Build Your Own AI Chatbot (No OpenAI): End-to-End Guide

A practical, production-focused playbook to design, deploy, and operate your own LLM-backed chatbot using open-source models and a custom backend.

Table of Contents

1. Scope & Persona

- Define target users, supported tasks, tone, languages, and measurable KPIs (accuracy, CSAT, deflection).
- Write down non-goals and safety boundaries (e.g., no medical/financial advice).

2. Architecture Overview

- Frontend → Backend Gateway (auth/rate limit) → Orchestrator (prompts/tools/RAG) → Inference Server (LLM) → Vector DB → Observability.
- Gateway: FastAPI/Express; Orchestrator: Python service; Inference: vLLM/TGI; Data: Postgres/Redis/Vector DB; Telemetry: Prometheus/Grafana.

3. Choose & Host an Open-Source Model

- Models: Llama 3.x, Mistral/Mixtral, Gemma, Qwen, Phi-3. Check license & quality vs. size.
- Serving: vLLM (high throughput), TGI (streaming), llama.cpp (CPU/edge).
- Hardware: 7B fits ~8–16 GB VRAM (quantized); 13B ~20–24 GB; 70B needs multi-GPU.

4. Backend API Design (No OpenAI)

- Expose /v1/chat/completions and stream tokens via SSE/WebSockets.
- Add JWT/OAuth2, request validation, tenant quotas, and structured logs.
- Return request_id, token counts, model version in responses.

5. Retrieval-Augmented Generation (RAG)

- Ingest → chunk (400–1000 tokens) → embed → index (Qdrant/Weaviate/pgvector) → hybrid search + rerank → prompt with citations.
- Cache frequent queries and invalidate on data changes.

6. Fine-Tuning & Preference Optimization

- SFT on domain dialogs; LoRA/QLoRA for efficiency; DPO/ORPO for alignment.
- Track dataset lineage; keep separate base and tuned checkpoints.

7. Safety, Guardrails & Compliance

- Input/output classifiers, blocklists, PII scrubbing, refusal templates.
- Per-tenant policies, audit logs, and region-aware data residency.

8. Evaluation & Benchmarks

- Offline: domain QA accuracy, faithfulness; Online: A/B tests, CSAT.
- Golden sets + regression gates to block unsafe/low-accuracy releases.

9. Frontend Integration (Web & Mobile)

- React/Next.js with SSE; RN/Flutter for mobile; Slack/Teams connectors.
- Show source citations, feedback buttons, and session history.

10. Observability, Logging & Cost Control

- Metrics: QPS, p95 latency, tokens/s, cache hit; Tracing with OpenTelemetry.
- Autoscale, quotas, and batch jobs on spot instances where feasible.

11. CI/CD & Release Management

- Containers + Helm; Canary/blue-green deploys; IaC with Terraform.
- Data pipeline CI (schema checks) and model registry with versioning.

12. Security & Data Governance

- mTLS between services; Vault/KMS for secrets; least-privilege RBAC.
- PII minimization, retention rules, backups, and DR playbooks.

13. Reference Stack & BoM

- Inference: vLLM/TGI; Orchestrator: FastAPI + tools; Vector: Qdrant/pgvector; Embeddings: e5/bge/GTE; Observability: Prometheus/Grafana.

14. Step-by-Step Setup Checklist

- Provision GPU/CPU; install drivers.
- Serve model with vLLM/TGI; verify /health.
- Stand up API gateway with SSE streaming.
- Create vector DB; implement ingestion, embeddings, hybrid search.
- Add safety filters and audit logging.
- Build frontend with streaming + citations.
- Run evals and red-team tests; set regression gates.
- Containerize and deploy with CI/CD.

Appendix: Minimal FastAPI Backend

```
from fastapi import FastAPI, Request
from fastapi.responses import StreamingResponse
import httpx, json, os

app = FastAPI()

@app.post('/v1/chat/completions')
async def chat(req: Request):
    body = await req.json()
    async with httpx.AsyncClient(timeout=60) as client:
        r = await client.post(os.environ['INFER_URL'], json=body)
        r.raise_for_status()
        data = r.json()
        return
{'choices':[{'message':{'content':data['choices'][0]['message']['content']}}]}
```

Appendix: Docker Compose Sketch

```
services:
  gateway:
    build: ./gateway
    ports: ['8080:8080']
    environment:
      - INFER_URL=http://inference:8000/v1/chat/completions
  inference:
    image: vllm/vllm-openai:latest
    command: --model mistralai/Mistral-7B-Instruct-v0.3 --gpu-memory-
utilization 0.9
    ports: ['8000:8000']
  qdrant:
    image: qdrant/qdrant:latest
    ports: ['6333:6333']
  postgres:
    image: postgres:16
    environment:
      - POSTGRES_PASSWORD=secret
    ports: ['5432:5432']
```