PRANJAL SRIVASTAVA

Corpus Christi, TX | +1 346 375 2373 | pranjal6004@gmail.com | linkedin.com/in/pranjal-srivastava07 | pranjalmax.github.io/pranjal-portfolio/

SUMMARY

Full-stack/AI engineer who ships privacy-first, production-ready web and AI experiences end to end. I build on-device RAG apps (WebLLM/WebGPU), client-side hallucination guardrails, and voice-AI workflows, while also delivering robust RESTful backends (Node/Express, Java), SQL tuning, and CI/CD. Strengths include retrieval design (chunking, embeddings, Top-k tuning), prompt engineering/evaluation, accessible and fast JS/TS frontends, and pragmatic DevOps (Docker, health checks, GitHub Actions) for repeatable demos and reliable releases.

CORE SKILLS

Languages & Frameworks: Python, Java, TypeScript/JavaScript, Node.js/Express, React, SQL
AI & Retrieval: RAG, embeddings (Transformers.js/MiniLM), cosine similarity, Top-k tuning, chunking strategies, guardrails
Model & Prompting: Prompt libraries, eval sets, safety/refusal patterns, grounding via citations, answer-diffing and red-teaming
Frontend & UI: Component architecture, responsive layouts, accessibility (ARIA, focus), forms/validation, routing
Backend & APIs: RESTful design, authentication/authorization (RBAC), pagination, validation, Swagger/OpenAPI
Data & Storage: SQL modeling/tuning, caching strategies, client storage (IndexedDB/localForage), pdf.js ingestion
Cloud & Deployment: Static hosting (GitHub Pages), Vercel serverless APIs, ngrok, CORS, Docker, health checks
Dev Workflow: Git/GitHub Actions CI, code reviews, PR templates, issue triage, Agile/Scrum
Security & Governance: RBAC, input validation, audit logging, change control

EXPERIENCE

Tinker Tech Logix - Software Developer **Apr 2024 - Sep 2025**

- Built and enhanced modular web apps; shipped features across admissions, fees, exams, and results modules.
- Designed and tuned SQL schemas/queries with caching for high-load screens to improve responsiveness and reduce load spikes.
- Implemented RBAC with server-side validation, audit logging, and change control for accountable operations.
- Documented API contracts with Swagger/OpenAPI; improved error handling and retry patterns across the stack.
- Wrote unit/integration tests and added GitHub Actions CI; containerized services with Docker and added health checks.
- Raised accessibility with ARIA roles, focus outlines, keyboard flows, and reduced-motion support across core screens.
- Collaborated via code reviews and PR templates; supported staged rollouts with rollback steps and runbooks.

ECS - Intern, Cloud Server & Data Management **May 2019 - Aug 2019**

- Assisted cloud migration planning, stability hardening, and performance tuning of data platforms.
- Supported backup/recovery procedures and contributed to scale-out planning and data handling policies.

PROJECTS - HIGHLIGHTS

Private Doc Chat - On-Device RAG (Browser-Only) | GitHub Pages

- Pure client-side PDF Q&A;: local embeddings (Transformers.js MiniLM) + WebLLM (WebGPU) + cosine Top-k retrieval; zero backend, zero paid APIs.
- Sliding-window chunking with overlap and similarity thresholding; strict quote-only answers with inline citations to reduce hallucinations.
- IndexedDB vector persistence with Clear Data, Service Worker offline shell, accessible UI; static CI/CD to GitHub Pages.

Hallucination Guard - Client-Side LLM Evidence Checker (React + TS)

- Verifies each claim in an LLM answer against user-provided PDFs/text entirely in-browser via embeddings + retrieval pipeline.
- Review -> Sources -> Report flow with clickable evidence and grounded fix drafts; designed for clarity and trust.
- Privacy-first and zero-ops; static deployment with CI/CD for reproducible demos.

Vanessa - Voice AI Acquisitions Agent (Node/Express + React Dashboard)

- Backend endpoints (/vapi/events, /api/leads, /health) with deterministic rules; CORS/ngrok setup for stable dev.
- Frontend dashboard with readable tables/cards, filters, and empty/error states; live lead badges for quick triage.
- Ops runbook and health feeds; packaged for repeatable demos and reliability.

Portfolio Site - Modern Static Frontend (GitHub Pages)

- Responsive layout with sticky header, dark/light toggle, accessible navigation; keyboard-friendly dialogs and focus rings.
- Core Web Vitals: preload hero, explicit image dimensions, content-visibility, deferred non-critical JS via requestIdleCallback.
- Projects gallery with tag filters and search; high-conversion contact funnel with honeypot, thanks page, and confetti.

ServiceNow - Dog Adoption Portal App

- Custom tables and forms, flow designer for approvals, list views, and basic role-based access; clean navigation.
- Built adoption request lifecycle and notifications; demo-ready seed data for quick trials.
- Documented setup steps and included screenshots/runbook for reviewers.

