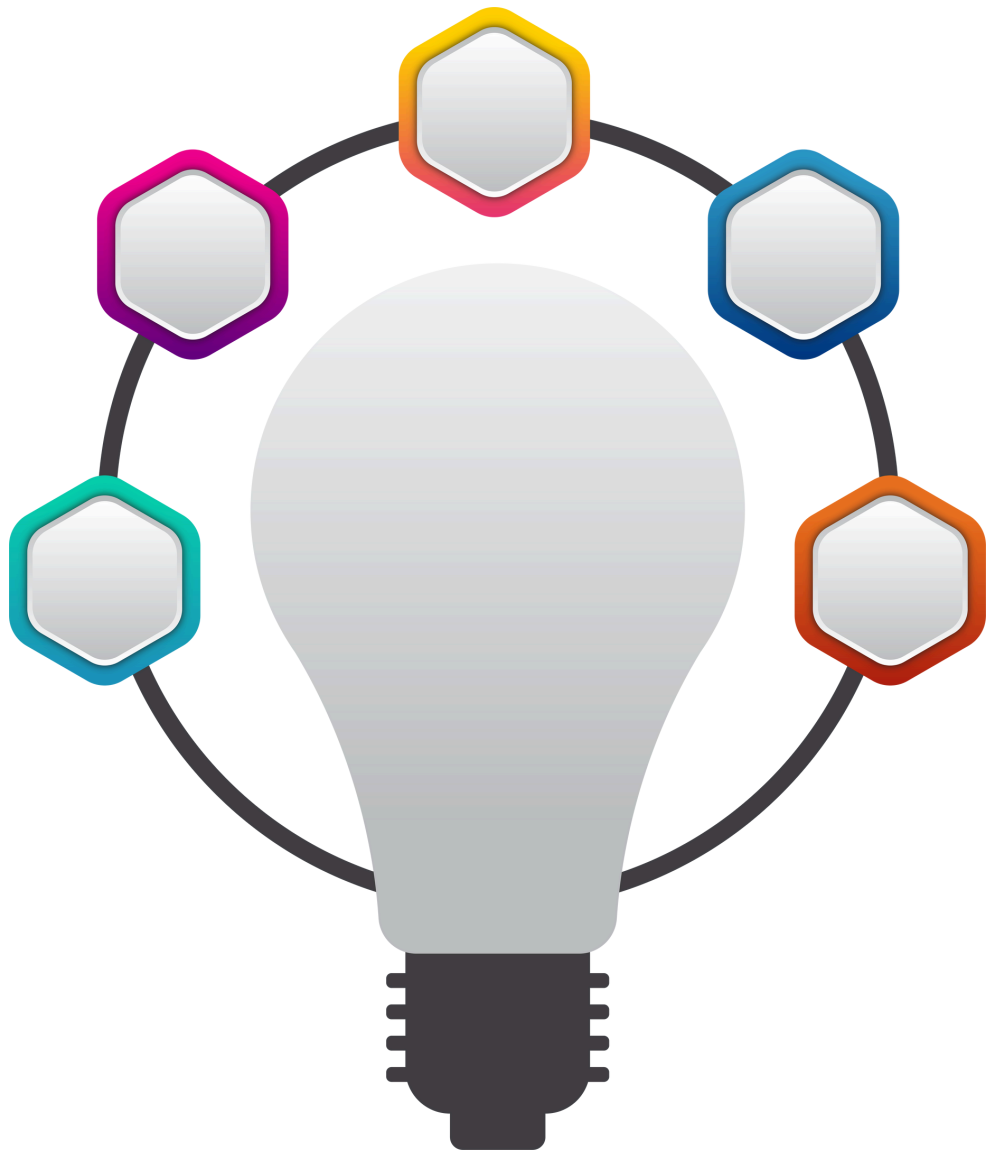
LLM SYSTEM DESIGN

Learn to build scalable LLM Application

Bhavishya Pandit

What is LLM System Design?

