

PROJECT 1 (AI / ML CATEGORY)

Project Title

Domain-Specific Small Language Model (SLM) with Retrieval-Augmented Generation (RAG)

Project Category

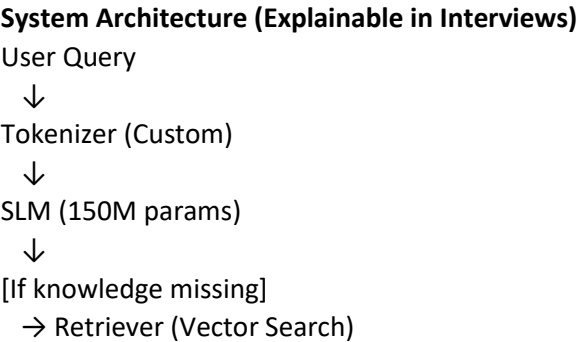
AI / Machine Learning / NLP Systems

One-Line Resume Summary (Impact Line)

Designed and trained a CPU-based Small Language Model from scratch and enhanced it using Retrieval-Augmented Generation (RAG) to deliver accurate, domain-specific responses with controlled hallucination.

- Problem Statement
- General-purpose LLMs are:
- Expensive
  - Overkill for narrow domains
  - Unreliable for factual, domain-specific queries
- Goal:
- Build a **lightweight, domain-focused language model** that:
- Runs on limited hardware (CPU, low RAM)
  - Understands domain context deeply
  - Uses external knowledge safely via RAG instead of memorization

- Implemented Solution (What You Actually Built)
- Phase 1: SLM from Scratch
- Trained a **Small Language Model (~150M parameters)** without using any hosted LLM APIs
  - Rebuilt tokenizer to fit **English + Hinglish** corpus
  - Cleaned and deduplicated dataset manually to remove noise and repetition
  - Fine-tuned model behavior using QA-style training
- Phase 2: Domain Specialization
- Created a **custom dataset focused on the 2023 Cricket World Cup**
  - Generated 1000+ unique, non-repetitive Q&A samples
  - Enforced strict semantic diversity rules
  - Evaluated factual accuracy using automated QA tests
- Phase 3: RAG Integration
- Implemented **Retrieval-Augmented Generation** to:
    - Fetch external context dynamically
    - Reduce hallucination
    - Extend knowledge beyond training data
  - Compared outputs:
    - SLM-only responses
    - SLM + RAG responses
- Phase 4: Evaluation & Auditing
- Built QA test scripts for factual validation
  - Logged incorrect reasoning patterns
  - Iteratively retrained model with cleaned data



- Context Injection
  - Answer Generation
- 

## Tech Stack

### AI / ML

- Python
- PyTorch
- Custom Tokenizer
- Transformer Architecture
- Retrieval-Augmented Generation (RAG)
- Vector Embeddings

### Data & Evaluation

- Custom Dataset Generation Scripts
- Regex-based Deduplication
- QA Evaluation Pipelines

### Infrastructure

- CPU-based Training
  - Local File System Checkpoints
  - Lightweight inference setup
- 

## Key Challenges Solved

- Training stability on CPU-only hardware
  - Tokenization issues with Hinglish text
  - Hallucination control without massive models
  - Maintaining semantic diversity in datasets
  - Balancing model size vs accuracy
- 

## Results & Outcomes

- Achieved **domain-accurate conversational responses**
  - Reduced hallucination using RAG instead of overtraining
  - Built a reusable architecture for other domains
  - Demonstrated end-to-end understanding of LLM internals
- 

## What This Project Proves (Interview Angle)

- You understand **how LLMs work internally**
- You can build AI **without APIs**
- You think in **systems, not shortcuts**
- You can explain trade-offs clearly

PROJECT 2 (AI / ML CATEGORY)

Project Title

AI Friend / Personal AI Companion (Voice-Enabled, RAG-Backed System)

Project Category

Artificial Intelligence · Conversational Systems · Voice AI

Project Overview (What this project actually is)

Built a **personal AI companion** designed to behave like a real conversational partner rather than a generic chatbot. The system combines **language understanding, voice interaction, retrieval-based knowledge access, and personality control** to deliver contextual, human-like conversations in **English, Hindi, and Hinglish**. The focus was **accuracy, controllability, and natural interaction**, not entertainment demos.

- Problem Statement
- Most conversational AIs fail in three areas:
1. **Inconsistent personality** across sessions
  2. **Hallucinated answers** when knowledge is missing
  3. **Unnatural voice interactions** that break immersion

- The goal was to design an AI companion that:
- Maintains a stable personality
  - Admits uncertainty when required
  - Uses external knowledge safely
  - Communicates naturally via voice

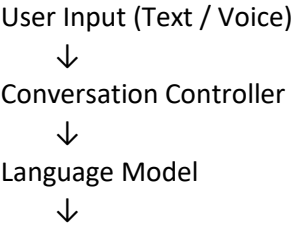
- Implemented Solution (Execution Breakdown)
- Phase 1: Core Conversational Engine
- Designed a structured conversational flow instead of open-ended chat
  - Controlled responses using system-level prompting
  - Ensured consistent tone and behavior across interactions

- Phase 2: Accuracy & Knowledge Handling
- Integrated a **Retrieval-Augmented Generation (RAG)** layer
  - Used retrieval when the model lacked internal knowledge
  - Prevented forced or fabricated answers
  - Built logic to switch between:
    - Direct response
    - Retrieved response
    - Clarification request