ServiceNow - Helpdesk Ticketing App

- Users raise tickets (hardware/software/other); triage to teams; auto-assignments based on category/priority rules.
- Notifications on resolution and status changes; reports for SLA visibility.
- Well-labeled forms, validation rules, and list filters for fast support handling.

ECG Heart Failure Detection (ML)

- Signal preprocessing, feature extraction, and classical ML to classify heart failure risk.
- Training/validation workflows and metrics reporting; documented limitations and next steps.

- Academic presentation and write-up for the project.

DDoS Detection on SDN (Mininet/Ryu/Python)

- Traffic capture, labeled flows, and ML model to detect volumetric anomalies in SDN environments.
- Implemented pipeline scripts and evaluation harness; reproducible experiments.
- Reported precision/recall and ablation insights; notes on deployment considerations.

Database Optimizations via DSA

- Analyzed index strategies, query plans, and data structures to improve query latency.
- Proposed algorithmic tweaks and caching policies; measured improvements with benchmarks.
- Documented migration steps and risks; wrote a concise runbook.

Fake News Detection (Python)

- Text preprocessing, vectorization, and classifier training; baseline comparisons across models.
- Precision/recall and confusion matrix reporting; notes on data bias and generalization.
- Readable CLI scripts and a short README with results.

Weather Forecasting with Deep Learning

- Feature engineering and deep learning architecture for short-term forecast predictions.
- Training curves and error metrics; comparisons with simpler baselines.
- Careful notes on data leakage and evaluation design.

EDUCATION

M.S., Computer Science - Texas A&M; University - Corpus Christi **2022 - 2023**

B.Tech., Computer Science - SRM Institute of Science and Technology **2017 - 2021**

CERTIFICATIONS

- micro1 - Certified Software Engineer (AI Interview), Sep 2025
- Complete ServiceNow Developer Course (Udemy)
- Programming in Python for Everyone (Coursera)
- ChatGPT Prompt Engineering for Developers
- Building Real-Time Video AI Applications (Nvidia)

EXTRA-CURRICULAR

- Scheduler, International Student Organization (ISO) - Texas A&M; University-Corpus Christi Planned and maintained the ISO master calendar
- handled room bookings and university approvals. Coordinated end-to-end event logistics (venue, AV, catering, permits/risk forms)
- created run-of-show and contingency plans. Recruited, scheduled, and led volunteer teams
- defined roles and SOP checklists for smooth execution. Negotiated with campus/external vendors
- ensured on-time delivery and policy compliance. Drove email/social promotions (Canva/Instagram)
- tracked RSVPs/feedback to refine programming and increase attendance.

