Overall Verdict

29 December 2025 18:29

Core Technical Requirement

↓

ML and DL Requirement(Optional)

↓

GenAI Foundation

↓

LLMs/ SLMs/Multimodal

↓

Fine-Tuning

↓

RAG

↓

Agents

↓

MCP

↓

Cloud Services for GenAI

↓

▨ Deployment Strategy

↓

▨ Projects(AI Powered and Integrated App)

Common FAQs

29 December 2025 18:35

1) Who is this for?

- Software engineers ➡ **GenAI / LLM engineering roles**.
- Backend / full-stack developers who want to integrate **LLMs, RAG, and agents**.
- Data scientists/ML engineers aiming to ➡ **GenAI / LLM engineering roles or real production-grade AI systems**.
- Professionals preparing for **interviews, projects, or startup-level AI products**.

2) What about freshers?

- Yes, **freshers can do this**, but with the right expectations.
- Freshers should first be comfortable with **Python, basic APIs, and Git**.
- For freshers, this syllabus can act as a **career accelerator into GenAI roles**.
- Focus should be on **learning concepts + hands-on practice**, not rushing modules.

3) Who is this not for?

• People looking for **quick shortcuts or one-week crash courses**
• Those who want **theory only, without hands-on coding or deployment**
• Anyone expecting **copy-paste solutions** without understanding **system design**

4) How much time will it take to prepare?

- **8 month structured learning**.

  2–Months → Foundations + Core Understanding

  Covers these modules:

  - MODULE 1: Foundations of Modern GenAI
  - MODULE 2: Early Stage LLM
  - MODULE 3: Modern LLMs, SLMs & Multimodal
  - MODULE 4: API for access the LLM
  - MODULE 5: Training LLM from Scratch
  - MODULE 8: Prompt Engineering
  - MODULE 9: (Basics): RAG fundamentals
  - MODULE 11: (Basics): Agent fundamentals

  2–4 Months → Depth + GenAI Engineering Skills

Covers these modules:

- MODULE 6: Fine-Tuning LLMs
- MODULE 10: Context Engineering
- MODULE 9: (Advance): Data parsing, Multimodal RAG, Agentic RAG
- MODULE 11: (Advance): Multiagent, Deep Agent
- MODULE 12: Evaluation Strategies
- MODULE 13: Guardrails

2–Months → Cloud + Deployment + Reality Check

Covers these modules:

- MODULE 14: MCP
- MODULE 16: Cloud Services for GenAI
- MODULE 17: Deployment Strategies
- MODULE 7: LLM Hosting on your own server
- MODULE 18: End-to-End Projects

- With consistent effort (1–2 hrs./day), you'll be **industry-ready in 6 – 8 months**.

5) Do you need ML & DL to *start* GenAI?

NO — ML & DL are NOT mandatory before starting GenAI.
GenAI work do NOT require classic ML/DL.

You can **start GenAI directly** and build:

- RAG systems
- Agentic systems
- Production GenAI apps

Then Why People Say "Learn ML/DL First"?
Goal : GenAI Application Engineer (Most people)
Build systems(apps) using LLMs
👉 **ML/DL NOT mandatory**
But if you Train models, Modify models architectures, Finetune Model then **ML & DL Basics should be REQUIRED**

One-Line Industry Truth
ML/DL is a *booster*, not a barrier.

Stats→ ML → DL → Math → Theory → (Burnout)
(Most people fail here)
…and **never reach GenAI systems**.

Final Recommendation (Crystal Clear)

- Don't wait for ML/DL
- Learn ML/DL **in parallel or later**
- Start **GenAI now**

- Build systems early

Core Technical Requirements

29 December 2025 18:36

#	Skill Area	Priority	What to Learn	Level Required
1	Programming Language	MUST	**Python**	Basic → Intermediate → Advanced (Strong)
2	SQL	Required	—	Basic → Intermediate
3	SQL Database	Required	MySQL / **PostgreSQL (Recommended)** / Oracle	Working Knowledge
4	NoSQL Databases	Required	MongoDB, Redis, Neo4j	Basic → Intermediate
5	DSA	Required	—	Basic → Intermediate
6	API Creation	MUST	**FastAPI (Preferred)**, Django	Production-ready APIs
7	Docker & CI/CD	Required	Docker, GitHub Actions / CI pipelines	Working Knowledge
8	Cloud Knowledge	MUST	**AWS / Azure / GCP (Any ONE)**	GenAI Services & Deployment-level
9	Version Control	MUST	**Git, GitHub**	Daily-use, collaboration-ready
10	IDE / Code Editor	Required	**VS Code (Preferred), PyCharm**	Comfortable for daily development
11	Writing Code with Copilot	MUST	GitHub Copilot, MS Copilot, Claude, ChatGPT	Master for highest productivity

In Today's Era: AI Is the Heart of the Product

Whenever we build AI-enabled product, they generally fall into **two categories**.

1 AI-Native Applications

AI-native applications are systems where **AI is the core of the product**.
Without AI, the application **cannot exist**.
Examples:

- ChatGPT
- Manus
- Perplexity

2 AI-Integrated Applications

AI-integrated applications are **existing systems** where AI is added as a **feature or enhancement**, not the core product.
Examples:

- IRCTC
- Amazon customer support

The Most Important Point

In **both cases**, the **fundamental software engineering building blocks remain the same**.
You still need strong knowledge of:

- UI / UX frontend development
- APIs
- Backend
- Databases
- Cloud infrastructure/ Deployment & DevOps
- Data Integration Layer
- Software Development Life Cycle (SDLC)

AI adds intelligence to a system, but designing scalable and reliable systems still depends on strong software engineering fundamentals.

PRODUCT generally be categorized into two types.
Consumer-scale products (millions of users)
OpenAI- ChatGPT
Google-YouTube
Meta- Facebook
Netflix
Perplexity.ai

Business / Enterprise products (domain-specific)

Domain	Major Companies
Healthcare	UnitedHealth Group, CVS Health
Pharma	Pfizer, Merck
Finance	BlackRock, Visa
Banking	JPMorgan Chase(Fraud Detection & Monitoring, Trading & Risk Platforms, Chase), Morgan Stanley, Goldman Sachs

Retail Walmart, DMart

Education Coursera, Udemy

SDLC Steps:

1. **Requirement Analysis**
2. **Design**
3. **Development**
4. **Testing**
5. **Deployment**
6. **Maintenance & Monitoring**
7. **Support**

Agile Framework (with Devops):

Agile is a **software development approach** focused on iterative development, fast feedback, and continuous improvement.