- Phase 3: Voice Interaction Layer
- Implemented **Text-to-Speech (TTS)** for real-time narration
  - Focused on **Hindi / Hinglish voice delivery**
  - Designed system to read on-screen explanations aloud
  - Avoided audio file storage to keep interaction lightweight and live

- Phase 4: Evaluation & Auditing
- Created structured accuracy evaluation reports
  - Identified failure patterns in reasoning
  - Refined prompt and retrieval logic iteratively
  - Tested conversational depth, not just factual correctness

System Architecture (Explainable Clearly in Interviews)



[If knowledge insufficient]  
→ Retrieval System (RAG)  
→ Context Injection  
↓  
Response Generator  
↓  
Text-to-Speech Output

---

## Tech Stack

### AI / NLP

- Python
- Transformer-based Language Model
- Prompt-controlled behavior system
- Retrieval-Augmented Generation (RAG)
- Vector-based context retrieval

### Voice & Interaction

- Text-to-Speech (TTS) integration
- Real-time narration pipeline
- Multi-language (English / Hindi / Hinglish)

### Evaluation & Control

- Accuracy auditing scripts
  - Structured conversation testing
  - Prompt refinement loops
- 

## Key Engineering Challenges Solved

- Maintaining personality consistency over long conversations
  - Preventing hallucinations without killing conversational flow
  - Designing natural Hinglish responses
  - Synchronizing voice output with on-screen responses
  - Handling unknown queries gracefully
- 

## Outcome & Learnings

- Built a controllable, voice-enabled AI companion
  - Demonstrated safe knowledge handling using RAG
  - Learned how conversational UX differs from standard chatbots
  - Developed evaluation discipline for conversational AI
- 

## Why This Project Matters (Positioning Insight)

This project demonstrates:

- Applied conversational AI engineering
- Voice + language system integration
- UX thinking inside AI systems
- Responsible AI behavior (uncertainty handling)

It's **not a chatbot** — it's a **conversation system**.

PROJECT 3 (AI + SYSTEM SIMULATION CATEGORY)

Project Name

Saarni

Project Category

AI Systems · Simulation Engineering · Infrastructure Optimization

Project Overview

Built a **train traffic simulation system** that models real-world railway operations such as train movement, delays, and platform-level scheduling.

The project focuses on **decision-making under constraints**, not just animation or visualization.

It was designed as a **hackathon-ready infrastructure AI system**, where logic correctness mattered more than UI polish.

Problem Statement

Railway traffic management faces challenges like:

- Cascading delays
- Platform conflicts
- Poor real-time rescheduling
- Static, rule-based decision systems

The objective was to simulate:

- How delays propagate
- How platform allocation affects traffic flow
- How AI-based logic could later optimize decisions

Implemented Solution (Execution Breakdown)

Phase 1: Core Simulation Engine

- Built a **map-based train movement simulator**
- Modeled trains as independent entities with:
  - Route
  - Speed
  - Schedule
  - Delay state
- Ensured movement logic stayed consistent with time progression

Phase 2: Delay Handling Logic

- Introduced artificial and real delay scenarios
- Designed logic to:
  - Update downstream schedules
  - Prevent impossible overlaps
- Simulated realistic knock-on effects instead of isolated delays

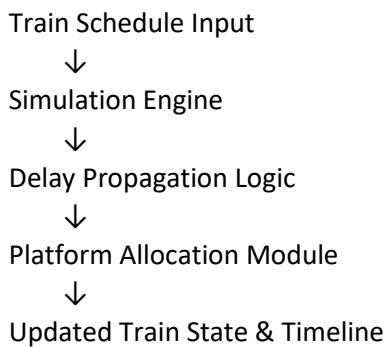
Phase 3: Station & Platform-Level Scheduling

- Modeled stations with **multiple platforms**
- Assigned trains dynamically to platforms
- Prevented conflicts like:
  - Two trains on one platform
  - Arrival before platform availability
- Allowed station-specific logic (e.g., medium-size stations like Indore)

Phase 4: AI-Ready Decision Layer (Design-Level)

- Structured the system so AI optimization could be plugged in later
- Clear separation between:
  - Simulation state
  - Decision logic
  - Visualization
- Prepared foundation for reinforcement learning or heuristic AI scheduling

System Architecture (Explainable in Interviews)



## Tech Stack

### Core Logic

- Python
- Data structures for time-based simulation
- Rule-based scheduling engine

### Simulation & Visualization

- Grid / map-based movement model
- Time-step driven simulation loop

### AI Readiness

- Modular decision layer
  - Future compatibility with AI optimizers (RL / heuristic models)
- 

## Key Engineering Challenges Solved

- Modeling real-world constraints realistically
  - Avoiding naive “one-delay-one-fix” logic
  - Handling simultaneous arrivals and departures
  - Designing a scalable simulation structure
  - Keeping logic deterministic and debuggable
- 

## Outcome & Learnings

- Built a working infrastructure simulation, not a toy model
  - Learned how **real systems fail under delays**
  - Understood why naive scheduling breaks at scale
  - Developed system-thinking applicable to traffic, logistics, and ops AI
- 

## Why This Project Matters (Positioning Insight)

This project shows:

- Systems thinking beyond CRUD apps
- Understanding of real-world constraints
- Readiness for AI-driven optimization
- Ability to model complex environments before applying ML

PROJECT 4 (MOBILE APP · AI-ASSISTED PRODUCTIVITY)

Project Name

AI Routine Planner – Smart Daily Scheduling App

Project Category

Mobile Application · AI-Assisted Productivity · Flutter

Project Overview

Built a **mobile-first routine planning application** that helps users structure their day realistically instead of creating ideal but unsustainable schedules.