APPENDIX A - Full Text: All AI Projects

Here's a crisp, recruiter-ready project summary of your portfolio build (Now Legacy, Older version, newer version of portfolio is available) : Modern, fast personal site (GitHub Pages, vanilla HTML/CSS/JS) Deployed as a static site for zero-cost hosting and instant global CDN delivery. Clean, responsive layout with sticky header, tidy nav, and persistent dark/light theme toggle . Performance & stability (Core Web Vitals-minded) Preload hero image, explicit image width/height + decoding="async" to eliminate CLS . content- visibility:auto + intrinsic size hints for below-the-fold sections to speed first render . Deferred non-critical JS with requestIdleCallback pattern; minimal dependencies (no frameworks). Lightweight service worker registration scaffold for future offline/cache tuning. Accessibility & UX polish Skip to content link , strong focus rings, keyboard-friendly dialogs, reduced-motion support. Improved contrast for key CTAs and dark-mode modal contrast so content remains legible. Back-to-top button, smooth section scrolling with scroll-margin to avoid header overlap . Projects gallery with discovery tools Tag filters + full-text search for 7 featured projects (Java, .NET, ServiceNow, ML). Hover states, case-study modals with explicit "Case study" buttons and improved readability. Consistent thumbs, aspect-ratio cropping, and lazy loading for fast browsing. Education section redesign (visual + scannable) Logo-led list items with TAMU-CC and SRM images + degree lines for instant comprehension. Contact funnel (high-conversion) Formspree AJAX submit with inline status , confetti micro-interaction , and redirect to a branded thanks page . Honeypot field for spam reduction; hidden UTM/referrer attribution fields auto-filled via JS. LocalStorage prefill for name/email to reduce friction; copy-to-clipboard for email/phone. Sticky mobile CTA ("Hire Me", "Resume") to keep conversion actions visible on small screens. SEO & shareability Canonical URL, Open Graph/Twitter cards, and JSON-LD (Person) schema. Descriptive titles/meta and consistent alt text for images. Max-AI Assistant (on-site chatbot) Floating chat widget (" Max-AI Assistant ") with typing indicator , quick prompt chips , optimistic UI, and cancel in-flight . Backed by a Vercel serverless API calling Groq with model fallbacks (avoids outages), strict CORS, concise/system prompts bound to portfolio scope. Persona tuned to be friendly and human , enriched with personal context (age, hobbies: football/Real Madrid, movies, Indian festivals) while staying professional and on-topic. Content & brand consistency Clear About and Core Skills aligned with your backend/API, ServiceNow, SQL tuning, DevOps, and ML strengths. Experience presented with concrete outcomes, tools, and process (tests, CI/CD, reviews). Resume links and assets organized under /assets, with a soft file-availability check. Reliability & safety touches 404 page with quick recovery links; graceful error states for chat/form networking hiccups. Client-side guards to avoid double sends, ignore late responses, and keep UI responsive. Result: A fast, accessible, and conversion-oriented portfolio that showcases real projects, guides recruiters smoothly to contact, and includes a delightful AI assistant that speaks your voice and domain-without adding paid infra or heavy frameworks. Project Title (choose one) Vanessa - Voice AI Acquisitions Agent (MVP) Voice AI Acquisitions Assistant (Vapi + Node) One-line Summary Built a voice AI that calls homeowners (browser dialer), detects seller intent in <90s, captures price/timing/condition, and logs qualified leads to a live dashboard. Core Bullets (6-8 lines) Designed and shipped a Voice AI acquisitions agent using Vapi (browser calls) + Node/Express webhook. Implemented a conversation flow (owner -> openness to sell -> price -> timing -> condition) with polite exits for No / CallLater / DNC , capped at <=180s . Built a tool-invocation pipeline (API Request -> / vapi /events) to capture structured fields : owner, intent, price, timing, condition, notes. Created a qualification rule (Yes/Maybe + "weeks/1-2 months/soon" ⇒ Qualified) and surfaced outcomes with badges on a live /dashboard (auto-refresh + "time ago"). Added defensive prompts (short, warm, lightly assertive tone; wrap at ~160s) to keep calls natural and concise. Solved CORS/ ngrok issues and added health checks to ensure reliable webhook delivery in