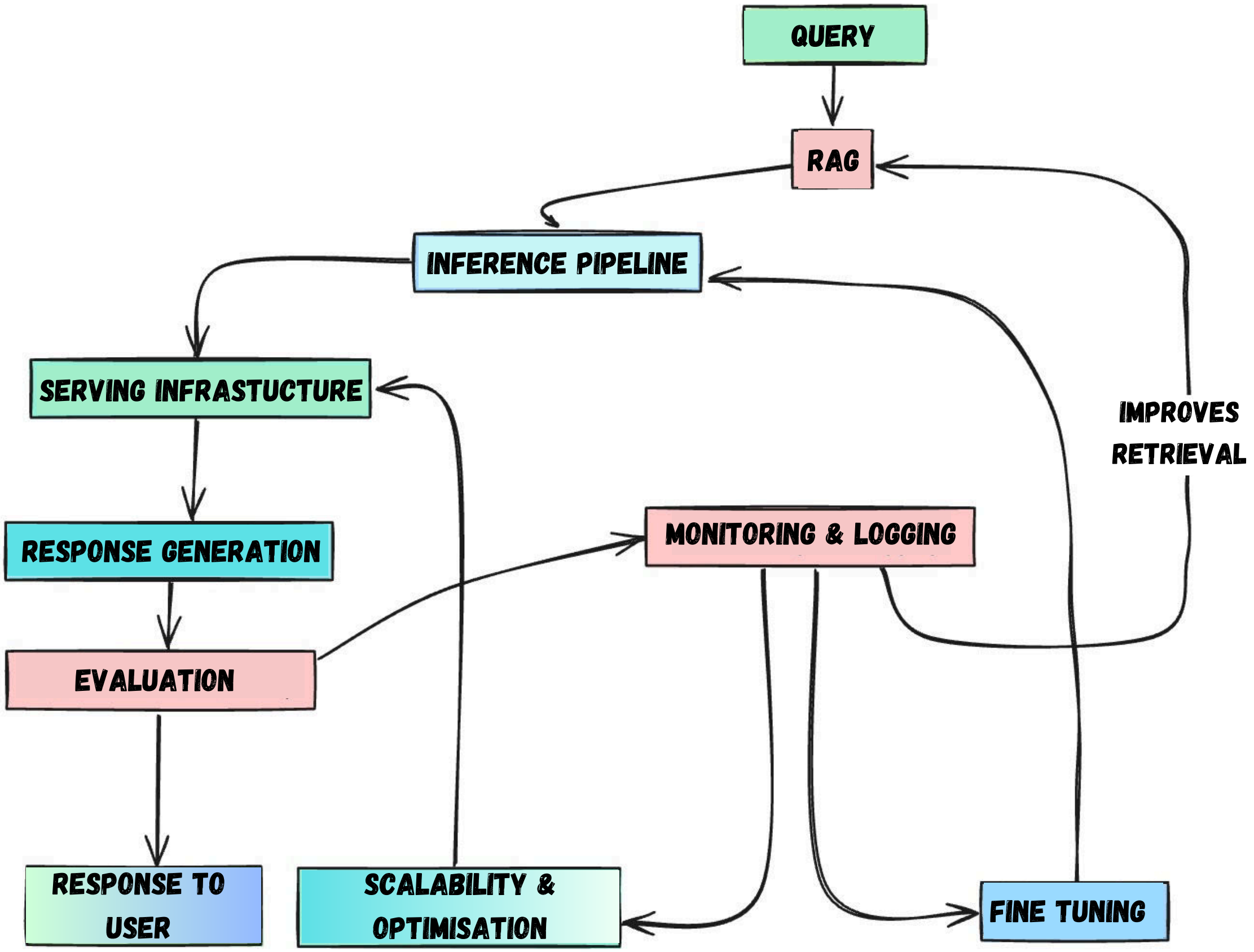
LLM system design is the process of architecting and optimizing large language model applications for efficiency, scalability, and real-world deployment.



1. **Infrastructure planning**: Choosing the right compute resources.
2. **Inference optimization**: Reducing latency and cost using quantization, caching etc.
3. **Prompt Engineering** : Designing & optimizing prompts to guide AI models.
4. **Scalability & deployment**: Deciding b/w cloud, edge, or on-prem solutions.
5. **Cost vs Performance trade-offs**: Balancing accuracy, speed, and affordability.

