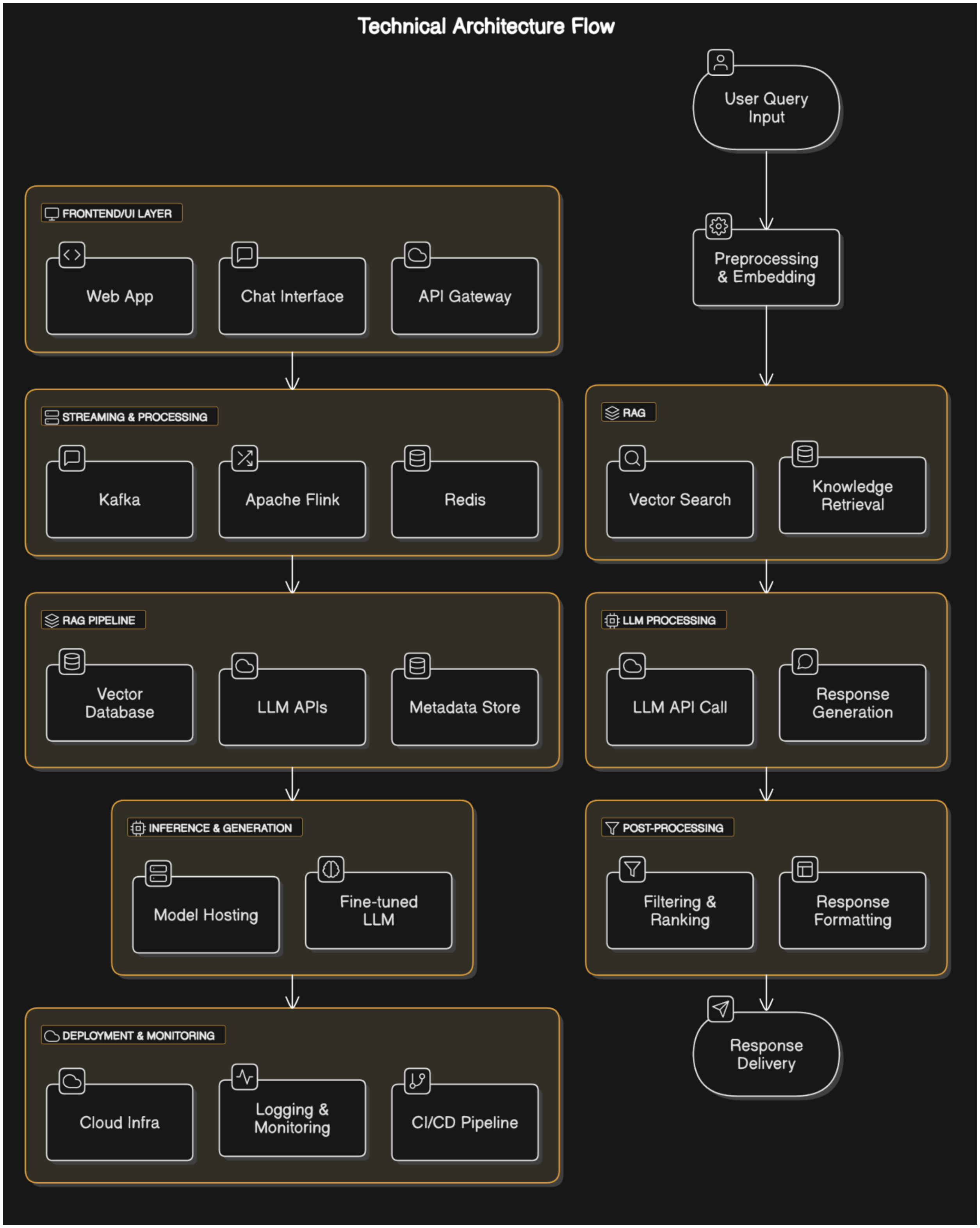
1.streaming LLM-Based Chatbot with RAG

Functional Architecture Flow

1. **User Query Input** → User sends a query to the chatbot via UI or API.
2. **Preprocessing & Embedding** → Tokenization, cleaning, and conversion to vector embeddings.
3. **Retrieval-Augmented Generation (RAG)**
   - **Vector Search** → Query embeddings are used to retrieve relevant documents from a vector store.
   - **Knowledge Retrieval** → Additional metadata/context is fetched from an indexed knowledge base.
4. **LLM Processing**
   - **LLM API Call** → Query, retrieved context, and chat history are sent to the LLM.
   - **Response Generation** → LLM generates a contextual response.
5. **Post-Processing**
   - **Filtering & Ranking** → Ensures response relevance.
   - **Response Formatting** → Final response is structured for UI display.
6. **Response Delivery** → The chatbot returns the generated response.

Technical Architecture Flow

1. **Frontend/UI Layer**
   - Web App (React.js, Next.js)
   - Chat Interface (WebSocket for real-time)
   - API Gateway (FastAPI, Flask)
2. **Streaming & Processing**
   - Kafka (Real-time message streaming)
   - Apache Flink (Streaming data processing)
   - Redis (Caching for session storage)
3. **RAG Pipeline**
   - Vector Database (FAISS, Pinecone, Weaviate)
   - LLM APIs (OpenAI GPT, LlamaIndex, Mistral)
   - Metadata Store (MongoDB, PostgreSQL)
4. **Inference & Generation**
   - Model Hosting (Hugging Face Inference Endpoint, Triton Inference Server)
   - Fine-tuned LLM or API-based Inference
5. **Deployment & Monitoring**
   - Cloud Infra (AWS Lambda, Kubernetes, Databricks for ML workflows)
   - Logging & Monitoring (Prometheus, Grafana)
   - CI/CD Pipeline (GitHub Actions, Jenkins)