dev. Produced a recruiter-ready doc (architecture, screenshots, runbook) and a GitHub repo with setup instructions. (Constraint) PSTN unavailable in workspace -> optimized for browser-call demo with a clear, simulated handoff script; transfer can be wired later (Vapi PSTN/Twilio). Impact / Results (pick 2-3) Under-90s intent detection demonstrated consistently in live tests. Zero-infrastructure demo : local server + ngrok + browser dialing; fully reproducible from README. Instant visibility : dashboard auto-refresh shows Qualified leads in real time for handoff. Technical Highlights Stack: Vapi (LLM voice, browser call), Node.js, Express, Day.js, CORS, HTML/CSS dashboard, ngrok . Endpoints: / vapi /events (webhook), / api /leads (JSON feed), /dashboard (UI), /health (status). Data model: { owner , intent, price, timing, condition, notes, updated_at , disposition } . Rules/Logic: deterministic qualification; polite branch handling; 160-180s time cap; idempotent logging. Challenges & Solutions (optional bullets) Webhook visibility: used /health and / api /leads to debug ngrok + event flow. CORS in tool testing: added cors middleware and/or used Vapi's "Use CORS Proxy". Non-PSTN constraint: kept UX strong via browser call + spoken transfer script and dashboard proof. Keywords / ATS Tags Vapi, Voice AI, LLM, Speech, Node.js, Express, Webhooks, REST, ngrok , CORS, Real-time Dashboard, Lead Qualification, Prompt Engineering, Outbound Calling, Twilio (future), Supabase (future). Links (add to resume) GitHub: https://github.com/pranjalmax/vanessa-voice-ai Demo Packet (doc): "Vanessa_Voice_AI_Demo_Packet_v2.docx" (screenshots + runbook) Compact 3-5 Bullet Version (if space is tight) Built a Voice AI acquisitions agent (Vapi + Node/Express) that detects seller intent <90s and captures price/timing/condition. Logged leads via webhook -> /dashboard (auto-refresh + "Qualified/ NotQualified " badges) with a deterministic qualification rule. Implemented polite branches (No / CallLater / DNC) and <=180s call cap to keep conversations crisp. Resolved ngrok /CORS issues and added health/JSON feeds for reliable testing. Delivered a recruiter-ready doc + repo ; PSTN not enabled -> optimized for browser-call demo ; transfer ready to add later. STAR-style Mini Narrative (1-2 lines) Situation/Task: Needed a working Voice AI to qualify home-seller leads quickly for a job test. Action: Used Vapi (browser) + Node webhook + prompt design to capture fields and compute qualification; built a dashboard. Result: Live demo with <90s intent detection and real-time Qualified badges; clean handoff path prepared for PSTN. Project: Private Doc Chat - On-Device RAG (Browser-Only) One-line Privacy-first PDF Q&A; app that runs 100% in the browser: local embeddings, local LLM, grounded answers with clickable citations, zero servers and zero API keys. Why I built it Prove real on-device AI: demonstrate Retrieval-Augmented Generation without cloud costs or data exposure. Showcase practical AI engineering: end-to-end pipeline (ingest -> embed -> retrieve -> generate) running in a standard browser. Recruiter-friendly demo: GitHub Pages live link + repeatable verification checklist. Key capabilities PDF ingest with text extraction, sliding-window chunking (≈900 char, 150 overlap). In-browser embeddings with Transformers.js (MiniLM) and cosine-similarity retrieval (Top-k 3-8). Local LLM generation via WebLLM (WebGPU) using a tiny instruct model. Strict quote-only mode with similarity threshold to prevent hallucinations and force citations like [C#]. Clickable citations that scroll and highlight the exact source chunk. IndexedDB persistence via localForage , Clear Data privacy control, offline app shell (Service Worker). Accessible, keyboard-navigable UI with progress indicators and graceful error states. Technologies used LLM inference: WebLLM (MLC) on WebGPU Embeddings: @ xenova/transformers (MiniLM , ONNX/WASM/ WebGPU) PDF parsing: pdf.js Storage: IndexedDB via localForage Frontend: Vanilla HTML/CSS/JS (no bundler), Service Worker for offline Retrieval: cosine similarity over normalized vectors Hosting: GitHub Pages (static) Resume bullets (pick 4-6) Built a browser-only RAG app that answers questions over PDFs with clickable citations, using WebLLM (WebGPU) for local generation and Transformers.js (MiniLM) for in-browser embeddings; no servers or API keys. Implemented PDF ingest and chunking (≈900 char with 150 overlap), vectorized via MiniLM , and

