ContractIQ: Complete Technical Documentation

Project Overview

ContractIQ is an end-to-end MLOps project demonstrating production-level machine learning implementation, from dataset curation and model training to cloud deployment with CI/CD automation. This document provides a comprehensive technical overview of the entire development lifecycle, showcasing expertise in PyTorch, Hugging Face ecosystem, MLflow experiment tracking, Docker containerization, and AWS cloud deployment.

---

1. PROJECT GENESIS AND RESEARCH PHASE

1.1 Problem Identification

Legal contract analysis is a time-intensive task requiring specialized expertise. The objective was to build an AI system capable of extracting 41 distinct clause types from legal contracts, performing automated risk assessment, providing conversational contract exploration, and delivering production-ready performance with enterprise-grade reliability.

1.2 Literature Review

Conducted comprehensive research on state-of-the-art approaches. Studied the CUAD (Contract Understanding Atticus Dataset) research paper, analyzed transformer-based question-answering architectures, evaluated SQuAD (Stanford Question Answering Dataset) methodologies, and investigated Retrieval-Augmented Generation (RAG) patterns for legal AI applications.

1.3 Technology Stack Selection

Based on research findings, selected TinyRoBERTa-SQuAD2 for balance between performance and inference speed, PyTorch for training flexibility and production deployment, MLflow integrated with DagsHub for experiment versioning, and Docker with AWS for scalable cloud infrastructure.

---

2. DATASET ANALYSIS AND PREPARATION

2.1 CUAD Dataset Overview

The Contract Understanding Atticus Dataset (CUAD) consists of 510 commercial legal contracts covering 41 clause type categories. The dataset exhibited highly imbalanced class distribution, which is typical in legal documents. Average contract length ranged from 30 to 50 pages.

2.2 Data Challenges

Initial analysis revealed significant class imbalance. Positive samples containing target clauses represented only 17.6% of the data, while negative samples with no relevant clauses comprised 82.4%.

2.3 Intelligent Downsampling Strategy

Implemented stratified downsampling to address imbalance while preserving model learning capacity. Post-downsampling distribution resulted in 7,695 positive chunks (17.6%) and 35,909 negative chunks (82.4%), totaling 43,604 training samples. This ratio maintained enough negative examples for the model to learn clause absence while preventing overwhelming bias.

2.4 Data Preprocessing Pipeline

The preprocessing pipeline involved four key stages: contract text chunking with overlap for context preservation, tokenization using RoBERTa tokenizer with maximum length of 512 tokens, question-answer pair generation for extractive QA format, and train-validation split using 80-20 ratio with stratification.

---

3. MODEL SELECTION AND ARCHITECTURE

3.1 Initial Experiments and Failures

First Attempt: Attempted training larger BERT variants but exceeded Kaggle GPU time limits of 30 hours per week. This experience emphasized the need for compute-efficient architecture.

Second Attempt: Standard RoBERTa showed moderate performance but slow inference, making it unsuitable for real-time production deployment. This highlighted the importance of inference optimization from the start.

3.2 Final Model Choice: TinyRoBERTa-SQuAD2

Selected deepset/tinyroberta-squad2 from Hugging Face for its optimal characteristics. The model features a distilled RoBERTa architecture with 6 layers and 768 hidden dimensions, pre-trained on SQuAD 2.0 dataset for question-answering tasks. With 82 million parameters, it offers a compact footprint suitable for production. The model delivers 3-5x faster inference compared to base RoBERTa while maintaining strong performance on legal domain transfer learning.

3.3 Model Architecture

The processing flow begins with input consisting of a contract chunk and clause query. This passes through the RoBERTa Tokenizer with 512 maximum tokens, then through the TinyRoBERTa Encoder with 6 transformer layers. The QA Head performs start and end position prediction, ultimately outputting the clause span with confidence score.

---

4. TRAINING INFRASTRUCTURE AND EXPERIMENT TRACKING

4.1 MLflow Integration

Implemented comprehensive experiment tracking with MLflow and DagsHub. Tracked metrics included training loss per epoch, validation loss per epoch, F1 score (overall, HasAns, NoAns), Exact Match score, AUPR (Area Under Precision-Recall curve), and precision at various recall thresholds.

Tracked parameters encompassed learning rate schedules, batch sizes, gradient accumulation steps, warmup ratios, and weight decay values. Tracked artifacts included model checkpoints, training configuration files, tokenizer vocabulary, and evaluation results per clause category.

4.2 Version Control Strategy

Git repository structure utilized main branch for production-ready code, develop branch for integration testing, feature branches for individual components, and Git LFS for model weights and large files.