Jira (Tool / Software)

- **Jira**
- Jira is a **project & issue tracking tool** used to implement Agile / Scrum in real projects.

Sprint (Fixed time period: usually 2 weeks)

- **Product Backlog** → Complete list of all work
- **User Story** → Requirement from user's perspective
- **Epic** → Large feature broken into stories
- **Task** → Small unit of work
- **Bug** → Defect or issue

Scrum

- **Scrum** is an **process** used to manage work using sprints.

Scrum Master (Role & Responsibilities)

- Facilitates **Daily Stand-ups**
- Conducts **Sprint Retrospective**
- Removes blockers
- Ensures Scrum process is followed

Product Owner(AI Required)

- Owns **product vision**

Product Manager(AI Required)

- Manage the product

Business Analyst(AI Required)

- Converts business requirements into **user stories**

Tech Lead(AI Must)

- Defines **technical architecture**

Development Team (Cross-Functional Team)

- **Frontend Developers**
  - Build UI (web / mobile)
  - Integrate AI features into the user experience
- **Backend Developers(AI Required)**
  - Build APIs and business logic
  - Integrate AI/LLM services into backend systems
- **Database Engineers**
  - Design and manage databases
  - Handle structured data, logs, and vector databases
- **AI / ML Engineers (AI Developers)(AI Must)**

    - Build and integrate AI models
    - Design prompts, RAG pipelines, and agents
    - Optimize model performance, cost, and accuracy

Testing Team

- Performs **manual & automated testing**

DevOps Team

- CI/CD pipelines
- Cloud infrastructure
- Monitoring & deployments

Support Team

- Handles production issues
- User support & incident management

ML and DL Important Concept(Optional)

29 December 2025 18:40

◇ Supervised Learning

Regression Models (Continuous Output)

- Linear Regression
- Multiple Linear Regression
- Polynomial Regression

Use cases:

House price, salary prediction, demand forecasting

Classification Models (Discrete Output)

- Logistic Regression
- k-Nearest Neighbors (KNN)
- Support Vector Machines (SVM)
- Naive Bayes
- Decision Trees
- Random Forest
- Gradient Boosting
- XGBoost / LightGBM / CatBoost

Use cases:

Spam detection, fraud detection, disease

prediction

◇ Unsupervised Learning

Clustering Techniques

- K-Means
- Hierarchical Clustering
- DBSCAN

Use cases:

Customer segmentation, document grouping

◇ Dimensionality Reduction

- PCA (Principal Component Analysis)
- LDA (Linear Discriminant Analysis)
- t-SNE (Visualization focused)

Use cases:

Feature reduction, visualization, noise removal

◇ Evaluation Metrics

- Accuracy
- Precision

- Recall
- F1 Score
- ROC–AUC
- Confusion Matrix
- Mean Squared Error (MSE)
- R^2 Score

◇ Other Imp Concepts

- Bias–Variance Tradeoff
- Overfitting & Underfitting
- Cross Validation (K-Fold, Stratified)
- Data Leakage
- Data Drift
- Hyperparameter Tuning

DEEP LEARNING (DL) – CORE TOPICS

◇ Types of Neural Networks

- **ANN** → Basic neural networks
- **CNN** → Image & vision tasks
- **RNN** → Sequential data
- **LSTM** → Sequential data Long-term dependencies

- **GAN** → Data generation
- **Autoencoders** → Data generation, Feature learning, compression
- **RL (Reinforcement Learning)** → Agent-based learning

◇ Deep Learning Fundamentals

- Activation Functions (ReLU, Sigmoid, Softmax)
- Loss Functions
- Forward Propagation
- Backpropagation
- Gradient Descent
- Learning Rate

◇ Optimization & Regularization

- Optimizers: SGD, Adam, RMSprop
- Weight Initialization
- Batch Normalization
- Dropout
- Early Stopping
- Learning Rate Schedulers
- Vanishing / Exploding Gradients

- Loss Curves analysis

◇ **Training & Evaluation in DL**

- Epochs, Step & Batch Size
- Model Checkpointing
- Hyperparameter Tuning

◇ **Practical Deep Learning**

- Transfer Learning
- Fine-Tuning
- Model Compression
- Model Distillation
- Model Quantization
- Inference vs Training

Category	**Tools**
Data Processing	NumPy, Pandas
Visualization	Matplotlib, Seaborn
Classical ML	Scikit-learn
Deep Learning	TensorFlow, Keras, PyTorch

Main AI Road Map | Core AI Skills

29 December 2025 18:44

MODULE 1: Foundations of Modern GenAI
MODULE 2: Early stage LLM
MODULE 3: Modern LLMs, SLMs & Multimodal
MODULE 4: API for accessing the LLM
MODULE 5: Training an LLM from Scratch
MODULE 6: Fine-Tuning LLMs
MODULE 7: LLM Hosting on your own server and exposing as a API
MODULE 8: Prompt Engineering
MODULE 9: Context Engineering
MODULE 10: RAG & Multimodal RAG (The Important)
MODULE 10.1: Data Parsing
MODULE 10.2: Chunking and Embedding
MODULE 10.3: Vector Databases
MODULE 10.4: Retrieval Strategies for RAG Systems
MODULE 11: Agentic AI – Agent & Multiagent system
MODULE 11.1: Deep Agents
MODULE 12: Evaluation Strategies
MODULE 13: Guardrails
MODULE 14: MCP
MODULE 15: No Code Low Code Tools
MODULE 16: Cloud Services for GenAI
MODULE 17: Deployment Strategies
MODULE 18: Projects

Tools & Frameworks to Use for Gen AI

29 December 2025 18:45

LLM / GenAI Development Tools & Frameworks

LLM Foundations

- Huggingface
- PyTorch

Finetuning

- Unsloth
- LLaMA Factory

Vector Databases & Search (RAG Backbone)

- FAISS
- Chroma
- Qdrant
- Supabase
- pgvector
- Milvus
- Pinecone
- AWS OpenSearch
- Azure AI Search

Databases & Storage

- Neo4j
- MongoDB
- PostgreSQL
- Redis
- SQL Alchemy

LLM Application Building Framework

- LangChain
- Llama Index
- Haystack
- Lang Smith

Guardrails, Validation & Safety

- Guardrails.ai
- Nvidia Nemo
- OpenAI Guardrail

Local LLMs & Runtime

- Ollama

Experiment Tracking

- MLflow

Data Modelling

- Pydantic

MCP

- FastMCP

Document AI & Parsing

- Llama Parser
- Docling
- Google Document AI
- Amazon Textract
- python-docx
- PyMuPDF
- pdfplumber
- Parsio
- Unstructured.io
- Azure Document Intelligence