retrieved Top-k results using cosine similarity with a tunable threshold to prevent low-confidence answers. Designed a Strict quote-only mode (temp=0) that copies sentences verbatim from retrieved chunks and enforces inline citations like [C#], improving grounding and eliminating hallucinations. Persisted vectors and metadata in IndexedDB (localForage) with a Clear Data control; added a Service Worker to enable offline app-shell loading and fast repeat runs. Shipped an accessible, keyboard-friendly UI with progress indicators (model load, parsing, embedding), error handling, and citation anchors that scroll/highlight the source text. Deployed as a pure static site on GitHub Pages; no bundler, zero backend, and zero cloud cost while maintaining privacy-by-design. Optional ATS-dense variants RAG; WebGPU ; WebLLM ; Transformers.js; MiniLM embeddings; cosine similarity; vector store (IndexedDB); pdf.js; localForage ; Service Worker; accessibility; prompt engineering; chunking; Top-k retrieval; zero-backend; GitHub Pages. Implemented similarity thresholding and extractive QA prompt templates to enforce grounded answers and reduce hallucination risk. Portfolio / LinkedIn summary (short paragraph) I built a private, browser-only RAG app that lets you drop a PDF and ask questions with grounded answers and clickable citations. The entire pipeline runs on-device: pdf.js for text extraction, sliding-window chunking, MiniLM embeddings via Transformers.js, cosine-similarity Top-k retrieval, and a tiny WebLLM model on WebGPU for generation. Data stays local in IndexedDB with a Clear Data control, and the app works offline after first load. Strict quote-only mode plus a similarity threshold ensures faithful, citation-rich answers. Deployed as a static site on GitHub Pages. Skills demonstrated Applied AI / RAG: Chunking strategies, embeddings, retrieval, prompt design for extractive vs. abstractive answers. On-device inference: WebGPU , WebLLM model initialization, lightweight models for commodity laptops. Information retrieval: Cosine similarity, vector normalization, Top-k tuning, guardrails (min similarity). Frontend engineering: Vanilla JS architecture, async pipelines with progress UX, accessibility, error handling. Local persistence & privacy: IndexedDB design, data lifecycle (ingest, update, wipe), offline UX via Service Worker. Product thinking: Zero-backend constraint, privacy-first controls, recruiter-ready documentation and demo flow. Talking points (interview) Trade-offs: Chose a 1B-ish model for responsiveness; added Strict mode and similarity threshold to maintain factuality under small-model limits. Reliability: Progress bars and idle yielding to keep UI responsive during embedding; Esc kill-switch for overlays; clear fallbacks and error messages. Privacy & cost: No uploads or API keys; everything computed client-side and persisted locally. Extensibility: Easy to add URL/paste ingest, a tiny in-browser reranker , or multi-document libraries with tags. Project naming line (for resume header or Projects list) Private Doc Chat - On-Device RAG (Browser-Only): Local embeddings (Transformers.js, MiniLM), local LLM (WebLLM on WebGPU), cosine Top-k retrieval, strict quote-only mode with citations, IndexedDB persistence, offline app shell; pure static deploy on GitHub Pages. Hallucination Guard - Client-Side LLM Hallucination Checker Live Demo: https://pranjalmax.github.io/hallucination-guard/ Code: https://github.com/pranjalmax/hallucination-guard 1. What this project is Hallucination Guard is a fully client-side web app that takes: An LLM-generated answer, and One or more source documents (PDF / text), and then: Checks each factual claim against the ingested sources, Highlights supported vs uncertain claims inline, Shows clickable evidence chunks for transparency, and Builds a grounded fix draft + review report , all inside the browser , with no backend , no paid APIs , and no data leaving the user's machine . It's designed to look and feel like a polished AI product, not a throwaway demo. 2. Why I built it (story for recruiters) Companies that use LLMs for content, customer support, or internal tools have two recurring concerns: "Can we trust what the model just said?" "Can we verify it against our own documents-without shipping confidential data to another server or vendor?" Most examples online: either depend on cloud APIs and server infrastructure, or are minimal scripts that don't look or behave like production tools. My goal with Hallucination Guard : Prove that you can build a real, usable hallucination guardrail : using on-device embeddings and retrieval , packaged as a static React app , with a strong UX and clear explanations of what's happening. Demonstrate end-to-end skills: product thinking + frontend architecture + ML/retrieval basics + CI/CD + privacy-first design . This is the kind of problem many AI product teams actually have . This project is my "I can design and ship that" proof. 