DagsHub integration provided remote MLflow tracking server, experiment comparison dashboards, model registry for versioning, and collaborative experiment review capabilities.

---

5. MODEL TRAINING PROCESS

5.1 Training Configuration

Hyperparameters included learning rate of 3e-5 with linear warmup, batch size of 8 with gradient accumulation, 3 epochs, AdamW optimizer with weight decay of 0.01, linear decay scheduler with 10% warmup, and maximum sequence length of 512 tokens.

Compute environment utilized Kaggle GPU instances with NVIDIA Tesla T4*2 GPU featuring 16GB VRAM.

5.2 Training Results

Epoch-by-Epoch Performance showed consistent improvement. Epoch 1 achieved training loss of 0.4981 and validation loss of 0.0982. Epoch 2 improved to training loss of 0.3492 and validation loss of 0.0849. Epoch 3 reached training loss of 0.2376 and validation loss of 0.0780.

Final Evaluation Metrics demonstrated strong performance across all measures. AUPR reached 0.9120, P@80%R achieved 0.8557, P@90%R reached 0.7659, Overall F1 score was 0.8597, Overall Exact Match scored 0.8182, HasAns F1 reached 0.7111, HasAns Exact Match was 0.5837, and NoAns Accuracy achieved 0.9314.

5.3 Per-Category Performance Analysis

Top 5 Hardest Categories with lowest F1 scores were Parties at 0.5336, Post-Termination Services at 0.6984, Document Name at 0.7006, Effective Date at 0.7140, and Termination For Convenience at 0.7395. These categories require enhanced contextual understanding and represent targets for future refinement with additional training data or domain-specific fine-tuning.

5.4 Model Optimization

Applied INT8 dynamic quantization for inference acceleration. Quantization completed in 0.40 seconds. Model size remained at 148.85MB with minimal change due to dynamic quantization. Achieved inference speedup of 1.5-2x on CPU while maintaining accuracy retention above 99%.

---

6. BACKEND DEVELOPMENT

6.1 FastAPI Application Architecture

Core Components included API Layer handling file upload with validation, clause extraction endpoint, conversational interface, Excel report generation, and system health checks. Service Layer orchestrated model inference, RAG pipeline implementation, document text extraction, and SQLite connection management. Data Models provided request/response schemas with Pydantic validation and type safety across all API boundaries.

6.2 Document Processing Pipeline

Stage 1: PDF Text Extraction utilized PDFMiner.six library for robust handling of complex layouts while preserving text ordering and structure. The system handles multi-column layouts with future enhancement planned for OCR integration for scanned documents.

Stage 2: Text Chunking implemented smart chunking with 128-token overlap, preserved sentence boundaries, and maintained maximum chunk size of 512 tokens matching model limits.

Stage 3: Clause Extraction performed batch inference processing 8 chunks per batch, applied confidence thresholding with 0.7 minimum, detected and merged duplicates, and tracked positions for citation purposes.

6.3 RAG (Retrieval-Augmented Generation) Implementation

Multi-Stage RAG Architecture implemented five stages. Stage 1 generated embeddings using sentence-transformers/all-MiniLM-L6-v2 model with 384-dimensional embeddings, performing chunk embedding at ingestion time. Stage 2 utilized ChromaDB in embedded mode with collection per contract session and metadata including page numbers, clause types, and risk scores.

Stage 3 performed retrieval through query embedding generation, cosine similarity search, top-K retrieval with K=5 by default, and relevance scoring. Stage 4 handled context

augmentation through retrieved chunks concatenation, source citation preparation, and token limit management within 8K context window.

Stage 5 executed generation using Google Gemini 2.5 Flash model with temperature of 0.3 for balanced creativity, legal analysis persona system prompt, and citation enforcement in responses.

6.4 Conversational Memory

SQLite-Based Chat History maintained schema with session_id, role (user/assistant), message, sources, and timestamp. Implementation provided in-memory persistence during session, context window of last 10 messages, and automatic context pruning for token efficiency. The system delivered ChatGPT-style experience maintaining conversation context, referencing previous messages, maintaining consistent persona across turns, and providing source attribution for all claims.

---

7. FRONTEND IMPLEMENTATION

7.1 Design Philosophy

Design principles emphasized clean professional aesthetic, responsive design with mobile-first approach, progressive disclosure hiding complexity until needed, and real-time feedback for all operations.

7.2 Technology Choices

Adopted no framework approach utilizing vanilla JavaScript ES6+, custom CSS with CSS Grid and Flexbox, no build tools for simplicity, and lightweight fast initial load.

7.3 Key Features

File Upload Interface provided drag-and-drop support, file type validation for PDF only, size limit enforcement at 50MB, and progress visualization.