Key Components of LLM System

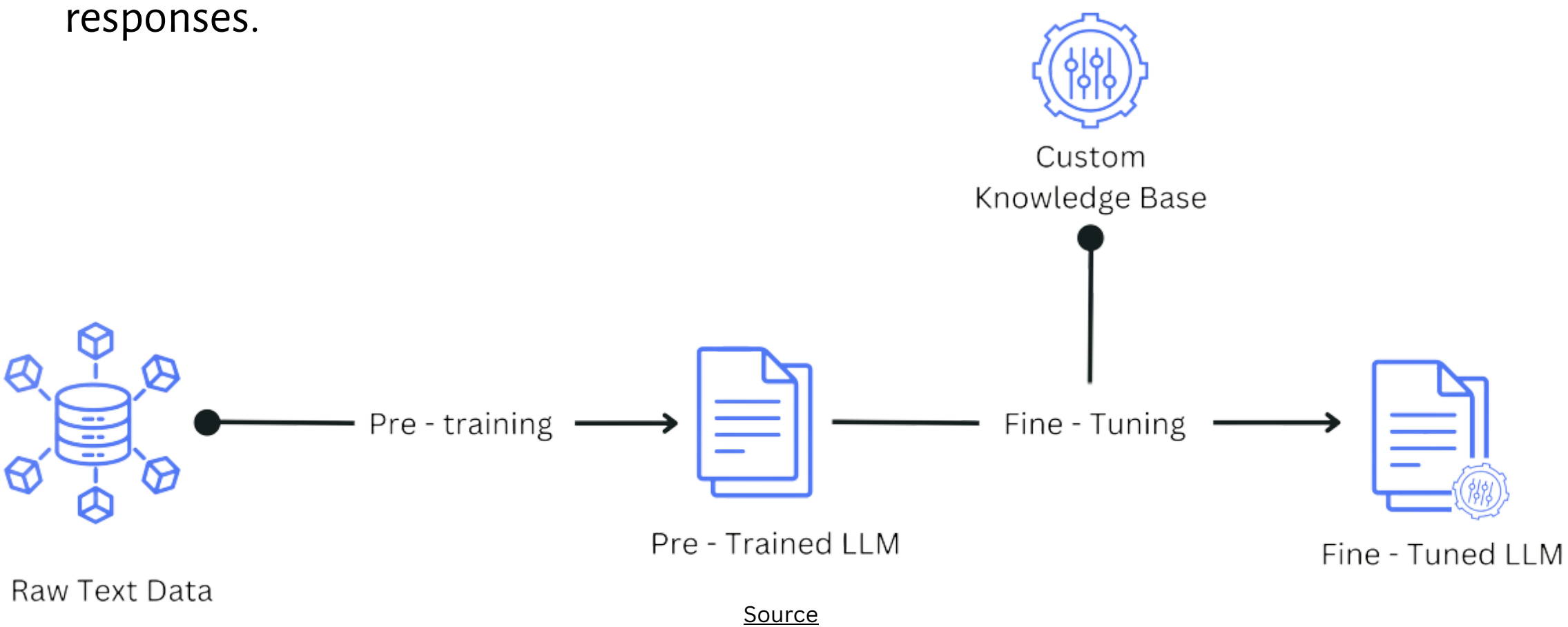
*RAG specific use case



- **User query** – The user inputs a query, which could be text, image, or multimodal data.

- **RAG** (Retrieval-Augmented Generation) – Enhances the query by fetching relevant contextual data from external sources before passing it to the model.

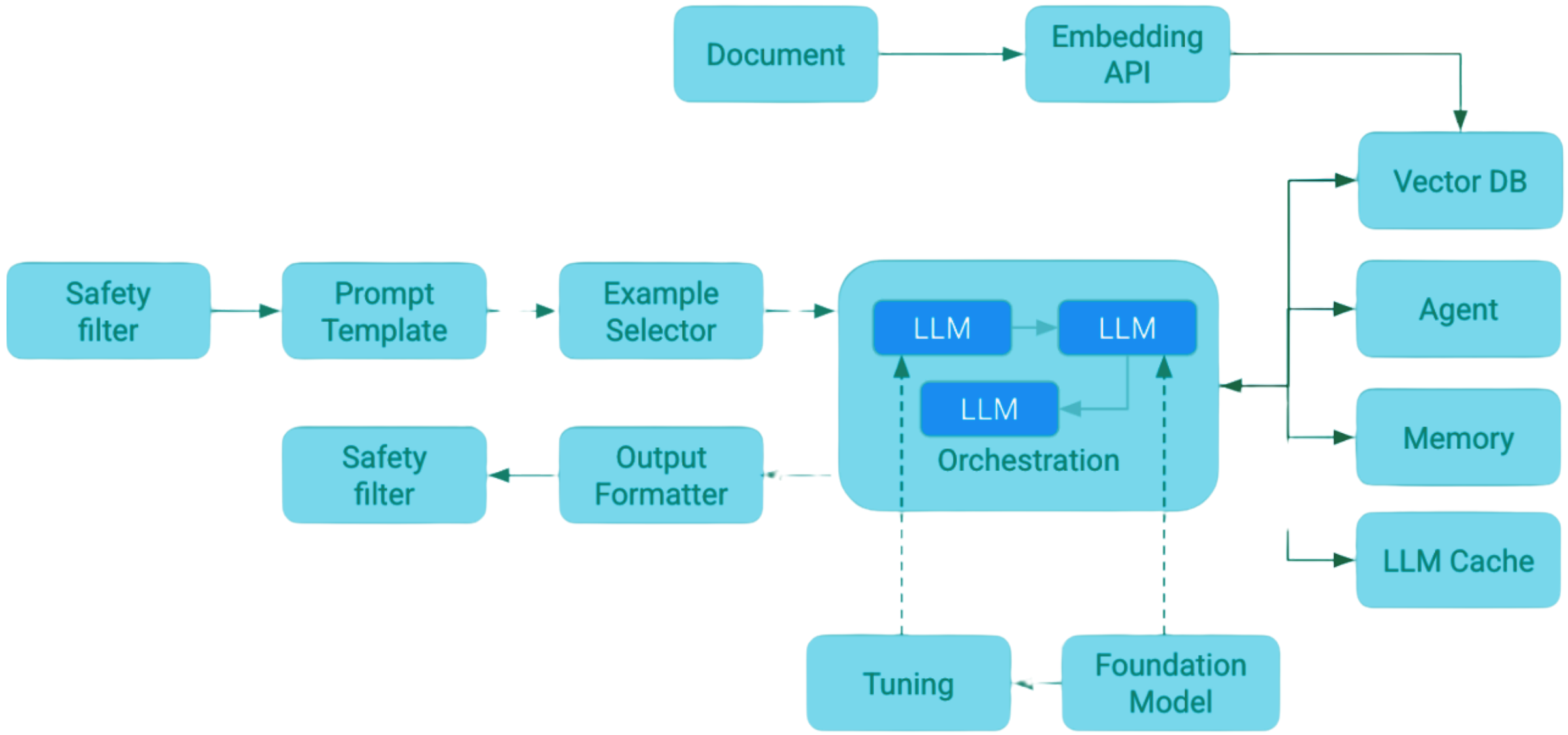
- **Inference pipeline** – Processes the query using the LLM, applying prompt engineering, model execution, and reasoning.
- **Serving infrastructure** – Manages API calls, load balancing, and efficient model deployment.
- **Fine-tuning** – Uses logged data to refine model performance and improve future responses.