2. AI-Powered Search Engine (Multi-modal RAG + Vector Search)

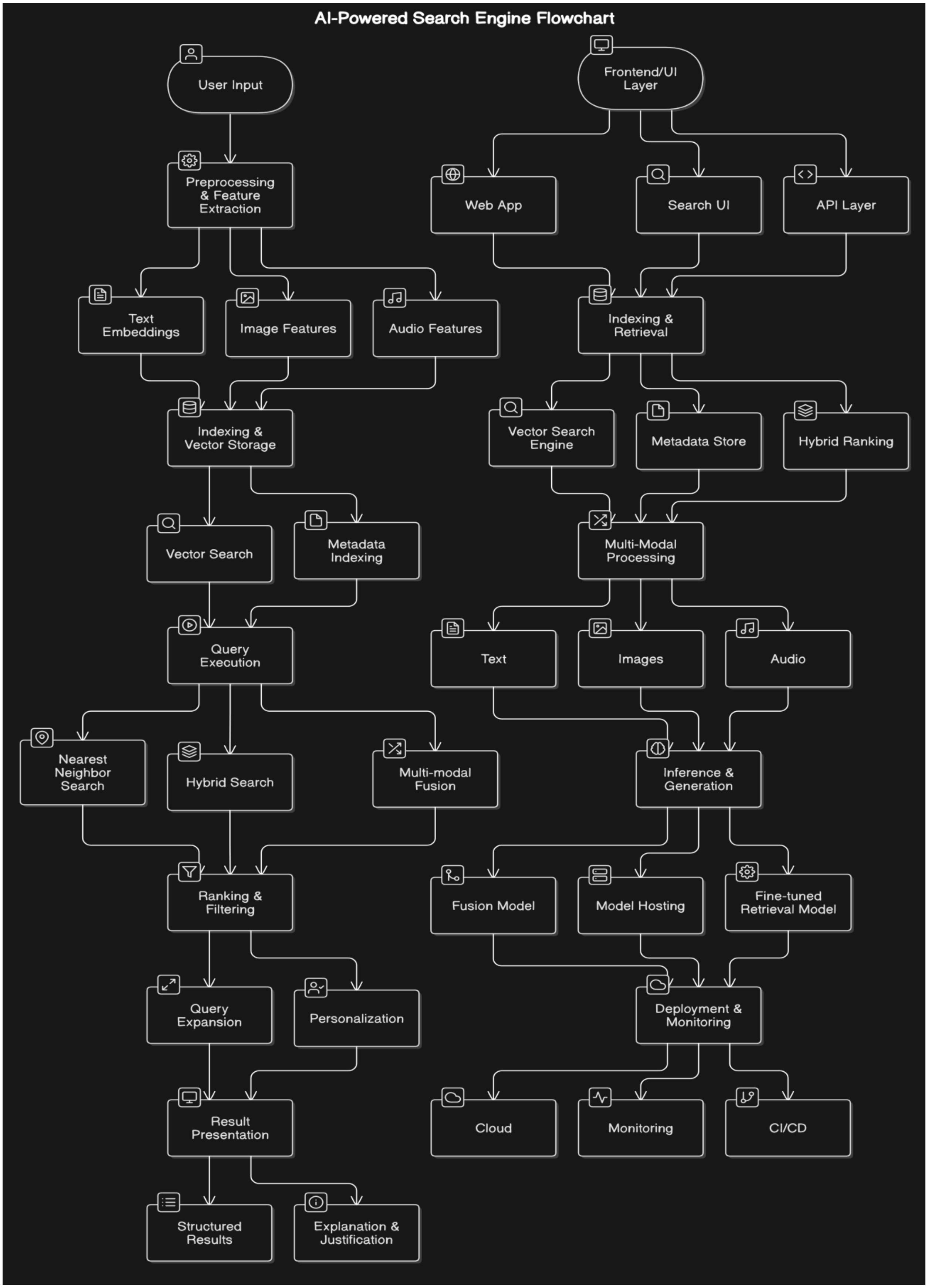
Functional Architecture Flow

1. **User Input (Text/Image/Audio)**
2. **Preprocessing & Feature Extraction**
   - Text Embeddings (BERT, Sentence-Transformers)
   - Image Features (CLIP, DINO, OpenAI Vision models)
   - Audio Features (Whisper, Wav2Vec)
3. **Indexing & Vector Storage**
   - Vector Search (FAISS, Pinecone, Milvus)
   - Metadata Indexing (Elasticsearch, PostgreSQL)
4. **Query Execution**
   - Nearest Neighbor Search (FAISS, ANN)
   - Hybrid Search (BM25 + Dense Retrieval)
   - Multi-modal Fusion (Combining text, image, and audio relevance)
5. **Ranking & Filtering**
   - Query Expansion (Reranking with ColBERT)
   - Personalization (Recommender system integration)
6. **Result Presentation**
   - Structured Results (UI ranking)
   - Explanation & Justification (Model interpretability)

Technical Architecture Flow

1. **Frontend/UI Layer**
   - Web App (Next.js, React)
   - Search UI (Elastic UI, Haystack UI)
   - API Layer (GraphQL, FastAPI)
2. **Indexing & Retrieval**
   - Vector Search Engine (FAISS, Pinecone)
   - Metadata Store (Elasticsearch)
   - Hybrid Ranking (BM25 + Neural Ranking)
3. **Multi-Modal Processing**
   - Text: BERT, GPT, ColBERT
   - Images: CLIP, DINOv2
   - Audio: Whisper, Wav2Vec2.0
4. **Inference & Generation**
   - Fusion Model (Multi-modal RAG pipeline)
   - Model Hosting (Triton Inference Server)
   - Fine-tuned Retrieval Model (ColBERT, DPR)
5. **Deployment & Monitoring**
   - Cloud (AWS Lambda, GCP Vertex AI)
   - Monitoring (Grafana, Prometheus)
   - CI/CD (GitHub Actions)