The app uses **AI-assisted logic** to recommend routines based on time availability, priorities, and user constraints, with a strong focus on **practical daily execution**.

This was not a reminder app — it was a **decision-support tool for daily planning**.

Problem Statement

Most routine and productivity apps fail because:

- They assume unlimited motivation
- They ignore time constraints
- They overload users with rigid plans

The objective was to build a system that:

- Respects real-world time limits
- Adapts routines dynamically
- Assists planning instead of enforcing it

Implemented Solution (Execution Breakdown)

Phase 1: Core Routine Planning Logic

- Designed a routine model based on:
  - Available time blocks
  - Task priority
  - Energy level assumptions
- Prevented over-scheduling by enforcing realistic limits

Phase 2: AI-Assisted Recommendations

- Integrated AI logic to:
  - Suggest task ordering
  - Balance work, rest, and personal activities
- Provided soft suggestions rather than hard enforcement

Phase 3: Mobile-First UX (Flutter)

- Built the app using **Flutter** for cross-platform support
- Focused on minimal input friction
- Designed screens for fast daily check-ins

Phase 4: State & Data Handling

- Implemented local state management for routine persistence
- Ensured fast startup and offline usability
- Allowed easy routine edits without re-planning everything

System Architecture (Explainable in Interviews)

User Inputs (Tasks, Time)



Routine Logic Engine



AI Recommendation Layer



Daily Schedule Output

Tech Stack

## Mobile Development

- Flutter
- Dart
- Local storage (device-based)

## AI Assistance

- Lightweight AI logic for task ordering
- Constraint-based scheduling rules

## App Architecture

- Modular widget structure
  - State management for daily routines
- 

## Key Engineering Challenges Solved

- Preventing unrealistic schedules
  - Balancing automation with user control
  - Designing UX for daily repeat usage
  - Making AI suggestions feel optional, not forced
- 

## Outcome & Learnings

- Built a usable productivity app, not just a UI demo
- Learned how small AI suggestions improve adoption
- Understood why simplicity beats feature overload
- Gained experience in mobile-first product thinking



PROJECT 5 (FULL-STACK · HEALTHCARE SOFTWARE)

Project Name

AayushCare – Doctor–Patient Healthcare Management System

Project Category

Full-Stack Software · Healthcare Systems · Data Management

Project Overview

Built a **fully functional healthcare management software** that connects **doctors and patients** on a single platform and securely maintains **patient medical history** over time.

The system was designed as a **real-world healthcare application**, not a prototype — focusing on **data integrity, role-based access, and long-term record management**.

Problem Statement

Traditional healthcare record handling suffers from:

- Fragmented patient history
- Manual or unsafe data storage
- No continuity between doctor visits
- Poor digital access for patients

The goal was to create a system that:

- Digitally stores patient medical history
- Allows doctors to manage records efficiently
- Gives patients controlled access to their own data
- Maintains data consistency across sessions

Implemented Solution (Execution Breakdown)

Phase 1: System Design & Roles

- Defined two primary roles:
  - Doctor**
  - Patient**
- Designed role-based workflows:
  - Doctors can create, update, and review medical records
  - Patients can view their own medical history

Phase 2: Patient Medical History Management

- Implemented structured storage for:
  - Personal details
  - Visit records
  - Diagnoses
  - Prescriptions
- Ensured historical records remain immutable once saved
- Allowed chronological tracking of patient health data

Phase 3: Backend & Database Integration

- Built backend logic to:
  - Handle user registration and login
  - Enforce access control
  - Validate medical data before storage
- Connected the system to a real database
- Ensured proper relationships between:
  - Doctors
  - Patients
  - Medical records

Phase 4: Application Flow & Usability

- Designed clean workflows for doctors to:
  - Quickly access patient history
  - Add new consultation data

- Designed patient views to:
    - Read medical history without editing rights
  - Focused on clarity and reliability over visual complexity
- 

## System Architecture (Explainable in Interviews)

User (Doctor / Patient)



Authentication Layer



Role-Based Access Control



Healthcare Logic Layer



Medical Records Database

---

## Tech Stack

### Frontend

- Web-based user interface
- Role-specific dashboards

### Backend

- Server-side application logic
- Authentication & authorization
- API-based data handling

### Database

- Structured medical records storage
  - Patient–doctor relationship mapping
- 

## Key Engineering Challenges Solved

- Designing secure role-based access
  - Maintaining long-term patient data integrity
  - Preventing unauthorized data modification
  - Structuring medical data for scalability
  - Building healthcare logic without overcomplication
- 

## Outcome & Learnings

- Delivered a **working healthcare software**, not just CRUD pages
- Gained experience in **domain-critical software design**
- Learned how data sensitivity changes system decisions
- Understood the importance of clean data models in healthcare

PROJECT 6 (AI / ML · INFORMATION VERIFICATION)

Project Name

Fake News Detection System using Web Search + RAG

Project Category

Artificial Intelligence · NLP · Information Verification Systems

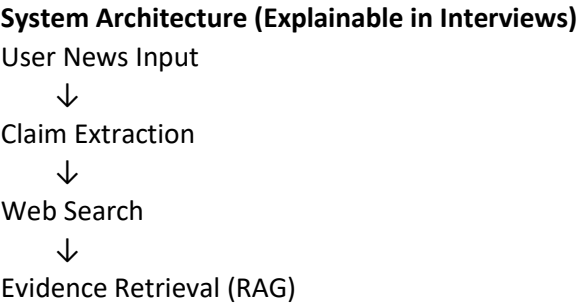
**Project Overview**

Built a **fake news detection system** that verifies claims by combining **live web search** with **Retrieval-Augmented Generation (RAG)** instead of relying only on static training data.

