Guía técnica: Construcción de un Chatbot Bancario con LLM/NLP (Python-first)

Este documento describe de forma práctica y enfocada cómo construir un chatbot bancario conversacional basado en LLM/NLP. Se centra exclusivamente en la arquitectura, componentes técnicos, flujos de datos, herramientas recomendadas y pasos de implementación necesarios para desarrollar, desplegar y operar la solución.

Resumen del objetivo

Construir un chatbot que responda libremente a consultas relacionadas con productos y procesos bancarios, explique procedimientos, ejecute consultas (saldo, movimientos) tras autenticación, y escale interacciones a un sistema de tickets administrado por un panel Admin/Agent. La solución debe usar retrieval-augmented generation (RAG), NLU (intents + NER), y estar implementada con Python como lenguaje principal.

Arquitectura general

Arquitectura propuesta (componentes principales):

- Frontend conversacional (widget web / mobile).
- API Gateway / BFF (FastAPI en Python).
- Core Chat Service (orquestador RAG, NLU, Dialog Manager).
- Vector DB para embeddings (Qdrant / Milvus / Pinecone / FAISS).
- Ingestor de conocimiento (ETL y generador de embeddings).
- Integración segura con Core Banking (microservicio de APIs).
- Sistema de Ticketing / Admin / Agent (integrado o externo).
- Analytics & Dashboard (métricas en tiempo real).
- Observabilidad y seguridad (logs, auditoría, DLP).

Tecnologías recomendadas (stack concreto)

Backend y orquestación:

• FastAPI (API/BFF) — Python. • Uvicorn/Gunicorn para ASGI. • Celery/Redis o RQ para tasks asíncronas.

NLU & LLM orchestration:

• LangChain o LlamaIndex para pipelines RAG. • Transformers / Hugging Face / sentence-transformers para embeddings y modelos cuando se self-host. • spaCy para NER y procesamiento de entidades.

Vector DB:

• Qdrant, Milvus, Weaviate, Pinecone o RedisVector. FAISS para soluciones on-prem.

LLM / Inferencia:

• Inicial: OpenAI/Anthropic/Cohere (rápido para POC). • Para uso ‘ilimitado’: Llama 2 / Mistral u otros modelos open-source desplegados en infra propia con vLLM/Triton/Hugging Face Inference.

Bases de datos y cache:

• PostgreSQL para metadatos y tickets. • Redis para sesiones y cache.

Frontend y Admin:

• React + Tailwind (widget y paneles). • WebSocket para realtime (estadísticas, typing, actualizaciones).

Observability:

• Prometheus + Grafana, Sentry, ELK/Opensearch para logging y trazabilidad.

Seguridad y secretos:

• Hashicorp Vault o KMS (AWS/GCP/Azure) para secretos y gestión de claves.

Componentes y responsabilidades (detallado)

Ingestor de conocimiento (Knowledge Ingestion)

• Fuentes: PDFs, KB, documentación de producto, transcripciones anonimizadas. • Pipeline: extracción → normalización → chunking semántico → embeddings → indexación en Vector DB. • Metadatos: origen, fecha, confidencialidad, producto.

Retriever + RAG

• Retriever: búsqueda semántica por embeddings con filtros por metadata (producto, fecha, confidencialidad). • Opcional: reranker (cross-encoder). • Generator: plantilla de prompt que incluye instrucciones de seguridad y cita de fuentes. • Resultado: respuesta fundamentada con referencias a documentos indexados.

NLU & Dialog Manager

• Intent classifier (modelo ligero fine-tuned o clasificación sobre embeddings). • NER (spaCy + reglas) para identificar y enmascarar datos sensibles. • Slot filling y manejo de estados multi-turno (sessions en Redis). • Lógica de fallback: thresholds para derivar a humano.

Integración con Core Banking

• Microservicio seguro que expone llamadas a APIs internas (solo tras verificación/auth). • Endpoints típicos: consultar saldo, movimientos, iniciar pagos simulados en sandbox, bloquear tarjeta. • Políticas: rate-limits, auditing y verificación MFA para acciones sensibles.

Escalación y Ticketing (Admin/Agent)

• Modelo: cuando el chatbot detecta necesidad humana (baja confianza, PII, transacciones complejas) crea un ticket con el historial. • Opciones: integrar con Zendesk/Jira/ServiceNow o

implementar un microservicio propio (Postgres + panel en React). • Roles: Agente y Supervisor con RBAC y SLAs.

Feedback y mejora continua

• Recolectar CSAT/NPS tras interacciones. • Guardar feedback y etiquetarlo para retraining. • Pipeline de A/B testing para prompts y variaciones de respuesta.