Source

- **Response generation** – Formats the model's output into a structured and coherent response.
- **Evaluation & safety** – Applies filters for bias detection, hallucination reduction, and security checks.
- **Final response to User** – The refined output is sent back to the user.
- **Monitoring & logging** – Tracks system performance, logs errors, and records feedback for improvements.
- **Scalability & Latency optimization** – Ensures low-latency response times and system efficiency.

Architecting the LLM Application



Source

Frontend & User interaction

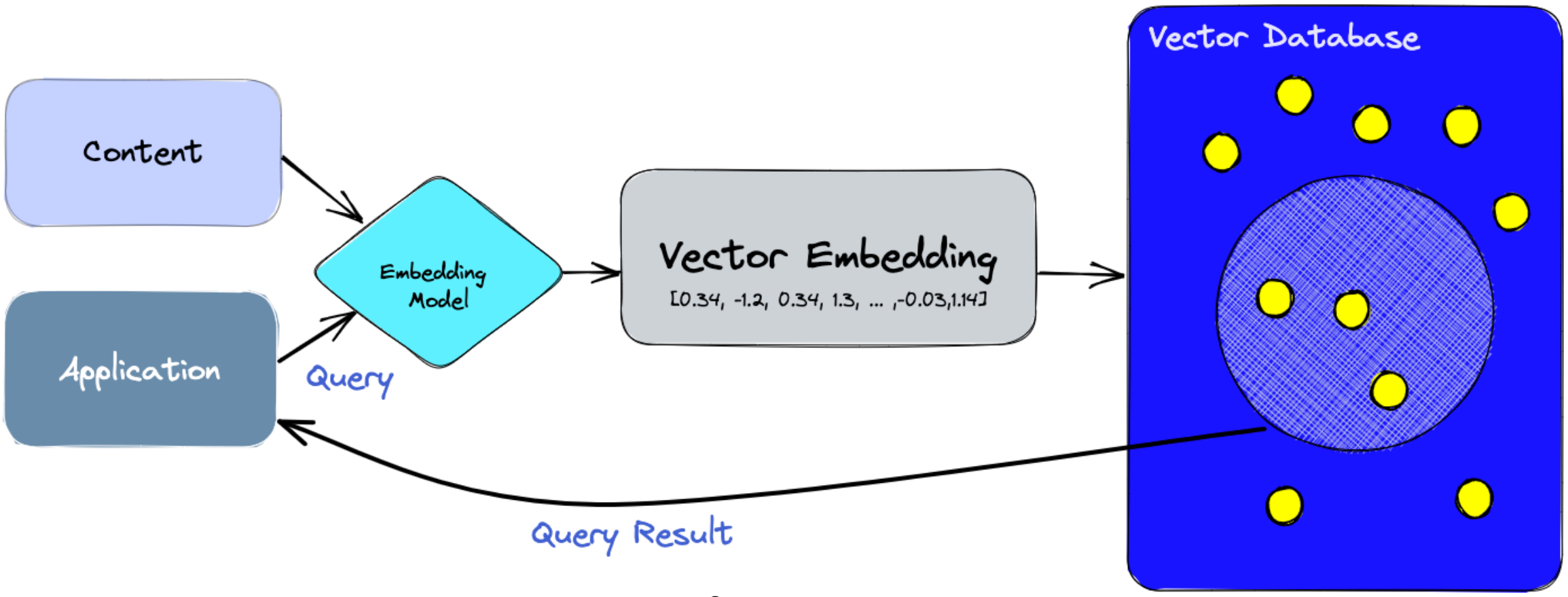
- Interface design – Web, mobile, chatbot, or API-based system?
- User input handling – Text, voice, or multimodal queries?