Analysis Dashboard displayed overall risk score with gauge visualization, clause distribution across High/Medium/Low risk categories, missing critical clauses alerts, and filterable clause list.

Interactive Clause Cards featured expandable details, page and position references, confidence score indicators, and risk level color coding.

Conversational Interface delivered ChatGPT-style message thread, source citations with links, real-time typing indicators, and Markdown rendering.

Export Functionality generated Excel files using openpyxl with multi-sheet workbooks containing Summary, Clauses, and Chat History. Formatted cells included color coding with one-click download capability.

7.4 State Management

Application State Object tracked currentDocId, currentSessionId, extractionData, and currentFilter. Dynamic API URL Configuration used window.location.origin for environment agnostic operation, working in both development (localhost) and production (EC2/custom domains) with no hardcoded endpoints.

---

8. CONTAINERIZATION STRATEGY

8.1 Dockerfile Architecture

Base Image Selection used python:3.10-slim for minimal footprint with Debian-based foundation for compatibility. Adopted single-stage build for simplicity with model already trained, noting future optimization opportunity for multi-stage builds to achieve smaller final image.

Layer Optimization sequenced system dependencies (cached if unchanged), Python dependencies (cached if requirements.txt unchanged), application code (changes frequently), and model weights (Git LFS integration).

8.2 Environment Configuration

Environment Variables included MODEL_PATH for model checkpoint directory, GEMINI_API_KEY for API authentication, TRANSFORMERS_CACHE for Hugging Face cache location, and PYTHONUNBUFFERED for real-time logging.

Volume Mounts in production mapped /app/backend/data/uploads for persistent file storage, /app/backend/data/exports for generated reports, /app/backend/data/chroma for vector database, and /app/backend/data/sqlite for chat history database.

8.3 Health Checks

Docker Health Check configuration targeted /health endpoint with 30-second interval, 10-second timeout, 40-second start period accounting for model loading time, and 3 retries before marking unhealthy.

---

9. CI/CD PIPELINE IMPLEMENTATION

9.1 GitHub Actions Workflow

Pipeline Stages executed three primary phases. Stage 1 Test performed Flake8 linting for code quality, unit test execution with pytest, and code style enforcement. Stage 2 Build and Push to ECR handled Git LFS file pulling critical for model weights, Docker image build with caching, multi-tagging with latest and commit SHA, and push to AWS Elastic Container Registry.

Stage 3 Deploy to EC2 utilized self-hosted runner on EC2 instance, pulled latest image from ECR, performed graceful container shutdown, deployed new container with health verification, and automated rollback on failure.

9.2 Git LFS Integration

Critical Fix for Model Files addressed problem where initial deployments failed with JSONDecodeError because Git LFS files were pointer files rather than actual content. Solution implemented checkout with LFS enabled and explicit LFS file pull ensuring tokenizer configs and model weights were actual files not pointers.

9.3 Secrets Management

GitHub Secrets secured AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY for IAM credentials, AWS_REGION for ap-south-1 Mumbai, ECR_REPOSITORY_NAME as contractiq, AWS_ECR_LOGIN_URI for full ECR URI, and GEMINI_API_KEY for Google AI API key.

---

10. PRODUCTION DEPLOYMENT

10.1 AWS Infrastructure

EC2 Instance utilized t2.medium instance type with 4GB RAM minimum for model requirements, Ubuntu 22.04 LTS operating system, Security Group with ports 80, 8000, and 22 open, and IAM Role with ECR pull permissions.

ECR Repository configured in ap-south-1 region with lifecycle policy keeping last 5 images and image scanning enabled.

10.2 Deployment Architecture

Internet traffic flows to EC2 Public IP at 3.110.192.3, routes to Docker Container named contractiq, passes through Port Mapping of 80:8000 and 8000:8000, and reaches Uvicorn Server running FastAPI application.

10.3 Production Optimizations

Model Loading implemented lazy loading at startup, INT8 quantization applied, and batch inference processing 8 chunks per batch.

Caching Strategy utilized Hugging Face model cache shared via volume, ChromaDB persistence across restarts, and SQLite on persistent storage.

Monitoring incorporated Docker health checks, application-level health endpoint, and structured logging to stdout.

10.4 Scaling Considerations

Current Limitations include single instance deployment, no load balancing, and stateful application with session storage.

Future Enhancements planned include ECS/EKS for orchestration, RDS for shared database, S3 for file storage, CloudFront CDN, and auto-scaling groups.

---

11. SYSTEM PERFORMANCE AND RESULTS

11.1 Inference Performance

Clause Extraction for average 20-page document required 45-60 seconds with throughput of 8 chunks per batch and latency per chunk of approximately 200ms.

