🧠 Jarvis AI — Advanced Technical System Design Report

A next-gen multimodal AI agent platform for productivity, support, and safety.

🧩 1. Problem Statement & Context

🧠 Purpose:

Most AI assistants today are fragmented — Google Assistant does basic home tasks, ChatGPT does language processing, and dev tools are mostly separate. There's no **single integrated agent** that:

- Handles **daily life & productivity**
- Offers **developer-grade contextual support**
- Provides **emergency response** assistance
- Is **personally customizable** in behavior and personality

Our goal: Build a **modular, hybrid AI system** that is accessible, efficient, safe, and feels emotionally engaging.

🧠 2. Requirements Gathering & Feature Prioritization

🔍 Functional Requirements:

- **Voice-controlled smart agent** ("Hey Jarvis")
- **Task & calendar management**
- **Emergency keyword detection + alerting**
- **Screen-aware help (desktop)**
- **Custom personalities (e.g., Tony Stark)**
- **IoT/Home control**

- **Mobile + Desktop app parity**

🛡️ Non-Functional Requirements:

- **Low latency**
- **Privacy-first**
- **Cost-efficient**
- **Scalable agent orchestration**
- **Personalization without data leakage**

🧪 Design Philosophy:

- Start lean with **critical functionality** (P0), build a system that is **modular and pluggable**
- Aim for a **production-ready baseline**, not an academic prototype

🏗️ 3. System Architecture Overview

⚙️ Architecture Style: Modular Microservices

We chose **microservices** over monolith for these reasons:

Microservice Rationale	Benefit
Decoupled features	Each agent (e.g., DevHelp, LifeSaver) scales independently
Fault isolation	Crash in Emergency Agent won't affect Task Agent
Parallel development	Multiple engineers can work without merge conflicts
Easier A/B testing + rollout	Ship beta features to select users

🔁 4. Communication & Control Flow

✨ Agent Flow Overview:

1. **Input** (voice, text, screen)

2. **Router** decides intent: scheduling, help, emergency, personality
3. **LLM layer** parses and routes to appropriate **agent**
4. Agent may query vector DB (RAG), access APIs, or trigger downstream actions
5. **Response generated**, optionally with voice synthesis

📡 Inter-Process Communication:

- **REST**: Simple user requests (e.g., schedule, reminders)
- **gRPC**: Internal fast messaging (agent <-> agent or orchestrator)
- **Kafka/NATS**: Emergency triggers and real-time context changes

🧠 5. LLM Strategy: Hybrid Deployment Model

✅ What We Did:

We built a **hybrid stack** combining:

- **API-based LLMs**: GPT-4 (for fallback, complex reasoning)
- **Open-source LLMs**: Mixtral, Phi-3, LLaMA 3 (local/private tasks)

Reasoning	Details
Cost Reduction	Open-source models for ~80% of day-to-day queries
Privacy	Emergency context, screen-aware prompts handled locally
Latency Optimization	On-device or edge hosting to avoid round-trips for trivial tasks
API Budget Offloading	GPT-4 for high-accuracy, fallback only on demand (e.g., major planning decisions)

🧱 Local Hosting

- Model hosted via **vLLM or TGI** (Text Generation Inference)
- GPU provisioning via **Runpod / Modal** with **A100 spot instances**
- **Cold start mitigated** via queue warmers

🧰 6. Component Breakdown

🗣️ Voice Agent (Input Layer)

- **Speech-to-text**: OpenAI Whisper (local inference)
- **Wake word**: "Hey Jarvis" via Vosk / Porcupine
- **TTS**: ElevenLabs (API) or Coqui (local fallback)

🧠 Orchestration Layer

- **LangGraph (or Haystack Agents)** for stateful multi-agent flows
- Each agent has its own memory + function toolkit

```
User → Orchestrator → [Agent A (Dev)] or [Agent B (IoT)] → Tools →
Response → Synthesized Output
```

🧑‍💻 Screen-aware Agent (Desktop Only)

- Uses **Tesseract/OCR** or **Electron hooks** to read current screen
- Context passed to LLM as a prompt ("User has VSCode open, with Python file showing a bug in line 42")

🔐 Emergency Response Agent

- Always-on lightweight listener
- Detects critical phrases: "Fire", "Help me", "Call ambulance"
- Sends pre-configured alerts + survival tips + dials contact
- Works offline too using **on-device models + SMS fallback**

💾 Vector DB (RAG + Personal Memory)

- Embedding: Instructor or E5 models
- DB: Qdrant or Weaviate
- Indexed: Past chats, notes, documents, websites, code snippets

🎛 Persona Engine

- Loads fictional personality presets from structured JSON (e.g., "Tony Stark" = sarcastic, witty, confident)
- Adjusts **tone, verbosity, emotion, and knowledge bias** of agent
- Uses **LLM + Rule Layer** for reinforcement

📱 Clients: Mobile + Desktop

Platform	Use Case	Stack
Mobile	Life automation, voice assistant	Flutter + Whisper + REST APIs
Desktop	Technical + screen-aware support	Electron + Node.js + Python Agent Bridge

🔐 Security & Privacy Architecture

Feature	Implementation
Sensitive data isolation	No raw PII sent to cloud APIs
Encryption	AES-256 at rest, TLS in transit
Agent sandboxing	Docker + restricted scopes
Prompt injection prevention	Preprocessing + output sanitizer
User control over memory	Can delete / reset local memories anytime

Cost Optimization Strategy

Layer	Optimization
LLM Inference	API fallback only; use Phi-3, Mixtral as primary
GPU Usage	Shared GPUs (Runpod), spot pricing, model batching
Vector Search	Run on CPU or cheap GPU (small index)
Emergency Agent	Run as tiny Python listener, 10MB RAM footprint
Speech Layer	Use Coqui or Vosk for offline inference when feasible

Engineering Decisions Breakdown

Decision	Why This, Not That
Hybrid LLM	Combines cost-savings + API power
Microservices > Monolith	Resilience, scalability, isolation of agents
REST + gRPC	Simplicity for frontend; speed for internal agents
LangGraph > LangChain	Cleaner graph-based control flow + better memory handling
Flutter (Mobile)	Cross-platform and GPU-accelerated speech models
Qdrant (Vector DB)	Open-source, scalable, easy local deployment

Future Roadmap Ideas

- Holographic UI (WebAR SDK or Apple Vision Pro integration)
- Multiplayer Agents (collaborative planning with others)
- Fine-tuned open-source LLM based on task analytics
- Jarvis Store: user-submitted personalities & agent plugins
- Federated learning (on-device fine-tuning per user)

☑ Conclusion

This design shows how you can:

- Balance **cost**, **efficiency**, and **security**
- Think **modularly** with multi-agent orchestration
- Leverage **open-source + cloud AI hybrids**
- Offer a **high-utility and emotionally resonant product.**