Backend & API layer

- API gateway: Manage requests efficiently (e.g., FastAPI, Flask, GraphQL).
- Orchestration: Route queries b/w LLMs, fallback mechanisms, error handling(LangChain, Ray Serve)
- Logging & monitoring: Track model performance, API usage, and failures(Prometheus + Grafana, OpenTelemetry)

Model selection & processing

- Choosing the right LLM – Open-source vs proprietary, fine-tuned vs general-purpose.
- Inference optimization – Quantization, distillation, RAG, caching to reduce costs.
- Context management – Use embeddings and history tracking for better responses.



Source

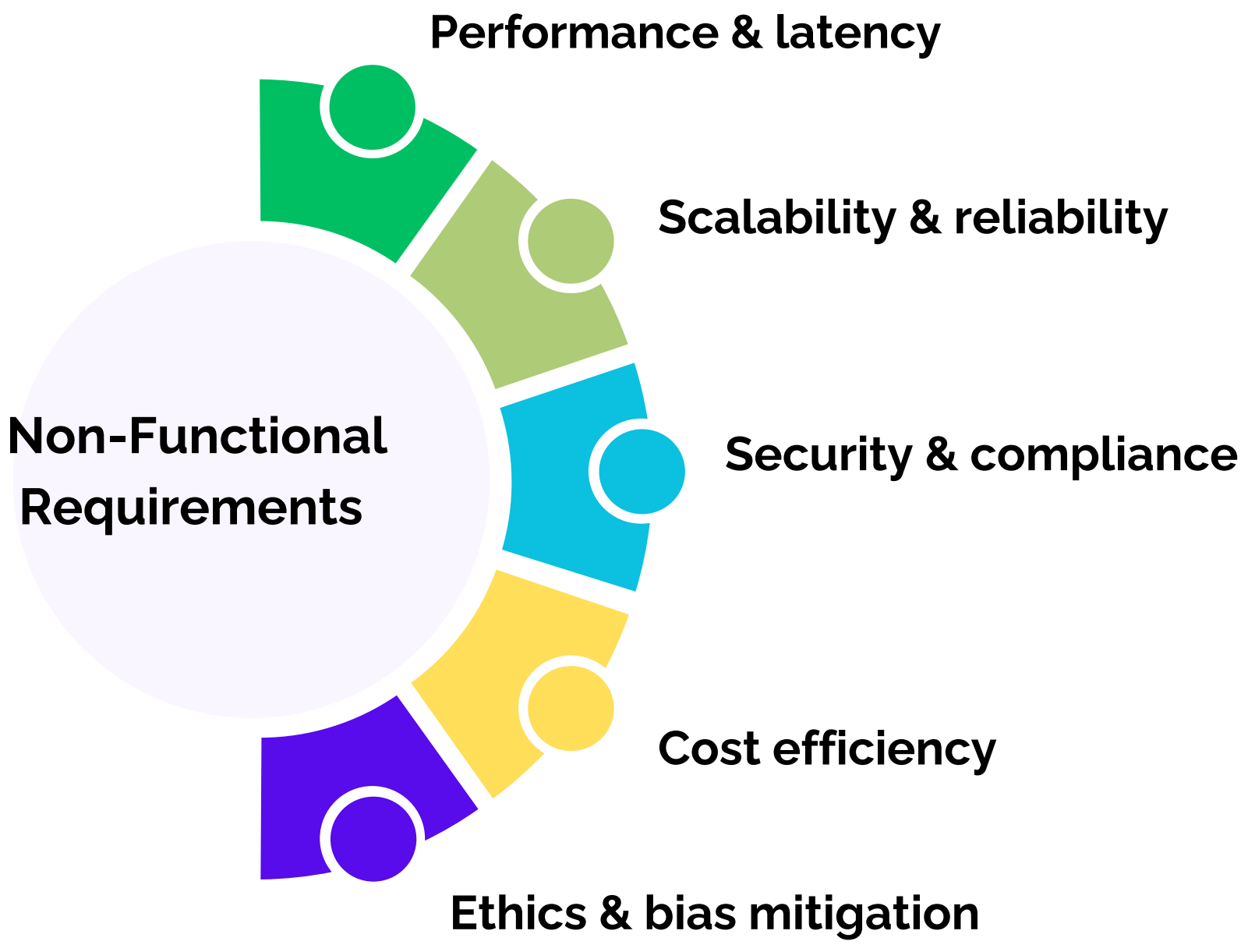
Data storage & retrieval

- Vector database (for RAG) – FAISS, ChromaDB, Weaviate for context retrieval.
- Traditional databases – Store user queries, responses, and metadata (MySQL, MongoDB)
- File storage – Cloud storage for documents, images, and structured datasets.