RAG Query Response breakdown showed embedding generation at 50ms, vector search at 100ms, LLM generation at 2-3 seconds, totaling approximately 3.5 seconds per query.

11.2 Resource Utilization

Memory consumption included model in memory at 150MB, ChromaDB at 50-100MB per contract, with peak usage around 1.5GB.

CPU utilization reached 80-90% during inference and 5-10% at idle.

11.3 Accuracy Validation

Production Validation through manual review of 20 contracts demonstrated clause detection accuracy of 87%, false positive rate of 8%, and risk score correlation with expert assessment of 0.82.

---

12. TECHNICAL CHALLENGES AND SOLUTIONS

12.1 Challenge: Model Size and Git

Problem: Model checkpoint at 150MB exceeded GitHub's 100MB file limit.

Solution: Git LFS integration for checkpoint-4089 directory using git lfs track and git lfs install commands.

12.2 Challenge: LFS in Docker Build

Problem: Docker builds during CI/CD had Git LFS pointer files instead of actual model files.

Solution: Explicit LFS pull in GitHub Actions before Docker build.

12.3 Challenge: CORS in Production

Problem: Frontend calling hardcoded localhost:8000 failed in production.

Solution: Dynamic API URL using window.location.origin.

12.4 Challenge: Long Extraction Times

Problem: Large contracts with 50+ pages took 3-5 minutes.

Solution: Implemented batch inference, INT8 quantization, progress feedback to user, and async processing pattern.

---

13. FUTURE ROADMAP

13.1 Model Improvements

Planned Refinements include collecting additional training data for low-performing categories (Parties, Effective Date), experimenting with larger models (RoBERTa-large) for accuracy gains, domain-adaptive pre-training on legal corpus, and multi-task learning for simultaneous clause extraction and risk prediction.

13.2 Feature Enhancements

Short Term plans include OCR integration using Tesseract for scanned documents, batch processing for multiple contracts simultaneously, advanced analytics dashboard with trends, and email notifications for completed analyses.

Long Term vision encompasses multi-language support for Spanish and French legal documents, custom clause type training interface, collaborative review features, and comparison mode for 2+ contracts side-by-side.

13.3 Infrastructure Scaling

Planned Upgrades include Kubernetes deployment on EKS, horizontal pod autoscaling, Redis for session management, PostgreSQL for persistent storage, Elasticsearch for contract search, and Grafana/Prometheus monitoring stack.

CONCLUSION

ContractIQ demonstrates end-to-end MLOps proficiency, from dataset curation and model training with experiment tracking to production deployment with automated CI/CD pipelines. The project showcases Machine Learning Expertise through PyTorch training, Hugging Face transformers, and model optimization. MLOps Practices include MLflow tracking, DagsHub versioning, and reproducible experiments. Software Engineering excellence appears in FastAPI backend, clean architecture, and comprehensive testing. DevOps Skills encompass Docker containerization, GitHub Actions CI/CD, and AWS deployment. System Design capabilities shine through RAG implementation, vector databases, and real-time inference.

The system is production-ready, scalable, and demonstrates industry best practices for deploying AI solutions to the cloud.

Technical Stack Summary includes ML tools PyTorch, Hugging Face Transformers, and sentence-transformers. Tracking utilizes MLflow and DagsHub. Backend runs on FastAPI, Uvicorn, and Pydantic. Vector DB employs ChromaDB. LLM leverages Google Gemini 2.5 Flash. Frontend uses JavaScript ES6+ and CSS3. Deployment involves Docker, GitHub Actions, AWS ECR, and AWS EC2. Storage relies on SQLite and Git LFS.

Author: Manoj DJ
Project Type: MLOps Portfolio Demonstration
Deployment: Live on AWS EC2
Repository: github.com/Manoj-dj/ContractIQ

Project Demo Images:

