**Detailed Projects Information**

**Automation, Scripting, Scrapping and Machine Learning Jobs**

**In all the below stated applications these are the main applications and challenges**

**Applications**

* Whatever a human does, machines can do it better, provided there’s some consistency in that task

**Challenge**

* The detail in which a task can be automated is imported.
* But outlier detection and handling is a challenge since it is unforeseen. In such cases, machine may give wrong output.
* Properly defined scenarios are important like detection of textbox, checking it, login boxes etc
* Low maintenance automation means that client should spend more money so that automation can be done in as much minute detail as possible by inculcating proxy health checkers etc
* Scrapers for social media platforms require multiple strategies because they can detect if robot is browsing or human is. These strategies are Robot.txt file (everyone can avoid these), 3rd party recaptcha, rotational proxies, fake browsing history (has to be imported to suggest a human is browsing because its pretty obvious if a robot is browsing, his browsing history will be clear/fresh)
* Real estate data scrapping system can be developed alongwith parcel data information
* Data Consistency
* Proxies and Server Costs

1. **Dataflik**

**Problem Statement**

Creation and maintenance of scoring system for historical real estate properties to analyze which property is most to least likely to get sold in X amount of time.

**Solution**

We used basic ML models to train some prediction models and settled on one of them, based on last 40 years of historical data. Additionally, we developed some scrappers that gathered stress data (a game changer) including inheritance, bankruptcy, foreclosure. For example, calculating high probability of an inherited house to immediately go on the market for selling. Another example was that a bankrupted company/ business would immediately want to sell the houses under their ownership.

We also provided them leads that ABC property was being sold and XYZ was their owner’s name, also selling his number on additional rates. Also, we developed 2 business models. We had mailing information for individuals’ data i.e., the home owners mailing address and their contact information. One model was approaching the individuals by mail and the other was to approach them via phone. The company that wanted this system already had 90% coverage of individual information i.e., upto 5 phone numbers per each person. To sell all accumulated data and code, we developed an end to end product on ElasticSearch, to make it searchable and we deployed it too. Skiptracing for their database of phone numbers was also made searchable by us, where if data was purchased of 10,000 and they wanted to script trace it, then upon calculating and receiving payment, we displayed data till the required number

We used ElasticSearch because it’s a NoSQL database, flexible and is search centric. Since we had 200+ million rows and each row had 250+ million columns, for search on this data of huge architecture and instant display. Also, for text search for addresses is done better by Elastic Search, because even if you type somewhat wrong address, it still provides similar results just like Google

**Industry**

Real Estate

**Technology Stack Used:**

* + Linode cloud & hosted on ElasticSearch & area based display on map
  + Backend in ElasticSearch
  + React for Frontend
  + Python Django in backend
  + Cloud was Linode
  + combo VPS and Kubernetes cluster (K8) was used to host ElasticSearch, as Database ELK
  + AI/ML techniques were employed.

Website: <https://www.dataflik.com/>

**Similar Applications:**

* Development
* Scrapping
* Prediction systems
* Big data processing

**Software skills used in this project:**

**In this project, several software skills and technologies were utilized:**

**1. Machine Learning (ML):** Basic ML models were trained for predicting the likelihood of real estate properties getting sold within a certain timeframe. This involved data preprocessing, model training, and evaluation.

**2. Web Scraping:** Scraping tools and techniques were employed to gather stress data, including information related to inheritance, bankruptcy, and foreclosure. This involved web scraping libraries and methods to collect and organize relevant data.

**3. Data Analysis:** Data analysis techniques were used to process and analyze historical real estate data spanning 40 years. This helped in understanding patterns and trends that could inform the predictive models**.**

**4. Data Integration:** The project involved integrating various data sources, such as historical real estate data and stress data, into a unified system for analysis.

**5. Database Management:** ElasticSearch, a NoSQL database, was employed to handle the massive amount of data efficiently. This involved setting up, managing, and querying the ElasticSearch database.

**6. Data Engineering:** Handling a dataset with 200+ million rows and 250+ million columns required data engineering skills to ensure data quality, scalability, and performance.

**7. Data Enrichment:** The project involved enriching the data with additional information, such as property owner details and contact information, which likely required data cleansing and merging techniques.

**8. Data Visualization:** While not explicitly mentioned, data visualization tools and libraries may have been used to create visual representations of the data for analysis and reporting.

**9. Software Development:** Custom software development was required to create the prediction models, develop web scrapers, and build the end-to-end product on ElasticSearch. This may have involved programming languages like Python and libraries such as Scikit-learn for ML, and Elasticsearch for database management.

**10. Deployment:** Deploying the end-to-end product on ElasticSearch and making it searchable for users would have required deployment and server management skills.

**12. Text Search and Similarity Matching:** ElasticSearch was used for text-based searches, which involves text indexing, search queries, and similarity matching algorithms.

**13. Data Security:** Given the sensitivity of the data, data security measures would have been necessary to protect user information and payment data.

**Overall, this project required a combination of data science, machine learning, web development, database management, and software engineering skills to create a comprehensive system for analyzing and predicting real estate property sales.**

**Criminal Scrapper**

**Problem Statement**

Developing a lawyer friendly system that recommends lawyers, reminds about appointments and court hearings. This was done for a lawyer for his inhouse team

**Solution**

It was an end to end job for automation, which scrapped legal US websites, i.e., since all filed case information is publically available, so what a lawyer would do was that if somebody from his area was incriminated, he would attain scrapped contact info and send his details to that person telling him that he was willing to represent him via mailed letter or an email.

**Industry**

* Legal
* Law Firm

**Tech Stack**

* Used Beautiful Soap library
* Selenium
* Scrapy.
* Mainly, selenium was used to load dynamic content, as middleware for things that could not be done by Scrapy.

**Software Skills used in this prject:**  
  
**1. Web Scraping:** The project involved web scraping from legal websites to collect information about filed cases, lawyers, and contact details. This required skills in web scraping tools and techniques.

**2. Data Collection and Integration:** Data collected from various legal websites needed to be integrated and structured for further processing. This includes organizing the scraped information in a usable format.

**3. Automation:** The goal was to create an end-to-end automation system, which likely involved scripting and automation of various tasks, such as sending appointment reminders and court hearing notifications.

**4. Recommendation Systems:** Developing a lawyer recommendation system required knowledge of recommendation algorithms and machine learning

**5. Middleware:** The project utilized Selenium as middleware to interact with websites and load dynamic content, enhancing the capabilities of the Scrapy web scraping framework.

**7. Email Integration:** Sending appointment reminders and court hearing notifications via email would require knowledge of email integration and communication libraries.

**Estateza**

**Problem Statement**

Enhancing and Maintaining Real Estate Data and Development of that Data according to client’s requirements.

**Solution**

We had three main jobs:

* Scrapping
* API development
* Migration

We also migrated their system from AWS to Linode. DevOps and kubernetes mainly used, Postgres SQL and Cloud functions too. In their Backend, we scrapped their database and two APIs were developed: one was connected to the website (this displayed all information) and the other was for data selling. The scrapper we developed was the biggest database of US residential and commercial properties that worked 24/7 and shared an hourly report on Slack. Also, if any scrapper failed, it would highlight this too on Slack via notifications. Also, the proxy manager maintained proxies and their rotation. The Rent Estimator engine provided estimates for land, rent and lease on a property through other properties’ information to estimate probable rent.

**Industry**

* Real Estate

**Tech Stack**

* Django
* ReactJS
* Scrapy
* AWS
* Azure
* BigData
* ELK

Website: <https://estateza.com/>

**Applications:**

* Development
* Scrapping
* Prediction systems
* Big data processing

**Proposed Solutions:**

* Real estate data scrapping system can be developed alongwith parcel data information
* Combined features: mortgage calculator can be used to develop a prediction system

**Challenges:**

* Data Consistency
* Proxies and Server Costs

**Software skills required for the project:**

**1. Web Scraping (Scrapy):** Scraping real estate data from various sources to build and maintain a comprehensive database required expertise in web scraping techniques.

**2. API Development:** Developing APIs to access and serve real estate data to the client's website and for data selling purposes.

**3. Database Management (PostgreSQL):** Managing the database that stores the real estate data efficiently and securely.

**4. DevOps and Kubernetes:** Setting up and managing DevOps processes and Kubernetes for deployment and scaling of the applications and services.

**5. Cloud Computing (AWS, Azure):** Utilizing cloud platforms like AWS and Azure for hosting and managing the infrastructure and services.

**6. Proxy Management:** Developing a proxy manager to handle and rotate proxies for web scraping, ensuring continuous data collection.

**7. Data Migration:** Migrating the client's system from AWS to Linode while ensuring data integrity and minimal downtime.

**8. Backend Development (Django):** Developing the backend system for data processing, storage, and management.

**9. Frontend Development (ReactJS):** Creating the user interface and frontend components for the website or data visualization.

**10. Big Data Processing:** Handling and processing large volumes of real estate data, which falls under big data, and possibly using big data tools and frameworks for analysis.

**11. Elasticsearch, Logstash, Kibana (ELK):** Utilizing the ELK stack for logging, monitoring, and analyzing system and application data.

**13. Continuous Integration and Continuous Deployment (CI/CD):** Implementing CI/CD pipelines to automate the deployment process and ensure code quality.

**14. Notification Systems (Slack):** Integrating notification systems like Slack to receive alerts and reports on the status of scrapers and system health.

**15. Data Reporting:** Creating hourly reports on real estate data and sharing them through appropriate channels.

**16. Data Security:** Implementing security measures to protect sensitive real estate data and user information.

**Revelator**

**Problem Statement**

Development of Royalty Attribution Engine, that analyzes consumed data to distribute royalties to the artists according to the music consumption.

