**Research Agent: Agente de IA con Búsqueda Web y Cálculo en Tiempo Real**

Agente de IA construido en Python desde cero (sin frameworks como LangChain) que responde preguntas usando Gemini 3.5 Flash mediante el patrón ReAct (Reasoning + Acting), decidiendo dinámicamente en cada paso si responder directamente o ejecutar una herramienta externa antes de continuar. Implementa dos tools: búsqueda web en tiempo real vía la API de Tavily (para datos actuales que exceden el conocimiento del modelo, como precios o eventos recientes) y una calculadora matemática que evalúa expresiones parseando el árbol sintáctico con el módulo `ast` de Python en vez de `eval()`, evitando ejecución de código arbitrario. El historial de conversación se reconstruye en cada iteración del loop (contents con roles user/model), permitiendo encadenar múltiples herramientas en una sola consulta (ej: buscar un dato y luego calcular con él), con un límite de 6 iteraciones como salvaguarda contra loops infinitos. Cada tool está desacoplada del loop principal mediante un schema de function-calling (nombre, descripción, parámetros tipados), permitiendo agregar nuevas herramientas sin modificar la lógica central del agente. El diseño prioriza mitigar alucinaciones mediante instrucciones explícitas en el system prompt: citar la fuente de cada dato usado y declarar explícitamente cuando no se encontró información suficiente, en vez de completar la respuesta con conocimiento no verificado. Stack: Python, google-genai SDK, Tavily API, python-dotenv. Tecnologías clave: function calling, patrón ReAct, tool use, integración de APIs externas, manejo de contexto conversacional, mitigación de alucinaciones, evaluación segura de expresiones matemáticas

**Proyecto: Clasificador de Tickets de Soporte con Fine-Tuning de LLM**

Fine-tuning de Llama-3.2-3B-Instruct con QLoRA (cuantización 4-bit + LoRA) para clasificar tickets de soporte al cliente en español en 5 categorías: facturación y pagos, problemas técnicos, cuenta y acceso, consultas de producto, y cancelaciones. Generé un dataset sintético de ~600 ejemplos usando la API de Groq (Llama 3.3 70B), anclado en 39 ejemplos escritos a mano para mantener realismo, con deduplicación y split estratificado train/val/test. Entrené el adapter LoRA (r=16, alpha=32) sobre GPU T4 gratuita de Google Colab, entrenando solo ~1-2% de los parámetros del modelo. Evalué 3 métodos sobre el mismo test set: TF-IDF + Regresión Logística (95.0% accuracy), Llama-3.2-3B zero-shot (53.3% accuracy), y Llama-3.2-3B + QLoRA fine-tuneado (91.7% accuracy). Stack: transformers, peft, bitsandbytes, trl, scikit-learn. Tecnologías clave: fine-tuning, QLoRA, LoRA, cuantización, generación de datos sintéticos, clasificación de texto, evaluación comparativa de modelos.

**DocChat RAG — Asistente de documentación con citas verificadas**

Sistema de preguntas y respuestas sobre documentación técnica (PDFs) que combina recuperación semántica con generación aumentada (RAG), diseñado para responder citando la fuente exacta (archivo y página) de cada afirmación en vez de confiar ciegamente en el modelo. Implementé un pipeline completo de ingestión (chunking por tokens con tiktoken, embeddings locales con Ollama/nomic-embed-text, almacenamiento en ChromaDB) y generación (Groq API con Llama-3.3-70B), con verificación de citas: las referencias que genera el LLM se reemplazan automáticamente por metadata real del vector store, garantizando que la cita nunca esté mal aunque el modelo se equivoque en el número. Para evaluar la calidad más allá de comparación de keywords, construí un pipeline de evaluación con RAGAS que usa un LLM-as-judge (el mismo Groq) para medir faithfulness (qué proporción de la respuesta está realmente respaldada por el contexto recuperado, detectando alucinaciones) y context precision (qué proporción de los chunks recuperados es realmente relevante, detectando ruido en el retriever), sobre un set de preguntas de prueba con fuentes esperadas conocidas. Toda la aplicación (API con FastAPI, frontend con Streamlit, servicio de embeddings) corre containerizada con Docker Compose para despliegue reproducible. Stack: Python, FastAPI, Streamlit, ChromaDB, Ollama, Groq API, RAGAS, Docker. Tecnologías clave: RAG, retrieval semántico, embeddings, LLM-as-judge, evaluación de alucinaciones, citas verificadas, containerización.