Agentic AI Frameworks

- Lang Graph
- Deepagent
- AutoGen
- n8n

Cloud GenAI Platforms

- Google ADK
- Agent Core
- AWS Bedrock
- Azure Cloud Foundry
- GCP Vertex AI

Evaluation & Observability

- RAGAS
- TruLens
- Llama Index Eval
- DeepEval
- OpenAI Evals

Prompt Engineering & Prompt Management

- LangChain Prompt Hub

- **PromptLayer**
- OpenAI Cookbook / Prompt Examples
- PromptHub (Community-driven repos)
- DSPy (Prompt-as-Code approach)
- Guidance (Microsoft)

Deployment, Infrastructure & CI/CD

- ECS + Fargate + EC2
- AWS SageMaker
- GitHub Actions
- Azure Virtual Machines (VMs)
- Azure Kubernetes Service (AKS)
- Azure Container Apps / ACI

Testing

- pytest

Pipeline Orchestration

- Apache Airflow

MODULE 1: Foundations of Modern GenAI

29 December 2025 18:45

- **Sequence Learning Basics**
- **Sequence Mapping Types (With Examples)**
- **Early Sequence Models**
- **Encoder–Decoder Architecture**
- **Transformer Architecture (Heart of LLMs)**
- **Why Text Needs Encoding**
- **Tokenization (Step-1 of Encoding , vocabulary creation)**
- **Classical Encoding Techniques (Quick History)**
- **Word Embedding Techniques**
- **Contextual Embeddings → Context awareness →Modern Approach**
- **Vector Space & Similarity**

- **Tools:**
- Python
- PyTorch
- Keras
- Hugging Face
- NLTK
- Spacy

MODULE 2: Early stage LLM

29 December 2025 18:47

- **Timeline: 2018 – 2021**
- **BERT (Bidirectional Encoder Representations from Transformers)**
- **RoBERTa**
- **DistilBERT**
- **T5 (Text-to-Text Transfer Transformer)**
- **GPT-1**
- **GPT-2**
- **GPT-Neo / GPT-J**

- Encoder and Decoder
- Masked Language Modeling (MLM)
- Autoregressive text generation
- Attention mechanism
- Cross-attention
- Multi-head attention
- Tokenization
- Positional encoding
- Pretraining vs fine-tuning
- inference
- loss functions
- Activation function

- **Tools& Frameworks:**
- Python
- PyTorch
- Keras

- Hugging Face
- NLTK
- Spacy

MODULE 3: Modern LLMs, SLMs & Multimodal

29 December 2025 18:50

Text / Vision-Language / Multimodal Models / Audio / Speech Models /Code-Focused LLMs

- GPT family
- Gemini family
- Deepseek family
- Llama Family
- Mistral Family
- Claude Family
- Qwen Family
- Gemma Family
- Phi family
- LLaVA
- BLIP / BLIP-2
- CLIP
- Whisper
- Dalle
- VALL-E
- CodeLLaMA
- StarCoder / StarCoder2
- DeepSeek-Coder
- Phi-3-Mini (code)

- LLM Leader boards:
- https://lmarena.ai/leaderboard
- https://lmarena.ai/
- https://www.vellum.ai/llm-leaderboard?utm_source=chatgpt.com
- https://llm-stats.com/?utm_source=chatgpt.com
- https://scale.com/leaderboard

- Embedding models leader board:
- https://huggingface.co/spaces/mteb/leaderboard

Feature	LLM	SLM
Parameters	Billions–Trillions (Wikipedia)	Millions–Few Billions (Wikipedia)
Resource Needs	High (GeeksforGeeks)	Low (redhat.com)
Scope	Broad & General (Wikipedia)	Specific & Domain-focused (redhat.com)
Training Data	Very Large (GeeksforGeeks)	Smaller/Curated (Splunk)

Speed & Cost	Slower, Expensive (Microsoft)	Fast, Cost-Efficient (Microsoft)

SLMs:

1. **Mistral NeMo** – Efficient small model
2. **Gemma 2** – Scalable smaller model
3. **Phi-3.5 / Phi-something Mini** – Lightweight models from Microsoft-led ecosystem
4. **Qwen 2 (0.5B / 1B / 7B)** – Scalable SLM sizes
5. **StableLM-Zephyr 3B** – Open-source smaller model
6. **MobileLLaMA / TinyLLaMA / MiniCPM-V** – Very compact variants

MODULE 4: API for access the LLM

29 December 2025 18:53

API Ecosystem

- OpenAI SDK
- Anthropic SDK
- Google (Gemini Ecosystem) AI Studio SDK
- Meta API / Open-Source Hosting SDK
- GROQ API SDK

Other Notable GenAI APIs

- Cohere API
- Together AI API
- Fireworks AI API
- Replicate API
- Perplexity API
- DeepSeek API
- Mistral API
- Open router API

Cloud Provider Managed APIs

- Microsoft Azure Cloud foundry(Azure OpenAI)
- Amazon AWS Bedrock
- Google Cloud Vertex AI

Tools& Frameworks:

- LangChain
- Llama Index
- Haystack

MODULE 5: Training an LLM from Scratch

29 December 2025 18:57

- LLM Pretraining Training pipeline
- Data Collection for LLMs, Data Sources & Challenges
- Data Preprocessing & Text Cleaning
- Vocabulary Creation, Tokenization & Vocabulary Design
- Data Augmentation for LLM Training
- Transformers & LLM Architecture
- Model Scaling Strategies
- Pretraining Objectives
- Causal Language Modeling
  - Autoregressive generation
  - Next-token prediction
- Training Process, Training Loop & Optimization
- Distributed Training & Infrastructure
- Hardware, Software Stack
- Monitoring, Debugging & Checkpointing
- LLM Evaluation
- Evaluation Types
- Metrics
- Benchmarks
- Ethical & Responsible AI
- Topics
  - Bias detection
  - Fairness
  - Privacy leakage
  - Hallucination
  - Truthfulness (TruthfulQA)

- **Tools& Frameworks:**
- Python
- PyTorch
- Hugging Face
- NLTK
- Spacy
- DeepSpeed
- Megatron-LM
- Experiment tracking (W&B, MLflow)
- GPU Knowledge