The system focuses on **evidence-backed verification**, not just text classification, making it suitable for real-world misinformation scenarios.

- Problem Statement**
- Traditional fake news detectors often:
- Rely only on text patterns
  - Fail on new or evolving news
  - Classify without showing evidence
- The objective was to build a system that:
- Actively searches for supporting or contradicting information
  - Grounds responses in real sources
  - Explains *why* a piece of news is likely fake or real

- Implemented Solution (Execution Breakdown)**
- Phase 1: Claim Extraction & Understanding**
- Designed logic to extract the **core claim** from user-provided news text
  - Removed emotional or irrelevant language
  - Focused verification on the factual assertion
- Phase 2: Web Search Integration**
- Integrated web search to fetch:
    - News articles
    - Official statements
    - Trusted sources
  - Avoided reliance on outdated training data
  - Ensured the system reacts to **current events**
- Phase 3: Retrieval-Augmented Generation (RAG)**
- Stored fetched information as retrievable context
  - Injected verified content into the response generation pipeline
  - Prevented hallucinated explanations by forcing evidence grounding
- Phase 4: Verification & Reasoning Layer**
- Compared user claim against retrieved evidence
  - Classified outcomes into:
    - Likely True
    - Likely False
    - Insufficient Evidence
  - Generated reasoning explaining the decision clearly





## Reasoned Verdict + Explanation

---

### Tech Stack

#### AI / NLP

- Python
- Transformer-based language model
- Retrieval-Augmented Generation (RAG)
- Semantic similarity matching

#### Data & Search

- Web search integration
- Source filtering logic
- Context ranking

#### System Design

- Modular verification pipeline
  - Evidence-first reasoning logic
- 

### Key Engineering Challenges Solved

- Handling rapidly changing news
  - Preventing hallucinated justifications
  - Extracting verifiable claims from noisy text
  - Balancing speed with verification depth
  - Explaining uncertainty transparently
- 

### Outcome & Learnings

- Built a misinformation system grounded in evidence
- Learned why classification alone is insufficient
- Understood the importance of explainability in AI
- Developed practical experience with RAG in real-world use

PROJECT 7 (AI / NLP · CAREER INTELLIGENCE TOOL)

Project Name

AI Resume–Job Description Analyzer (ATS-Aware System)

Project Category

Artificial Intelligence · NLP · Career Automation

Project Overview

Built an **AI-powered resume analysis system** that compares a candidate’s resume against a given **job description (JD)** and provides **actionable improvement suggestions**, with a strong focus on **ATS (Applicant Tracking System) compatibility**. The tool was designed to help users understand **why** their resume gets filtered out and **what exactly needs to change**, instead of offering generic advice.

Problem Statement

Most job seekers face rejection because:

- Their resume doesn’t align semantically with the JD
- ATS systems fail to parse keywords correctly
- Feedback is vague or non-actionable

The goal was to create a system that:

- Analyzes resume–JD alignment
- Highlights missing or weak areas
- Suggests improvements that are ATS-friendly and realistic

Implemented Solution (Execution Breakdown)

Phase 1: Resume & JD Parsing

- Built logic to parse resumes and job descriptions into structured sections:
  - Skills
  - Experience
  - Responsibilities
  - Requirements
- Removed formatting noise to focus on semantic content

Phase 2: Semantic Matching & Analysis

- Compared resume content with JD requirements using NLP techniques
- Identified:
  - Missing skills
  - Weakly represented experience
  - Keyword mismatches
- Avoided exact keyword matching alone by using semantic similarity

Phase 3: AI-Based Suggestions

- Generated contextual suggestions such as:
  - Which skills to emphasize
  - Which experience points need clearer wording
  - Where alignment is weak but improvable
- Ensured suggestions remained truthful and not fabricated

Phase 4: ATS Awareness Layer

- Designed output to be **ATS-compatible**:
  - Simple language
  - Clear sectioning
  - Avoidance of misleading or stuffed keywords
- Focused on improving *machine readability* without harming human readability

System Architecture (Explainable in Interviews)

Resume + Job Description

↓

Text Parsing & Structuring



Semantic Comparison Engine



Gap Detection



Improvement Suggestions

---

## **Tech Stack**

### **AI / NLP**

- Python
- NLP-based semantic similarity
- Transformer-based language model
- Text preprocessing pipelines

### **System Logic**

- Resume & JD structure extraction
  - Rule-based + AI hybrid suggestion logic
- 

## **Key Engineering Challenges Solved**

- Handling diverse resume formats
  - Preventing misleading or fake resume suggestions
  - Balancing ATS optimization with honesty
  - Explaining gaps clearly instead of scoring blindly
  - Avoiding keyword stuffing recommendations
- 

## **Outcome & Learnings**

- Built a practical career-support AI tool
- Learned how ATS systems influence resume design
- Understood limitations of pure keyword matching
- Gained experience in explainable NLP systems

PROJECT 8 (CYBERSECURITY · BROWSER EXTENSION)

Project Name

Malicious Content Verification Browser Extension

Project Category

Cybersecurity · Browser Extensions · AI-Assisted Threat Detection

Project Overview

Built a **security-focused browser extension** that helps users verify whether **links, emails, messages, or images contain malicious or unsafe content** before interacting with them.

