { const w = JSON.parse(rows[0].dataJson); return { alpha: w.alpha ?? 0.8, beta: w.beta ?? 0.2 }; }
  catch { return { alpha: 0.8, beta: 0.2 }; }
}

export async function hybridRetrieveBase(tenantId: string, queryText: string, imageQueryVec?:
number[], k=12) {
  const W = await getWeights(tenantId);

  // Texto obrigatoriamente disponível
  const [qv] = await embedTexts([queryText]);
  const vText = await knnText(tenantId, qv, 60); // lista {chunkId, score}

  // Imagem opcional (quando houver vetor de consulta de imagem)
  const vImage = imageQueryVec ? await knnImage(tenantId, imageQueryVec, 60) : [];

  // Index por chunkId
  const map = new Map<string, { text:number; image:number }>();
  for (const r of vText) map.set(r.chunkId, { text: r.score, image: 0 });
  for (const r of vImage) map.set(r.chunkId, { ...(map.get(r.chunkId) || { text:0, image:0 }), image:
r.score });

  // Score final
  const arr = [...map.entries()].map(([id, s]) => ({
    id, score: W.alpha*(s.text||0) + W.beta*(s.image||0), textScore: s.text||0, imageScore: s.image||0
  })).sort((a,b)=>b.score-a.score).slice(0, k);

  const chunks = await getChunksByIds(arr.map(a=>a.id));
  // retorno com "evidência"
  return arr.map(a => ({
    ...a,
    chunk: chunks.find(c => c.id === a.id)
  }));
}

```

6) Ingest multimodal — /server/ai/ingest.ts

- **Texto:** chunking + embeddings + ANN (text)
- **Imagem:** recebe imageUri e **embedding visual** + ANN (image)
- (Etapa 2: parsers robustos e OCR virão aqui)

```

// server/ai/ingest.ts
import { db } from "../db";
import { aiDocuments, aiChunks } from "@shared/schema.ai.core";
import { embedTexts } from "../embeddings.text";
import { upsertTextEmbedding, upsertImageEmbedding } from "../vector-store";
import { Tensor } from "onnxruntime-node";
import { preprocessImageRGB, embedImages } from "../embeddings.image";

function chunkText(t: string, maxChars = 1200, overlap = 120) {
  const out:string[] = [];
  let i=0;
  while (i < t.length) {
    const end = Math.min(t.length, i+maxChars);
    out.push(t.slice(i, end));
    i = end - overlap;
    if (i < 0) i = 0;
    if (i >= t.length) break;
  }
  return out;
}

```

```

    }

export async function ingestText(tenantId: string, { source, uri, title, text, meta }:
  { source:string, uri?:string, title?:string, text:string, meta?:any }) {

  const [doc] = await db.insert(aiDocuments).values({
    tenantId, source, uri: uri || null, title: title || null, metaJson: meta? JSON.stringify(meta):
null
  }).returning();

  const chunks = chunkText(text);
  const vecs = await embedTexts(chunks);

  for (let i=0;i<chunks.length;i++) {
    const [ch] = await db.insert(aiChunks).values({
      tenantId, documentId: doc.id, modality: "text", pos: i,
      text: chunks[i], imageUri: null, tokens: chunks[i].split(/\s+/).length, metaJson: null
    }).returning();
    await upsertTextEmbedding(tenantId, ch.id, vecs[i], "minilm-384-onnx");
  }
  return { documentId: doc.id, chunks: chunks.length };
}

/** Ingest de imagem com tensor já pré-processado (RGB NCHW) */
export async function ingestImage(tenantId: string, { source, uri, title, rgbaData, width, height,
meta }:
  { source:string, uri?:string, title?:string, rgbaData:Uint8ClampedArray, width:number,
height:number, meta?:any }) {

  const [doc] = await db.insert(aiDocuments).values({
    tenantId, source, uri: uri || null, title: title || null, metaJson: meta? JSON.stringify(meta):
null
  }).returning();

  const input = preprocessImageRGB(rgbaData, width, height); // Tensor NCHW
  const vecs = await embedImages([input]); // retorna [vector]
  const vector = vecs[0];

  const [ch] = await db.insert(aiChunks).values({
    tenantId, documentId: doc.id, modality: "image", pos: 0,
    text: null, imageUri: uri || null, tokens: 0, metaJson: meta? JSON.stringify(meta): null
  }).returning();

  await upsertImageEmbedding(tenantId, ch.id, vector, "mobilenet-onnx");
  return { documentId: doc.id, chunks: 1 };
}

```

7) Rotas HTTP — /server/ai/routes.ts

- /api/ai/ingest → texto
- /api/ai/ingest.image → imagem RGBA (upload simples)
- /api/ai/ask.base → busca híbrida **texto+imagem** (nesta etapa só texto como consulta; imagem query virá na Etapa 2 com upload/URL)

```

// server/ai/routes.ts
import type { Express, Request } from "express";
import { ingestText, ingestImage } from "../ingest";
import { hybridRetrieveBase } from "../hybrid-score";
import multer from "multer";

const upload = multer({ storage: multer.memoryStorage() });

function ctx(req: Request) {
  const tenantId = (req as any).tenantId || process.env.PRIMARY_TENANT_ID!;
  const userId = (req.headers["x-user-id"] as string) || "web";

```