MODULE 6: Fine-Tuning LLMs

29 December 2025 19:03

- Fine-Tuning in Deep Learning (CNNs)
- Why Fine-Tuning is Limited in RNN / LSTM?
- Hugging Face Ecosystem vs LangChain
- Fine-Tuning Classical Language Models
- Knowledge Distillation in LLMs
- LLM Quantization
- Fine-Tuning Large Language Models (LLMs)
- API-Based Model Fine-Tuning
- Best Frameworks for LLM Fine-Tuning
- Types of Fine-Tuning
  - Full Fine-Tuning
  - LoRA (Low-Rank Adaptation)
  - QLoRA (Quantized LoRA)
  - PEFT (Parameter-Efficient Fine-Tuning) – Overview
- Dataset Preparation
- Model Packaging
  - Hugging Face checkpoints
  - Safetensors format
  - GGUF format
  - GGML format
- Fine-Tuning Vision-Language Models
- Reinforcement Learning with Human Feedback (RLHF), DPO, ORPO, GRPO
- Embedding Model Fine-Tuning

MODULE 7: LLM Hosting on your own server and exposing as a API

29 December 2025 19:06

Hosting Options (Where to Run LLMs)
Bare Metal Server: physical server dedicated only to you, rented from a provider. No virtualization layer. No shared hardware.
Cloud VM (GPU) or Cloud Managed service: Compute provided by cloud providers like **AWS, Azure, GCP**.
On-Prem Server: Servers owned and hosted by your organization

Inference Servers (How Models Are Served)
vLLM
llama.cpp
FastAPI Custom Server

Cloud Managed (AWS)
Fine-tune base model (LoRA) on **AWS SageMaker**
Deploy model as a **SageMaker Endpoint**
Configure access using

- **API Gateway** OR
- **Application Load Balancer (ALB)**

Perform inference using

- **AWS Lambda** (lightweight / burst)
- **ECS Fargate** (container-based scaling)

Call API from a client application

Inference Challenges (Production Reality)
Latency — time to first token
Throughput — requests per second
Resource Usage — GPU memory, VRAM, CPU
Cold start & scaling issues

MODULE 8: Prompt Engineering

29 December 2025 19:09

- Core Prompting Concepts
- System vs User Prompts
- Zero-Shot Prompting
- Few-Shot Prompting
- Chain-of-Thought (CoT)
- Self-Consistency
- ReAct (Reason + Act)
- Fallback Prompts

- Prompt Design Strategies
- Task-Wise Prompting
- Domain Prompting

- Prompt Management (Production Grade)
- Prompt Library
- Jinja2 Templates
- YAML-Based Prompt Configs

- Structured & Controlled Prompting
- Output Control
  - JSON / YAML outputs
  - Schema-based prompts
  - Guarded outputs (format enforcement)
- Always Remember ⚠ ⚠ VERY IMPORTANT
  - Token Cost Optimization
  - Context Window Optimization

- Tools & Frameworks to Follow
  - LangChain Prompt Hub
  - PromptLayer
  - OpenAI Cookbook / Prompt Examples
  - PromptHub (Community-driven repos)
  - DSPy (Prompt-as-Code approach)
  - Guidance (Microsoft)
  - LangChain Prompt Template
  - Llama index Prompt Template

MODULE 9: RAG & Multimodal RAG(The Important)

29 December 2025 19:23

What You Need to Master in Retrieval-Augmented Generation (RAG)

- Understanding why LLMs hallucinate
- End-to-end RAG system architecture
- Data ingestion and parsing
- Chunking strategies (when and when not to use them)
- Embedding generation and selection
- Vector database design and management(Local / Open-Source, 3rd party Managed / Cloud Managed)
- Metadata design and filtering
- Retrieval, ranking, and re-ranking strategies
- Prompting with retrieved context
- Context engineering for reliable outputs
- Memory management
- Caching strategies for performance and cost
- Evaluating RAG quality and reliability
- How multimodal RAG systems work
- Common failure modes in RAG systems
- **Tools & Frameworks to Follow**

MODULE 10: Context Engineering

29 December 2025 19:18

Context Engineering is the art of curating and structuring the right input information for LLMs so they produce useful, accurate, and context-aware outputs.**Main Components of Context Engineering**

- Chunking Data
- Prompt Templates
- Retrieval Selection
- Context Ranking
- Context Injection Strategy
- Maximizing Utility without Exceeding Context Window

MODULE 11: Agentic AI - Agent, Multiagent system and Deep Agent

29 December 2025 19:28

- Single-agent architectures
- Multi-agent systems(Supervisor, Network, Hierarchical)
- Deep agents
- LLMs as the reasoning and decision-making core
- Tools as agent interfaces
- Agent orchestration layers
- Memory and state management
- Prompting strategies for agents
- Human-in-the-loop mechanisms
- Loop prevention and safety controls
- Execution limits (max steps)
- Token usage and cost budgeting
- Agentic RAG architectures
- Inter-agent collaboration and coordination
- **Tools & Frameworks to Follow**

MODULE 12: Evaluation Strategies

29 December 2025 19:35

observability & Debugging (Foundation)

- **Logging and monitoring for GenAI systems**
- **Conversation-level logs for debugging**
- Tracing prompts, context, tools, and responses
- Understanding *why* a model behaved a certain way

Why Classical Evaluation Breaks in GenAI

- Why **traditional ML evaluation** fails for GenAI
- Limits of **model-level metrics** for system behavior
- Difference between **model-level vs task-level evaluation**
- Why static benchmarks don't reflect real usage

Core Evaluation Strategies for GenAI

- **LLM-as-a-judge**: how it works and where it fails
- **Human-in-the-loop evaluation**
- Combining **automated + human feedback**
- **Offline vs online evaluation** strategies
- Evaluating outputs against **task-specific goals**

Evaluating Complex GenAI Systems

- **End-to-end evaluation of RAG pipelines**
- Evaluating **single-agent and multi-agent systems**
- Measuring grounding, relevance, and faithfulness
- Detecting hallucinations and reasoning failures

Beyond Accuracy: System-Level Metrics

- **Cost metrics** (token usage, inference cost)
- **Latency metrics** (response time, tool delays)
- **User experience (UX) metrics**
- Trade-offs between quality, speed, and cost

Classical Metrics (And Their Limits)

- **Perplexity and loss** (research-focused)
- **Token-level metrics** (high-level view)
- Why these are useful for research but **not sufficient for production**

Task-Specific Metrics (When They Work)

- **Accuracy** (QA, classification tasks)
- **BLEU / ROUGE** (summarization, translation)
- **Exact match vs semantic match**
- Why many metrics:
  - Reward surface similarity
  - Penalize valid alternative answers

  Correct intent ≠ exact wording.