AI-Powered Search Engine Flowchart



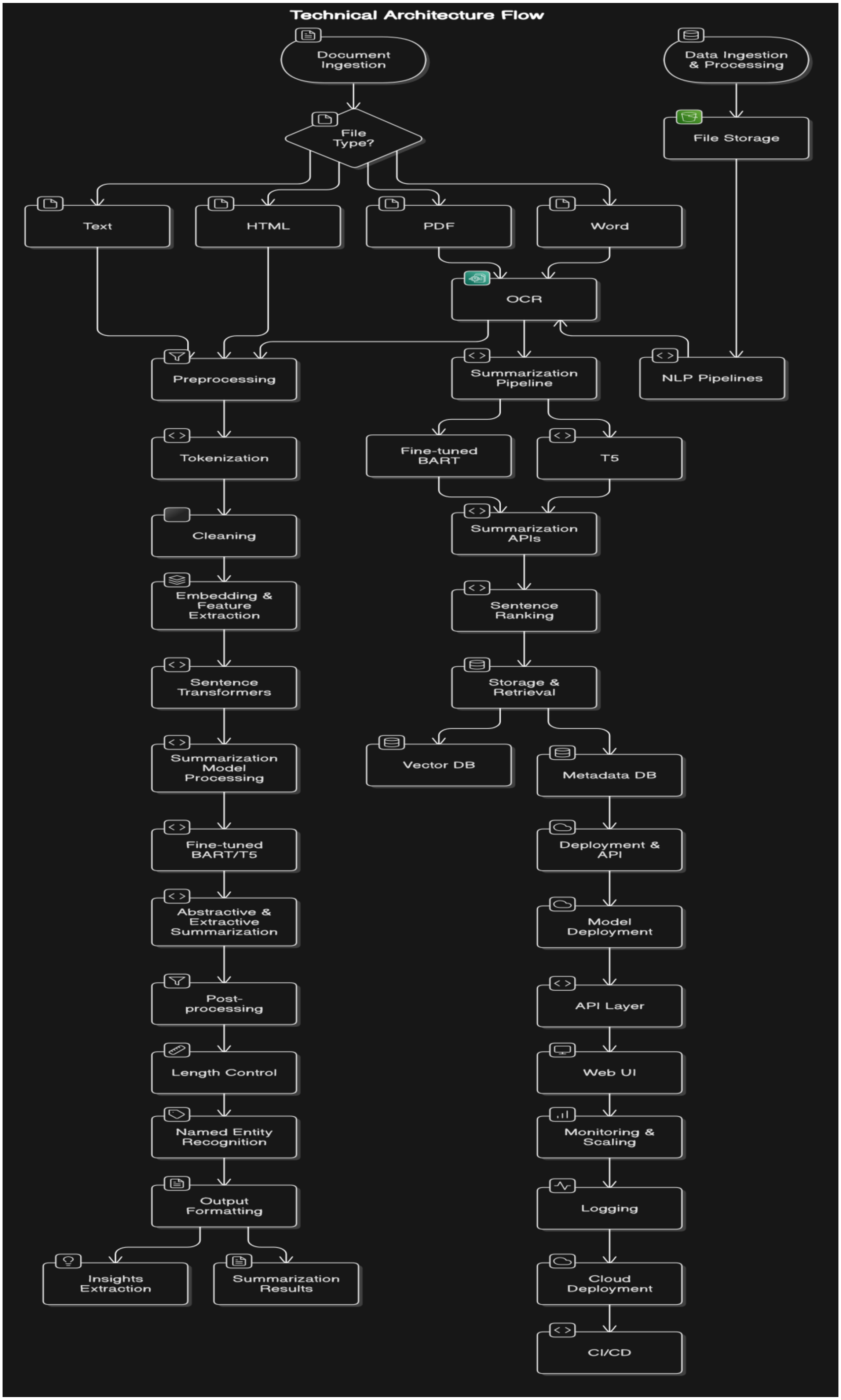
3. Document Summarization System (Fine-tuned BART/T5)

Functional Architecture Flow

1. **Document Ingestion**
   - PDF, Word, Text, HTML files
   - OCR (Tesseract, AWS Textract)
2. **Preprocessing**
   - Tokenization (Hugging Face Transformers)
   - Cleaning (Removing unnecessary symbols, HTML tags)
3. **Embedding & Feature Extraction**
   - Sentence Transformers (SBERT, Universal Sentence Encoder)
4. **Summarization Model Processing**
   - Fine-tuned BART/T5 model
   - Abstractive & Extractive Summarization Pipeline
5. **Post-processing**
   - Length control (Short, Medium, Long)
   - Named Entity Recognition (NER tagging)
6. **Output Formatting**
   - Summarization Results (Text, JSON, Markdown)
   - Insights Extraction (Key Topics, Sentiment Analysis)

Technical Architecture Flow

1. **Data Ingestion & Processing**
   - File Storage (S3, Google Drive API)
   - NLP Pipelines (spaCy, NLTK)
   - OCR (AWS Textract, Tesseract)
2. **Summarization Pipeline**
   - Fine-tuned BART, T5 (Hugging Face, OpenAI)
   - Summarization APIs (Google T5, Pegasus)
   - Sentence Ranking (TextRank, BERTScore)
3. **Storage & Retrieval**
   - Vector DB (FAISS, Pinecone)
   - Metadata DB (Elasticsearch, MongoDB)
4. **Deployment & API**
   - Model Deployment (Hugging Face Inference API, Triton)
   - API Layer (FastAPI, Flask)
   - Web UI (React, Streamlit)
5. **Monitoring & Scaling**
   - Logging (Prometheus, Grafana)
   - Cloud Deployment (AWS Lambda, GCP Vertex AI)
   - CI/CD (Docker, Kubernetes)