**Solution**

It was mainly an automation and scripting job done for music streaming platforms like Spotify, Apple Music etc. It would collect the number of times songs were streamed, collected all revenue for all streamed data and then it would distribute the total revenue generated among all parties involved. We also had the digital copies of artist’s contracts, hence revenue was divided according to percentages mentioned in the contracts. Everything was predefined and trailing balance adjustments features were developed. If song was a single, the revenue cut was sent to the artist according to his personal contract. However, if song was from an album of multiple artists, their contracts were studied and the cut was sent to all according to that.

Link for Info: <https://www.revelator.com/why-revelator/business-services/royalties-payments>

**Industry**

Music

**Tech Stack**

* Python
* Azure Platform API was used for fetching data from all platforms

Website: <https://www.revelator.com/>

Software Skills used:

1. Automation and Scripting: Automation scripts were used to collect data from music streaming platforms (e.g., Spotify, Apple Music) and perform calculations for royalty distribution.

2. Data Collection (Platform APIs): Utilizing platform APIs (Application Programming Interfaces) to fetch data from music streaming platforms. These APIs provide access to data on song streams, revenue, and other relevant metrics.

3. Data Analysis and Processing: Processing and analyzing the collected data to calculate royalties for artists and other parties involved. This likely involved data manipulation and calculations based on predefined rules and contract percentages.

4. Digital Contracts Management: Managing digital copies of artist contracts and using them as a basis for dividing revenue according to the contract terms and percentages specified therein.

5. Predefined Rules: Implementing predefined rules for distributing revenue, including adjustments for trailing balances, which may be necessary for certain contractual agreements.

6. Revenue Distribution Logic: Developing logic to distribute revenue differently based on the nature of the music, whether it's a single song or part of an album with multiple artists. Different contract terms would apply accordingly.

7. Programming Language (Python): Python was used as the primary programming language for developing the automation scripts and performing data processing tasks.

8. Cloud Platform (Azure): Utilizing Microsoft Azure as the cloud platform for hosting and managing the application or service responsible for processing and distributing royalties.

9. Third-Party Services (Revelator): Using third-party services like Revelator for managing royalties and payments, as indicated in the provided link.

This project involved a combination of software development, data analysis, and automation skills to create an efficient and accurate system for tracking music consumption and distributing royalties to artists and other stakeholders in the music industry.

**Nuitee (Handsfree Automation)**

**Problem Statement**

Development of a booking platform that generated definite profits

**Solution**

Online booking platforms like booking.com, all over the world they had listed properties and they have scrappers that check and maintain rates. When customer made bookings through any booking site, before confirming their booking, we checked, if cheaper room was available through nuitee and the profit was atleast 15% only then that room was booked through nuitee. However, if profit of 15% was not generated, the booking was automatically cancelled, and the customers were notified. For the amenities that were not an exact match, there was a manual dashboard to enter and search for such amenities.

**Industry**

Hotel Management, Hospitalilty

Customer Service

**Tech Stack**

* Python backend
* React was used at frontend
* hosting done on AWS. Scrapping

Website: <https://handsfreeautomation.com/>

Software Skills used:

In this project aimed at optimizing the hotel booking process and maximizing profit, a combination of software skills and technologies were utilized:

1. Web Scraping (Scrapping): Developing web scrapers to extract data from online booking platforms like booking.com. These scrapers were used to check and maintain property rates.

2. Python Backend: Implementing a Python-based backend to handle the business logic of checking and confirming bookings. This included checking for cheaper rooms through the Nuitee platform and ensuring a minimum profit margin of 15% before booking.

3. React Frontend: Developing a React-based frontend for user interaction, which likely included interfaces for monitoring bookings and managing amenities.

4. AWS Hosting: Utilizing Amazon Web Services (AWS) for hosting the application and its components, ensuring scalability and reliability.

5. Automated Booking and Cancellation: Developing automated processes to confirm bookings when cheaper rooms were available through Nuitee and cancel bookings if the desired profit margin was not met. Customers were also notified of these actions.

6.Manual Dashboard: Creating a manual dashboard for handling amenities that did not have an exact match, allowing users to enter and search for such amenities.

7. Nuitee Integration: Nuitee played a central role in the project, serving as the provider for cheaper room options and helping to maximize profit by ensuring bookings met a minimum profit margin of 15%.

8. Hotel Management and Hospitality: Implementing industry-specific features and logic related to hotel management and hospitality, including rate optimization and customer service.

Overall, this project required expertise in web scraping, backend development, frontend development, hosting on AWS, and industry-specific knowledge in hotel management and hospitality, with a critical focus on the role of Nuitee in the booking process optimization.

**Social Manager (Sparrow Charts)**

**Problem Statement**

Developing an application to filter out customers by giving them automated responses and recording their complaints so that service representatives can get back to them later.   A solution to fetch comments and messages on basis of their priority.

**Solution**

Brand would login to all their social media accounts on our platform, and we would instantly fetch all their posts and comments, providing them a single platform to manage everything. We would classify their comments based on their urgency and content like urgent, important, not important, positive, neutral and negative etc.

Website: <https://www.sparrowcharts.com>

**Industry**

Customer Service

Marketing

**Tech Stack**

* Tensorflow
* Twilio API
* Facebook API
* Twitter API
* Instagram API
* SwaggerUI
* Python in backend in Flask,
* NLP for analysis of comments
* React for frontend

**Software skills used:**

**1. Social Media Integration:** Creating a platform that allows brands to log in to all their social media accounts, including Facebook, Twitter, and Instagram. Integration with these social media platforms was essential to fetch posts and comments.

**2. Comment Classification:** Developing a system to classify comments based on various criteria such as urgency, content sentiment (positive, neutral, negative), and importance. This likely involved natural language processing (NLP) techniques for comment analysis**.**

**3. Automated Responses:** Implementing automated response functionality to provide instant replies to customers based on the classification of their comments. This helps in addressing urgent issues promptly.

**4. Complaint Recording:** Recording customer complaints and issues for later follow-up by service representatives. This ensures that important matters are not overlooked.

**5. Frontend Development (React):** Building the user interface using React for a user-friendly and responsive frontend.

**6. Backend Development (Flask):** Developing the backend of the application using Flask, a Python web framework. This backend is responsible for handling data processing, communication with APIs, and managing the application's core logic.

**7. Natural Language Processing (NLP):** Utilizing NLP techniques, possibly with the help of TensorFlow, to analyze the content and sentiment of customer comments.

**8. API Integration (Twilio, Facebook, Twitter, Instagram):** Integrating with various APIs, including Twilio for messaging and Facebook, Twitter, and Instagram APIs for fetching social media posts and comments.

**9. SwaggerUI:** Implementing SwaggerUI for API documentation, making it easier for developers to understand and use the APIs.

**11. Customer Service and Marketing:** Addressing customer service needs by developing a tool for managing customer interactions, and using marketing strategies to promote the platform to potential clients.

Overall, this project involved a blend of social media integration, NLP, API communication, frontend and backend development, and the creation of a user-friendly platform catering to customer service and marketing needs in the industry.

**SearchLook**

**Problem Statement**

Developing a system to provide background checks on individuals for potential hiring

**Solution**

We did multiple components for this client. We developed an end to end website for a company that provides background checks to employers for potential employees. If someone wants to hire a candidate, they request searchlook for their background check including their criminal history, debt record etc. We also developed and automated 100+ scrappers for a Mexico based company. Also generated timely automation reports and we’re continuously adding new features to their backend. Implemented a monitor for monitoring all scrappers. Crone jobs made sure scrappers work in order, starting them one by one every 1-2 hours.

**Industry**

Background Check

**Website:** <https://searchlook.mx/>

**Tech Stack**

* Python for development
* Flask for backend,
* Angular.
* Pipelines and Mturk were used for automation and scrapping
* AWS for Automation cloud.

**Software Skills used:**  
  
In this project aimed at developing a system for conducting background checks on individuals for potential hiring, a combination of software skills and technologies were employed:

1. Website Development: Creating an end-to-end website for a company specializing in background checks. The website likely provided a user-friendly interface for employers to request and access background checks on potential employees.

2. Background Check Automation: Developing an automated system for conducting background checks, including criminal history and debt records, on individuals. This involved the automation of data collection and analysis.

3. Web Scraping (100+ Scrappers): Building over 100 web scrapers to collect data, from various online sources or databases, to provide comprehensive background check reports.

4. Automation Reports: Generating automated reports to provide insights into the background check process, including the results of individual checks and the overall status of the automation system.

5. Backend Development (Flask): Developing the backend of the website using Flask, a Python web framework, to handle data processing, user requests, and interactions with the database.

6. Frontend Development (Angular and Flask): Using Angular for the frontend. This involved creating user interfaces for employers to request background checks and view results.

7. Automation Tools (Pipelines and Mturk): Utilizing automation tools and pipelines for efficient data collection and analysis. Amazon Mechanical Turk (Mturk) may have been used for tasks that required human input.

8.Cloud Computing (AWS): Using Amazon Web Services (AWS) to host and manage the automation system in the cloud, including the execution of cron jobs for scheduling tasks and ensuring timely scrapping.

9. Data Monitoring: Implementing a monitoring system to keep track of all scrappers and ensure they are functioning correctly. Monitoring tools would help identify and address issues in real-time.

10. Cron Jobs: Setting up cron jobs to schedule and manage the execution of scrappers at specific intervals (e.g., every 1-2 hours).

Overall, this project involved a combination of web development, automation, web scraping, and cloud computing to create a comprehensive system for conducting background checks on individuals, with a focus on efficient data collection and reporting for potential employers in various industries.

**Blockchain Jobs**

**In all the below stated applications these are the main applications and challenges**

**Applications**

Smart Contract Development is one function, which is audited by auditing firms. This is an alternative for DOWS, in which an idea is picted, asking for xyz amount of money till abc dats. If by then, whole amount is not used, it is sent back to the money’s original owners. Smart contract is made in solidity language. Whoever is maing the smart contact, enters complete information about the original owner and broker etc. Money is separately demanded for making the smart contract live.