3. How it works (clear, non- handwavey) High-level pipeline: Sources Ingest User uploads PDFs or pastes text. PDFs parsed with pdf.js. Text is split into overlapping chunks (~800-1000 chars). Documents + chunks stored in IndexedDB (via localForage). Embeddings & Vector Index On click, the browser loads a tiny Transformers.js model (MiniLM -like). Each chunk is embedded client-side; vectors and metadata are stored locally. This forms a lightweight vector store per document-no server. Claim Extraction User pastes an LLM answer. A small analysis module finds candidate factual claims: dates, numbers, entities, phrases. Each claim is tracked with type and text span -> used for highlighting. Evidence Retrieval & Scoring For each claim: Embed the claim, Retrieve top-k similar chunks from the local vector store. A custom scoring layer (scoring.ts) labels evidence as: supported if strong lexical overlap & consistent dates/numbers. unknown if weak or mixed. Internally detects obvious contradictions (e.g. "February 2023" vs "March 2023" in same context) and downgrades to unknown. UI shows per-claim status + top evidence chunks. Highlighting, Fix Draft & Report The original answer is re-rendered with inline highlights: green for supported , amber for uncertain . Users can generate a grounded revision (template-based when no model). Report tab exports: claim table, factuality bar, markdown summary, JSON report for tooling. All of this is implemented in React + TypeScript , bundled with Vite , and deployed as static assets on GitHub Pages through GitHub Actions . 4. What this project shows about my skills You want recruiters to see capabilities , not just "I followed a tutorial". This is what Hallucination Guard demonstrates: Product & UX Can translate a fuzzy idea ("catch hallucinations") into a clear user flow : Sources -> Review -> Report. Designs interfaces that show why a decision was made (evidence, chunks, statuses) instead of magic scores. Balances visual polish (gradients, glassmorphism , motion) with usability and clarity. Frontend Engineering Built with React + Vite + TypeScript : modular structure (lib/ vs components/), typed utilities for chunks, storage, retrieval, scoring. Uses Tailwind CSS + shadcn / ui + Framer Motion + lucide -react for a cohesive, modern AI dashboard feel. Handles: file uploads, streamed parsing (PDF), complex state across tabs, toasts, status indicators, and error handling. AI / Retrieval / "LLM Ops" Thinking Implements client-side embeddings using Transformers.js . Designs a small vector store API around IndexedDB : saving vectors, top-k retrieval, document-scoped indexes. Builds a custom claim + evidence scoring heuristic : lexical overlap, month/year checks, conservative labeling (never over-claims support). Understands the basics of retrieval-augmented verification , not just "call /v1/chat". DevOps / Constraints Strict non-negotiables: No backend. No paid APIs. Static deploy on GitHub Pages . Configures GitHub Actions to: build on push, deploy automatically, keep the project zero-ops. Privacy & Safety Mindset All data & models run in-browser. IndexedDB usage surfaced (storage meter + clear data button). No hidden network calls; easy for a company to reason about compliance. 5. How I'd pitch it quickly (talk track) You can reuse this in interviews or your doc: Hallucination Guard is a client-side hallucination checker I built to show how we can verify LLM outputs directly against local documents without sending data to a server. Users upload PDFs or paste text, the app builds a local vector index with a tiny Transformers.js model in the browser, and every claim in an answer is checked against those chunks. I designed the UX (Review -> Sources -> Report), implemented the React + Tailwind + shadcn UI, wired up transformers.js embeddings, added a simple but safe scoring layer to detect mismatches (like wrong dates), and automated deploys with GitHub Actions + GitHub Pages. The result is a polished, zero-cost, privacy-preserving hallucination guardrail demo that behaves much closer to a real product than a coding exercise. 6. Resume-ready bullet points Pick 2-4 of these for your CV / LinkedIn under "Projects": Built "Hallucination Guard" , a fully client-side hallucination checker that