Technical Architecture Flow
Document Ingestion
File Type?
Text
HTML
PDF
Word
OCR
Data Ingestion & Processing
File Storage
NLP Pipelines
Preprocessing
Tokenization
Cleaning
Embedding & Feature Extraction
Sentence Transformers
Summarization Model Processing
Fine-tuned BART/T5
Abstractive & Extractive Summarization
Post-processing
Length Control
Named Entity Recognition
Output Formatting
Insights Extraction
Summarization Results
Summarization Pipeline
Fine-tuned BART
T5
Summarization APIs
Sentence Ranking
Storage & Retrieval
Vector DB
Metadata DB
Deployment & API
Model Deployment
API Layer
Web UI
Monitoring & Scaling
Logging
Cloud Deployment
CI/CD

4. LLM-Based Code Assistant

Technical Architecture

Core Components:

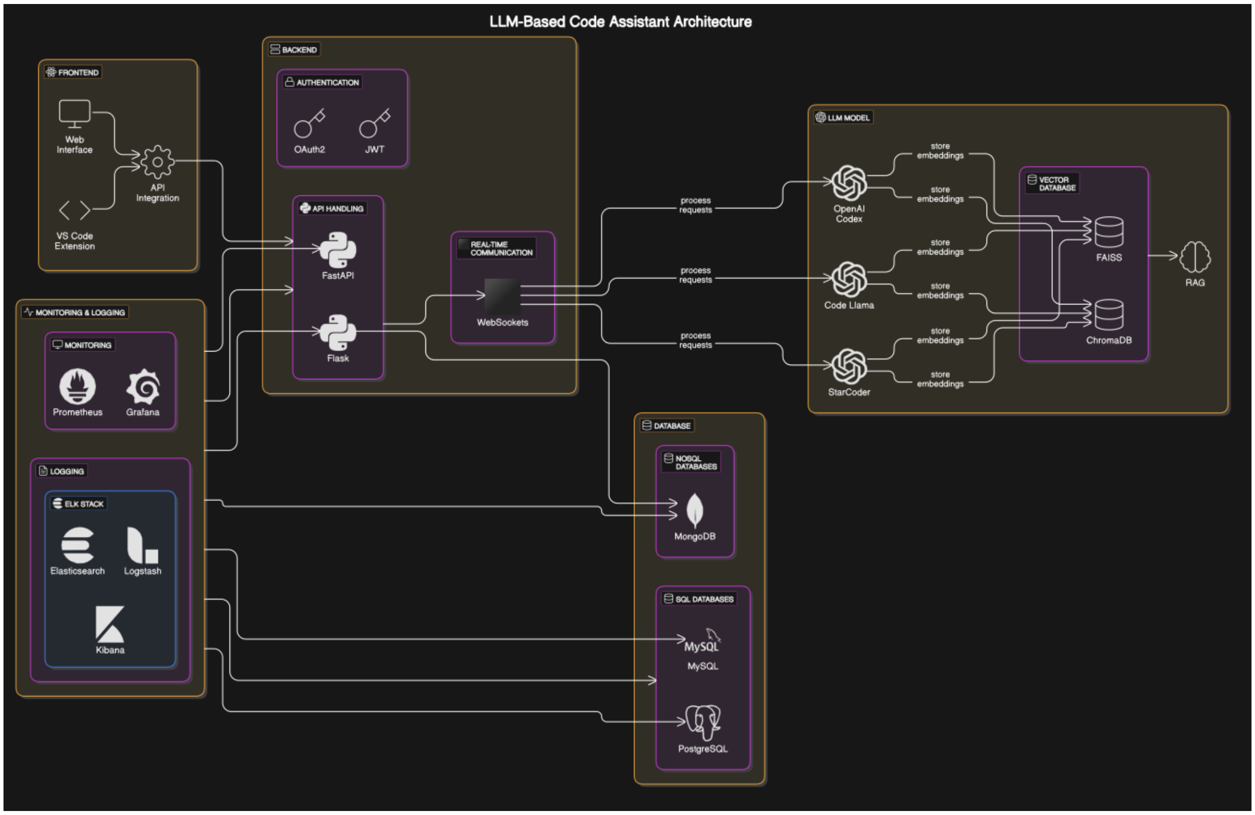
1. **Frontend:**
   - Web interface (React.js/Next.js) or VS Code Extension
   - API integration for LLM inference
2. **Backend:**
   - FastAPI/Flask (Python) for handling API requests
   - WebSockets for real-time code completion and suggestions
   - Authentication (OAuth2, JWT)
3. **LLM Model:**
   - OpenAI Codex / Code Llama / StarCoder (Fine-tuned if necessary)
   - Vector Database (FAISS, ChromaDB) for context-aware suggestions
   - RAG (Retrieval-Augmented Generation) for better accuracy
4. **Database:**
   - PostgreSQL/MySQL (storing user preferences, prompts, feedback)
   - MongoDB (storing conversation history)
5. **Monitoring & Logging:**
   - Prometheus + Grafana for monitoring
   - ELK Stack (Elasticsearch, Logstash, Kibana) for logs

Functional Flow

1. User enters a coding query in the UI (e.g., "Generate a Python function to sort an array").
2. The frontend sends the request to the backend API.
3. Backend queries LLM with context-aware embeddings.
4. If necessary, retrieves past interactions from the vector database.
5. LLM generates a code snippet.
6. The response is displayed in the UI with options to refine, test, or modify.
7. Logs and feedback are stored for continuous improvement.