The tool acts as a **pre-click safety layer**, especially useful for phishing prevention and social-engineering attacks, where users often trust content blindly.

Problem Statement

Most users fall victim to scams because:

- Malicious links look legitimate
- Verification happens *after* clicking
- Existing tools are either too technical or too passive

The objective was to create a system that:

- Actively verifies suspicious content
- Works across multiple content types
- Gives clear risk signals instead of technical jargon

Implemented Solution (Execution Breakdown)

Phase 1: Content Extraction

- Built logic to extract potential threats from:
  - URLs
  - Email text
  - Messages
  - Embedded links in images or web pages
- Normalized extracted data for consistent analysis

Phase 2: Verification Logic

- Implemented checks to identify:
  - Suspicious domains
  - Redirect-heavy URLs
  - Known scam patterns
- Flagged risky content before user interaction

Phase 3: AI-Assisted Risk Assessment

- Used AI logic to analyze:
  - Context around the link or message
  - Language patterns commonly used in phishing
- Combined rule-based detection with AI reasoning
- Avoided binary “safe/unsafe” judgments by indicating risk level

Phase 4: Browser Integration

- Implemented the system as a **browser extension**
- Enabled scanning directly within the browser context
- Ensured minimal performance impact during browsing

System Architecture (Explainable in Interviews)

User Content (Link / Email / Message / Image)



Content Extraction



Rule-Based Security Checks



AI Risk Analysis



Risk Verdict & User Warning

---

**Tech Stack**

**Extension Development**

- JavaScript
- Browser Extension APIs
- DOM inspection logic

**Security & AI Logic**

- URL analysis techniques
  - Pattern-based phishing detection
  - AI-assisted contextual analysis
- 

**Key Engineering Challenges Solved**

- Extracting links reliably from varied content types
  - Reducing false positives
  - Making warnings understandable to non-technical users
  - Balancing security checks with browsing performance
  - Handling obfuscated or shortened URLs
- 

**Outcome & Learnings**

- Built a real-world cybersecurity utility
- Gained experience in phishing detection logic
- Learned how attackers exploit trust and urgency
- Understood browser security limitations and workarounds



PROJECT 9 (BROWSER AUTOMATION · COMPLIANCE & TRANSPARENCY)

Project Name

Website Policy & Compliance Scanner Browser Extension

Project Category

Browser Extensions · Web Compliance · Automation Tools

Project Overview

Built a **browser-based compliance scanner** that automatically detects whether a website provides essential legal and transparency documents such as **Privacy Policy, Terms & Conditions, and Cookie disclosures**.

The tool was designed for **real-world browsing**, not controlled demos, and was tested across live websites with different layouts and structures.

Problem Statement

Many websites:

- Hide or poorly expose legal policies
- Mislead users about cookie usage
- Fail basic transparency standards

Manual verification is slow and inconsistent.

The goal was to create a tool that:

- Automatically checks compliance signals
- Works across varied website structures
- Gives instant, understandable feedback to users

Implemented Solution (Execution Breakdown)

Phase 1: Page Content Detection

- Implemented DOM scanning to locate:
  - Privacy Policy references
  - Terms & Conditions links
  - Cookie-related disclosures
- Handled variations in naming (e.g., “Privacy”, “Legal”, “Data Policy”)

Phase 2: Navigation & Dynamic Content Handling

- Designed logic to:
  - Detect policies even when loaded dynamically
  - Follow internal links where policies were not on the landing page
- Addressed cases where content loads after user interaction or scrolling

Phase 3: Verification Logic

- Differentiated between:
  - Actual policy pages
  - Placeholder or misleading links
- Ensured detection was based on content presence, not link labels alone

Phase 4: Browser Extension Integration

- Integrated detection logic into a browser extension
- Provided real-time feedback while browsing
- Focused on low overhead to avoid slowing page loads

System Architecture (Explainable in Interviews)

Active Webpage



DOM & Link Scanner



Policy Identification Logic



Compliance Status Output

## **Tech Stack**

### **Extension Development**

- JavaScript
- Browser Extension APIs
- DOM traversal and mutation observers

### **Automation Logic**

- Pattern-based text detection
  - Link validation heuristics
  - Dynamic content handling
- 

### **Key Engineering Challenges Solved**

- Handling inconsistent website structures
  - Detecting policies hidden behind multiple clicks
  - Avoiding false positives from dummy links
  - Working with dynamically loaded content
  - Keeping the extension lightweight and fast
- 

### **Outcome & Learnings**

- Built a practical transparency-checking tool
- Learned how websites differ widely in compliance implementation
- Gained strong DOM analysis and browser automation experience
- Understood real-world limitations of automated compliance checks

PROJECT 10 (AI / NLP · SENTIMENT ANALYSIS)

Project Name

Sentiment Analysis System for Large-Scale User Comments

Project Category

Artificial Intelligence · Natural Language Processing · Text Analytics

**Project Overview**

Built a **sentiment analysis system** to analyze and classify user opinions from real-world textual data.

The system was tested on **700+ real comments**, focusing on **robustness, consistency, and practical evaluation**, not just model accuracy claims.

The project emphasized **end-to-end NLP workflow** — from raw text handling to result interpretation.