verifies LLM answers against user-provided PDFs/text using in-browser embeddings and retrieval (React + TypeScript + Transformers.js). Designed a privacy-first architecture : all parsing, embeddings, and scoring run in the browser with IndexedDB storage; no backend or paid APIs required. Implemented a lightweight vector store on top of IndexedDB and a custom evidence scoring engine (lexical overlap + date/number checks) to classify claims as supported or uncertain. Crafted a production-style UI using Tailwind, shadcn / ui , Framer Motion, and GitHub Pages CI/CD, emphasizing clarity, explainability, and an "AI-native" aesthetic. Demonstrated end-to-end ownership : product thinking, UX, frontend architecture, on-device ML integration, and automated deployment via GitHub Actions. MAX - Portfolio Copilot (Live Portfolio + Serverless Chat API) (Newer Portfolio site) What's going on (TL;DR) I built a recruiter-friendly portfolio that doesn't just show projects-it answers questions about me in real time. A lightweight serverless API (Vercel) powers MAX , a portfolio-scoped assistant that knows my skills, projects, and experience. The site is fast, modern, and uses a "breathing" background with a product-style AI Lab section that makes my work skimmable in seconds. Why I made this Recruiters and hiring managers skim fast. I wanted a portfolio that: Explains my impact in under 30 seconds, not just lists tech. Shows AI thinking end-to-end: frontend craft, backend API design, guardrails, and shipping polish. Keeps control of my narrative via an on-page AI copilot (MAX) that only answers about my public profile and projects. What skills this demonstrates AI product thinking: scoped, safe assistant bound to a portfolio domain. Serverless backend engineering: rate-limit, CORS, input guards, model fallbacks. Frontend engineering: React + Tailwind, component architecture, animated UX (breathing orbs, clean sections). Security & safety: input sanitation, blocked patterns, length caps, origin checks. DevOps hygiene: environment-based config, graceful errors, recoverable states. Purpose & outcomes Make evaluation easy: AI Lab cards + visual summaries + "view details" modes. Answer FAQs instantly: MAX explains relevant projects for a role (e.g., "What matches an AI Engineer role?"). Demonstrate production habits: telemetry mindset, guardrails, structured prompts, rollback-friendly code. How it works (clear flow) User flow Visitor lands on portfolio (GitHub Pages) -> sees AI Lab and project tiles. The floating MAX button opens chat. User asks anything about my skills/work. MAX (client) sends message + short history to pranjal -chat- api . API validates, rate-limits, applies persona + public profile prompt, chooses a Groq model from a fallback list , returns a concise answer. UI renders response; errors degrade gracefully with friendly messages. System flow (serverless API) Endpoint: api /chat.js (Vercel , Node 18+). CORS: restricted to https://pranjalmax.github.io. Rate limit: simple token bucket (burst 16, 8/min), keyed by IP. Input guards: size cap, spam/URL patterns, non-ASCII "weird" char filter. Persona binding: curated BIO (public info + project inventory). Model fallbacks: tries env-configured model -> then safe fallbacks (llama-3.3-70b-versatile, etc.). Resilience: returns helpful error text; client shows fallback copy. Frontend highlights Sections split into components (Hero, AI Lab, Build Stack, Foundations, Experience, About, Playground, Contact). Breathing background (custom keyframes) for a lively feel. Compact, skimmable cards with images, dot-coded labels, expandable details. Contact form wired to Formspree (name, email, message) + copy buttons for each handle. Architecture (high level) Client: React + TypeScript + Tailwind -> static build on GitHub Pages. API: Vercel serverless function (api /chat.js), Groq Chat Completions API. Config: GROQ_API_KEY, optional GROQ_MODEL. Security: Origin allowlist, rate limiting, input sanitation. UX polish: animated progress bar, floating MAX avatar with dynamic badge ("AI Lab guide", " Experience lane ", etc.). What makes it " hireable " Real guardrails: input caps, spam/URL filters, origin locks, model fallback logic. Recruiter UX: immediately shows what I built, why it matters, and where it fits . Maintainable: clear components, readable CSS utilities, no framework bloat. Extensible: swap persona, add embedded demos, or plug a vector DB later. Suggested demo script (60-90s) 0:00 Open homepage -> point at breathing background and AI Lab . 0:10 Expand Hallucination Guard -> mention evidence alignment. 0:25 Click MAX -> ask "Which two projects best fit an AI Engineer role?" 0:45 MAX response -> click into Foundations -> show case-study buttons (if applicable). 1:05 Scroll to Experience -> reveal bullets + narrative paragraph. 1:20 End at Contact -> show copy buttons + Formspree form. Talking points for interviews Why serverless for this? Instant deploys, scale-to-zero, and simple API surface for a single-purpose assistant. How I constrain the assistant: persona prompt, non-portfolio redirect, token limits, and rate-limit to avoid abuse. Fallback strategy: avoids model outages breaking the UX; returns a useful message even on failure. Future: optional embeddings store for deeper Q&A; analytics on questions to prioritize portfolio improvements. Resume bullets (paste-ready) Built a portfolio-native AI copilot (MAX) with a serverless chat API (Vercel + Groq) that answers only about my skills and projects, improving recruiter clarity and time-to-signal. Implemented safety guardrails (CORS allowlist, token-bucket rate limiting, input sanitation, model fallbacks) for reliable responses and graceful degradation. Shipped a high-performance React/TS site with animated "breathing" background, AI Lab project tiles, expandable details, and Formspree contact form. Designed a curated persona + prompt strategy that keeps responses accurate, scoped, and recruiter-friendly . Structured the codebase into clean, reusable components with Tailwind utility styling and minimal bundle overhead. Optional "Case Study" (1-pager skeleton) Problem: Portfolios rarely communicate impact fast; recruiters skim and bounce. Solution: A product-like portfolio with project-first architecture , crisp visuals, and an on-page assistant that answers only relevant questions. Implementation: React/TS frontend (GitHub Pages) + Vercel serverless API (Groq). Input guards, CORS, rate limiting, and model fallbacks ensure safe, resilient answers. Impact: Faster understanding of my fit; clearer mapping from project to role; stronger signal on AI product thinking and engineering hygiene. What's next: Add vector-backed retrieval for deeper answers; track anonymous question themes to continuously refine the portfolio.

