VAIBHAV KOLEKAR

AI Systems Engineer

+91 7892225626 · va18havrk3@gmail.com · Bengaluru, Karnataka

HIGHLIGHTS

- Built 3 production backend systems using Go, Python, and TypeScript. These included a real payment SaaS (interviu.pro) and a Go service that can handle 1,000 req/sec with <1 ms (p50) latency.
- Designed an agent using LangGraph to autonomously resolve incidents in Kubernetes. This had dual safety mechanisms to ensure prompt injection cannot change state of the cluster. It also allowed routing to 5 different LLMs based on the stage of the process and ensured that there was a limit to the number of tokens retrieved.
- Always reduced any redundant infrastructure. This included removing Redis by using PostgreSQL as a transactional job queue. I also removed distinct STT/TTS steps and streamed audio directly. I ensured correctness by using atomic transactions for concurrent payments, FOR UPDATE SKIP LOCKED for exactly-once delivery, and race-condition-free status consistency between HTTP and pub/sub.

PROJECTS

Autage – Autonomous Agentic SRE & Incident Response Copilot

Python · FastAPI · LangGraph · SQLAlchemy · PostgreSQL · Redis · React.js · TypeScript · Kubernetes

Code: github.com/va18hav/Autage

- **4-Stage Agentic Pipeline:** A 4-stage agentic pipeline built using LangGraph, including Triage, Cluster Diagnostics, Runbook Lookup, and Remediation stages. This process pulls in raw monitoring webhooks from any provider (Datadog, Prometheus, Grafana, Sentry, PagerDuty). We perform zero schema adaptation across data providers and perform concurrent read-only kubectl commands based on LLM plans. Each stage provides remediation step-by-step plan details to a React 19 dashboard using Redis Pub/Sub → SSE. The remediation is race-condition-aware and ensures we are always keeping our statuses up to date using HTTP lookups and pub/sub.
- **Double-Layer Read-Only Enforcement:** We have a double-layer read-only approach. The agent's tools only expose static Python functions. The LLM will never provide any shell commands. Instead the LLM will provide parameters to the functions. The Kubernetes identity that connects to the cluster is a view-role ServiceAccount. This means that even if there was some prompt injection or hallucination from the LLM, there would be no way to change the cluster state. If there is no kubectl binary present on the host, we will instead make direct HTTPS calls to the K8s API Server using TLS validation with the cluster's CA certificate.
- **Two-Pass Runbook Retrieval:** We have a double-stage runbook lookup. First we only pull in runbook names and section headings. The LLM will provide us with the exact sections they want to open and we will only pull in that section of the runbook. This reduces the context window used depending on how relevant the information is. Our operational runbooks can be 30+ pages long.
- **Per-Stage Dynamic Model Routing:** Each stage of the pipeline has been designed to dynamically route to any model provider (Google, OpenAI, Anthropic, OpenRouter, Ollama). We do this by using a LlmResolver which is scoped to a ContextVar. This means that for every incident run we will only make one call to the resolver. We have a dynamic discovery layer which looks up the models provided by each provider. This will cache the results (15-minute TTL) and invalidate based off the credentials. This means that the UI will always show what models are available.

Interviu – An asynchronous technical screening platform

TypeScript · React.js · Node.js · WebSockets · Gemini Flash Live API · PostgreSQL · Redis · AWS S3 · Docker · AWS EC2

Link: interviu.pro

- **Real Time Voice AI:** Created a real time voice interviewing tool where audio was streamed directly using the Gemini Live API over bi-directional web sockets, instead of through separate STT and TTS steps. Audio was buffered as PCM and synchronised, and the connection was re-established if it had dropped.
- **Credit Billing & Transaction Ledger:** Created a pay as you go credit system using Razorpay, whereby webhooks were used to trigger actions based on payments. Credit was protected against being incorrectly allocated through using HMAC to verify the payments and atomic transactions within PostgreSQL.
- **Background Processing:** Moved LLM scorecard generation, video processing and emails to be handled by BullMQ workers using Redis. This provided automatic retries and exponential backoffs so that errors in one queue did not affect the main app.
- **Multi-Tenant & CI/CD:** Set up data isolation at an organisation level alongside RBAC. We deployed our app automatically using Docker, GitHub Actions and AWS. We also set up Prisma migrations, built our TypeScript app and checked our containers were working correctly before deployment.

PingDeck – Fail-safe webhook proxy, retry engine & visual replay deck

Go · PostgreSQL · React.js · TypeScript · Server-Sent Events · HMAC · Docker

Code: github.com/va18hav/pingdeck

- **Ingest Performance:** Sustains 1,000 webhooks/sec with p50 < 1 ms and 100% success on a single container — every request signature-checked and committed in one ACID PostgreSQL insert, preventing the provider-side timeout retries that duplicate events (reproducible load-test scripts in docs/BENCHMARKS.md).
- **Backlog Recovery:** Cut worst-case queue recovery from ~40 min to under 2 sec (~800×) by parallelizing the dispatcher with a bounded goroutine pool and adaptive claim loop — preserving exactly-once claiming via atomic FOR UPDATE SKIP LOCKED so concurrent workers never double-deliver.
- **Redis-Free Queue Architecture:** Eliminated Redis by making PostgreSQL both the system of record and the transactional job queue, removing the dual-write consistency failure mode between store and queue and cutting the deployment from three services to two.
- **Real-Time Replay Deck:** Built a replay-first delivery dashboard (React 19/TypeScript/TanStack Query) over nudge-only SSE with a non-blocking broker that drops slow-client updates instead of stalling delivery — shipped as one self-contained binary (embedded migrations + SPA) in a 23 MB scratch image idling at ~12 MiB RAM.

TECHNICAL SKILLS

Languages	Go · Python · TypeScript · SQL
AI & Agentic Systems	LangGraph · Multi-Provider LLM Routing · Tool-Calling Agents · Structured Outputs
Backend & Real-Time	FastAPI · Node.js · Prisma · SQLAlchemy · WebSockets · SSE · BullMQ · ARQ
Frontend	React.js · TanStack Query · Zustand · Tailwind CSS
Infra & DevOps	PostgreSQL · Redis · Docker · Kubernetes · Vercel · AWS (EC2, S3) · GitHub Actions

EDUCATION

B.Tech – Electronics & Communication Engineering — Dec 2021 – Aug 2025

University Visvesvaraya College of Engineering (UVCE), Bengaluru — CGPA: 7.3 / 10.0

LinkedIn linkedin.com/in/vaibhav-k-968883385 · **GitHub** github.com/va18hav · **Portfolio** vaibhavk.devs.surf