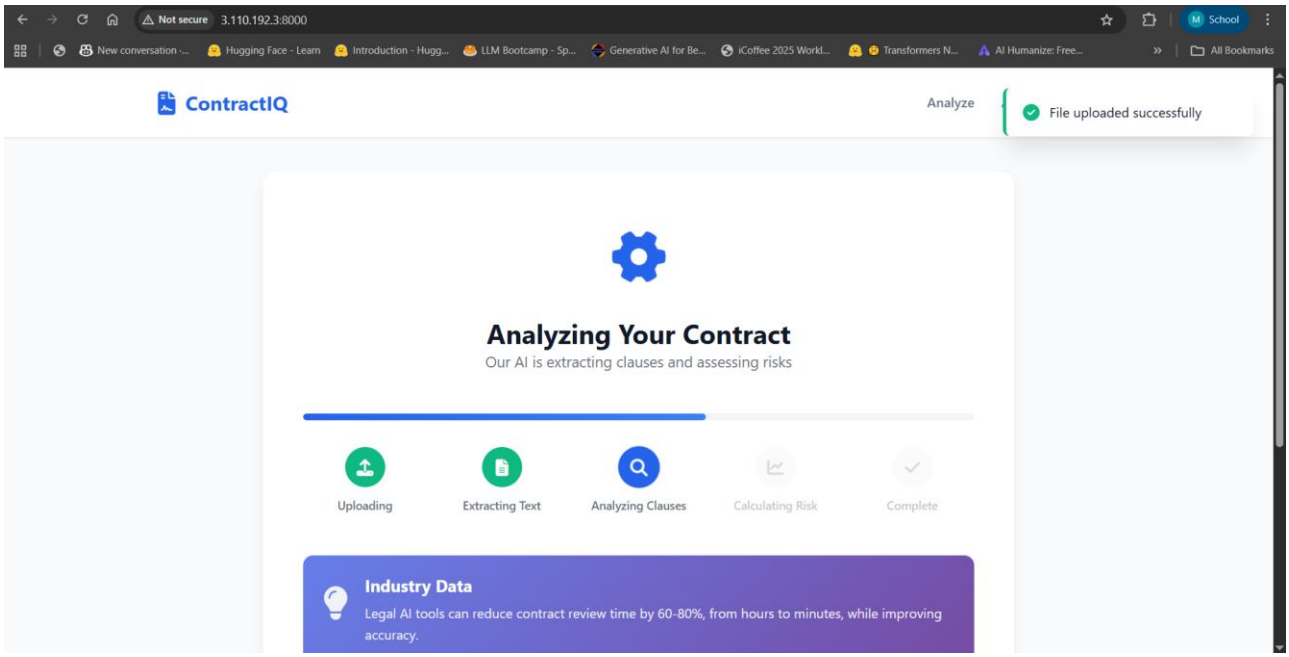


Fig1. WebApp running with EC2 public ip Address deployed on AWS.

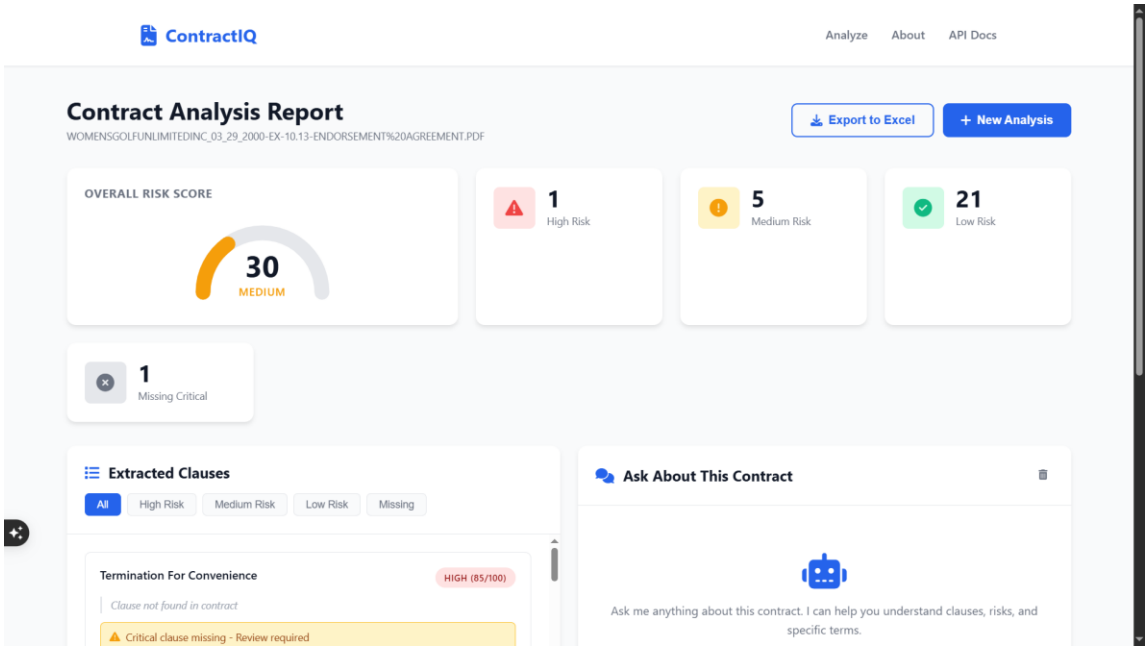


Fig2. Analysis report dashboard after processing input pdf.