Deployment & scaling

- Cloud vs edge vs On-Prem – Choose based on cost, security, and latency needs.
- Load balancing & autoscaling – Handle high traffic with distributed inference(NGINX, AWS Auto Scaling)
- Containerization (Docker, Kubernetes) – Ensure portability and efficient scaling.

Considering Non-Functional Req.

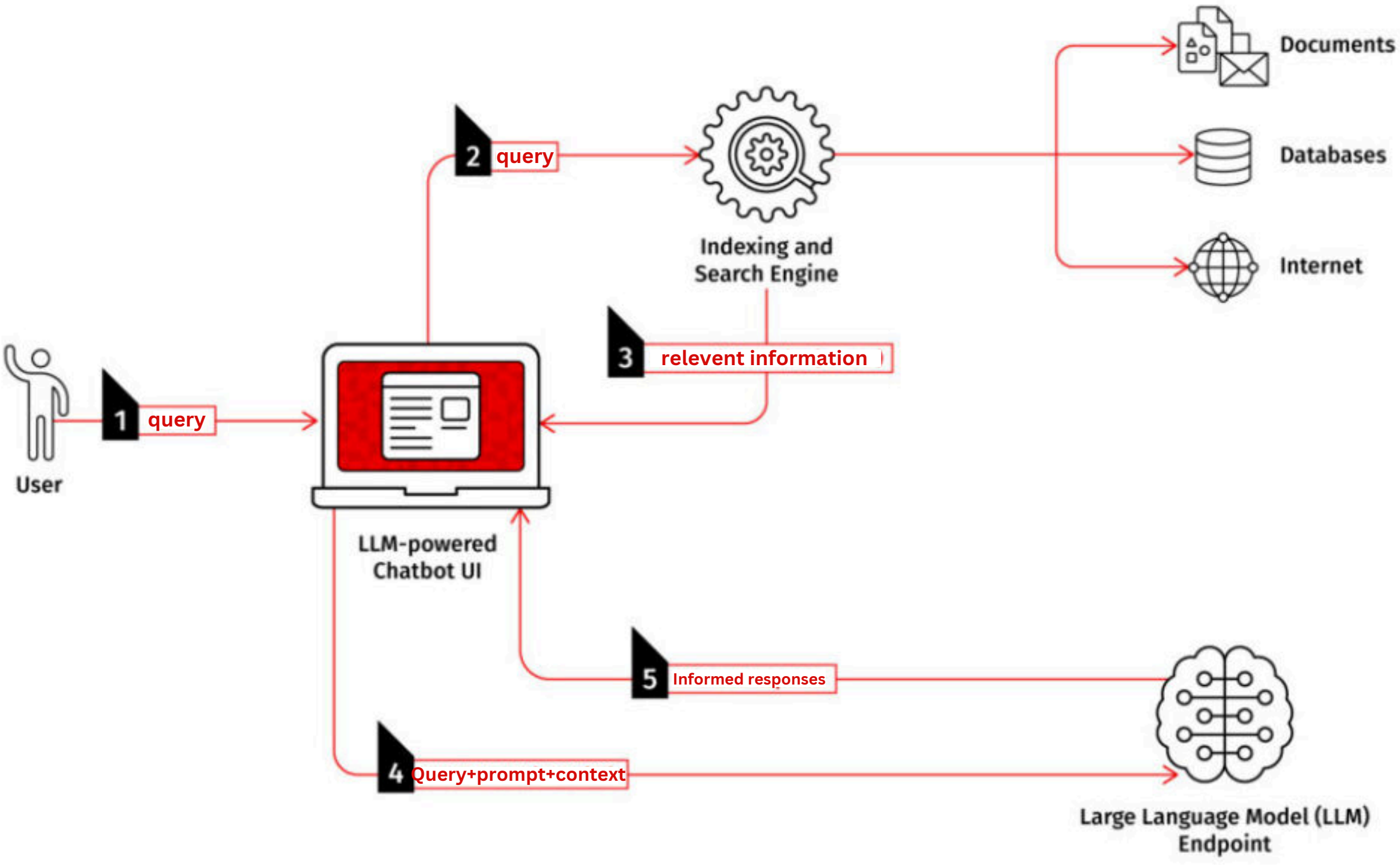


- **Performance & latency** – Optimize inference speed with quantization, caching, and batch processing to reduce delays.
- **Scalability & reliability** – Use horizontal scaling (adding more servers) and vertical scaling (upgrading resources) to handle demand.
- **Security & compliance** – Encrypt user data, restrict access, follow GDPR, HIPAA for compliance. Use authentication (OAuth, API keys) and rate limiting to prevent abuse.
- **Cost efficiency** – Reduce expenses with lightweight models, dynamic scaling, and serverless computing. Continuously monitor API usage and optimize deployments.
- **Ethics & bias mitigation** – Regularly audit model outputs to detect biases and improve fairness. Implement human-in-the-loop mechanisms for accountability.