**Challenges**

Once contract is live, new contract has to be made in case of changes, since modification is not allowed

Auditing firms need to make sure that no loopholes exist

**GatsbyNFT**

**Problem Statement**

Developed a platform for buying and minting NFTs, with pre-sale functionalities, compatible with open sea.

**Solution**

An end to end platform was developed where backend portal was given to client to upload photos and convert them to NFTs and the customers could mint the NFTs through the front portal. The platform was compatible with open sea, a marketplace for NFTs, the minted NFTs were automatically listed on open sea so the customers could buy the NFTs from their as well.

**Industry**

Finance, NFT

Stock

**Tech Stack**

* Backend: Node
* frontend was in React
* Ethereruem was Javascript library used to access the cyptocurrency wallet for purchasing of NFTs

**Website:** <https://www.gatsbynft.io/>

**Software Skills used:**  
  
**In this project focused on creating a platform for buying and minting NFTs with pre-sale functionalities and compatibility with OpenSea, the following software skills and technologies were employed:**

**1. Blockchain Integration (Ethereum):** The project required integrating with the Ethereum blockchain, which is a public blockchain often used for NFTs. The Ethereum JavaScript library was used to access cryptocurrency wallets for purchasing NFTs.

**2. Backend Development (Node.js):** Developing the backend of the platform, including functionalities for uploading photos and converting them into NFTs. Node.js was chosen as the backend technology.

**3. Frontend Development (React):** Building the user-facing frontend of the platform, where customers could mint NFTs. React was used to create the frontend interface.

**4. NFT Minting:** Implementing the functionality for customers to mint NFTs through the frontend portal. This likely involved interactions with the Ethereum blockchain to create and store NFTs.

**5. Pre-Sale Functionalities:** Developing features for pre-sale functionalities, which could include options for setting reserve prices or allowing early access for specific users or groups.

**6. OpenSea Compatibility:** Ensuring compatibility with OpenSea, a popular NFT marketplace. The minted NFTs were automatically listed on OpenSea, allowing customers to buy NFTs from there as well.

**7. Finance and Cryptocurrency Handling:** Handling financial transactions involving cryptocurrencies, including the purchase of NFTs. This required security measures to protect users' assets.

**8. Tokenization:** Tokenizing digital assets (e.g., photos) to create NFTs, which represent ownership of these assets on the blockchain.

**9. Smart Contracts:** Potentially using Ethereum smart contracts to manage NFT ownership and transactions securely**.**

**10. Web3.js:** Web3.js or similar libraries might have been used to interact with the Ethereum blockchain from the frontend.

**12. Testing and Security:** Ensuring the security of the platform by conducting testing, including audits of smart contracts and cybersecurity measures.

**14. NFT Metadata:** Managing metadata associated with NFTs, including images and descriptions.

**Overall, this project combined blockchain technology, backend and frontend development, cryptocurrency handling, and integration with a popular NFT marketplace (OpenSea) to create a comprehensive platform for buying and minting NFTs in the finance and NFT industry.**

**Snipping Bot**

**Problem Statement**

Developing an advanced cryptocurrency trading bot with real-time monitoring capabilities to seize trading opportunities in highly volatile markets by executing precise buy and sell orders based on price fluctuations. The bot should support various features, including sniping, multiple buy and sell options, approval mechanisms for trades, and multi-currency support. This was for client internal needs

**Solution**

Scripting job done where bot would be monitoring the drop of a coin and the bot would snipe the first block as soon as the price drops. Also, when the price even slightly hiked, fluctuation of price increased our investment by hundreds. We also added sniping function, multiple sell, multiple buy, approve before or after (Price check before selling), open trade buying and selling. First we used BNB currency then BUSD

**Industry**

Trading

**Tech Stack**

* Ethereum support
* lazy mint: different process for minting
* Javascript in Node
* EtherJS
* Solidity (blockchain language) used for development

**Applications:**

Property Transfer Record

**Proposed Solutions:**

Transparent Data Record Blockchain Solutions

Transaction history and Record is maintained

**Challenges:**

Delay in transactions so u would get space in second block instead of block 1 resulting in reduced profit, so threading and multiprocessing was done to counter this

**Software Skills used:**  
  
 **Scripting and Automation**: Proficiency in scripting languages and automation techniques to create and manage the trading bot's functionalities.

* **Cryptocurrency Trading Algorithms**: Knowledge of cryptocurrency trading algorithms and strategies to design the bot's trading logic and decision-making processes.
* **Real-time Data Analysis**: Skills in real-time data analysis to monitor cryptocurrency market data, assess price movements, and identify trading opportunities.
* **Smart Contracts (Solidity)**: Expertise in Solidity, a blockchain-specific programming language, to develop smart contracts that automate cryptocurrency transactions and trades.
* **Blockchain Interaction (EtherJS)**: Understanding of EtherJS, a JavaScript library for Ethereum, to interact with the Ethereum blockchain for executing trades and managing digital assets.
* **Backend Development (Node.js)**: Proficiency in Node.js for backend development to support the bot's operations, data processing, and communication with blockchain networks.

**Other Jobs**

**Oone**

**Problem Statement**

Developing a hassle free parking lot management system

**Solution**

We developed an app for parking spaces management. All parking spaces in parking lots have APIs that show available parking spaces from where you can book the space and pay for it too. The developed app showed list of open spaces, you could open the gate for the parking lot from this app.

Website: [www.oone.me](http://www.oone.me)

**Industry**

* **Automobile/Parking**

**Tech Stack**

* Flutter

**Software Skills used:**

**In this project aimed at developing a hassle-free parking lot management system, the following software skills and technologies were employed:**

**1.** Mobile App Development (Flutter): Creating a mobile app for parking space management using the Flutter framework. Flutter allows for cross-platform development, making the app accessible on both Android and iOS devices.

**2. Parking Space APIs:** Integrating with parking space APIs that provide real-time information about available parking spaces in parking lots. These APIs enable users to view and book parking spaces.

**3. Booking and Payment System:** Implementing a booking and payment system within the app, allowing users to reserve parking spaces and make payments for their bookings.

**4. Gate Control:** Enabling users to open the gate for the parking lot directly from the app, providing a seamless entry and exit experience.

**5. User Interface Design:** Designing a user-friendly interface for the app, ensuring that users can easily view available parking spaces and perform actions such as booking and gate control.

Overall, this project involved mobile app development, API integration, payment processing, gate control, and user interface design to create a hassle-free parking lot management system accessible via the Flutter-based mobile app and the website.

**ASR (Automated Speech Recognition)**

**Problem**

Developing an automated speech recognition application.

**Solution**

Deployed an audio to text conversion application that takes in the recording from a browser, sends it to the server in 5 second chunks, to be processed through an application at the backend, eventually transcribing the information on the screen

**Industry**

* AI

**Tech Stack**

* Flask for creating APIs
* HTML, CSS and Bootstrap for webpage designing
* Whisper-small OpenAI model used since application was lightweight, it also provided cleaning of data

**DocQA**

**Challenge**

Developing a system to extract information through questions from the link of an image or a document

**Solution**

(Mainly) Document questioning and answering system developed that when once the link of a document or image is uploaded, it extracts all the information from there and upon questioning, provides relevant answers.

For example, if you provide the link of an invoice and ask the invoice number, it takes less than 5 seconds and displays the invoice number on the screen

**Industry**

**Tech Stack**

Impira LLM Model mainly used with 217 other models. Some main models:

* [microsoft/HuggingGPT](https://huggingface.co/spaces/microsoft/HuggingGPT)
* [freddyaboulton/gradio\_pdf](https://huggingface.co/spaces/freddyaboulton/gradio_pdf)
* [taesiri/HuggingGPT-Lite](https://huggingface.co/spaces/taesiri/HuggingGPT-Lite)
* [huggingfacejs/doc-vis-qa](https://huggingface.co/spaces/huggingfacejs/doc-vis-qa)
* [Gokul14/impira-layoutlm-document-qa](https://huggingface.co/spaces/Gokul14/impira-layoutlm-document-qa)
* [king007/docquery](https://huggingface.co/spaces/king007/docquery)
* [Alfasign/HuggingGPT-Lite](https://huggingface.co/spaces/Alfasign/HuggingGPT-Lite)
* [kevinbednarek/docquery](https://huggingface.co/spaces/kevinbednarek/docquery)
* [TensoraCO/docquery](https://huggingface.co/spaces/TensoraCO/docquery)
* [aaravlovescodes/fastapi-document-qa](https://huggingface.co/spaces/aaravlovescodes/fastapi-document-qa)
* [vvmnnnkv/doc-vis-qa](https://huggingface.co/spaces/vvmnnnkv/doc-vis-qa)
* [Vito99/HuggingGPT-Lite](https://huggingface.co/spaces/Vito99/HuggingGPT-Lite)
* [Faizanshaikh/pdf-chatbot](https://huggingface.co/spaces/Faizanshaikh/pdf-chatbot)
* [AnshulVijay/impira-layoutlm-document-qa](https://huggingface.co/spaces/AnshulVijay/impira-layoutlm-document-qa)
* [dhruv1710/impira-layoutlm-document-qa](https://huggingface.co/spaces/dhruv1710/impira-layoutlm-document-qa)
* [Sobit/chat\_with\_document](https://huggingface.co/spaces/Sobit/chat_with_document)
* [sarim/documentOCR](https://huggingface.co/spaces/sarim/documentOCR)
* [warven/impira-layoutlm-document-qa](https://huggingface.co/spaces/warven/impira-layoutlm-document-qa)
* [jg7288/impira-layoutlm-document-qa](https://huggingface.co/spaces/jg7288/impira-layoutlm-document-qa)
* [NagasRepo/impira-layoutlm-document-qa](https://huggingface.co/spaces/NagasRepo/impira-layoutlm-document-qa)
* Flask on backend
* HTML, CSS and Bootstrap for webpage styling

**Trading Jobs**

**Here are some of the challenges from these Jobs**

**Challenges**

Data Availablity, Trading Strategy, Target Market (Crypto, FOREX, Stock), Well Defined Targeting Strategy

**Trading Bot**

**Problem**

Developing a trading system that would select, buy and sell stocks itself

**Solution**

An end-to-end trading system was developed based on the US stock market that would automatically select stocks and find the right time to buy and sell those stocks. Fundamental analysis was done on the stocks of S&P 500 index to filter out the fundamentally strong companies to create a dataset on which investment entry and exit were done based on technical indicators like DOW theory, Fibonacci, Moving Average and Trend Lines etc.

