GUIA PRÁTICO PARA DESENVOLVEDORES

IA NO SEU SISTEMA

Do primeiro token ao RAG com citações: engenharia prática e determinística de LLMs em produção.

MARCOS MORAES · OUTUBRO DE 2026

01

Made with GAMMA

ARQUITETURA DE PRODUÇÃO

SOBRE ESTE LIVRO

Como colocar um LLM dentro de um sistema de verdade: do token ao RAG com citações verificáveis exposto por API HTTP segura.

ORIGEM EM DOIS PILARES

O livro nasceu da união entre os fundamentos mecânicos da computação neural (tokens, BPE, limites de contexto, precificação, amostragem e contratos tipados) e o **DocMind**, um projeto de referência construído em fases para demonstrar cada trade-off em código real.

PÚBLICO-ALVO

Feito para engenheiros de software e arquitetos de sistemas. Não exige conhecimento prévio de matemática pesada nem machine learning: trata o modelo de linguagem como um subsistema probabilístico de rede, integrado com testes e observabilidade.

STACK TECNOLÓGICA

Código limpo e tipado com **TypeScript e Node.js**, integrando com Claude API (Anthropic), PostgreSQL com extensão **pgvector**, EmbeddingGemma local, Fastify como gateway HTTP, Zod para parsing e Docker em toda a infraestrutura.

A REGRA DE OURO

Meça com os seus próprios dados. Evite heurísticas de redes sociais: meça distribuição de tokens, registre custo real de chamadas e teste a taxa de acerto de cada prompt com benchmarks reprodutíveis.

RUNTIME

Node.js 24 + TypeScript

ARMAZENAMENTO VETORIAL

PostgreSQL + pgvector

VALIDAÇÃO DE CONTRATOS

Fastify + Zod schemas

Made with GAMMA

MECÂNICA DE TOKENIZAÇÃO

CAPÍTULO 1: O TOKEN É A UNIDADE DE CUSTO, LIMITE E LATÊNCIA

Modelos não operam sobre texto: operam sobre inteiros gerados por um tokenizador.

SUB-PALAVRAS E BPE

Criado em 1994 para compressão e adaptado para LLMs, o Byte Pair Encoding funde iterativamente os pares de bytes mais frequentes. Em "baixo", "baixar" e "caixa" funde ai e depois aix; "faixa" vira f + aix + a.

INGLÊS X PORTUGUÊS

Inglês: ~4 caracteres por token ("performance": 1 token). Português: ~2,5 a 3 ("desempenho": [des] + [empen] + [ho]).

O TRIÂNGULO DE IMPACTO

01 CUSTO DE INFRAESTRUTURA

A fatura soma exatamente os tokens enviados e gerados.

02 JANELA DE CONTEXTO

O limite por requisição é medido em slots de tokens, não em megabytes.

03 LATÊNCIA E TTFT

O Time to First Token depende da leitura do prompt; a geração depois é sequencial.

Regra prática: contar caracteres estima mal o orçamento; use o campo usage da resposta.

Made with GAMMA

CAPÍTULO 2: A FATURA É POR TOKENS PROCESSADOS

Custo = entrada × preço de entrada + saída × preço de saída

SAÍDA

Custa cerca de **5× a entrada**; o raciocínio interno (*thinking*) é cobrado como saída.

HISTÓRICO

System prompt, histórico e contexto do RAG entram como entrada e se acumulam.

Modelo	Entrada	Saída	Leitura de cache
Opus 5.5	4,00	20,00	0,20
Sonnet 5.5	2,00	10,00	0,20
Haiku 5.5	0,10	0,50	0,01

US$ por 1 milhão de tokens, outubro de 2026: o maior modelo custa 40× o menor na entrada. Confira a tabela oficial.

Use o campo usage para medir, nunca estime por caracteres. Registre consumo por funcionalidade, use prompt caching e escolha o modelo pela tarefa.

Made with GAMMA

MECÂNICA DE INFERÊNCIA

CAPÍTULO 3: COMO O MODELO ESCOLHE O PRÓXIMO TOKEN

A cada passo, a rede gera logits que o softmax transforma em probabilidades. Ex.: após *"Design Pattern:"*, "Constructor" 65%, "Singleton" 20%, "Factory" 10%, outros 5%.

TEMPERATURE (T)

Escala os logits antes do softmax. Em **T = 0** a escolha é gulosa (*greedy*) e determinística; valores altos geram variedade, criatividade ou alucinações.

TOP_P (NUCLEUS SAMPLING)

Sorteia apenas entre os tokens cuja soma de probabilidades atinge p (ex.: 0.9).

PENALIDADES DE REPETIÇÃO

Frequency: proporcional às repetições. **Presence:** binária, se o token já apareceu.

Regra de ouro: ajuste temperature ou top_p, nunca os dois ao mesmo tempo.

Caso de uso	Temperature	Top_p	Frequency	Presence
Código / JSON	0,0	1,0	0,0	0,0
Análise / extração	0,0	1,0	0,0	0,0
Refatoração com alternativas	0,5	0,9	0,2	0,0
Brainstorming / redação livre	0,8	0,9	0,5	0,6

Temperatura alta gasta mais e gera JSON inválido, com retry pago em dobro. Em modelos com raciocínio, controle o effort (low a max).