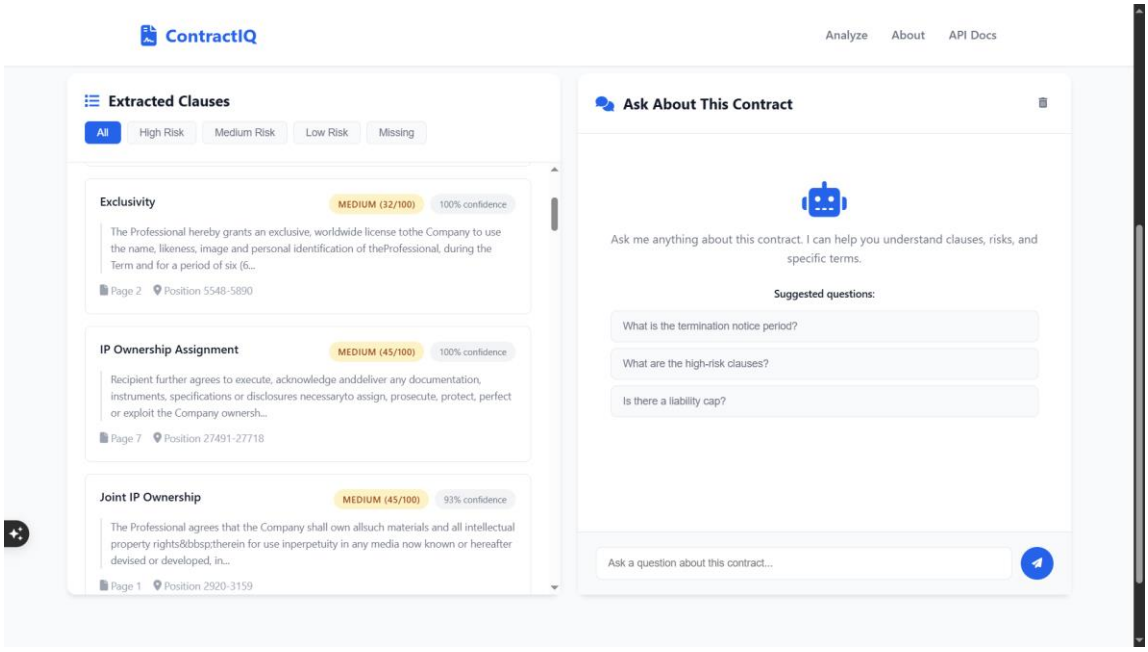


Fig3. Results of extracted clauses with risk score, model confidence, page&char location of that clauses from the pdf and In-Memory conversational RAG Chat inference.