Common Evaluation Anti-Patterns

- Evaluating models instead of systems

- Relying on a single metric
- Ignoring cost and latency
- Over-trusting LLM-as-a-judge
- Optimizing metrics instead of user outcomes

Tools & Frameworks to Use in Practice

- **RAGAS** – RAG-specific evaluation
- **TruLens** – tracing + feedback functions
- **LangSmith** – prompt, agent, and chain evaluation
- **Custom evaluation pipelines** for production systems

Tool / Framework	Type	Best Use Case
IntellAgent	Open-source evaluation framework	Multi-agent simulation & diagnosis (arXiv)
AgentBench	Benchmark suite	LLM agent benchmarking (GitHub)
Azure AI Evaluation SDK	Cloud SDK	Production batch evaluation & telemetry (Microsoft Learn)
Maxim AI / Langfuse / Comet / Arize	Production observability tools	Monitoring and continuous evaluation (Maxim AI)
DeepEval	OSS evaluation module	Traceable agent component evaluation (Confident AI)
Custom simulation + LLM-as-Judge	DIY approach	Flexible, domain-specific testing (Fiddler AI)

MODULE 13: Guardrails

29 December 2025 20:00

- Introduction to Guardrails (Foundations)
- Guardrails in Traditional Software Engineering (Pre-GenAI)
- Guardrails in AI & LLM Applications
- Core Objectives of Guardrails
- Input Validation Guardrails
- Output Validation Guardrails
- Schema-Based Guardrails with Pydantic
- Prompt Injection Attacks
- Guardrails Tools & Frameworks

MODULE 14: MCP

29 December 2025 20:02

MCP Topics (Verified & De-duplicated)

- What is Model Context Protocol (MCP)?
- Why MCP was introduced (Problems with traditional tool calling)
- MCP vs Plugins vs Function Calling
- MCP in GenAI ecosystem (RAG, Agents)
- Real-world use cases of MCP
- MCP architecture overview (Client ↔ Server ↔ LLM)

MCP Core Components

- MCP has 3 parts
- MCP Host
- MCP Client
- MCP Server

MCP Transports

- STDIO transport (local execution)
- SSE transport (event streaming)
- Streamable HTTP transport (production standard)
- Stateful vs Stateless MCP servers
- When to use which transport
- Security implications of transports

MCP Python SDK

- MCP Python SDK overview
- MCP Python SDK architecture
- FastMCP vs Low-Level MCP Server
- Installing MCP using uv and pip
- MCP CLI tools (mcp dev, mcp run, mcp install)
- Project structure best practices

- Building MCP Servers with FastMCP (Core)

MCP Capabilities

- MCP Tools
- Structured Output in MCP
- MCP Prompts (Reusable Intelligence)
- What are MCP Prompts?
- Prompt templates vs Tool prompts
- Multi-message prompt responses
- Context Object deep dive

Advanced MCP

- User Interaction in MCP (Advanced)
- MCP Sampling & LLM Interaction
- Authentication & Security in MCP
- Running MCP Servers (Dev → Prod)
- Mounting MCP to existing applications
- Building MCP Clients (Python)
- OAuth-enabled MCP Clients
- Advanced MCP Server Patterns
- Pagination & Large Data Handling
- MCP for Agentic AI Systems

MODULE 15: No Code Low Code tool(Optional)

29 December 2025 20:06

N8N Documentation

- AI Automation & n8n Foundations
- n8n Setup & Interface
- n8n Concepts & Nodes
- Automation with n8n
- JSON with n8n
- API Explained in Depth
- API with n8n
- AI in Automation
- Agents in Automation
- Multi-Agent Architecture (Chain, Parallel, Controller, React, Hierarchical)
- AI Personal Assistant using Swarm Agents
- MCP (Model Context Protocol) Explained
- MCP in n8n Cloud
- MCP in n8n Self Hosted
- RAG explained
- RAG on n8n
- RAG using Supabase
- RAG using Pinecone
- Running n8n on Local and Self-Hosted Cloud
- Mastering Local n8n
- n8n Self Hosting Kit
- RAG with Self Hosted Kit
- Social Media Automation
- GitHub PR Automation Agent

- WhatsApp Agent
- Video Creation with Gemini
  This proves:
  - Automation
  - AI
  - Agents
  - APIs
    all together

Tool	Best For	Tech Level
Zapier	Business workflows, many integrations	No-code
Make	Detailed automation with logic	No-code / Low-code
Power Automate	Enterprise + Microsoft ecosystem	Low-code
Apache Airflow	Data pipelines & scheduling	Dev/Engineer
Pipedream	API + script-centric automation	Low-code / Pro-code
Activepieces	Open-source automation	No-code / Dev
Gumloop	AI/LLM workflows	No-code / AI focus
Latenode	Custom low-code workflows	Low-code

Integrately	Simple app automations	No-code
IFTTT	Personal automation	No-code

MODULE 16: Cloud Services for GenAI

29 December 2025 20:07

AWS AI Services (Must-Know)

Amazon SageMaker – Core platform for ML training, deployment, and MLOps
Amazon Bedrock – Managed GenAI / foundation model platform
Amazon OpenSearch Service – Vector database for search and RAG
Amazon Textract – Document intelligence (OCR + extraction)
Amazon Comprehend – NLP services (entities, sentiment, topics)
Amazon Rekognition – Computer vision (image & video analysis)
Amazon Transcribe – Speech-to-text services
Amazon Personalize – Recommendation systems
Agent Core – For Building Agent

Azure AI Services (Must-Know)

Azure AI Foundry (Top-Level Platform)

Azure OpenAI Service
Azure Machine Learning
Azure AI Vision
Azure AI Custom Vision
Azure AI Speech Services
Azure AI Translator
Azure AI Document Intelligence
Azure Bot Service
Azure AI Personalizer
Azure AI Anomaly Detector
Azure AI Search

GCP AI Services
Vertex AI *(Most important platform)*
Gemini (via Vertex AI)
Vertex AI Search *(formerly Gen App Builder / Search)*
Document AI
Vision AI
Speech-to-Text & Text-to-Speech

Natural Language AI

Use Case	AWS	Azure	GCP
GenAI models	Bedrock	Azure OpenAI	Gemini
ML Platform	SageMaker	Azure ML	Vertex AI
Vector / RAG	OpenSearch	AI Search	Vertex AI Search
OCR / Docs	Textract	Doc Intelligence	Document AI
Vision	Rekognition	AI Vision	Vision AI
Speech	Transcribe	Speech	Speech-to-Text
NLP	Comprehend	Text Analytics	Natural Language AI