Case Study: Building a Scalable LLM-Based Customer Support Chatbot

Background

A leading e-commerce company wanted to deploy an AI-powered chatbot to handle customer queries, reduce support costs, and improve response times.



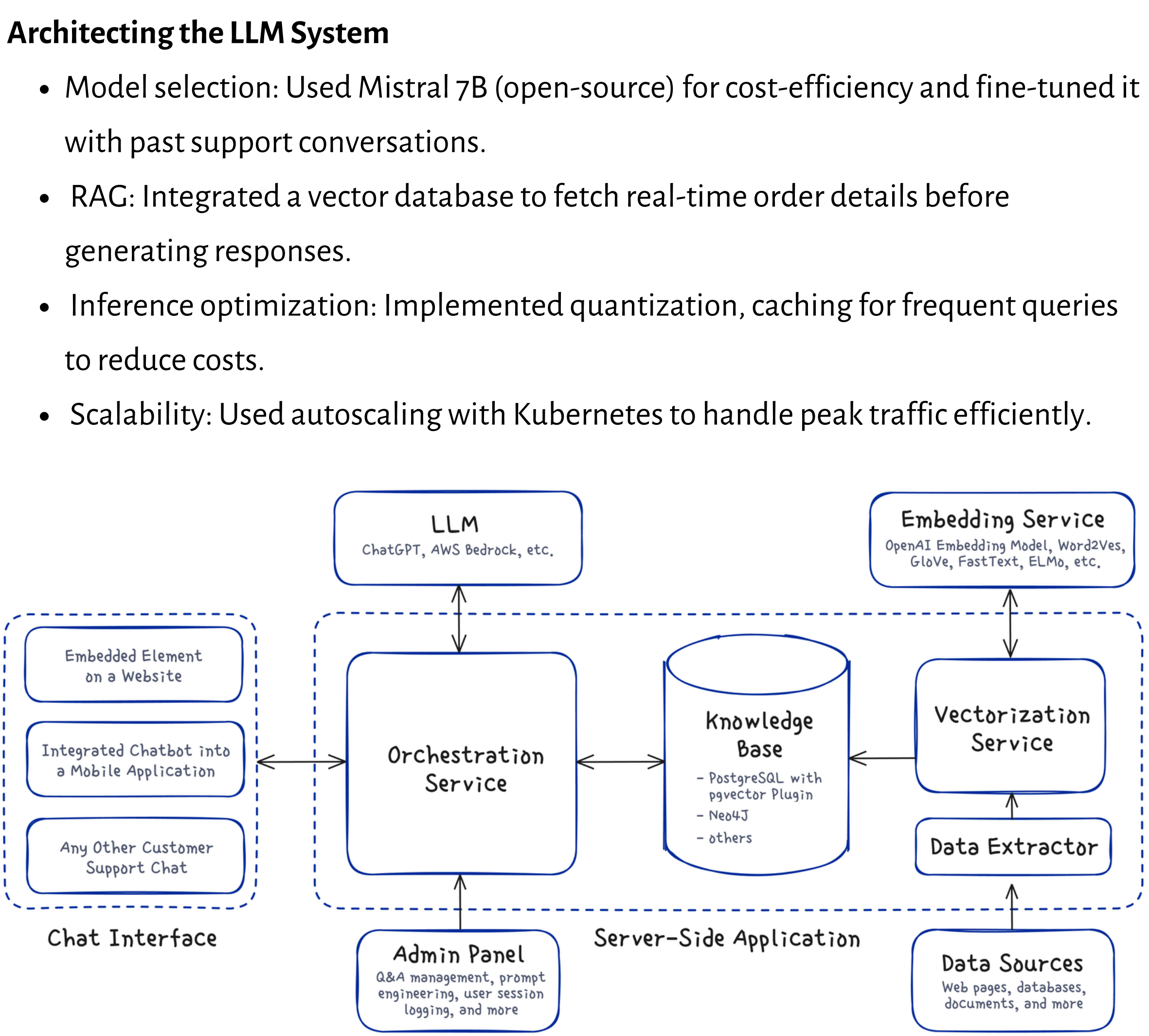
Source

Challenges

- **High query volume**: The chatbot needed to handle thousands of simultaneous users.
- **Personalization**: It had to provide relevant responses based on order history.
- **Cost optimization**: Running a large LLM continuously was expensive.
- **Latency**: Customers expected instant responses without long processing times.

Architecting the LLM System

- Model selection: Used Mistral 7B (open-source) for cost-efficiency and fine-tuned it with past support conversations.
- RAG: Integrated a vector database to fetch real-time order details before generating responses.
- Inference optimization: Implemented quantization, caching for frequent queries to reduce costs.
- Scalability: Used autoscaling with Kubernetes to handle peak traffic efficiently.