Seguridad, privacidad y cumplimiento

Puntos clave a implementar:

- Autenticación fuerte (OAuth2, OIDC, MFA) para acciones sensibles.
- DLP y redaction: detectar y enmascarar PAN/CVV/PII en tiempo real.
- No almacenar texto con información sensible sin tokenización.
- Auditoría completa (quién, cuándo, qué).
- TLS/mTLS para comunicaciones internas y externas.
- Revisión de cumplimiento (PCI-DSS, regulaciones locales).

Preparación de datos

• Reunir documentos del banco (productos, contratos, manuales de operación), transcripciones anonimizadas y FAQ. • Anonimizar PII y crear dataset ejemplo de intents y utterances para entrenar el intent classifier. • Diseñar ontología de productos y taxonomía para el indexado semántico.

Diseño de prompts y guardrails

• Plantilla de system prompt que define rol, tono, límites de respuesta y obligue a citar fuentes. • Incluye instrucciones para redaction y comportamiento ante baja confianza (fallback a humano). • Control de temperature y longitud según tipo de consulta.

Pruebas y QA

• Unit tests para pipelines. • Integration tests con mocks de core banking. • Pruebas de seguridad: verificación de redaction y DLP. • Pruebas de usuario en sandbox con agentes reales para validar SLAs y experiencia.

Despliegue y operación

• Contenerizar servicios (Docker). • Orquestación: Kubernetes (recomendado) o servicios gestionados. • Infra para inferencia: GPUs dedicadas para modelos self-hosted o uso de proveedores de inference. • CI/CD: GitHub Actions/GitLab CI con despliegue a staging antes de prod. • Backup y plan de rollback.

Roadmap de implementación (fases)

1. 1. Descubrimiento y mapear intents, fuentes, SLAs y restricciones de cumplimiento.
2. 2. POC RAG mínimo: ingest de 1-2 documentos + LLM cloud para validar flujo básico.
3. 3. Implementar NLU y Dialog Manager (intents, NER, sesiones).
4. 4. Integrar llamadas read-only al core banking en sandbox.
5. 5. Implementar escalación y ticketing + Agent UI.
6. 6. Añadir analytics y dashboard en tiempo real.
7. 7. Hardening de seguridad y pruebas de cumplimiento.
8. 8. Migración parcial o total a self-hosted LLM si se desea control / coste fijo.

Ejemplo mínimo de endpoints (MVP)

• POST /v1/chat → envía mensaje al chatbot y recibe respuesta (texto, acciones, fuentes).

• POST /v1/auth/verify → verifica identidad para operaciones sensibles.

• POST /v1/tickets → crea ticket de escalación con contexto de conversación.

• GET /v1/session/{id} → recupera historial de la sesión.

Snippet básico (estructura en Python - skeleton)

El siguiente esqueleto muestra la idea general. Implementar retriever, embeddings y llamada al LLM es necesario.

from fastapi import FastAPI from pydantic import BaseModel app = FastAPI() class ChatRequest(BaseModel): session_id: str user_id: str message: str @app.post("/v1/chat") async def chat(req: ChatRequest): # 1. validar sesión # 2. intent + NER # 3. recuperación de documentos (vector DB) # 4. construcción de prompt y llamada LLM # 5. aplicar guardrails y devolver respuesta return {"text": "Respuesta generada (ejemplo)", "sources": []}

Recomendación para IA 'ilimitada'

• Ruta 1 (recomendado para control): Self-hosted LLMs (Llama 2, Mistral, etc.) desplegados en infra propia con servidores GPU e inference server (vLLM, Triton, Hugging Face Inference). Esto permite uso ilimitado sujeto a la capacidad de infra.

• Ruta 2 (híbrida): Empezar con servicios cloud (OpenAI, Anthropic) para desarrollo rápido y, en paralelo, preparar la migración a infra propia para producción con alto tráfico.

Riesgos clave y mitigaciones técnicas (breve)

- Hallucinations: mitigar con RAG + citar fuentes + threshold para fallback.
- Data leakage: DLP, redaction y no almacenar PII.
- Costos de inferencia: planificar infra o modelo híbrido.

- Cumplimiento: pruebas y auditorías periódicas.

Conclusión

Esta guía presenta una hoja de ruta técnica y un listado de herramientas y prácticas para construir un chatbot bancario robusto con LLM/NLP usando Python como eje principal. Siguiendo las fases propuestas y aplicando los guardrails de seguridad y pruebas recomendadas, es posible desplegar una solución que ofrezca respuestas abiertas fundamentadas y una integración segura con los sistemas bancarios existentes.