MODULE 17: Deployment Strategies

29 December 2025 20:12

#	Method	Type	Best For	Infra Mgmt	Complexity
1	Lambda	Serverless	APIs, events	❌	★
2	API GW + Lambda	Serverless	REST APIs	❌	★★
3	ALB + Lambda	Serverless	Routing APIs	❌	★★
4	ECS (EC2)	Container	Control heavy apps	☑	★★★
5	ECS Fargate	Container	Modern backend	❌	★★
6	App Runner	Container	Fast deploy	❌	★
7	EKS	Kubernetes	Enterprise scale	☑	★★★★
8	EKS + Fargate	Kubernetes	Serverless K8s	❌	★★★★
9	EC2	VM	Legacy / GPU	☑	★★★
10	EC2 + ASG	VM	Scalable VM apps	☑	★★★
11	S3 Static	Frontend	Static sites	❌	★
12	S3 + CloudFront	Frontend	Prod frontend	❌	★★
13	Elastic Beanstalk	PaaS	Simple apps	❌	★★
14	Step Functions	Workflow	Orchestration	❌	★★
15	EventBridge	Event	Event driven	❌	★★
16	CodePipeline	CI/CD	AWS CI/CD	❌	★★
17	GitHub Actions	CI/CD	Modern CI/CD	❌	★★
18	SageMaker	ML	ML production	❌	★★★

Core Cloud Building Blocks (Quick)

- Compute
- Storage
- Networking
- IAM / Security

- Amazon S3
- Amazon EC2
- Amazon CloudFront
- AWS CloudFormation
- AWS Identity and Access Management (IAM)
  - IAM Users
  - IAM Roles

    - IAM Policies
- Authentication (via AWS IAM / Cognito)
- Amazon DynamoDB
- Amazon RDS
- Amazon OpenSearch Service
- Amazon Redshift
- Amazon Bedrock
- Amazon Q

#	Method (Azure)	Type	Best For	Infra Mgmt	Complexity
1	Azure Functions	Serverless	APIs, events	✗	★
2	API Management + Functions	Serverless	REST APIs	✗	★★
3	Application Gateway + Functions	Serverless	Routing APIs	✗	★★
4	AKS (VM-based nodes)	Container	Control heavy apps	☑	★★★★
5	Azure Container Apps	Container	Modern backend	✗	★★
6	Azure App Service (Container)	Container	Fast deploy	✗	★
7	AKS	Kubernetes	Enterprise scale	☑	★★★★
8	AKS + Virtual Nodes	Kubernetes	Serverless K8s	✗	★★★★
9	Azure VM	VM	Legacy / GPU	☑	★★★
10	VM Scale Sets	VM	Scalable VM apps	☑	★★★
11	Azure Static Web Apps	Frontend	Static sites	✗	★
12	Blob Storage + Azure CDN	Frontend	Prod frontend	✗	★★
13	Azure App Service	PaaS	Simple apps	✗	★★
14	Logic Apps	Workflow	Orchestration	✗	★★
15	Event Grid	Event	Event driven	✗	★★
16	Azure DevOps Pipelines	CI/CD	Azure CI/CD	✗	★★
17	GitHub Actions	CI/CD	Modern CI/CD	✗	★★
18	Azure Machine Learning	ML	ML production	✗	★★★

#	Method (GCP)	Type	Best For	Infra Mgmt	Complexity
1	Cloud Functions	Serverless	APIs, events	✗	★
2	API Gateway + Cloud Functions	Serverless	REST APIs	✗	★★
3	Cloud Load Balancer + Functions	Serverless	Routing APIs	✗	★★
4	GKE (Node pools)	Container	Control heavy apps	☑	★★★★
5	Cloud Run	Container	Modern backend	✗	★★

6	App Engine (Flex/Std)	Container	Fast deploy	☒	★
7	GKE	Kubernetes	Enterprise scale	☑	★★★★
8	GKE Autopilot	Kubernetes	Serverless K8s	☒	★★★★
9	Compute Engine	VM	Legacy / GPU	☑	★★★
10	Managed Instance Groups	VM	Scalable VM apps	☑	★★★
11	Firebase Hosting	Frontend	Static sites	☒	★
12	Cloud Storage + Cloud CDN	Frontend	Prod frontend	☒	★★
13	App Engine	PaaS	Simple apps	☒	★★
14	Workflows	Workflow	Orchestration	☒	★★
15	Eventarc	Event	Event driven	☒	★★
16	Cloud				

AWS	**Azure**	**GCP**
Lambda	Azure Functions	Cloud Functions
API Gateway	API Management	API Gateway
ECS Fargate	Container Apps	Cloud Run
App Runner	App Service	App Engine
EKS	AKS	GKE
EC2	VM	Compute Engine
ASG	VM Scale Sets	Managed Instance Groups
S3 + CloudFront	Blob + CDN	GCS + CDN
Step Functions	Logic Apps	Workflows
EventBridge	Event Grid	Eventarc
SageMaker	Azure ML	Vertex AI

MODULE 18: Project
29 December 2025 20:15

LLM = Generation(Text, Code, Image, Audio, Video) means LLM-based AI projects fundamentally involve generation.
Regardless of the type of application we build—conversational system, automation systems, report generators—fundamental remains same: That is generation

that is why they are called **Generative AI**.

Conversational AI / Chatbots/Question Answering Systems
- Human-like chat assistants
- Customer support bots
- Voice assistants
- Coding Assistants
- Language Translation & Language Services

Content Generation Systems
- Report generation
- Code Generation & Coding Assistants
- Synthetic Data / Data Augmentation
- Blog / article writing

Document Processing & Understanding
- Data extraction or parsing
- Summarization of long docs
- document interpretation

Enterprise Automation
SDLC Automation
Jira Ticket→Design(AI generates)→ Development (AI generates Boilerplate)→Code Review(AI Powered)→testing(AI generates test cases / scripts)→Deployment→ Monitoring & Alerts

Customer Service & Support Automation
Customer Query / Ticket→ Intent Detection (AI)→ Context Retrieval (KB / CRM / RAG)→ Response Generation (LLM)→ Confidence / Safety Check (AI)→ Auto-Reply OR Human Escalation→ Ticket Update & Closure→ Monitoring & Feedback

HR Automation
Employee Request / Resume / HR Ticket→ Requirement Classification (AI)→ Data Extraction (AI parses resume / form)→ Decision Support (Policy / Rules + AI)→ Action Execution (Approval / Rejection / Scheduling)→ Notification to Employee→ Audit Logs & Monitoring

Report Generation & Decision Support
Data Sources→ Data Aggregation→ Data Cleaning & Validation→ Insight Generation (AI)→ Report Draft Generation (LLM)→ Human Review (Optional)→ Final Report Delivery→ Monitoring & Feedback