APPENDIX B - Full Text: Resume Personal Everything

PRANJAL SRIVASTAVA +1 (346) 375 - 2373 Corpus Christi, TX pranjal6004@gmail.com https://www.linkedin.com/in/pranjal-srivastava 07 https://pranjalmax.github.io/pranjal-portfolio/ CAREER OBJECTIVE Full stack Software Developer with hands on experience across enterprise web apps, ServiceNow platform work, and data driven services. Build scalable APIs and modular UIs in Java/Spring Boot, .NET, and JavaScript/React; automate ITSM processes on ServiceNow (forms, flows, scripts); and integrate cloud backed solutions on AWS and Azure. Strong in SQL design and reporting (SSRS/Power BI), CI enabled quality (tests, code reviews, linting), and change controlled releases with documentation and runbooks. Comfortable switching contexts between feature delivery, integration work, and operational support, and partnering with stakeholders in Agile teams to land reliable, secure, and usable software. EDUCATION Master 's in Computer Science : Texas A&M; University Corpus Christi 202 2 - 2023 Bachelor's in Computer Science : SRM Institute of Science & Technology 2017-2021 SKILLS Technical Skills Languages & Frameworks: Java, Spring Boot, C#, .NET, Python, JavaScript, TypeScript, C/C++, React basics, Glide APIs ServiceNow: App Engine Studio, Form Designer, tables/dictionary, views/related lists, client scripts, UI policies/actions, Script Includes, business rules, Flow Designer, ATF basics Web & APIs: HTML, CSS, responsive UI, accessibility, REST APIs, JSON/XML, Swagger/ OpenAPI , input validation, error handling Data & SQL: SQL Server, PostgreSQL; schema design basics, indexing, query tuning; reporting with SSRS, Power BI, Crystal basics Cloud & DevOps: AWS and Azure basics, IIS, Docker, Git, GitHub Actions/Jenkins, environment configs and secrets management Quality & Process: unit/integration tests (JUnit/Mockito/ PyTest), debugging, regression checklists, code reviews, Agile/Scrum (Jira) Security & Governance: RBAC, audit logging, least privilege, change control, documentation and runbooks EXPERIENCE Tinker Tech Logix - Software Developer April 2024 - Sept 2025 Built and enhanced features for internal web apps using clean, modular code (OOP/SOLID). Developed RESTful APIs (CRUD, auth, pagination) and integrated third-party services. Wrote unit/integration tests (JUnit/Mockito/ PyTest) and fixed regressions from CI results. Collaborated via Git (PRs, code reviews, branch strategy) and resolved merge conflicts. Designed and optimized SQL queries and schemas; improved reliability and query times. Containerized services with Docker and set up basic CI/CD pipelines (GitHub Actions). Participated in Agile ceremonies-daily stand-ups, sprint planning, and retrospectives in Jira. Debugged issues with logs and API tracing (Postman/Insomnia); improved error handling. Documented endpoints with Swagger/OpenAPI and maintained setup/runbooks for teammates. Assisted deployments to dev/staging and monitored app health checks. Built core university ERP modules (Admissions, Fees, Attendance, Exams, Results) in an Agile/Scrum team. Designed & implemented RESTful APIs in Java/Spring Boot; added unit/integration tests and CI/CD (GitHub Actions). Implemented CBCS (Choice Based Credit System) rules-credits, SGPA/CGPA, backlog progression; automated exam workflows (timetable, seating, hall-tickets, result publish). Integrated payment gateways (Razorpay , Paytm) with GST invoices and ledger reconciliation. Optimized SQL and caching for high-load screens (attendance, timetable clash detection). Built secure role-based portals (RBAC) with validation and audit logging. ECS - Intern (Cloud Server & Data Management) May 2019 - Aug 2019 Implemented and managed cloud-based server solutions, enhancing data management and storage capabilities. Engineered data migration strategies to transition enterprise systems to cloud platforms, ensuring seamless operations during the transfer period. Coordinated with the data management team to develop and enforce policies for effective data handling and security. Assisted in optimizing server performance through maintenance and updates, leading to improved system stability and uptime. Participated in cloud infrastructure planning and scaling, contributing to the company's capacity to handle increased data loads. Supported senior data engineers in creating robust backup and recovery procedures, minimizing potential data loss. PROJECTS Dog Adoption Portal App (2025) Designed and implemented a custom Dog Adoption Portal using ServiceNow App Engine Studio, enabling users to submit dog profiles and adoption requests via a user-friendly Service Portal. Built and managed custom tables for Dogs and Adoption Centers with relationships and form-level validations. Configured Record Producers, reference fields, choice lists , and form visibility rules to streamline data entry and record accuracy. Created workflows to send automated email notifications upon adoption form submission and ensured status tracking for each dog. Used UI Policies and Service Catalog integration to dynamically control behavior based on record conditions . Helpdesk Ticketing App (202 5) Worked on designing an optimal database with the help of customizable data structures based on specifics of data. Developed a full-stack Helpdesk Ticketing Application using ServiceNow, supporting submission, categorization, and resolution tracking of IT issues. Designed custom tables for Tickets, Departments, and Technicians, established data relationships for efficient ticket routing. Implemented auto-assignment logic based on category and priority using Business Rules and Flow Designer. Integrated notifications to alert users and technicians on ticket updates, resolution, and reassignment. Utilized catalog items and Service Portal for intuitive ticket creation and status monitoring by end-users. Heart Failure Detection Using ECG Signals (2017-2021) Developed a system to detect signs of heart failure through the analysis of ECG signals using signal processing techniques and machine learning algorithms. A Neural Network was used to identify and classify different types of Arrhythmias to determine risk of heart failure in a person. The data set had 4 primarily classes of heartbeats with an additional class of unknown heartbeats. The model would have a higher accuracy than achieved by having a larger dataset as the number of unknown samples in the used dataset were a problem in generating more False Positives to skew the accuracy. Detection of DDOS Attacks on SDN networks using Machine Learning (2021-2023) Built a Software Defined Network and Leveraged Mininet to build SDN network on local machine in kali Linux environment. Simulated Network traffic on SDN network using tools such as iperf for regular traffic and hping3 for DDoS network packets. Network logs of Normal traffic and DDoS traffic were collected using RYU -controller. Monitored all the DDoS packets and sent alarm to the SDN network. Optimizations in Databases through Data Structures and Algorithms (2021-2023) Worked on designing an optimal database with the help of customizable data structures based on specifics of data. Wrote relevant algorithms for each data structure embedded in the database for every CRUD application. Fake News Detection using Python (2021-2023) Created a Python-based tool to perform Text preprocessing using natural language processing techniques, including tokenization, stopword removal, and stemming, and then applied to refine the textual data and prepare it for analysis. V alidate d the model's efficacy, by conduct ing extensive evaluation and validation using performance metrics like accuracy, precision, recall, and F1-score. By splitting the dataset into training and testing sets and applying cross-validation techniques, we gain ed a n understanding of the model's performance in real-world scenarios. Predict Model of Weather Forecasting using Deep Learning architecture (2021-2023) D eveloped a Python script focusing on advanced data preprocessing, feature engineering, and the implementation of a feedforward neural network. Key techniques employed are label encoding, integration of early stopping, and visualizations of activation functions. The script demonstrated high accuracy on test data. CERTIFICATIONS 1) Programming in Python for Everyone (Coursera) 2) Learn C++ Programming- Beginner to Advance - Deep Dive in C++ 3) Complete ServiceNow Developer Course (Udemy) 4) micro1 - Certified Software Engineer (AI Interview), Sep 2025 5) ChatGPT Prompt Engineering for Developers 6) Building Real-Time Video AI Applications (Nvidia) EXTRA CURRICULAR Scheduler, International Student Organization (ISO) - Texas A&M; University-Corpus Christi Planned and maintained the ISO master calendar; handled room bookings and university approvals. Coordinated

end-to-end event logistics (venue, AV, catering, permits/risk forms); created run-of-show and contingency plans. Recruited, scheduled, and led volunteer teams; defined roles and SOP checklists for smooth execution. Negotiated with campus/external vendors; ensured on-time delivery and policy compliance. Drove email/social promotions (Canva/Instagram); tracked RSVPs/feedback to refine programming and increase attendance.