Step-by-Step Real-Time Build

1. Set up a FastAPI backend with authentication.
2. Integrate LLM API (OpenAI, Code Llama).
3. Implement a vector database for retrieval.
4. Build a React.js/Next.js frontend.
5. Set up WebSockets for real-time suggestions.
6. Add logging, monitoring, and database integration.



5. Image Generation & Editing (Stable Diffusion + ControlNet)

Technical Architecture

Core Components:

1. **Frontend:**
   - Streamlit/Web app (for easy user interaction)
   - Upload/Edit images with control parameters (Pose, Depth, Scribble, etc.)
2. **Backend:**
   - FastAPI for API calls
   - Integration with Diffusers (Hugging Face)
   - WebSockets for real-time updates
3. **Stable Diffusion Pipeline:**
   - Pretrained models (SD 1.5 / SDXL)
   - ControlNet (Depth, Pose, Edge, etc.)
   - Text-to-Image & Inpainting models
4. **Storage & Databases:**
   - MinIO / S3 for storing images

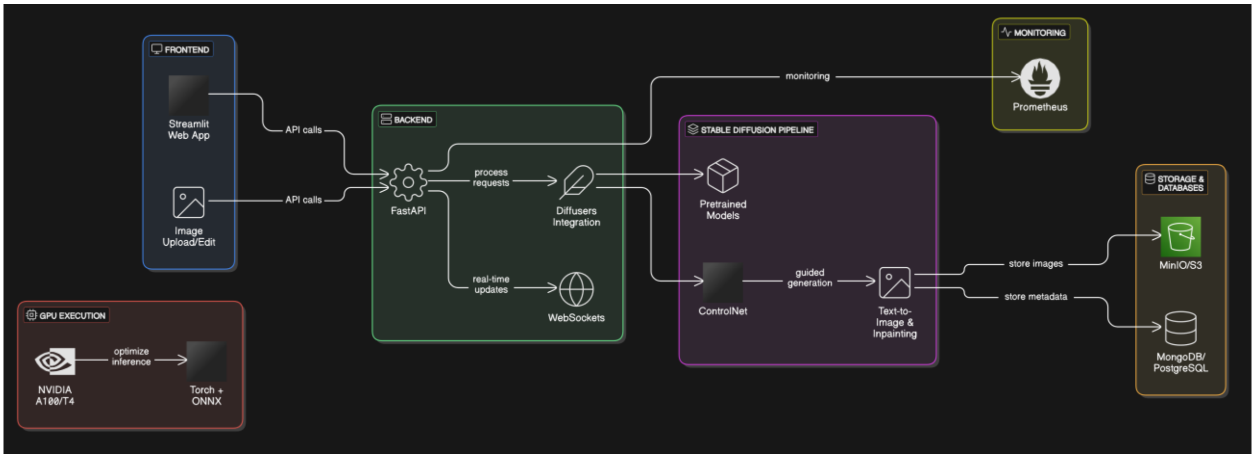
    - MongoDB/PostgreSQL for metadata
5. **GPU Execution:**
    - Deploy on NVIDIA A100/T4 instances
    - Use Torch + ONNX for acceleration

Functional Flow

1. User uploads an image or provides a text prompt.
2. Backend preprocesses input & applies ControlNet.
3. Stable Diffusion generates the output.
4. Post-processing (upscaling, filtering).
5. UI displays the generated image with editing options.
6. Logs and results are stored.

Step-by-Step Real-Time Build

1. Deploy FastAPI backend with SDXL pipeline.
2. Integrate ControlNet for guided generation.
3. Build a frontend for user interaction.
4. Optimize inference with ONNX/TorchScript.
5. Deploy monitoring with Prometheus.



---

6. Speech-to-Text & Text-to-Speech (Whisper + Tacotron/VITS)

Technical Architecture

Core Components:

1. **Frontend:**

    - Web interface (React.js, Flask)
    - Audio recording and file upload

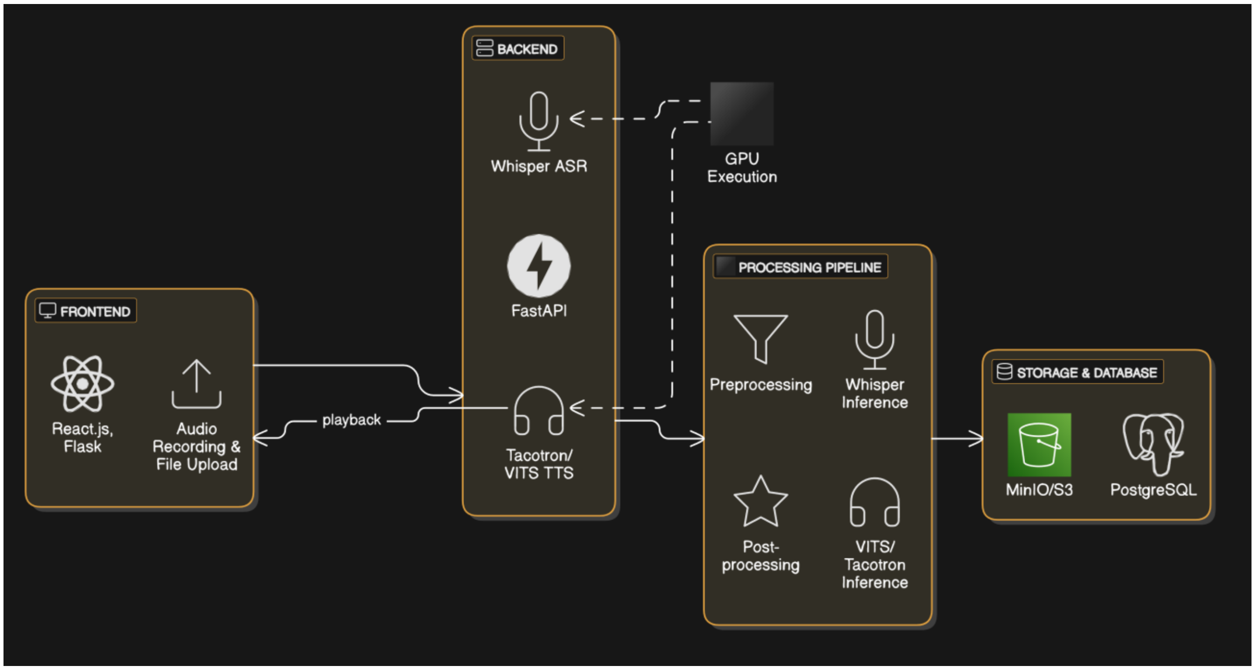
2. **Backend:**
    - FastAPI for handling requests
    - Whisper (ASR) for Speech-to-Text
    - Tacotron/VITS for Text-to-Speech
3. **Processing Pipeline:**
    - Preprocessing (Noise Reduction)
    - Whisper inference (Transcription)
    - VITS/Tacotron inference (TTS)
    - Post-processing (Enhancements)
4. **Storage & Database:**
    - MinIO/S3 for storing audio files
    - PostgreSQL for storing transcriptions
5. **GPU Execution:**
    - CUDA-optimized Whisper & VITS models