High-Level Design (HLD)
The architect decides:
- How many floors the house will have
- Where the kitchen, bathroom, and rooms will be located
- Number of rooms and overall layout
- Flow of water and electricity

Low-Level Design (LLD)
The engineer decides:
- Grade of cement
- Size of electrical wires
- Diameter of pipes
- Model of switches

HLD → overall system architecture
LLD → Classes, methods, DB tables, APIs(get, post), logic

High-Level Design (HLD COMPONENTS)
System architecture/ Tech lead / App architecture / strategy
- Frontend (UI / UX)
- API Layer(REST, gRPC, WebSocket, GraphQL)(**Versioning, Rate limiting, Idempotency)**

- Asynchronous Processing / background workers/Message queues/ Event-driven systems/ Workers & consumers
- Security & Auth Layer/Authentication & Authorization/API keys / OAuth / JWT/Encryption at rest & in transit/Secrets management /Role checks/ Permission logic
- Backend Services
- Databases(Relational, NoSQL, Vector DB)
- Data Integration Layer (Data Engineering: ETL, streaming, batch pipelines)
- AI / LLM Layer
- Caching Layer
- Observability, Monitoring & Error handling(Logs, metrics, tracing, alerts)
- Cloud Infrastructure / Deployment Architecture / Blue-green / Canary deployments/ Infrastructure as Code

- **Scalability**
  - Auto-scaling(vertical or horizontal)
  - Stateless services (JWT token / DB / Redis)
  - Sharding
- **Availability**
  - Load balancers
  - Single point of failure removal
  - Multi-AZ / Multi-region(server)
  - Failover strategies
- **Reliability & Fault Tolerance**
  - Retries
  - Timeouts
  - Circuit breakers
  - Graceful degradation
- **Latency & Performance**
  - Caching (Redis, CDN)
  - Async processing
  - Background jobs
  - Pagination & batching
  - Indexing(DB)
  - Read replicas(DB)
  - Batch processing
  - Time complexity
  - DB query optimization

- **Consistency Models**
  Distributed system
  C – Consistency
  A – Availability
  P – Partition tolerance (network failure)
- **Cost Optimization**

Trade-offs(No perfect design exists.)
- Consistency vs Availability
- Cost vs Performance
- Simplicity vs Flexibility
- Accuracy vs Latency

LLD COMPONENTS
Engineer/ Devs
1. **API Design**: routes, HTTP methods, request/response schemas, status codes, error payloads, versioning
2. **Data Models**: DTOs/schemas (Pydantic/dataclasses), enums, validation rules
3. **Class Design**(Attributes / Fields/ Methods / Functions)
4. **Service Layer**: business logic, orchestration, workflows
5. **DAO/Repository Layer**: DB access patterns, queries, transactions
6. **Database Schema Design**: tables/collections, PK/FK, indexes, constraints, migrations
7. **Method-Level Data Flow**: validation → processing → persistence → response
8. **Error Handling**: exceptions, error mapping, retries/timeouts, fallbacks
9. **Async/Concurrency (if any)**: queues, background jobs, race handling
10. **Logging** : logs, metrics, tracing, Error handing
11. **Config Management**: env vars, feature flags, config files

12. **Testing Plan**: unit/integration tests, test data, mocking strategy
13. **Performance Notes**: pagination, batching, limits, query optimization
14. **Fallbacks**

Let's Design a project and understand what all concept will be used here:
Project Name Document portal: Upload, Document Analysis, Chat, Comparison
Single Doc Chat
Multi Doc Chat
Analysis
Compare (Doc A vs Doc B)
Two modes:

1. **Semantic compare**:
   - retrieve top topics from both docs
   - align similar sections
   - summarize differences with citations
2. **Q-driven compare**:
   - user asks: "Compare eligibility criteria"
   - system retrieves relevant chunks from both docs
   - generate side-by-side response

Frontend
Streamlit (fast) or React (later) Upload UI, chat UI, comparison UI, citations viewer
API Layer

- FastAPI REST endpoints
- WebSocket for streaming responses (optional)

Backend Services (Core)

- Auth/User service (optional)
- Document service (upload, versioning, access control)
- Ingestion pipeline service (parse → chunk → embed → index)
- Retrieval service (vector search, filters, reranking)
- Chat orchestration service (memory, query rewrite, context condense)
- Comparison service (alignment, diff summary, similarity)
- Evaluation service (offline eval + feedback)

Databases

- Relational DB (Postgres): users, sessions, docs, metadata, audit
- Object storage (S3): original files, extracted artifacts, thumbnails
- Vector DB:
  - Local/dev: FAISS/Chroma
  - Prod: Pinecone/Qdrant/Weaviate (recommended Qdrant for self-host)
- Cache (Redis): session cache, CAG cache, rate-limit, job status

Async Processing

- Queue (SQS / Redis Queue / Celery)
- Workers: ingestion jobs, embedding jobs, OCR, comparison batch, evaluation

AI/LLM Layer Components

A) Model Router
Chooses model based on:

- latency budget
- cost budget
- context length required
- safety policy
- task type (summarize vs QA vs compare)

Example policies

- chat: Groq (fast)
- heavy compare: OpenAI (better reasoning)
- offline eval: local LLM
- vLLM for self-host (optional)

B) Prompt + Context Engine

- Prompt templates (system + developer + user)
- Context policies:
  - max tokens for context
  - citation enforcement
  - "no answer if not in sources"

- Context condensation:
  - compress chat history into “memory summary”
- Query rewriting:
  - multi-query (RAG-fusion)
  - hyde (optional)
- Tool calling (optional)
  - citation formatter
  - chunk viewer

C) Retrieval Engine (Advanced RAG)

Pipeline:

1. Query Normalize
2. Query Rewrite (optional)
3. Retrieve K chunks (vector search + metadata filters)
4. **MMR** (diversity)
5. Rerank (cross-encoder / LLM reranker)
6. Context packer (token-aware packing)
7. Answer generator with citations

D) CAG (Cache-Augmented Generation)

Cache key options:

- normalized query + doc-set + topK + prompt version

Cache value:

- final answer + citations + chunk ids

TTL:

- per doc version

Best used for:

- repeated enterprise queries
- repeated compliance questions

E) MM-RAG (Optional)

If doc contains images/charts:

- image captioning (BLIP/InternVL)
- store captions as text chunks + image embeddings (CLIP)
- retrieve both text + image captions

Advanced RAG Design (Your listed topics mapped)

6.1 Chunking strategies

- Recursive character splitter baseline
- Header-aware chunking (DOCX headings)
- Page-aware chunking (PDF page boundaries)
- Overlap: 10–20%