The back testing showed impressive results: in 3 and a half years, the results were 100% showing that the investment was successfully doubled in almost 4 years.

**Industry**

Finance

Stock Market

**Tech Stack Used**

GoLang for APIs, Python for data crunching, Javascript for frontend and Polygon.io for data gathering

# Project: Cursor SxS Human Eval

## Problem Statement

The client needed to evaluate the real-world performance of a coding AI tool called Cursor. They needed to assess its features like Composer and Agent in a way that mimicked how a developer would naturally use them.

This project was required to provide human feedback on Cursor's functionality.

## Solution

Performed a side-by-side human evaluation of Cursor’s features including Composer and Agent mode. Used the tool to interact with codebases, ask questions, make edits, and run terminal commands.

Received prompts and code repositories, allowed Cursor to index the codebase, used features like Composer, Agent, and Web Search to complete tasks.

## Industry

AI/ML Development, Software Development Tools, Human-in-the-Loop AI Evaluation.

## Tech Stack

JavaScript, Cursor, and command-line tools.

## Results

Provided critical human-driven data to the Cursor development team for tool improvement.

## Challenges

Mimicking natural workflow and managing multiple tools within Cursor.

## Challenge Solution

Followed protocol carefully and tested each feature systematically.

## Live Link

Not applicable.

**Project: Synthkit Human Evals V4**

## Problem Statement

The client needed to verify the quality of 'Augmented Code' generated by an AI model. The main concern was correctness and executability. To ensure AI-generated code is reliable before deployment or training.

## Solution

Human evaluation to measure correctness and executability of augmented code. Reviewed and ran code samples to check for correctness and errors, measured quality across various dimensions.

## Industry

AI/ML Development, Software QA, Human-in-the-Loop AI Training.

## Tech Stack

JavaScript, development/evaluation platforms.

## Results

Direct measure of AI model’s performance, enabling refinement.

## Challenges

Evaluating complex code consistently and accurately.

## Challenge Solution

Established clear criteria and applied them rigorously.

## Live Link

Not applicable.

# Project: SWEAP Simplified

## Problem Statement

The client needed to validate test suite modifications for GitHub 'SWE problems.' To ensure tests were robust enough for automated validation of future solutions.

## Solution

Reviewed GitHub issues, code changes, and tests to validate their suitability.

## Steps

Analyzed issue descriptions, code changes, and new tests; checked adequacy and coverage.

## Industry

AI/ML Development, Automated Software Testing, QA.

## Tech Stack

JavaScript, GitHub, test frameworks.

## Results

Curated high-quality dataset of verified tests for training/evaluation of AI models.

## Challenges

Understanding complex issues and ensuring test comprehensiveness.

## Challenge Solution

Thoroughly analyzed problems and critically evaluated tests.

## Live Link

Not applicable.

# Project: Creating Ideal Code & Unit Tests from EduArena

## Problem Statement

The client needed high-quality 'Ideal Code' and 'Ideal Unit Tests' for failed model prompts.

To create reliable training data for RLEF (Reinforcement Learning from Human Feedback).

## Solution

Created ideal JavaScript solutions and unit tests for EduArena prompts. Checked prompt relevance, wrote ideal solution and unit test, updated function signatures, validated tests.

## Industry

AI/ML Development, Educational Technology, Human-in-the-Loop AI Training.

## Tech Stack

JavaScript, EduArena, test environments.

## Results

Produced a valuable dataset for improving AI coding tasks.

## Challenges

Writing robust code and tests for problems that stumped models.

## Challenge Solution

Meticulous coding and writing thorough unit tests covering all edge cases.

## Live Link

Not applicable.

# Project: Open Evals - Challenging Coding Questions on EduArena

## Problem Statement

The client needed a challenging bank of coding questions to compare AI models. To create a rigorous benchmark for AI models in educational tech.

## Solution

Created mid-to-advanced computer science questions on EduArena.

## Steps

Designed algorithm/data structure questions, used them to compare models.

## Industry

AI/ML Development, Educational Technology, AI Model Evaluation.

## Tech Stack

JavaScript, EduArena, coding environments.

## Results

Created a valuable benchmark bank of coding questions.

## Challenges

Designing diverse, solvable but challenging questions.

## Challenge Solution

Applied deep CS knowledge and creativity to design questions.

## Live Link

Not applicable.

**STS Marine Solutions (STSMS)**

**Industry:** Marine Solutions

**Project Overview**

The STSMS project was designed to provide customers with **real-time tracking of their cargo** through an intuitive and user-friendly dashboard. The company’s objective was to enhance customer satisfaction by giving full visibility of cargo status during transport on large vessels.

**Solution**

We developed a **multi-tiered system** that included:

* **Customer Dashboard:** Monitor cargo in real-time with clear status updates.
* **Web Portal for Operators:** STSMS staff can send updates efficiently to customers.
* **Mobile Application for Mooring Masters:** A lightweight app with an offline database to store updates during poor connectivity; updates are automatically synced once the connection is restored.
* **Automated Notifications:** Email and in-app alerts for customers, operators, and Mooring Masters about important events.
* **Chat Feature:** Enables real-time communication between customers and operators for quick issue resolution.

The system ensures that all stakeholders remain informed, improving operational efficiency and customer experience.

**Technologies Used**

* **Frontend:** React.js
* **Backend:** Django

**Results**

The STSMS project delivered a **fully functional, real-time cargo tracking system** that significantly improved customer satisfaction by providing transparency. It also gave STSMS a comprehensive view of all active jobs, enhancing operational oversight and workflow efficiency.

**Client**

**STS Marine Solutions (STSMS):** A global leader in marine services with over 40 years of experience in energy transfer solutions.

Live Link: <http://173.249.57.177:8502/>

**Haeussler Queue Management System**

**Industry:** Medical Supplies

**Project Overview**

Haeussler required a comprehensive **queue management system** to streamline customer handling at their medical supply store. The goal was to efficiently issue tokens to incoming customers, assign them to the appropriate employees, and allocate consultation rooms based on real-time availability, all while maintaining detailed tracking of customer interactions and internal workflows.

**Solution**

We developed a **web-based queue management application** with two primary modules: **In-store Operations** and **Production OT Management**.

**1. In-store Module**

The in-store module focuses on improving customer experience and employee efficiency. Key features include:

* **Ticketing System:** Issue tokens to customers upon arrival.
* **Employee Assignment:** Automatically or manually assign customers to employees for assistance.
* **Room Allocation Dashboard:** Real-time tracking and allocation of consultation rooms.
* **Queue Management:** Monitor current queues and wait times efficiently.
* **Feedback Collection:** Capture customer feedback at checkout for service improvement.
* **Time Tracking:** Record timestamps for ticket creation, reception, and checkout for detailed analytics.

**2. Production OT Module**

The Production OT module manages the internal workflow of Haeussler’s product handling and operational tasks. It ensures that orders move efficiently through various stages of production. Features include:

* **Client Ticket Management:** Log client requests with product details, deadlines, and ERP identifiers.
* **Workflow Tracking:** Monitor progress across multiple stages including cutting, assembly, testing, and quality checks.
* **To-Do Management:** Create and assign tasks to employees, ensuring accountability and efficient task tracking.
* **Dashboard & Progress Logs:** Provide a centralized view of ongoing tasks, process statuses, and employee assignments.
* **Internal Communication:** Helps employees stay on top of priorities and ensures smooth collaboration.

**Challenges**

* Tracking employee assignments and task completion was initially difficult.
* Implementing a clear **to-do system** resolved these issues, improving accountability and workflow transparency.

**Technologies Used**

* **Frontend:** Vue.js, Tailwind CSS
* **Backend:** Django (Python), JavaScript

**Results**

The Haeussler Queue Management System successfully delivered:

* A **real-time token issuance system** to improve customer handling.
* Efficient **employee allocation** and **room booking** based on availability.
* A robust **internal workflow management module** for production tasks.
* Enhanced operational efficiency and **improved customer service quality**.

Live Link: <http://173.249.57.177:6502/>

**Ninja Store(Greece Client)**

**Industry:** Ecommerce

**Project Overview:**

Ninja Store is a store on the Skroutz.gr to sell the products online. In this project, client wants to automate the whole process of listing, price updating, invoice automation etc. Also include the scraping of products from other stores for listing purpose.

**Solution:**

An admin panel was developed to manage the products, distributors, store status and invoices.

The admin panel allows the client:

* To manage the products
* To manage the product distributors.
* Upload the CSVs to make the invoices.
* Change the store status.

**Technologies Used:**

**Backend:** Python

**Results:**

The project is successfully fulfilling the requirements. We can extract the new products through scraping, get the listing from the distributor’s website through automation. Make the invoices through the CSV file, updating the price of products by comparing with the competitors at skroutz.gr, automatically making the XML file of products and saving it according to the status of store.

**Client:**

Chris(Greece).

**Square Up**

**Industry:** Beauty Saloon and Spa

**Project Overview:**

To extract the information regarding the sales of each employee through the square-up APIs.