Functional Flow

1. User uploads or records an audio file.
2. Whisper transcribes speech to text.
3. If text-to-speech is needed, Tacotron/VITS converts text to speech.
4. Processed audio is available for download/playback.
5. Logs and analytics are stored.

Step-by-Step Real-Time Build

1. Deploy FastAPI backend with Whisper integration.
2. Build a frontend for user interaction.
3. Optimize inference with CUDA and batching.
4. Implement database storage for audio files.
5. Set up monitoring and logging.



7. Multi-Agent AI System (CrewAI-based)

Technical Architecture

Core Components:

1. **CrewAI Agents:**
   - Role-based agents (Planner, Researcher, Coder, Validator)
   - LLM integration (GPT-4, Claude, Mistral)
2. **Backend:**
   - FastAPI for orchestration
   - WebSockets for real-time interactions
   - Celery for task execution
3. **Vector Database:**
   - Pinecone/FAISS for memory storage
4. **Frontend:**
   - Dashboard to interact with agents
   - Task management UI
5. **Monitoring & Logging:**
   - Prometheus + Grafana for monitoring
   - ELK Stack for logs

Functional Flow

1. User defines a task (e.g., "Write a research report on AI ethics").

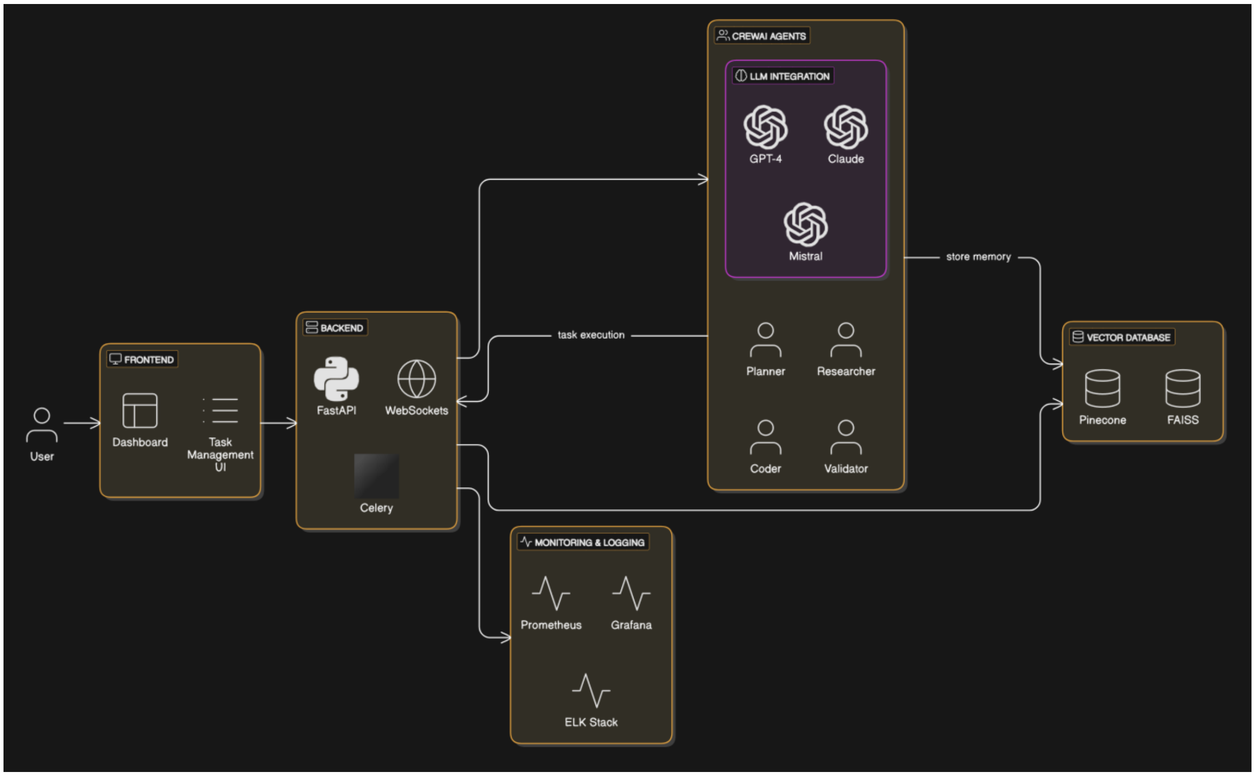
2. CrewAI assigns the task to agents:
    - **Researcher:** Gathers information.
    - **Planner:** Outlines the document.
    - **Coder:** Writes the draft.
    - **Validator:** Reviews and refines.
3. Final output is delivered to the user.

