RUSHI DESAI

- Problem : 2

  i)

  LLM: B
  Deep Learning: A
  AI (general): A
  ML (classical): A

  ii)

  - User Interface

  Web, mobile, WhatsApp etc., wherein the user sends and receives messages.

  Sends plain text and occasionally metadata like user ID, channel, locale to the backend over HTTP.

  - Backend service

  Exposes a Chat endpoint, handling authentication, rate limiting and routing.

  Normalizes incoming raw messages to a uniform format, such as user_id, message, context, and sends them to the Orchestrator.

  - Orchestration

  Central "brain" - makes the decisions of what to do for each turn: Call LLM directly, run retrieval (RAG), call external tools

  Manages conversation state-history, slots, and user profile-and builds the final prompt for the LLM.

  - LLM Engine:

  An API hosted by someone else - OpenAI, Anthropic, etc.

Can be used zero-shot, few-shot, fine-tuned or as part of an “agent” that reasons and calls tools.

- Knowledge and Tools layer: RAG & APIs

Vector database or search index utilized for Retrieval-Augmented Generation: embed the user query, fetch relevant documents, and inject them into the prompt to ground answers.

Domain APIs, databases, or business services such as booking, CRM, and payments that the orchestrator can call and then feed results back to the LLM for explanation.

- Data stores and memory

Conversation history store such as Redis or PostgreSQL; Long-term user memory (preferences, previous tickets) if needed. Configuration store for prompts, tool definitions and policies.

- Safety, governance and monitoring.

Input/Output filters for PII, toxicity, jailbreaks and policy enforcement. Logging, analytics, collection of feedback and A/B testing to continuously improve prompts, retrieval and model choice.

High -level:

User sends a message via UI - for instance, chat widget.

The backend receives it, performs user authentication, and forwards it to the orchestrator with context.

Orchestrator analyzes the request and decides to:

Call LLM directly:

Run Retrieval against a vector store (RAG)

call external tools/APIs, or

Using rules/fallback flows.

Relevant data is then gathered: documents from RAG, API responses, and previous turns are all combined in a structured prompt.

The LLM will produce a response, sometimes (in more sophisticated agent setups) containing instructions to call more tools.

Safety filters are applied and formatting is done. The final text - along with the rich content, if any - is returned to the UI. Logs and metrics are kept for monitoring, personalization, and continuous improvement. :

iii)

What is a vector database?
Data, like text and images, is changed into embeddings, which are lists of numbers capturing meaning. Similar items are represented by nearby vectors.

The database organizes these vectors and supports similarity search using metrics like cosine similarity or Euclidean distance. This helps return the top-k closest items to a query vector.

It supports use cases such as semantic document search, recommendations, and Retrieval-Augmented Generation for LLM chatbots.

Hypothetical problem

Problem: An internal support chatbot for a mid-sized company answers questions from 5,000 employees using around 100,000 pages of policies, SOPs, and FAQs. It needs fast semantic search, metadata filters (like department and country), and RAG for an LLM.

The scale is moderate, involving millions of vectors but not billions, and it should be self-hosted on standard cloud VMs.

Choice and reasoning

For this case, Qdrant would be a good choice. It is designed for high-performance vector similarity search, offering good recall and low latency for millions to low billions of vectors, using Rust and ANN indexes.

It has rich payload and metadata filtering, which is ideal for filtering documents in RAG workflows by department, country, or document type. It is open source, easy to deploy via Docker/Kubernetes, and integrates well with common LLM/RAG frameworks like LangChain and LlamaIndex.

Alternatives include Pinecone if a fully managed SaaS with minimal operations and strong SLAs is more important than self-hosting, and Weaviate if you need hybrid search using both keywords and vectors, with a built-in schema/GraphQL API. This is also suitable if you want tight integration of structured and vector data.