**Solution:**

*[Not Documented Yet]*

**Technologies Used:**

**APIs:** Square up APIs

**Script:** Python

**Results:**

*[Not Documented Yet]*

**Client:**

*[Not Documented Yet]*

**GameXPanel (Streaming API)**

**Industry:** Live Streaming of Games and Channels

**Project Overview:**

GameXPanel is a CMS for managing the live games and Live Channels. Client wants to get a system that will manage the all live games, channels, subscribers, devices, CDNs, websites, Teams, etc. He wants the secure URL to be generated through different CDNs such as Akamai, GCore, and Fastly.

**Solution:**

An admin panel was developed to manage the live games, channels, subscribers and all other requirements.

**The admin panel allows the client:**

· To manage the Livegames, channels, subscribers, sites, authentication tokens, CDNs

· Sites API – It returns the all associated resources, like livegames and channels with any site through specific CDN

· Integrate the Akamai, Fastly and GCore CDNs to provide the secure stream URL.

· BetsAPI is used to dump our DB with the Livegames and their Teams.

**Technologies Used:**

**Backend:** Python

**APIs:** BetsAPI

**DB:** Postgres

**Results:**

The project is successfully fulfilling the requirements. A CMS is provided in the form of an Admin Panel. All the live games are successfully coming through the API and being dumped in our database, also they are automatically associated with the sites. On hitting the API, the stream\_url is converted into the secure\_stream\_url through the requested CDN.

**Client:**

*Company Name Not Known. Done for a Pakistani Client (Rana Hammad)*

**Ford Dealership**

**Problem Statement**

Ford dealerships needed a centralized system to manage vehicle inventory and dealership information across multiple locations. Manual processes for updating availability, managing sales records, and tracking dealerships were slow, fragmented, and error-prone**,** making it difficult to deliver a smooth customer experience.

**Solution**

I developed a **multi-location dealership management platform** with the following key features:

* **Inventory & Dealership Data Import** – All dealership and vehicle inventory data was originally maintained in **Google Spreadsheets**. This data is fetched into the database automatically, and a cron job runs every 12 hours to keep the platform updated.
* **Inventory Management** – Manage vehicle data (e.g., Ford F-150, Bronco Sport, Escape), availability, and sales records in real time.
* **Dealership Map Integration** – An interactive map displays all dealership locations, allowing users to search by ZIP code to find the nearest dealership.
* **Multi-Location Access** – Each dealership has a dedicated page (e.g.,  
    
  + https://localford.store/tricities
  + https://localford.store/louisville
  + https://localford.store/lexington
  + https://localford.store/bluefield
  + https://localford.store/charleston
  + https://localford.store/evansville)
* **Search & Filtering** – Users can filter inventory by model, availability, and location.
* **Reporting Tools** – Admin dashboards with inventory and dealership reports.
* **Footer Dealership Listings** – All dealerships are listed in the footer with name, address, phone number, and Google Maps link for easy navigation.

**Industry**

Automotive / Dealership Management

**Tech Stack**

* **Frontend:** Vue.js
* **Backend:** Django REST Framework
* **Database:** PostgreSQL
* **APIs:** Vehicle Information APIs & Dealership Data Integration
* **Automation:** Cron jobs for Google Sheets sync

**Results**

* Streamlined inventory management across multiple dealership locations.
* Reduced manual errors by introducing an automated, API-driven sync system.
* Improved customer experience with faster search and accurate dealership details.
* Enhanced data accessibility for dealership staff and customers.

**Challenges Faced**

* **Synchronizing Data Across Dealerships:** Solved by API-driven synchronization and background tasks (scheduled jobs).
* **Location-Specific Customization:** Each dealership URL dynamically pulls localized data while maintaining a consistent UI/UX.

**Live Link:**

<https://localford.store/>

**Software Skills Used**

* **Frontend:** Vue.js (UI, map integration, search & filter components)
* **Backend:** Django REST Framework (API design, data sync)
* **Database:** PostgreSQL (schema design for inventory, dealerships)
* **Integration:** API integration, Google Maps API, workflow automation

**Automated Calendly Follow-Up Calls & Reschedule Alerts**

**Problem Statement**

High no-show rates and last-minute cancellations reduce meeting efficiency for sales teams. Manually confirming attendance, resending scheduling links, and validating lead information is time-consuming and often leads to missed opportunities.

**Solution**

I built an automated scheduling assistant using n8n, Calendly API, Vapi.ai, and OpenAI that proactively manages follow-ups and reschedules.

Key features include:

* **Automated Meeting Scan** – Every 15 minutes, the system checks all upcoming Calendly meetings within 24 hours.
* **AI-Powered Voice Calls** – Uses **Vapi.ai + GPT-4o** to call invitees and confirm if they will attend, want to cancel, or need to reschedule.
* **Reschedule Handling** – If a reschedule is requested, both client and lead receive an email with a fresh Calendly link automatically.
* **Website Validation** – Verifies the lead’s submitted website; if invalid/unreachable, a manual verification alert is sent to the client.
* **Smart Decision Logic** – AI interprets responses from voice calls and routes workflows accordingly.

**Industry**

* Scheduling Automation
* Client Management

**Tech Stack**

* **Automation:** n8n
* **Scheduling:** Calendly API
* **AI Voice Calls:** Vapi.ai + GPT-4o
* **Emailing:** Gmail API
* **Integration:** JavaScript, JSON, HTTP Requests

**Results**

* Reduced no-show rates through automated pre-meeting confirmations
* Reschedule alerts instantly sent with new booking links
* Real-time website validity checks to catch incorrect lead submissions
* AI-powered decision-making ensures accurate call outcome handling
* Complete hands-off scheduling oversight, running every 15 minutes

**Challenges Faced**

* **AI Call Accuracy:** Tuned GPT-4o prompts to interpret diverse responses reliably.
* **Time-Sensitive Execution:** Used n8n cron scheduling for **frequent 15-min scans** without missing updates.
* **Error Handling:** Built retries and manual alerts for unreachable websites or failed call outcomes.

**Software Skills Used**

* n8n workflow automation
* API integration
* OpenAI GPT-4o prompt engineering

**Auto-Cancel Invalid Calendly Leads with Email Alerts**

**Problem Statement**

Calendly meetings often attract unqualified leads—such as those using fake emails or outside the target region. Manually reviewing and cancelling these events is time-consuming, leading to wasted time on calls that don’t meet business requirements.

**Solution**

I created an automated lead validation workflow using n8n, Hunter.io, and Calendly’s API to automatically cancel unqualified events before they consume time.

Key features include:

* **Email Validation with Hunter.io** – Each new Calendly booking triggers a validation check of the invitee’s email.
* **U.S. Citizenship Check** – Ensures only **U.S.-based leads** are allowed through.
* **Auto-Cancellation via Calendly API** – If the email is invalid or the lead is non-U.S., the event is cancelled instantly using:  
   https://api.calendly.com/scheduled\_events/{uuid}/cancellation
* **Email Notification Alerts** – In case of cancellation (e.g., invalid email), the system sends you a **detailed Gmail alert**.

**Industry**

* Scheduling Automation
* Lead Qualification

**Tech Stack**

* **Automation:** n8n
* **Scheduling:** Calendly API
* **Validation:** Hunter.io API
* **Emailing:** Gmail API

**Results**

* Automatically **filters out invalid or unqualified leads**
* Instantly **cancels unwanted meetings** in Calendly
* Sends you **cancellation alerts** for transparency
* Saves hours of manual review, keeping calendars clean
* Fully automated, no manual intervention needed

**Challenges Faced**

* **Accurate Email Validation:** Integrated Hunter.io with retry logic for reliable results.
* **Error Handling in Cancellations:** Managed failed API calls with backup notifications.
* **Regional Filtering:** Added conditional workflows to flag and cancel non-U.S. leads.

**Software Skills Used**

* n8n workflow automation
* Calendly API integration
* Hunter.io email validation
* automated lead qualification

**Auto-Sync LinkedIn Leads to Airtable from Google Sheets**

**Problem Statement**

Sales and outreach teams often struggle with manual lead tracking across multiple tools. Accepted LinkedIn connections usually require someone to manually cross-check spreadsheets and update CRMs, which is time-consuming, error-prone, and inefficient.

**Solution**

I developed an automated lead sync workflow using n8n, HeyReach, Google Sheets, and Airtable that keeps CRMs updated in real time without manual effort.

Key features include:

* **Webhook Trigger via HeyReach** – Every time a LinkedIn connection is accepted, a webhook notifies n8n.
* **Google Sheets Validation** – The workflow checks a Google Sheet of pre-existing customer data and matches leads by **LinkedIn profile URL**.
* **Airtable Sync** – If a match is found, the lead’s details (name, email, company, blog summary) are **instantly pushed to Airtable**.
* **Real-Time Updates** – Ensures that CRM records are automatically enriched and stay accurate.

**Industry**

* Sales Automation
* CRM
* Lead Generation

**Tech Stack**

* **Automation:** n8n
* **Lead Capture:** HeyReach (LinkedIn webhook)
* **Data Source:** Google Sheets
* **CRM:** Airtable
* **Integration:** Webhooks, APIs

**Results**

* Seamless **syncing of LinkedIn leads** with Airtable when connections are accepted
* **Instant validation and matching** against existing Google Sheets data
* Elimination of manual entry from spreadsheets to CRM
* Streamlined CRM population for **sales and outreach teams**
* Fully automated, **scalable workflow** triggered by HeyReach

**Challenges Faced**

* **Lead Matching Accuracy:** Implemented profile URL-based matching to ensure precise validation.
* **API/Webhook Reliability:** Built retries and error handling into n8n to avoid data loss.
* **Scalability:** Optimized workflow to handle bulk LinkedIn connection activity efficiently.

**Software Skills Used**

* n8n workflow automation
* HeyReach webhook integration
* Google Sheets API
* Airtable API
* CRM data synchronization

**Automated Blog Post Generation & posting to Wordpress**

**Problem Statement**

Marketers, niche site owners, and SEO agencies often spend significant time and resources on blog content creation researching topics, writing, formatting, sourcing images, and publishing. These manual workflows are time-consuming, repetitive, and prone to inconsistency.