```

    return { tenantId, userId };
  }

export function registerAiRoutesV2(app: Express) {

  app.post("/api/ai/ingest", async (req, res) => {
    try {
      const c = ctx(req);
      const { source="manual", uri, title, text, meta } = req.body || {};
      if (!text) return res.status(400).json({ error: "text required" });
      const r = await ingestText(c.tenantId, { source, uri, title, text, meta });
      res.json(r);
    } catch (e:any) { res.status(400).json({ error: e?.message }); }
  });

  app.post("/api/ai/ingest.image", upload.single("image"), async (req, res) => {
    try {
      const c = ctx(req);
      if (!req.file) return res.status(400).json({ error: "image required" });
      // req.file.buffer é RGBA? Em produção, faça decode (PNG/JPEG→RGBA). Aqui assumimos RGBA já.
      // Se vier binário PNG/JPEG, decodificaremos na Etapa 2 (parser completo).
      const width = Number(req.body.width || 224);
      const height = Number(req.body.height || 224);
      const rgba = new Uint8ClampedArray(req.file.buffer);
      const { source="file", uri, title, meta } = req.body || {};
      const r = await ingestImage(c.tenantId, { source, uri, title, rgbaData: rgba, width, height,
meta });
      res.json(r);
    } catch (e:any) { res.status(400).json({ error: e?.message }); }
  });

  app.post("/api/ai/ask.base", async (req, res) => {
    try {
      const c = ctx(req);
      const { query, k=12 } = req.body || {};
      if (!query) return res.status(400).json({ error: "query required" });
      const r = await hybridRetrieveBase(c.tenantId, query, undefined, k);
      res.json(r);
    } catch (e:any) { res.status(400).json({ error: e?.message }); }
  });

}

```

No **/server/routes.ts** principal, registre:

```

// server/routes.ts
import type { Express } from "express";
import { registerAiRoutesV2 } from "../ai/routes";

export function registerRoutes(app: Express) {
  // seu middleware Single-Tenant aqui
  app.use("/api", (req, _res, next) => { (req as any).tenantId = process.env.PRIMARY_TENANT_ID!;
next(); });
  registerAiRoutesV2(app);
}

```

8) Matemática – por que esta fundação é sólida?

8.1 Embeddings (texto)

- **Encoder BERT-like** $\text{ftext:RT} \times \text{dtok} \rightarrow \text{RD}$
- Pooling **média** sobre *tokens válidos* (máscara de atenção), seguido de **normalização L2**
- Similaridade por **cos seno**:

$$\cos(u,v) = \frac{\|u\| \cdot \|v\|}{\langle u,v \rangle}$$

8.2 Embeddings (imagem)

- **Encoder CNN/Vision ONNX** $\text{fimg:RH}\times\text{W}\times 3\rightarrow\text{RD}'$
- **GAP** (Global Average Pooling) quando necessário; **L2**
- Similaridade por cosseno

8.3 HNSW (ANN)

- Grafo hierárquico de *small-world*: busca $O(\log N)$ aproximada com parametrização:
 - M (grau médio), $efConstruction$ (qualidade construção), $efSearch$ (recall da busca)
- **Incremental**: cada vetor inserido $\rightarrow addItem()$.
- **Checkpoint**: grava `.bin` + `.meta.json` periodicamente e em **SIGINT/SIGTERM**.

8.4 Score híbrido base

$$S=\alpha\cdot S_{\text{text}}+\beta\cdot S_{\text{image}}, \alpha+\beta=1$$

- Nesta etapa, garantimos **robustez multimodal** mesmo sem BM25/grafos/frescor.
- Na Etapa 2, incorporamos:
 - $\gamma\cdot S_{\text{graph}}$ (grafo semântico/entidades),
 - $\delta\cdot S_{\text{fresh}}$ (tempo de atualização),
 - $\zeta\cdot S_{\text{auth}}$ (autoridade/fonte).

9) Teste rápido (sanity) — em 5 passos

1. **Migrate** (adicione o `schema.ai.core.ts` nas migrations Drizzle).

2. **Coloque modelos**:

- `models/minilm.onnx`, `models/vocab.txt`
- `models/mobilenet.onnx`, `models/imagenet_labels.json`

3. **Suba o server** (Replit/Node).

4. **Ingest texto**:




```
curl -X POST http://localhost:3000/api/ai/ingest \
-H "Content-Type: application/json" \
-d '{"source":"manual","uri":"kb://intro","title":"Intro","text":"A Yes You Deserve é uma
plataforma ... (cole um texto longo)","meta":{"sourceRank":0.9}}'
```

5. **Consultar**:

```
curl -X POST http://localhost:3000/api/ai/ask.base \
-H "Content-Type: application/json" \
-d '{"query":"Quero saber sobre o marketplace e tours em Sintra"}'
```

(Para imagem, na Etapa 2 colocaremos **decodificação PNG/JPEG** e **OCR** completos; aqui a rota aceita um buffer RGBA já pronto.)

O que fica garantido ao final desta Etapa 1

-  **Fundação multimodal** sólida (texto + imagem) 100% local, com ONNX;
-  **Busca ANN HNSW incremental** por tenant e modalidade, com checkpoint automático;
-  **Score híbrido** (texto + imagem), já compatível com expansão para BM25/grafos/frescor/autoridade;

- 🚀 **Rotas HTTP** simples e claras para ingest/consulta;
- ✅ Compatível com Single-Tenant e fácil de integrar ao CRM/ERP/Marketplace.