1) Short Intro & MVP

MVP features (minimum viable Jarvis):

- Voice input (wake word optional) → ASR (speech-to-text).
- Text/voice → Large Language Model (LLM) with context window.
- Output voice (TTS) + short visual UI.
- Simple tools: web search, weather, calculator, open apps (desktop), run shell commands (sandboxed).
- Short-term memory (session) + simple persistent memory (user prefs).

Why phased approach: Complete Jarvis bahut bada kaam hai — isliye stepwise build karein: pehle working MVP, phir features add karo.

---

2) High-level Architecture (components)

1. **Input layer** — Mic, text input, wake-word detector.
2. **Preprocessing** — VAD (voice activity detection), noise reduction, ASR (Speech → text).
3. **Dialog/NLU** — Intent/entity extraction (optional), context manager.
4. **Core LLM + Reasoner** — Main generative model (LLM) + prompt engineering + tool invocation layer (agents).
5. **Knowledge & Memory** — Vector DB (embeddings), short-term convo buffer, long-term memory store.
6. **Tool Executors** — Web search, calendar API, system control, home automation, code execution sandbox.
7. **Postprocessing & Output** — TTS + UI response.
8. **Orchestration** — API gateway (FastAPI), queue (Redis), workers, streaming.
9. **Monitoring & Safety** — logging, content filters, user consent, permission model.

---

3) Phase-wise Step-by-Step Roadmap (Practical)

Phase 0 — Foundations (skillset + infra basics)

- **Skills to learn:** Python, Git, Linux basics, REST APIs, async programming, basic web (HTML/CSS/JS), Docker.
- **Math/ML basics:** Probability, linear algebra (vectors, matrices), basic ML concepts.
- **Mini projects:** CLI chatbot (simple rules), deploy a Flask/FastAPI app in Docker.

Deliverable: Local dev env + basic Python API running.

Phase 1 — Speech & Text pipeline (core I/O)

- **ASR:** Whisper/Vosk/Kaldi — integrate microphone input → `text.`
- **TTS:** Coqui/Mozilla TTS or any TTS API (neural voices).
- **Wake-word (optional):** Porcupine, Snowboy or VAD+custom model.
- **Microphone handling:** sounddevice/pyaudio, VAD, chunking, streaming.

Tasks:

1. Record mic input, convert to WAV, run ASR, print text.
2. Send text to a simple LLM (local small model or remote API) and get reply.
3. Convert reply to audio via TTS and play.

Deliverable: Basic voice loop: mic → `ASR → LLM → TTS → speaker.`

Phase 2 — Conversational core & state management

- **Context window & session memory:** Sliding window to keep recent n turns.
- **Prompt engineering:** System prompts, user/system messages, instruction tuning.
- **Dialog manager:** Maintain conversation state, turn-taking rules, fallback responses.
- **Fallback/NLU:** Simple intent classifier for tool calls (e.g., "search web for X").

Tasks:

- Implement session store (in-memory) with configurable token/turn limits.
- Build prompt builder to combine system prompt + memory + recent messages.

Deliverable: Multi-turn conversation that preserves context.

Phase 3 — Tools, Actions & Agents

- **Tool design:** Define a set of safe tools (web search, calendar, calculator, file open).
- **Agent orchestration:** Use an agent pattern (LLM decides to call tool; tool returns result; LLM continues).
- **Safety sandbox:** Any command execution must be sandboxed; implement permission checks.

Tasks:

- Implement a web search tool (serp API or simple scraper with caching).
- Implement a calculator tool and a limited shell command runner in chroot or docker sandbox.

Deliverable: Assistant can perform actions (search, calculate, open file) reliably.

Phase 4 — Knowledge, Retrieval & Personal Memory

- **Embeddings & Vector DB:** Use embeddings (open-source or API) and a vector DB (Milvus/Weaviate/Pinecone) for retrieval.
- **RAG (Retrieval Augmented Generation):** Retrieve relevant docs to augment prompts.
- **Memory model:** Short-term vs long-term memory; store facts with timestamps & tags.

Tasks:

- Build pipeline: user message → `embedding → vector search → include top K docs in prompt.`
- Create memory primitives: store(user_pref), recall(query), forget(key).

Deliverable: RAG-enabled assistant with personal memory.

Phase 5 — Multi-modal & Vision (optional advanced)

- **Vision models:** Image understanding (CLIP, vision-LLaMA, etc.) for describing images or screen.
- **Use cases:** Read screen, summarise images, webcam-based object detection.

Tasks:

- Integrate image upload → `captioning → include caption in LLM prompt.`

Deliverable: Assistant can process voice + images.