Step-by-Step Real-Time Build

1. Set up FastAPI backend with CrewAI.
2. Define multi-agent workflows.
3. Implement memory storage with FAISS.
4. Build a frontend for interaction.
5. Optimize execution with caching.

Final Notes

- **Deployment Considerations:** Use Kubernetes (K8s) for scalability.
- **CI/CD Pipeline:** GitHub Actions + Docker + AWS/GCP.
- **Security:** JWT-based authentication, rate limiting.



8. AI Personal Assistant - Architecture & Step-by-Step Approach

Objective

Develop an AI-powered personal assistant that integrates **LLM, RAG, and tool orchestration** to provide real-time information retrieval, task automation, and contextual assistance.

Technical Architecture

1. **User Interaction Layer (UI/UX)**
    - Web/Mobile App (React.js, Flutter, Next.js)
    - Voice/Text-based interaction (Twilio, WebRTC, Whisper for Speech-to-Text)
    - Chat Interface (LangChain, Streamlit)
2. **Orchestration Layer**
    - **Agentic Workflow** (LangChain/LLamaIndex)
    - **Tool Invocation** (APIs, Browser Automation, Plugins)
    - **Memory & Context Management** (Redis, ChromaDB, FAISS)
    - **Workflow Execution Engine** (Temporal.io, Celery)
3. **Retrieval-Augmented Generation (RAG)**
    - **Knowledge Base** (Elasticsearch, Pinecone, Weaviate, MongoDB Atlas Vector Search)
    - **Embedding Models** (OpenAI, BGE, Cohere, Hugging Face Transformers)
    - **Chunking & Indexing** (LangChain, Unstructured, LlamaIndex)
4. **LLM Orchestration**
    - **Model APIs** (GPT-4, Claude, Gemini, Mistral, LLaMA)
    - **Fine-Tuned Models** (T5, Falcon, Vicuna)
    - **Prompt Optimization & Fine-tuning** (LoRA, QLoRA)
    - **Multi-modal Integration** (Image, Voice, Text)
    - **Streaming Support** (FastAPI, WebSockets)
5. **External Tool Integration**
    - **Calendar & Email Automation** (Google APIs, Outlook API)
    - **Finance/News Data** (Alpha Vantage, OpenBB)
    - **Coding Assistant** (Copilot, Code Interpreter)
    - **Smart Home Controls** (IoT, Home Assistant)
6. **Storage & Persistence**
    - **User Context & Memory** (Redis, PostgreSQL, MongoDB)
    - **User Files/Notes** (Cloud Storage - AWS S3, Firebase)
    - **Authentication & Authorization** (OAuth2, Firebase Auth, Keycloak)

Functional Architecture Flow

1. **User Input Processing**
    - User asks a question or requests a task.
    - Input processed via **STT (Speech-to-Text) / Text-based UI**.

    - System identifies intent & determines action.
2. **Contextual Retrieval (RAG)**
    - Queries **Vector DB / Document Storage** for relevant data.
    - Uses embedding similarity search.
    - Retrieves past interactions for continuity.
3. **Agentic Workflow Execution**
    - Decides whether **LLM needs to respond** or **external tools need execution**.
    - Invokes appropriate APIs or databases.
4. **LLM Response Generation**
    - Constructs a prompt using **retrieved knowledge + context**.
    - Passes it to **LLM API** for response.
    - Performs **post-processing & validation**.
5. **Response Delivery & Continuous Learning**
    - Streams response to the user.
    - Updates **user memory & preferences**.
    - Learns from interactions (RLHF-based tuning).

Deployment & Scaling

- **Model Hosting**: On-premise (vLLM, TGI) / Cloud-based (OpenAI, Hugging Face Hub)
- **Microservices**: FastAPI, Flask, gRPC
- **Streaming**: Kafka, Redis Streams
- **Containerization**: Docker, Kubernetes
- **CI/CD**: GitHub Actions, ArgoCD

---

9. Finance AI Copilot - Architecture & Step-by-Step Approach

Objective

Real-time AI-driven assistant for **fraud detection and market analysis** using LLMs, Kafka, and predictive analytics.

Technical Architecture

1. **Data Ingestion Layer**
    - **Market Data** (Yahoo Finance, Alpha Vantage, Bloomberg APIs)
    - **Fraud Detection Streams** (Kafka, Confluent Cloud, Flink)
    - **Transaction Monitoring** (Banking APIs, Webhooks)
2. **Processing & Feature Engineering**
    - **Streaming Processing** (Apache Flink, Spark Streaming)

   - **ETL Pipeline** (Apache NiFi, Airflow)
   - **Data Aggregation** (Delta Lake, DuckDB)
3. **Machine Learning Layer**
   - **Fraud Detection Models** (Isolation Forest, XGBoost, LSTMs)
   - **Market Trend Prediction** (Transformer-based models, ARIMA, Prophet)
   - **Portfolio Optimization** (Reinforcement Learning)
4. **LLM-Based Insights**
   - **LLM for Market Sentiment** (GPT-4, Claude, Mistral)
   - **Real-time Analysis & Alerting** (LangChain + Streaming)
5. **User Interface**
   - **Dashboard** (Tableau, Streamlit, React.js)
   - **Alerting System** (Slack, Twilio)

Functional Architecture Flow

1. **Real-Time Data Streaming**
   - Kafka streams ingest market transactions & fraud signals.
   - AI models continuously monitor transactions.
2. **Feature Engineering & Model Execution**
   - Extracts financial patterns from historical & live data.
   - Predicts fraud likelihood and market trends.
3. **AI Copilot Analysis**
   - Context-aware assistant provides **insights & alerts**.
   - Generates **recommendations for financial actions**.
4. **Automated Execution**
   - Triggers **trade orders, fraud alerts** based on predictions.
   - Updates logs & dashboards.