**Solution**

I built an end-to-end content automation system using Make.com, OpenAI, and WordPress that converts a list of keywords into SEO-ready blog posts with zero manual effort.

Key features include:

* **Keyword Input via Google Sheets** – Keywords are added to a shared sheet as the starting point.
* **Automated Blog Generation** – For each keyword, the system generates a 7-heading outline and a 1500–2000 word article in a natural, human tone.
* **AI Image Creation** – Two relevant images are created with DALL·E 3 to enhance visual appeal.
* **Google Docs Integration** – Drafts are automatically uploaded to Google Docs for easy review and editing.
* **One-Click WordPress Publishing** – Finalized posts (text + images) are auto-published on WordPress via HTTP API.

**Industry**

* Content Marketing
* Digital Publishing

**Tech Stack**

* **Automation:** Make.com
* **AI Content:** OpenAI GPT-4o
* **AI Images:** DALL·E 3
* **Storage/Docs:** Google Sheets, Google Docs
* **Publishing:** WordPress (HTTP APIs)

**Results**

* Fully automated content creation pipeline from keyword → blog post → published article
* SEO-optimized, high-quality blog posts written in a human tone
* AI-generated images included forbetter engagement
* Seamless upload to Google Docs for collaboration and review
* Instant publishing to WordPress for immediate visibility

**Challenges Faced**

* **Maintaining Content Quality:** Balanced AI generation with structured outlines to ensure SEO and readability.
* **API Coordination:** Integrated multiple APIs (OpenAI, DALL·E, WordPress, Google) smoothly through Make.com workflows.
* **Scalability:** Optimized workflow to handle bulk keyword batches without timeouts or failures.

**Software Skills Used**

* Make.com workflows
* API integration
* OpenAI GPT-4o
* DALL·E 3 image generation
* Google Workspace automation (Sheets/Docs)
* WordPress publishing

**AI-Driven Workflow for Enhancing Sales Performance with HubSpot  
  
1. Problem Statement**

* **Client/User Problem:**Scott’s Directories had very low demo show-up rates, a lot of time wasted on fake/unqualified leads, and heavy manual workload on their sales team. Reps lacked proper pre-call context, which reduced conversion chances.
* **Why Project Was Needed:** To automate lead qualification, improve sales efficiency, and increase demo attendance while reducing manual effort from reps and Emma (sales ops).

**2. Solution**

* **What We Built:** An **AI-powered automation system** integrated with HubSpot that:  
    
  1. Analyzes leads in real time when forms are submitted
  2. Calls leads within 1–2 minutes for quick screening
  3. Uses **AI Voice Agents** to pre-qualify and log notes in HubSpot
  4. Transfers qualified calls live to sales reps or assigns them in round robin
  5. Sends immediate notifications and demo links to both reps and leads
  6. Automates nurture campaigns and retries if leads don’t answer
* **Step-by-Step Flow:**
  1. Lead submits form → AI validates email domain (personal vs business).
  2. AI calls lead instantly → asks 4–6 screening questions (intent, goals, company).
  3. If qualified → live transfer to rep or HubSpot assignment with notifications.
  4. If not qualified → AI ends the call politely, logs notes, and notifies Emma + Lori.
  5. If unanswered → retry schedule for 1 week with smart intervals.
  6. When demo booked → AI sends **pre-demo nurture sequences** (case studies, sample audits, questions).

**3. Industry**

* B2B Lead Generation & Sales Automation
* CRM & AI Workflow Automation (HubSpot + VOIP systems)

**4. Tech Stack**

* **CRM:** HubSpot (forms, workflows, email, lead management)
* **AI Voice Agent:** Vapi (for real-time AI calling & transfers)
* **Automation Platform:** n8n (for webhooks, routing, retries, and notifications)
* **Enrichment Tools:** Clay API (LinkedIn enrichment, company validation)
* **VOIP/Call Routing:** Multi-rep queue with simultaneous ring/transfer logic

**5. Results**

* Eliminated manual lead screening for reps and Emma.
* Improved demo show-up rates with pre-demo nurture sequences.
* Faster response time: leads contacted within 1–2 minutes of form submission.
* Reps now receive **context-rich notes in HubSpot** before talking to leads.
* Improved accuracy in identifying fake/unqualified leads.

**6. Challenges Faced**

* **Problem:** AI couldn’t detect which rep answered during live transfers.  
    
  + **Solution:** Send HubSpot notifications + email with lead data to all reps. Rep who picks up assigns lead to themselves in HubSpot.
* **Problem:** Avoiding over-calling while still following up effectively.  
    
  + **Solution:** Smart retry schedule (30 min, 2h, etc.) capped at 1 week.
* **Problem:** Handling reps out of office.  
    
  + **Solution:** Dynamic round robin assignment that removes unavailable reps.

**7. Software Skills Used**

* HubSpot CRM (forms, workflows, email automation)
* Vapi (AI voice assistant & call handling)
* n8n (workflow automation, webhooks, routing)
* Clay API (LinkedIn/company enrichment)
* VOIP/Telephony systems (rep call queue + transfers)

**AI Trading Bot**

**1. Problem Statement**

* The client needed an **automated trading bot** that could analyze **real-time stock market data** and execute trades based on user-defined strategies.
* Manual testing of strategies was inefficient, so a system was required to **test, optimize, and automate strategies**.

**2. Solution**

* Built a **FastAPI backend** that fetches **real-time market data** using the Polygon API.
* Designed a strategy execution engine where the bot checks stock performance in **user-defined time windows (e.g., 15 minutes)**. If stock acceleration exceeds the trigger threshold, a trade is executed.
* Implemented a **backtesting system** that allows users to test strategies on historical data (up to 1.5 years).
* Optimized backtesting by processing **day-by-day data** instead of loading full datasets at once, preventing RAM overload.
* Preserved trade continuity across days to simulate real trading conditions.
* Integrated **Celery jobs** to keep frontend testing sessions alive without timeouts.

**3. Industry**

* **Finance / Algorithmic Trading / Stock Market Automation**
* Extendable to **cryptocurrency trading**.

**4. Tech Stack**

* **Backend:** Python, FastAPI, Celery
* **Data Source:** Polygon API (real-time & historical stock data)
* **Testing Infrastructure:** Custom backtesting engine with interval-based analysis
* **Frontend:** using Dash(Python lib)

**5. Results**

* Automated trade execution system working on real-time data.
* Users can **test strategies efficiently on large datasets** without performance bottlenecks.
* Resolved **time zone mismatches** (Polygon ET → U.S. market hours).
* Improved system scalability by handling data in **smaller chunks (daily intervals)**.

**6. Challenges Faced**

* **Time zone handling:** Stock exchanges run in different market hours. Polygon API returned ET → needed conversion to U.S. market hours.
* **Large dataset processing:** One-year data backtests overloaded memory. → Solved by breaking it down into daily intervals with trade carryover.
* **Session timeout:** Long strategy tests caused timeouts. → Fixed using **Celery background workers**

**7. Live Link**

* <http://trading.coldev.com>

**8. Software Skills Used**

* Python
* FastAPI
* Dash
* Celery (task scheduling & background jobs)
* Polygon API integration
* Real-time data processing
* Algorithmic trading logic & backtesting
* API deployment & optimization

**Project Documentation: Podcast Automation**

**1. Problem Statement**

* **What problem did the client or user face?** The podcast team was struggling with a highly manual process for managing podcast guest leads. This included repetitive back-and-forth communication, inconsistent confirmations, missed follow-ups, and lack of structured tracking. These inefficiencies caused dropped bookings, no-shows, and limited guest engagement.
* **Why was this project needed?**To automate the entire guest journey from initial interest to post-podcast sales calls so the client’s team could focus on high-value conversations instead of repetitive scheduling and nurturing tasks.

**2. Solution**

* **What exactly did we build ?** We created an **end-to-end AI-powered automation flow** that manages podcast leads at every stage:  
    
  1. **AI Email Agent** to handle initial inquiries, FAQs, and bookings via HubSpot.
  2. **Pre-Podcast Nurture Sequence** with AI-driven personalized emails.
  3. **AI Voice Agent Confirmation Calls** before the recording to reduce no-shows.
  4. **Post-Podcast AI Growth Call** engagement sequence to encourage sales follow-through.
* **How did we solve the problem step by step?**
  1. Set up **n8n automation flows** to connect lead sources, HubSpot, and communication channels.
  2. Integrated **Vapi for AI voice calls** to confirm podcast bookings and Growth Calls.
  3. Built email and voice **AI agents** with custom prompts for FAQ handling, nurturing, and engagement.
  4. Used **Google Sheets** to track lead call status and engagement.
  5. Implemented fallback escalation to human team members if AI could not resolve queries.

**3. Industry**

This project belongs to the **Podcasting and Media** industry with applications in **marketing, sales enablement, and guest management automation**.

**4. Tech Stack**

* **AI Calling Agent:** Vapi
* **Automation Flow:** n8n
* **CRM & Scheduling:** HubSpot (Calendar integration via n8n form)
* **Data Storage & Tracking:** Google Sheets
* **Lead Source:** Instantly
* **AI Agents:** LLM-based Email and Voice agents

**5. Results**

* Reduction in manual email and scheduling work.
* Improved **guest show-up rates** through AI confirmation calls.
* **Faster response times** to guest inquiries (AI replies instantly instead of waiting for human response).
* Improved **post-podcast engagement**, leading to higher Growth Call bookings.
* Clear **tracking of lead journeys** via Google Sheets, eliminating data gaps.