- Problem Statement**
- User-generated content is:
- Noisy
  - Informal
  - Emotionally inconsistent
- Simple sentiment models often fail due to:
- Slang and mixed language
  - Sarcasm or short expressions
  - Poor preprocessing
- The objective was to build a system that:
- Handles real, unclean text
  - Produces stable sentiment outputs
  - Can be evaluated meaningfully

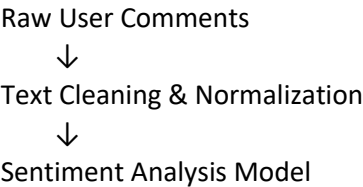
- Implemented Solution (Execution Breakdown)**
- Phase 1: Data Preparation**
- Collected and structured **700+ user comments**
  - Cleaned text by:
    - Removing noise and duplicates
    - Normalizing casing and spacing
  - Handled informal language patterns

- Phase 2: Sentiment Classification Logic**
- Designed sentiment categories such as:
    - Positive
    - Negative
    - Neutral
  - Applied NLP techniques to infer emotional polarity
  - Avoided overfitting to extreme language

- Phase 3: Evaluation & Validation**
- Tested model behavior across diverse comment styles
  - Checked consistency rather than isolated predictions
  - Analyzed misclassifications to refine preprocessing

- Phase 4: Result Interpretation**
- Converted raw predictions into understandable summaries
  - Focused on **trend-level insights**, not just individual labels

**System Architecture (Explainable in Interviews)**





## Sentiment Labels & Trends

---

### Tech Stack

#### AI / NLP

- Python
- NLP preprocessing pipelines
- Sentiment analysis model
- Text normalization utilities

#### Data Handling

- Structured text datasets
  - Evaluation scripts
- 

### Key Engineering Challenges Solved

- Handling noisy, informal user text
  - Preventing over-confidence on ambiguous comments
  - Evaluating sentiment meaningfully beyond accuracy
  - Managing class imbalance in real data
  - Interpreting results for non-technical use
- 

### Outcome & Learnings

- Built a stable sentiment analysis pipeline
- Learned importance of data quality over model size
- Gained experience in NLP evaluation
- Understood limits of sentiment classification in real scenarios

PROJECT 11 (FULL-STACK · CONTENT MANAGEMENT SYSTEM)

Project Name

Admin-Controlled CMS & Content Publishing System

---

Project Category

Full-Stack Development · Content Management · Web Systems

---

Project Overview

Built a **custom Content Management System (CMS)** that allows an **admin to fully control website content** through a dedicated dashboard.

The system was designed to replicate **real-world blogging and publishing workflows** (similar to platforms like Blogspot), with a strong focus on **data control, reliability, and real-data testing**.

This was not a static website — it was a **content-driven system** where every visible element is managed dynamically by the admin.

---

Problem Statement

Many basic websites fail to scale because:

- Content is hardcoded
- Non-technical users cannot update content
- Admin control is fragmented or unsafe

The objective was to build a system where:

- Admins can manage content without touching code
  - Published data flows cleanly from database to UI
  - Changes reflect instantly and reliably on the user-facing site
- 

Implemented Solution (Execution Breakdown)

Phase 1: Admin Dashboard Design

- Designed a dedicated **admin panel**
- Implemented editor-based content creation
- Enabled admin to:
  - Create posts
  - Edit existing content
  - Manage visibility

Phase 2: Backend & Database Integration

- Connected admin actions to a real database
- Ensured:
  - Data validation before saving
  - Clean separation between draft and published content
- Designed schemas suitable for long-term content storage

Phase 3: Frontend Content Rendering

- Built user-facing pages that:
  - Fetch content dynamically from backend
  - Render posts based on admin configuration
- Ensured no dummy or hardcoded data remained

Phase 4: System Testing with Real Data

- Tested the system using real admin-written content
  - Verified:
    - End-to-end data flow
    - Update reliability
    - No UI breakage on content changes
  - Identified and fixed loopholes between admin, backend, and frontend layers
- 

System Architecture (Explainable in Interviews)

Admin Dashboard



Backend API



Database



User-Facing Website

---

**Tech Stack**

**Frontend**

- Web UI for admin and users
- Editor-based content input
- Dynamic rendering logic

**Backend**

- Server-side APIs
- Content validation logic
- Admin authorization handling

**Database**

- Structured content storage
- Admin–content relationships

---

**Key Engineering Challenges Solved**

- Eliminating hardcoded content completely
- Maintaining consistency between admin edits and live UI
- Preventing invalid or broken content states
- Designing schemas flexible enough for future features
- Testing with real data instead of mock content

---

**Outcome & Learnings**

- Built a production-style CMS system
- Learned real-world admin workflow design
- Understood importance of content lifecycle management
- Gained experience in full-stack data flow ownership

PROJECT 12 (SIMULATION · VISUALIZATION ENGINEERING)

Project Name

Dynamic Visual Simulation System using Pygame

Project Category

Simulation Systems · Visualization Engineering · Python

Project Overview

Built a **dynamic visual simulation system** using Pygame that can render **multiple diagrams, animations, and explanatory text** without layout breakage.

The core focus was **automatic layout adaptation**—ensuring visuals never overflow the screen, overlap text, or go out of frame, regardless of content count or screen size.

This was a **logic-heavy visualization system**, not a static animation.