Deployment & Scaling

- **Data Processing**: Databricks, Spark, Flink
- **Real-time Analysis**: Apache Kafka, Druid, Flink SQL
- **Model Hosting**: SageMaker, Vertex AI, On-Prem MLFlow
- **Cloud Infrastructure**: AWS/GCP/Azure

10. Generative AI for Cybersecurity - Architecture & Step-by-Step Approach

Objective

Real-time AI-driven threat detection and response using LLM, anomaly detection, and SIEM integration.

Technical Architecture

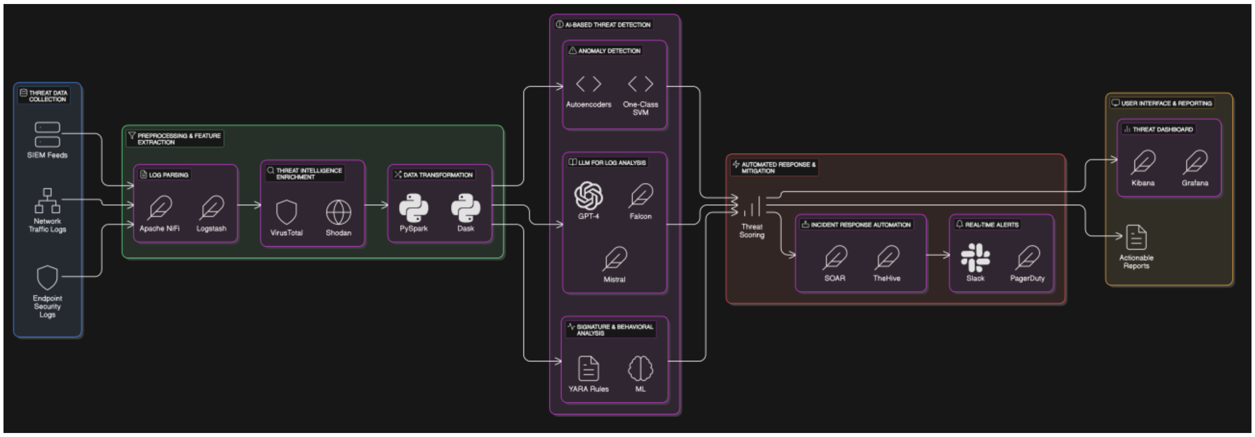
1. **Threat Data Collection**
   - **Network Traffic Logs** (Suricata, Zeek, Wireshark)
   - **Endpoint Security Logs** (EDR, XDR)
   - **SIEM Feeds** (Splunk, ELK Stack)
2. **Preprocessing & Feature Extraction**
   - **Log Parsing** (Apache NiFi, Logstash)
   - **Threat Intelligence Enrichment** (VirusTotal, Shodan)
   - **Data Transformation** (PySpark, Dask)
3. **AI-Based Threat Detection**
   - **Anomaly Detection** (Autoencoders, One-Class SVM)
   - **LLM for Log Analysis** (GPT-4, Falcon, Mistral)
   - **Signature & Behavioral Analysis** (YARA Rules, ML)
4. **Automated Response & Mitigation**
   - **Threat Scoring** (MITRE ATT&CK-based risk scoring)
   - **Incident Response Automation** (SOAR, TheHive)
   - **Real-time Alerts** (Slack, PagerDuty)
5. **User Interface & Reporting**
   - **Threat Dashboard** (Kibana, Grafana)
   - **Actionable Reports** (LLM-based summaries)

Functional Architecture Flow

1. **Threat Data Aggregation**
   - Streams logs from SIEM, network, and endpoints.
2. **Threat Analysis using AI**
   - Identifies suspicious patterns using **LLMs & anomaly detection**.
3. **Risk Scoring & Response**
   - AI determines risk level and executes automated mitigation.
4. **Reporting & Learning**
   - Generates **incident reports & real-time security alerts**.

Deployment & Scaling

- **SIEM Integration**: Splunk, ELK, Sentinel
- **AI Model Hosting**: On-prem / Cloud
- **Streaming Processing**: Kafka, Flink
- **Automation**: SOAR, AWS Lambda


THREAT DATA COLLECTION
SIEM Feeds
Network Traffic Logs
Endpoint Security Logs
PREPROCESSING & FEATURE EXTRACTION
LOG PARSING
Apache NiFi
Logstash
THREAT INTELLIGENCE ENRICHMENT
VirusTotal
Shodan
DATA TRANSFORMATION
PySpark
Dask
AI-BASED THREAT DETECTION
ANOMALY DETECTION
Autoencoders
One-Class SVM
LLM FOR LOG ANALYSIS
GPT-4
Falcon
Mistral
SIGNATURE & BEHAVIORAL ANALYSIS
YARA Rules
ML
AUTOMATED RESPONSE & MITIGATION
Threat Scoring
INCIDENT RESPONSE AUTOMATION
SOAR
TheHive
REAL-TIME ALERTS
Slack
PagerDuty
USER INTERFACE & REPORTING
THREAT DASHBOARD
Kibana
Grafana
Actionable Reports