**6. Challenges Faced**

1. **Missed follow-up calls due to exact 24-hour window**
   * *Solution:* Added a 5-minute buffer window before and after 24 hours.
2. **AI LLM misclassifying positive/negative intent**
   * *Solution:* Optimized prompts with scenario-specific fine-tuning.
3. **No tracking of call status** (who received Growth/Podcast calls)  
     
   * *Solution:* Logged all calls into Google Sheets and updated call\_status as "succeeded".
4. **HubSpot lacked rescheduling link for meetings**
   * *Solution:* Built a custom rescheduling workflow using **n8n form + Google calendar integration**.  
     .
5. **Duplicate guest records in google sheets**
   * *Solution:* Created a **deduplication rule in n8n flow** to avoid duplication of data in google sheets.

**8. Software Skills Used**

* AI Conversational Design (Email & Voice agents)
* Automation Workflow Design (n8n)
* CRM Integration (HubSpot)
* Lead Source Integration (Instantly)
* Data Management & Tracking (Google Sheets)
* Prompt Engineering for Intent Classification
* Error Handling & Escalation Flows

**Lexgro – Law Firm Case & Lead Management System**

**Problem Statement**

The law firm needed a centralized system to manage incoming leads, active legal cases, and performance analytics. Existing workflows relied on scattered spreadsheets and manual updates, making it difficult to track case progress, monitor KPIs, and make data-driven decisions. Handling large volumes of sensitive legal data also required a secure and role-based access system.

**Solution**

I developed a comprehensive lead and case management platform that combines secure authentication, case tracking, and powerful analytics dashboards.

Key features include:

* **Lead & Case Management** – End-to-end system for capturing, assigning, and tracking legal cases.
* **JWT Authentication** – Secure login and session management with role-based access control to ensure confidentiality.
* **Analytics Dashboards** – Interactive visualizations to provide real-time insights:  
    
  + **Total Leads (Current Month)**
  + **Total Cases (Current Month)**
  + **Average CPA (Cost Per Acquisition) – Current Month**
  + **Monthly Lead Trends**
  + **Best Performing Source**
  + **Most Common Case Type**
  + **Assigned Cases**
  + **Revenue by Source**
  + **Cost per Acquisition by Source**
  + **Total Cases Per Month**
  + **Leads by Status**
  + **Lead Source Monthly Comparison**
  + **Cost per Lead by Source**

* **Data Visualization** – Implemented using **Chart.js** with responsive, interactive charts for better usability.
* **Security Measures** – Data encryption for sensitive case files and activity logging for compliance.

**Industry**

* Legal
* Data Analytics

**Tech Stack**

* **Backend:** Django (REST APIs, JWT authentication)
* **Frontend:** React.js (UI, dashboards, chart rendering)
* **Database:** PostgreSQL (structured case & lead storage)
* **Data Visualization:** Chart.js (charts, graphs, trend analysis)

**Results**

* Improved case tracking and lead management with structured workflows
* Enhanced decision-making using real-time dashboards and analytics
* Increased data security with JWT authentication and role-based access control
* Provided clear visibility into marketing ROI through CPA and lead source analysis

**Challenges Faced**

* **Handling Sensitive Legal Data** – Solved with role-based access control, JWT authentication, and encryption.
* **Scalable Dashboard Design** – Optimized queries and caching for smooth rendering of large datasets in charts and tables.
* **Integration Across Sources** – Built APIs to unify lead data from multiple sources into a single dashboard.

**Live Link**

<https://dashboard.lexgro.com/>

**Software Skills Used**

* Web application development (Django + React.js)
* Secure authentication with JWT
* Data visualization using Chart.js (bar charts, pie charts, line graphs, tables)
* Database design for **case, lead, and financial tracking**
* Workflow automation and analytics dashboard design

**Vapi AI Assistants**

**Problem Statement**

Businesses needed AI-powered call assistants to handle both inbound and outbound calls.Traditional manual calling processes were inefficient, slow to scale, and often failed to provide consistent customer experiences.

**Solution**

I built two types of **AI call assistants** using **Vapi.ai and Twilio API**:

1. **Outbound AI Assistant**
   * Purpose: Reach out to prospects and introduce Sortsy.ai.
   * Capabilities:  
       
     + Calls new leads automatically.
     + Gives a brief introduction of Sortsy.ai..
     + Engages prospects to measure their level of interest.
     + If a prospect shows interest, their details are synced into the CRM for follow-up by the sales team and try to book meetings.
2. **Inbound AI Assistant**
   * Purpose: Act as a virtual receptionist for incoming calls.
   * Capabilities:  
       
     + Answers queries about services.
     + Asks detailed discovery questions to understand customer needs.
     + If qualified, offer to book a meeting/appointment directly on the client’s calendar.
     + Syncs collected data (customer intent, preferences, contact info) into the CRM for future tracking.

**Industry**

* AI / Sales Enablement
* Customer Support

**Tech Stack**

* **AI Calls:** Vapi.ai
* **Telephony:** Twilio API
* **Integrations:** CRM systems (via APIs & webhooks), Google Sheets

**Results**

* Automated client communication through inbound & outbound calls
* Reduced response time for incoming leads
* Increased outreach efficiency with outbound Sortsy.ai introduction calls
* Improved appointment booking rates with smart inbound call handling

**Challenges Faced**

* **Accents & Natural Conversation Variability** – Solved with Vapi’s AI tuning, prompt optimization, and fallback responses.
* **Data Sync Across Systems** – Implemented reliable CRM integrations to avoid data loss between calls and lead management systems.

**Software Skills Used**

* Conversational AI design (inbound & outbound flows)
* Vapi.ai assistant configuration & tuning
* Twilio telephony API
* CRM automation & lead tracking integrations

**1.Project: Sortsy**

**Problem Statement**

People often struggle with keeping their files and folders organized. Documents pile up across devices, and finding the right file at the right time becomes frustrating. Traditional manual organization by type, date, or project requires time and discipline, which most users lack.

**Solution**

Sortsy is an AI-powered desktop tool that automatically organizes files and folders into structured formats. Whether users want their data sorted by date, file type, project, or client—or even if they don’t know how to structure it—Sortsy intelligently creates an organized system using Nexa LLMs. It also integrates vector search to make file retrieval quick and context-aware.

**Industry**

Productivity Software / File Management Tools

**Tech Stack**

* Desktop: Tkinter (Python)
* Backend: Django REST Framework (DRF)
* Frontend: Next.js
* Database: Vector DB for intelligent search
* AI: Nexa LLMs for contextual file structuring

**Results**

* Automated file and folder organization for end-users.
* Reduced time spent manually searching or arranging files.
* Intelligent search allows users to locate files based on purpose and context rather than just file names.
* Improved productivity and reduced clutter across personal and professional use cases.

**Challenges Faced**

* Designing a universal structuring logic that works across different user needs.
* Ensuring the AI correctly interprets ambiguous file contexts.
* Balancing speed and accuracy while handling large file sets.
* Building an intuitive interface for a technically complex backend.

**Live Link**

https://app.sortsy.ai/

**Software Skills Used**

* Full-stack development (Python, Django REST, Next.js)
* Desktop app development (Tkinter)
* AI & Machine Learning integration (Nexa LLMs, Vector DB)
* Intelligent search optimization
* API design & integration

**2.Project: Turing – AI Model Evaluation**

**Problem Statement**

AI models often generate multiple answers for the same question, but it’s not always clear which one is better. There needs to be a proper way to check the responses for correctness, instruction following, and security before choosing the best one.

**Solution**

I worked on the **Turing project** where my job was to evaluate AI model responses. For every task, I had to compare outputs from Model A and Model B and decide which one was better. I checked different points like: did the model follow the instructions, was the answer correct, was it safe, and how well it was written. If both responses were wrong or incomplete, I also had to write the **ideal response** that should have been given.

**Industry**

AI Research / Model Quality Evaluation

**Tech Stack**

* Python (problem-solving, algorithms, handling CSVs, databases)
* Web frameworks (Flask, FastAPI, Django, Spring Boot in Java)
* CI/CD pipelines and automation tasks
* General programming (Java, Python)

**Results**

* Helped identify the stronger and more reliable model between two candidates.
* Provided **clear reasoning and comparisons** for why one response was better than the other.
* Created **ideal reference answers** to guide improvements when neither model performed well.

**Challenges Faced**

* Some tasks were very complex, covering advanced frameworks or CI/CD setups.
* Needed to balance **technical correctness** with **clarity and instruction following**.
* Writing an ideal response required deep understanding of multiple frameworks and tools.

**Live Link**

None

**Software Skills Used**

* Evaluation using structured metrics (correctness, security, instruction following, coherence)
* Python, Java, Flask, FastAPI, Django, Spring Boot
* Working with databases and CSV files
* Understanding of CI/CD pipelines

**3.Project: REOWN - WalletKit SDK Integration for Mobile Web Wallet**

**Problem Statement**

The client needed a reliable way to connect their React TypeScript Mobile Web Wallet with decentralized applications (dApps) using **Wallet Connect (via AppKit)**. The goal was to allow users to pair their wallet, approve sessions, and sign actions in a smooth and consistent way, without building a full custom UI or React Native app.

**Solution**

I integrated the **WalletKit SDK** into the client’s Mobile Web Wallet. The implementation covered session lifecycle management (proposing, pairing, approving, rejecting, expiring, and terminating) and supported signing actions once connected. I used the **Wallet Connect UI provided by Reown** to keep the integration lightweight and standardized.

**Industry**

Web3 / Blockchain / dApps Integration

**Tech Stack**

* React + TypeScript (Mobile Web Wallet)
* WalletKit SDK (docs.reown.com)
* Wallet Connect (via AppKit)
* Smart Accounts & USDC balance handling

**Results**

* Delivered a **working WalletKit SDK integration** in the client’s Mobile Web Wallet repo.
* Users can now:
* View their **Smart Account address** and USDC balance.
* Scan a Wallet Connect QR code to pair with supported dApps (e.g., Uniswap).
* Approve or reject sessions seamlessly.
* Take actions on dApps (like swaps) and confirm them directly from their mobile wallet.
* Handle expired sessions and easily reconnect when needed.
* The client was **highly satisfied** with the final delivery.