Problem Statement

In simulation and educational visuals, common issues include:

- Diagrams moving out of frame
- Text overlapping visuals
- Fixed layouts breaking with multiple elements
- Manual positioning becoming unscalable

The goal was to build a system that:

- Automatically adjusts layout based on content
- Handles multiple diagrams dynamically
- Keeps visuals readable and structured at all times

Implemented Solution (Execution Breakdown)

Phase 1: Screen-Aware Layout Engine

- Designed logic to detect:
  - Screen dimensions
  - Available rendering space
- Prevented fixed positioning assumptions
- Made layout responsive to screen size

Phase 2: Dynamic Diagram Management

- Implemented automatic placement for:
  - Single diagram
  - Multiple diagrams
- Distributed diagrams evenly within the available area
- Ensured no overlap between diagrams or text blocks

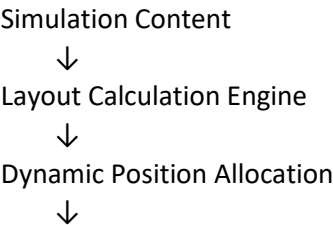
Phase 3: Text–Visual Synchronization

- Synchronized explanatory text with visuals
- Ensured text wraps correctly within boundaries
- Prevented collision between narration text and diagrams

Phase 4: Adaptive Rendering Loop

- Built a rendering loop that:
  - Recalculates layout on content change
  - Maintains frame stability
- Ensured smooth rendering without flicker or clipping

System Architecture (Explainable in Interviews)



**Tech Stack**

**Core**

- Python
- Pygame

**Logic & Layout**

- Dynamic coordinate calculation
  - Responsive layout rules
  - Collision & overflow prevention logic
- 

**Key Engineering Challenges Solved**

- Eliminating hardcoded positions
  - Managing multiple visual elements simultaneously
  - Preventing text–diagram overlap
  - Maintaining clarity under varying content loads
  - Keeping rendering logic deterministic
- 

**Outcome & Learnings**

- Built a reusable visual simulation framework
- Learned why layout logic is harder than rendering
- Gained experience in real-time rendering systems
- Improved ability to debug visual systems methodically



PROJECT 13 (VOICE AI · REAL-TIME NARRATION SYSTEM)

Project Name

Real-Time Text-to-Speech (TTS) Narration System

Project Category

Voice AI · Human-Computer Interaction · Accessibility Systems

Project Overview

Built a **real-time Text-to-Speech (TTS) system** that converts on-screen explanations into **live voice narration**, with a strong focus on **natural Hindi / Hinglish speech**.  
The system was intentionally designed **without audio file storage**—all speech is generated and played in real time—making it lightweight, privacy-aware, and suitable for interactive applications.

Problem Statement

Most TTS implementations:

- Generate and store audio files unnecessarily
- Add latency to user interaction
- Are poorly suited for live explanations

The goal was to design a system that:

- Speaks immediately as content appears
- Feels conversational rather than robotic
- Works seamlessly with dynamic on-screen text

Implemented Solution (Execution Breakdown)

Phase 1: Live Narration Pipeline

- Built a pipeline to:
  - Capture text as it appears on screen
  - Send text directly to the TTS engine
  - Play audio output instantly
- Avoided intermediate file storage completely

Phase 2: Language & Voice Handling

- Focused on **Hindi / Hinglish narration**
- Tuned speech parameters for:
  - Natural pacing
  - Clear pronunciation
- Ensured narration matched on-screen explanations

Phase 3: Integration with Applications

- Designed the system to plug into:
  - AI explanation tools
  - Educational simulations
  - Interactive demos
- Allowed narration to respond dynamically to content changes

Phase 4: Performance & UX Optimization

- Reduced latency between text generation and speech
- Ensured smooth playback without interruptions
- Maintained responsiveness even during rapid content updates

System Architecture (Explainable in Interviews)

On-Screen Text



TTS Request Handler



Live Voice Generation



Audio Output (No File Storage)

---

## **Tech Stack**

### **Voice & Interaction**

- Text-to-Speech engine
- Real-time audio playback

### **System Design**

- Event-driven text capture
- Low-latency processing pipeline

---

## **Key Engineering Challenges Solved**

- Eliminating audio file dependency
- Synchronizing narration with UI updates
- Handling rapid text changes smoothly
- Achieving natural-sounding Hindi / Hinglish speech
- Keeping the system lightweight and responsive

---

## **Outcome & Learnings**

- Built a production-style live TTS system
- Gained experience in real-time voice pipelines
- Learned how latency affects user trust
- Understood accessibility and UX considerations in voice systems

PROJECT 14 (AI SYSTEM · DESKTOP APPLICATION)

Project Name

MeaDocs – Offline AI-Powered Multimedia Search Engine (Desktop)

Project Category

AI Systems · Desktop Application Engineering · Offline Search Engines

Project Overview

Engineered **MeaDocs**, an **AI-powered, fully offline multimedia search engine** capable of finding **images, videos, documents, and audio files** from a user’s local device using **natural language queries**.

The system was designed specifically for **offline-first environments**, enabling professionals—especially in healthcare—to search large volumes of local data **without internet connectivity**, cloud APIs, or external dependencies.

This project transformed an existing **Python-based system** into a **production-ready native Windows desktop application** using **Electron**, demonstrating deep expertise in **system architecture, IPC communication, performance optimization, and deployment engineering**.