6.2 Embeddings

- HF: bge-small/large
- OpenAI embeddings
- Keep embedding model version in metadata

6.3 Vector Store choice

- Dev: FAISS (fast local)
- Prod:
  - Pinecone (managed)
  - Qdrant (self-host on ECS/EKS)

6.4 Retrieval strategies

- Similarity search: cosine/L2 (depends on embedding model)
- MMR for diversity
- RAG fusion: multi-query retrieval
- Reranker: cross-encoder (best accuracy)

6.5 Query rewriting & context condensation

- rewrite query if user refers to “that section”
- summarize chat history to keep token budget

7) Guardrails

- Prompt injection filters
- PII redaction (optional)
- “Answer only from sources”
- Refuse if insufficient context
- Citation required in response template

8) Evaluation

Metrics:

- Faithfulness (groundedness)
- Context precision/recall
- Answer relevance
- Latency
- Cost per query

Tools:

- RAGAS / TruLens (optional)

Observability

- Logs: CloudWatch
- Metrics: Prometheus/Grafana (optional) or CloudWatch metrics
- Tracing: OpenTelemetry (optional)
- Alerts: CloudWatch Alarms

Cloud Deployment

- AWS: ECR + ECS Fargate + ALB + S3 + RDS + Redis (Elasticache) + Secrets Manager
- CI/CD: GitHub Actions + SonarQube + IaC (Terraform/CloudFormation)

Scalability / Availability / Reliability / Latency

Scalability

- **Stateless FastAPI** behind ALB
- Horizontal autoscaling (ECS service autoscaling)
- Worker autoscaling separately
- Vector DB scaling:
  - managed Pinecone OR Qdrant cluster
- **Sharding** by tenant/collection for vector store (logical partition)

Availability

- Multi-AZ for RDS + Redis
- ALB + ECS across multiple AZs
- S3 inherently HA
- Failover: DB failover + retry

Reliability & Fault Tolerance

- retries + backoff for:
  - embeddings API
  - vector DB writes
  - LLM calls
- timeouts everywhere
- circuit breaker for provider outages (OpenAI down → Groq fallback)
- graceful degradation:
  - if reranker fails → skip rerank
  - if MM-RAG fails → text-only RAG

Latency & Performance

- caching:
  - Redis response cache (CAG)
  - embedding cache for duplicate chunks
- async:
  - ingestion off request path
- pagination/batching:
  - chunk upload in batches
  - vector upsert in batches
- indexing + DB tuning:
  - indexes on doc_id, tenant_id, status, created_at
- read replicas (optional):
  - analytics/usage read load

Consistency Model

- Doc ingestion: **eventual consistency**
  - indexing finishes after upload
- User expects:
  - “processing” state until READY

Cost Optimization

- Use Groq for fast cheap inference where possible
- Use OpenAI only for compare/summarize heavy tasks

- Chunk dedup + embedding reuse
- Compress old chat history
- Vector DB retention policies per tenant

Tradeoffs (explicit)

- Accuracy vs latency: reranker improves accuracy but adds latency
- Cost vs performance: OpenAI best reasoning but expensive
- Consistency vs availability: ingestion pipeline is async for availability

Low-Level Design (LLD)

API Design (Routes)

Documents

- POST /v1/documents/upload
- GET /v1/documents/{doc_id}
- GET /v1/documents?status=READY
- POST /v1/documents/{doc_id}/reingest
- DELETE /v1/documents/{doc_id}

Jobs

- GET /v1/jobs/{job_id}

Chat

- POST /v1/chat/sessions (create session)
- POST /v1/chat/query (doc_id / collection_id + question)
- GET /v1/chat/sessions/{session_id}/history
- WS /v1/chat/stream (optional streaming)

Compare

- POST /v1/compare
  Payload: {doc_a, doc_b, mode: semantic|question, question?}

Eval (optional)

- POST /v1/eval/run
- POST /v1/feedback

Data Models (Pydantic style)

Document

- doc_id (uuid)
- tenant_id
- name
- type
- s3_uri_raw
- status: UPLOADED|PROCESSING|READY|FAILED
- created_at, updated_at
- version

ChunkMetadata

- chunk_id
- doc_id
- page_no
- section_title
- source_span
- hash (dedup)
- embedding_model
- vector_id

ChatSession

- session_id
- tenant_id
- scope: SINGLE_DOC|MULTI_DOC
- doc_ids / collection_id
- created_at

ChatTurn

- turn_id
- session_id
- role: user/assistant
- content
- citations: list
- token_usage

- latency_ms

DB Schema (Postgres)
Tables:
- tenants
- users
- documents
- document_versions
- chunks (metadata only)
- chat_sessions
- chat_turns
- compare_runs
- jobs
- audit_logs

Indexes:
- documents(tenant_id, status)
- chunks(doc_id, page_no)
- chat_turns(session_id, created_at)
- jobs(status, created_at)

Core Classes (Backend)
- DocumentController
- IngestionService
- ParserService
- ChunkingService
- EmbeddingService (HF/OpenAI/BGE)
- VectorStoreService (FAISS/Qdrant/Pinecone interface)
- RetrievalService (MMR + rerank)
- ChatOrchestrator
- CompareService
- CacheService (Redis)
- ModelRouter
- PromptManager
- GuardrailService
- TelemetryService

Concrete "1M/hour readiness" checklist
- ☑ **CDN** for frontend
- ☑ **Stateless API** + ECS autoscaling
- ☑ **Redis rate limiting**
- ☑ **CAG response cache**
- ☑ **Retrieval cache**
- ☑ **Queue ingestion and heavy compare**
- ☑ **Managed vector DB** or clustered Qdrant + caching
- ☑ **RDS Multi-AZ + read replicas**
- ☑ **Multi-provider LLM router + fallback**
- ☑ **Observability**: p95 latency, queue depth, cache hit ratio, LLM error rate
- ☑ **Load testing** with k6/locust

Simple final answer

To handle 1M users/hour, we separate real-time and async paths, scale stateless APIs horizontally on ECS Fargate behind ALB, use Redis for rate limiting and caching (CAG + retrieval cache), offload ingestion/comparison to SQS workers, use a scalable vector DB (Pinecone/Qdrant cluster), add DB read replicas, and implement graceful degradation + multi-LLM fallback to maintain availability under spikes.

Project Sources:
https://github.com/ashishpatel26/500-AI-Agents-Projects,
https://github.com/GURPREETKAURJETHRA/END-TO-END-GENERATIVE-AI-PROJECTS