Source

Results

- Reduction in support costs by automating common queries.
- Response time improved by ensuring faster resolution.
- Lower operational costs due to optimized model inference.
- Higher customer satisfaction, with increase in CSAT(Customer Satisfaction) scores.

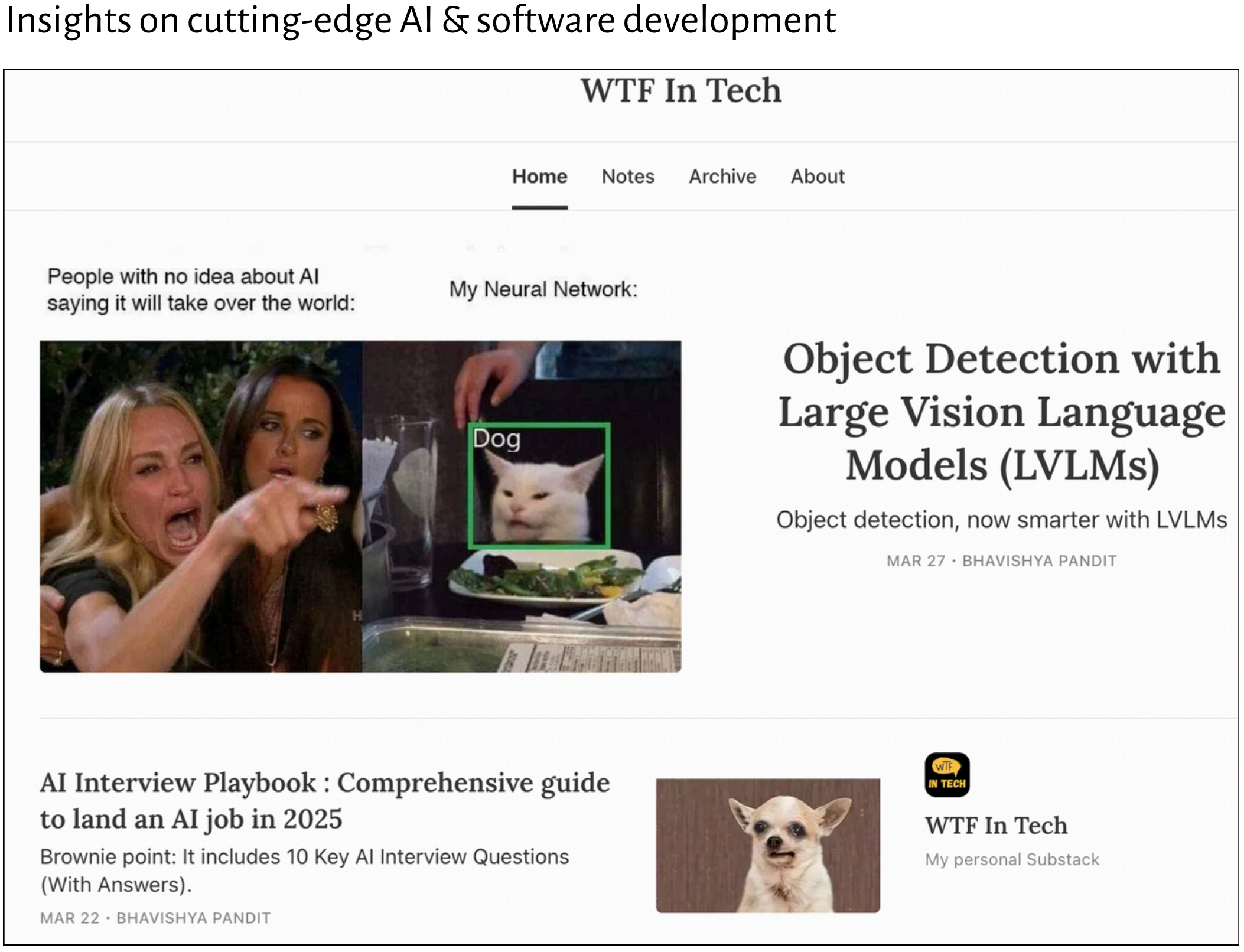
Stay Ahead with Our Tech Newsletter! 🚀

👉 **Subscribe now and never miss an update!**

🔗 **https://bhavishyapandit9.substack.com/**

Join our newsletter for:

- Step-by-step guides to mastering complex topics
- Industry trends & innovations delivered straight to your inbox
- Actionable tips to enhance your skills and stay competitive
- Insights on cutting-edge AI & software development



💡 Whether you're a developer, researcher, or tech enthusiast, this newsletter is your shortcut to staying informed and ahead of the curve.

Bhavishya Pandit

Follow to stay updated on Generative AI

SAVE

LIKE

REPOST

Bhavishya Pandit