- Problem Statement
- Most AI search tools:
- Depend on cloud connectivity
  - Upload sensitive data externally
  - Fail in low or no-internet environments
  - Are difficult to deploy outside developer machines
- The goal was to build a system that:
- Works **100% offline**
  - Searches across **multiple media types**
  - Understands **natural language queries**
  - Can be installed and run by non-technical users
  - Preserves **data privacy by design**

- Implemented Solution (Execution Breakdown)
- Phase 1: Offline AI Search Engine
- Built a natural language search engine capable of querying:
    - Documents
    - Images
    - Videos
    - Audio files
  - Indexed local files directly from the user’s device
  - Ensured zero cloud dependency for inference or retrieval
  - Designed the system to operate entirely on local compute

- Phase 2: Electron + Python System Architecture
- Converted a Python-based backend into a **native desktop application**
  - Used **Electron.js** for the desktop shell
  - Implemented a **robust IPC bridge** between:
    - Electron (Node.js)
    - Python backend (Flask)
  - Enabled **real-time communication** between UI and AI engine

- This architecture allowed:
- Clean separation of concerns
  - Independent scaling of UI and AI layers
  - Stable long-running background processes

- Phase 3: Performance Optimization
- Implemented **lazy-loading of AI models**
  - Parallelized backend initialization

- Achieved **~300% faster startup time**
- Prevented UI blocking during heavy model loading

Performance tuning focused on **user experience**, not benchmarks alone.

---

#### Phase 4: Native Windows Integration

- Integrated deep Windows OS features:
    - System tray support
    - Taskbar controls
    - Native file system access
  - Implemented **encrypted local storage** for sensitive data
  - Designed **custom process lifecycle management** to handle:
    - Graceful startup
    - Safe shutdown
    - Crash recovery
    - Error isolation between Electron and Python processes
- 

#### Phase 5: Production-Grade Packaging & Deployment

- Built an **automated build pipeline**
- Packaged:
  - Python runtime
  - All dependencies
  - Application assets
- Delivered as a **single ~150MB installer**
- Achieved **zero-dependency installation** on Windows 10+

Deployment complexity was reduced from **15+ manual steps to a single-click installer**.

---

#### System Architecture (Explainable in Interviews)

Electron Desktop UI (Node.js)

↓ IPC

Python Backend (Flask)

↓

Offline AI Search Engine

↓

Encrypted Local Storage + File Index

---

#### Tech Stack

##### Desktop & Frontend

- Electron.js
- Node.js
- JavaScript (ES6+)
- HTML5 / CSS3

##### Backend & AI

- Python
- Flask
- Offline AI inference & indexing

##### Build & Deployment

- Electron-BUILDER
  - PowerShell
  - Custom packaging scripts
- 

#### Key Engineering Challenges Solved

- Bridging Electron and Python reliably via IPC
- Packaging Python runtime inside a desktop installer
- Achieving fast startup with heavy AI models
- Designing offline-first AI architecture

- Managing multi-process lifecycle safely
  - Ensuring data privacy with encrypted local storage
- 

### Outcome & Learnings

- Delivered a **production-ready offline AI desktop application**
- Enabled AI-powered search in zero-connectivity environments
- Gained deep experience in:
  - Desktop system architecture
  - Cross-runtime communication
  - Build and release engineering
- Learned how real-world deployment differs from development setups

PROJECT 15 (MOBILE APP · OFFLINE AI MEDIA SEARCH)

Project Name

MeaDocs Mobile – Offline AI Photo Search App

Project Category

Mobile Application · AI-Powered Media Search · Flutter

Project Overview

Built a **lightweight mobile application** focused exclusively on **AI-powered photo search** from a user’s local device. The app allows users to find images using **natural language queries**, operating **entirely offline**, with no cloud dependency. This mobile app was designed as a **specialized companion** to the MeaDocs ecosystem, optimized for **quick visual retrieval** rather than full multimedia search.

Problem Statement

- On mobile devices, users often struggle to:
- Find specific photos from large galleries
  - Remember file names or folder locations
  - Rely on cloud-based search for private images
- The objective was to create a mobile app that:
- Searches photos semantically, not by filename
  - Works completely offline
  - Preserves user privacy
  - Delivers fast results on limited mobile hardware

Implemented Solution (Execution Breakdown)

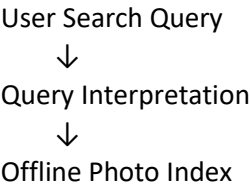
- Phase 1: Offline Photo Indexing
- Built logic to scan and index images stored locally on the device
  - Stored metadata and visual features locally
  - Ensured indexing did not require internet access

- Phase 2: Natural Language Query Handling
- Enabled users to search photos using plain language
  - Translated queries into search intent
  - Matched queries against locally indexed photo data

- Phase 3: Mobile-First Application Design
- Built the app using **Flutter** for cross-platform readiness
  - Designed a minimal UI focused on:
    - Search
    - Results
    - Image preview
  - Optimized for quick, one-hand mobile usage

- Phase 4: Performance & Privacy Optimization
- Ensured all processing happens on-device
  - Avoided background network calls
  - Focused on low latency for photo retrieval
  - Maintained strict user data privacy

System Architecture (Explainable in Interviews)





Matching & Ranking



Photo Results

---

## Tech Stack

### Mobile Development

- Flutter
- Dart

### AI / Search Logic

- On-device image indexing
- Semantic matching logic
- Offline inference pipeline

---

## Key Engineering Challenges Solved

- Running semantic search on mobile hardware
- Managing local storage efficiently
- Keeping the app responsive during indexing
- Designing privacy-first offline workflows
- Avoiding battery and memory overuse

---

## Outcome & Learnings

- Built a focused, production-style mobile app
- Learned constraints of on-device AI on mobile
- Understood trade-offs between accuracy and performance
- Gained experience in privacy-first mobile AI design

---

## Why This Project Matters (Positioning Insight)

This project shows:

- Ability to design **narrow, well-scoped products**
- Experience with offline AI on mobile
- Strong privacy and performance awareness
- Practical Flutter application development

It complements the **MeaDocs Desktop system** by demonstrating how the same AI philosophy can be adapted to **mobile constraints**.