Made with GAMMA

CONFIABILIDADE DE SOFTWARE

CAPÍTULO 4: CONTRATOS DE DADOS

4.1 COMO OS PROVEDORES FAZEM ISSO

Sem contrato

“O usuário se chama João, tem 30 anos e mora em SP.”

Com contrato

{ "nome": "João", "idade": 30, "cidade": "SP" }

Mecanismo	Como funciona	Garantia
Formato de saída nativo	response_format json_schema (OpenAI) e output_config.format (Anthropic): JSON Schema com constrained decoding	Formato na geração
Tool calling como contrato	Ferramenta fictícia com input_schema: o modelo preenche um formulário tipado	Formato na geração
Validação no backend	Zod ou JSON Schema no seu servidor	Defesa em profundidade: nunca dispense

As duas primeiras garantem o formato na geração; a terceira é defesa em profundidade e nunca deve ser dispensada.

Made with GAMMA

CONTRATOS DE DADOS

CAPÍTULO 4: FORMATO NATIVO COM OPENAI

4.2 EXEMPLO COM FORMATO NATIVO (OPENAI)

```
import OpenAI from "openai";
import { z } from "zod";
import zodToJsonSchema from "zod-to-json-schema";

const CurriculoSchema = z.object({
  nome: z.string(),
  email: z.string().email(),
  anos_experiencia: z.number().int().nonnegative(),
  habilidades: z.array(z.string()),
  disponivel: z.boolean(),
});

const response = await client.chat.completions.create({
  model: "gpt-4o",
  messages: [
    { role: "system", content: "Extraia os dados estruturados do currículo fornecido." },
    { role: "user", content: "João Silva, joao@email.com, 5 anos de experiência em Python..." },
  ],
  response_format: {
    type: "json_schema",
    json_schema: { name: "Curriculo", strict: true, schema: zodToJsonSchema(CurriculoSchema, "Curriculo") },
  },
});
const curriculo = CurriculoSchema.parse(JSON.parse(response.choices[0].message.content!));
```

Modo estrito: use nullable() e enum no lugar de optional() e uniões.

Made with GAMMA

CONTRATOS DE DADOS

CAPÍTULO 4: TOOL CALLING COM ANTHROPIC

4.3 EXEMPLO COM TOOL CALLING (ANTHROPIC)

```
const response = await client.messages.create({
  model: "claude-opus-5-5",
  max_tokens: 1024,
  tools: [{
    name: "extrair_curriculo",
    description: "Extrai dados de um currículo.",
    input_schema: {
      type: "object",
      properties: {
        nome: { type: "string" }, email: { type: "string" },
        anos_experiencia: { type: "number" },
        habilidades: { type: "array", items: { type: "string" } },
        disponivel: { type: "boolean" },
      },
      required: ["nome", "email", "anos_experiencia", "habilidades", "disponivel"],
    },
  }],
  tool_choice: { type: "tool", name: "extrair_curriculo" },
  messages: [{ role: "user", content: "João Silva, joao@email.com, ..." }],
});
```

```
const bloco = response.content.find(
  (b) => b.type === "tool_use"
);
const curriculo = CurriculoSchema.parse(bloco!.input);
```

O tool_choice força o uso da ferramenta. No Capítulo 7, a forma mais simples: o formato nativo da Anthropic com helper do SDK que recebe o schema Zod direto.

Made with GAMMA

CONTRATOS DE DADOS

CAPÍTULO 4: VALIDAÇÃO COM ZOD

4.4 DEFESA EM PROFUNDIDADE

Mesmo com structured outputs, valide sempre no backend: o modelo pode ter bugs, você pode trocar de provedor ou usar um modelo menor sem constrained decoding.

```
function parseSeguro<T>(schema: z.ZodType<T>, data: unknown):
 { success: true; data: T } | { success: false; errors: string[] } {
 const result = schema.safeParse(data);
 if (result.success) return { success: true, data: result.data };
 return {
  success: false,
  errors: result.error.errors.map((e) => `${e.path.join(".")}: ${e.message}`),
 };
}
```

Nunca dispense a validação: ela é a última barreira entre a LLM e o sistema.

Made with GAMMA

CONTRATOS DE DADOS

CAPÍTULO 4: O SCHEMA É A FONTE ÚNICA DA VERDADE

4.5 O SCHEMA COMO FONTE ÚNICA DA VERDADE

```
const AnaliseSchema = z.object({
  sentimento: z.enum(["positivo",
"negativo", "neutro"]),
  score: z.number().min(0).max(1),
  topicos:
z.array(z.string()).max(5),
  resumo: z.string().max(200),
});
type Analise = z.infer<typeof
AnaliseSchema>;
const schemaParaLLM =
zodToJsonSchema(AnaliseSchem
a);
```

Um schema Zod gera três saídas:

- JSON Schema para a LLM
- Tipo TypeScript (z.infer)
- Validação em runtime (.parse())

Regra de ouro: a LLM é a entrada; o Zod é a fronteira. Nenhum dado da LLM entra no sistema sem passar pelo schema.

4.6 EXERCÍCIO

Implemente extrairNotaFiscal(texto: string): receba o texto livre de uma nota fiscal e devolva um objeto validado com numero_nota, data_emissao (YYYY-MM-DD), valor_total, itens (descrição, quantidade, valor unitário) e cnpj_emitente (14 dígitos). Bônus: o mesmo schema funciona em dois provedores sem duplicar código.

Made with GAMMA