Phase 6 — Planning, Agents & Complex Tasks

- **Multi-step planning:** Task decomposition (LLM plans subtasks, executes tools, updates plan).
- **Chaining & memory of steps:** Save step outputs; checkpointing for resumability.
- **Human-in-the-loop:** confirmations for risky actions.

Tasks:

- Build a task planner that can create a step list and call tools for each step.

Deliverable: Assistant that can perform multi-step tasks (e.g., book meeting, fetch resources).

Phase 7 — Scaling, Deployment & Reliability

- **Infra:** Dockerize services, use Kubernetes for scaling, managed vector DB, Redis for queues.
- **Latency:** Use streaming outputs, edge inference for ASR/TTS for low latency.
- **Monitoring:** Logs, metrics, alerting, user analytics.

Tasks:

- Deploy microservices (ASR/TTS/LLM runner) with autoscaling and load balancing.

Deliverable: Productionized assistant with SLAs.

Phase 8 — UX, Personalization & Polishing

- **UI:** Desktop widget, web UI, mobile app, or voice-only with push notifications.
- **Personality:** Voice style, tone, customizable user persona.
- **Settings:** Privacy controls, deletion of memory, permission consent flows.

Deliverable: Polished, usable Jarvis with personalization options.

---

4) Learning Path (Level-wise with mini-projects) — Quick roadmap

Beginner (0 → 1):

- Learn Python, REST, basic data structures.
- Mini: Build a chat UI (text), host on local FastAPI.

Intermediate (1 → 2):

- Learn ML basics, Transformers, Hugging Face, embeddings.
- Mini: Integrate a small LLM via Hugging Face and do a question-answering RAG demo.

Advanced (2 → 3):

- ASR/TTS, vector DBs, agents frameworks (LangChain, LlamaIndex, Haystack).
- Mini: Build voice chat: mic → `Whisper (or API) → LLM → TTS.`

Expert (3 → 4) :

- Multi-modal models, RLHF, planner agents, infra (K8s), security & privacy.
- Mini: Production deploy with autoscaling + memory + complex tool integration.

---

5) Recommended Tools / Libraries (quick list)

- **LLM / Agents:** Hugging Face Transformers, LangChain, LlamaIndex, OpenAI API (if using).
- **Embeddings / Vector DB:** sentence-transformers, Milvus, Weaviate, Pinecone.
- **ASR / TTS:** Whisper, Vosk, Coqui TTS, Mozilla TTS.
- **Web / Orchestration:** FastAPI, Uvicorn, Redis, Celery or RQ.
- **Infra:** Docker, Kubernetes, NGINX, Traefik.
- **Databases:** PostgreSQL (structured), Redis (cache), S3 (blobs).
- **Monitoring:** Prometheus, Grafana, Sentry.

---

6) Data, Privacy & Safety Checklist (important!)

- **Consent:** User data must be stored only with consent; provide deletion.
- **Filters:** Profanity, hate speech filter; block dangerous tool use (e.g., remote shell without auth).
- **Sandbox:** All system commands run in sandbox with strict permissioning.
- **Audit logs:** Maintain logs for any action that affects user data or external systems.

---

7) Evaluation & Metrics

- **Functional tests:** correctness of tool outputs (web search accuracy, calendar updates).
- **Conversational metrics:** user satisfaction, completion rate, fallback rate.
- **Latency:** ASR → LLM → TTS total time.
- **Safety metrics:** number of unsafe responses, policy violations.

---

8) Minimal Code Skeletons / Examples (quick)

Mic → ASR → LLM → TTS (pseudo Python):

```
# pseudo-code: real code needs proper error handling

import sounddevice as sd

from whisper import load_model

# LLM API wrapper e.g., openai, or HF inference

# 1) record chunk, save

# 2) run ASR

# 3) send text to LLM API

# 4) get reply, run TTS, play
```

(Use this as starting template — production code needs streaming + async.)

---

9) Project Milestones (suggested deliverables chain)

1. Local voice chatbot (single machine).
2. Multi-turn context + session management.
3. Add tool-calling abilities (search, calc).
4. Add RAG + vector DB + personal memory.
5. Add multi-modal (images) and complex agents.
6. Productionize with Docker/K8s + monitoring.

---

10) Tips, Gotchas & Best Practices

- **Start small:** First make a reliable voice loop before adding complexity.
- **Token budget:** LLM prompts can get expensive — use retrieval, summarization, and memory compression.
- **Privacy first:** Don’t log sensitive info by default; allow users to opt out.
- **Test edge cases:** Interruptions, noisy audio, ambiguous instructions.