**Challenges Faced**

* Making sure session lifecycle events (approve, reject, expire, terminate) worked correctly across devices.
* Ensuring the **Smart Account** (not EOA) address stayed in sync with the connected dApp.
* Balancing integration speed while sticking to **Wallet Connect’s UI/UX standards**.

**Live Link**

None

**Software Skills Used**

* React & TypeScript development
* SDK integration (WalletKit, Wallet Connect)
* Blockchain wallet workflows (session lifecycle, signing actions)
* Testing across dApps (Uniswap, Reown test apps)

**4.Project: Vance – Cryptocurrency Arbitrage Bot**

**Problem Statement**

Crypto markets move fast, and price differences across decentralized exchanges can disappear in seconds. The client needed a bot that could automatically identify these gaps and execute trades on the Binance Smart Chain (BSC) to take advantage of arbitrage opportunities.

**Solution**

I developed **Vance**, a cryptocurrency arbitrage bot built to run on the **Binance Smart Chain (BSC)**. The bot leverages smart contracts and decentralized exchanges to detect price differences and execute trades automatically. Using **Ganache** for local blockchain forking and **Hardhat** for deployment, I set up a workflow that allows fast testing and reliable execution on the mainnet.

**Industry**

Blockchain / Cryptocurrency / Trading Automation

**Tech Stack**

* Node.js (v18.20.4+)
* Binance Smart Chain (BSC)
* Smart Contracts (Solidity)
* Hardhat (deployment)
* Ganache (local mainnet fork for testing)

**Results**

* Delivered a working **arbitrage bot** capable of scanning and exploiting price differences across exchanges.
* Implemented smart contract deployment and integration for executing flashloanbased trades.
* Built an **environment configuration system** (.env) to securely manage RPC URLs, keys, and testing variables.
* Provided full deployment workflow (forking mainnet with Ganache, redeploying contracts, running the bot).
* The client successfully tested and deployed the solution and was **highly satisfied** with the outcome.

**Challenges Faced**

* Managing flashloan repayment logic during testing and production.
* Ensuring contract redeployment worked smoothly on every new mainnet fork.
* Keeping the system stable while handling **real-time market fluctuations**.

**Live Link**

None

**Software Skills Used**

* Smart contract development & deployment (Solidity, Hardhat)
* Blockchain testing with Ganache
* Node.js scripting for bot execution
* Environment setup & configuration for secure keys
* Debugging and optimizing arbitrage strategies

**5.Project: Company Website – Professional Portfolio Site**

**Problem Statement**

The company needed a modern portfolio website to showcase its services and projects. The site had to be reliable, easy to update, and deployed in a way that supported development, staging, and production environments.

**Solution**

I built the company’s portfolio website using **Next.js** for the frontend and **Django REST Framework** for the backend. The project was managed end-to-end — from development to deployment. I also set up a full **CI/CD pipeline** to streamline updates, with separate stages for **development, staging, and production**. The website was hosted on **Google Cloud Platform (GCP)** for scalability and reliability.

**Industry**

Web Development / Company Branding

**Tech Stack**

* Frontend: Next.js
* Backend: Django REST Framework (DRF)
* Deployment: Google Cloud Platform (GCP)
* CI/CD: Automated pipelines for dev, staging, and production **Results**
* Delivered a **professional, responsive company website** with a clean UI.
* Implemented **robust CI/CD pipelines**, ensuring smooth and reliable

**Results**

* Delivered a **professional, responsive company website** with a clean UI.
* Implemented **robust CI/CD pipelines**, ensuring smooth and reliable deployments.
* Enabled the team to easily push changes across environments without downtime.
* Improved visibility of the company’s services and projects to clients and partners.

**Challenges Faced**

* Coordinating the pipeline setup across multiple environments.
* Ensuring frontend and backend integration remained smooth during deployments.
* Managing scalability on GCP to handle future growth.

**Live Link**

https://coaldev.com/

**Software Skills Used**

* Full-stack development (Next.js, Django REST)
* CI/CD pipeline setup and management
* Cloud deployment on GCP
* API integration and testing

**6.Project: Rubu – iOS & tvOS Streaming App**

**Problem Statement**

The client needed improvements to their existing streaming app, including UI fixes, multi-channel streaming, and compatibility with Apple TV. The app had to provide a smooth user experience across devices while keeping the existing functionality intact.

**Solution**

I worked on enhancing the **Rubu streaming app** for both **iOS and tvOS**. The updates included fixing UI constraints, refining layouts, and ensuring screens worked consistently across iPhone and Apple TV. I also implemented **multi-player support**, allowing users to add and stream up to 4 channels simultaneously. For Apple TV, I adapted the same set of screens to ensure a consistent experience on the bigger screen. Finally, I handled the **end-to-end deployment of the app to the App Store**, ensuring it passed all Apple review requirements.

**Industry**

Streaming / Media / Entertainment Apps

**Tech Stack**

* iOS Development (Swift)
* tvOS Development
* UIKit (UI improvements & constraints)
* Multi-player streaming support
* App Store Deployment & Release Management

**Results**

* Fixed and improved the app’s UI, ensuring smooth navigation and responsiveness.
* Added **support for up to 4 channels** in multi-player mode.
* Extended compatibility to **Apple TV (tvOS)** using the same screens for consistency.
* Successfully **deployed the app to the App Store**, making it available to end users.
* The client was **highly satisfied** with the delivery and launch.

**Challenges Faced**

* Maintaining existing functionality while fixing UI constraints.
* Adapting mobile UI screens for **Apple TV’s larger display**.
* Ensuring seamless performance when streaming multiple channels at once.
* Navigating App Store submission and approval processes.

**Live Link**

https://apps.apple.com/pk/app/rubu-player/id6463143367

**Software Skills Used**

* iOS & tvOS development (Swift)
* UI/UX optimization and layout handling
* Multi-channel streaming implementation
* Cross-device testing and deployment
* App Store submission & release management

**7.Project: Bittensor SN1 – Mining & Network Contribution**

**Problem Statement**

The goal was to participate in **Bittensor’s SN1 subnet** as a miner and earn incentives by improving the network. Bittensor is a decentralized machine learning network where participants contribute resources and models, and in return, they can earn rewards for valuable contributions.

**Solution**

I explored and worked on integrating with Bittensor SN1, setting up the mining environment and experimenting with improvements to subnet functionality. The focus was on understanding how miners contribute to the network, optimizing performance, and aligning with the incentive mechanisms. While the target of successfully earning incentives was not achieved, the project provided deep insights into decentralized AI and Bittensor’s ecosystem.

**Industry**

Decentralized AI / Blockchain / Mining

**Tech Stack**

* Bittensor framework (SN1 subnet)
* Python
* Apex (macrocosm-os/apex)
* Blockchain-based incentive mechanisms

**Results**

* Gained **hands-on experience** with decentralized AI systems and Bittensor’s incentive model.
* Understood how **subnets (SN1)** work and how miners contribute to improving the network.
* Built knowledge around blockchain-based AI incentive structures and mining setups.
* Although incentives were not earned, the project was **valuable for research and skill growth**.

**Challenges Faced**

* Meeting the **strict performance and contribution requirements** for SN1 incentives.
* Understanding and adapting to the **fast-evolving Bittensor ecosystem**.
* Optimizing mining setup to align with network expectations.

**Live Link**

None

**Software Skills Used**

* Python for mining and subnet integration
* Blockchain & decentralized AI systems (Bittensor)
* Working with open-source frameworks (Apex)
* Experimentation, testing, and optimization of mining setups

**8.Project: Zoho Project**

**Problem Statement**

The client needed to **transfer financial data** from their existing CRM into **Zoho CRM** while also having an easy way to analyze transactions. They required a quick solution that could give them **clear insights** without waiting for full-scale BI integration.

**Solution**

I handled the **migration of financial data** into Zoho CRM and built a lightweight **Python Streamlit dashboard** for transaction analysis. The dashboard allowed the client to track and visualize financial data, identify patterns, and make informed decisions faster.

**Industry**

CRM / Finance / Data Analytics

**Tech Stack**

* Zoho CRM (data migration & management)
* Python
* Streamlit (dashboard & data visualization)

**Results**

* Successfully migrated and structured financial data in **Zoho CRM**.
* Delivered a **Streamlit dashboard** that provided quick, interactive insights on transactions.
* Enabled the client to **analyze financial performance** without needing advanced BI tools.
* Improved decision-making and transparency of financial workflows.

**Challenges Faced**

* Ensuring **data consistency and accuracy** during migration.
* Handling different data formats between the two CRMs.
* Building a dashboard that was simple enough for **non-technical users** to navigate.

**Live Link**

None

**Software Skills Used**

* Zoho CRM integration
* Python (data processing)
* Streamlit (dashboard development)
* Data cleaning and transformation

**Capabilities in AI and LLM Projects**

In past projects, multiple facets of LLM and AI development were tackled by our team, covering essential elements like embedding functions, retrieval-augmented generation (RAG), fine-tuning, and reinforcement learning.

**1. Embedding Functions and Vector Databases**

Embedding functions were implemented to convert text into vectors, storing them in a vector database for optimized retrieval quality. Both offline and online embedding capabilities were included, enhancing lookup precision and retrieval efficiency.

**2. Retrieval-Augmented Generation (RAG)**

RAG was utilized to improve response accuracy, whereby data retrieved from vector databases was augmented before generating a final output, delivering enhanced contextual relevance.

**3. Supervised Fine-Tuning**

Supervised fine-tuning allowed the model’s behavior to be adjusted in alignment with specific conversational requirements. While not altering foundational knowledge, this approach enabled quicker behavioral adaptations for varied use cases.

**4. Reinforcement Learning**

Reinforcement learning was explored to guide the model’s responses based on feedback ratings, shaping desired behavioral adjustments through iterative fine-tuning on a rated scale. This was implemented where project needs aligned with extensive data requirements for iterative learning.