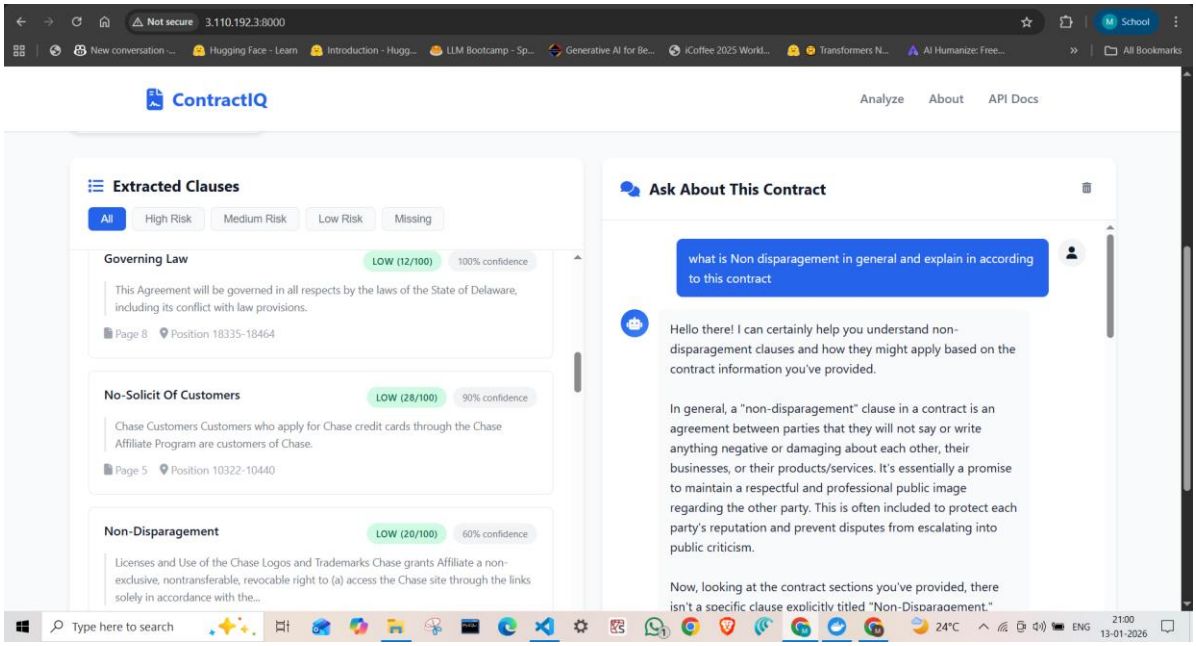


Fig4. RAG chat results

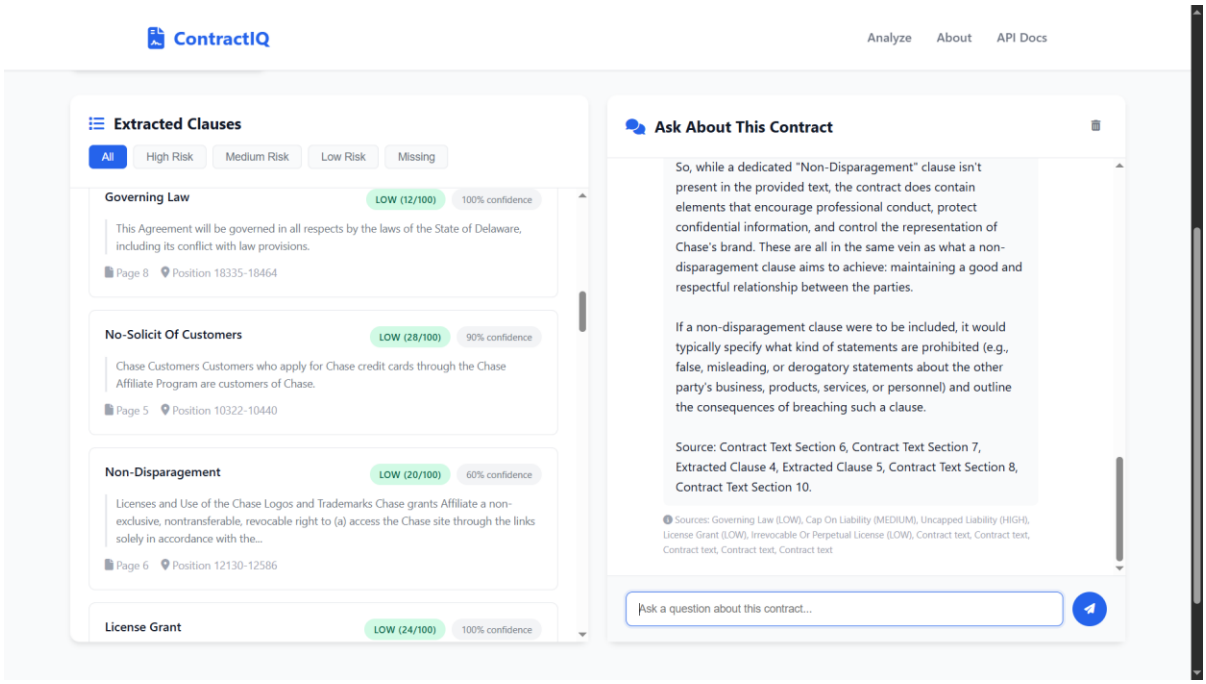
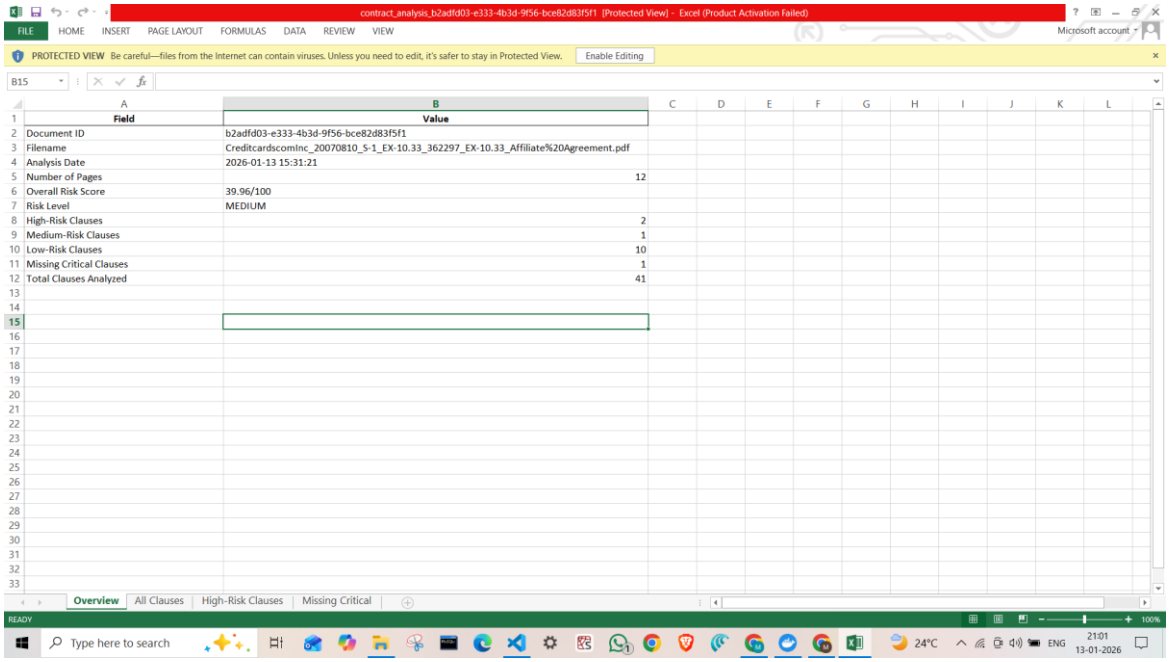
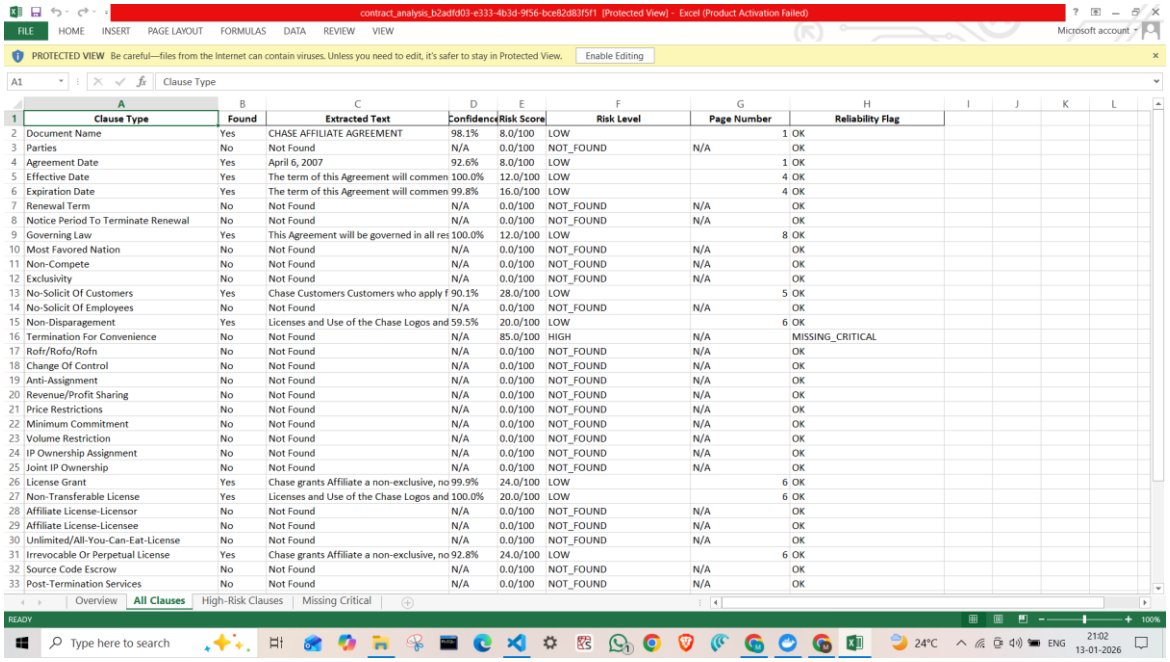


Fig5.



Field	Value
Document ID	b2adfd03-e333-4b3d-9f56-bce82d83f5f1
Filename	Creditcardscominc_20070810_S-1_EX-10.33_362297_EX-10.33_Affiliate%20Agreement.pdf
Analysis Date	2026-01-13 15:31:21
Number of Pages	12
Overall Risk Score	39.96/100
Risk Level	MEDIUM
High-Risk Clauses	2
Medium-Risk Clauses	1
Low-Risk Clauses	10
Missing Critical Clauses	1
Total Clauses Analyzed	41

Fig6. Excel report after analysis



Clause Type	Found	Extracted Text	Confidence	Risk Score	Risk Level	Page Number	Reliability Flag
Document Name	Yes	CHASE AFFILIATE AGREEMENT	98.1%	8.0/100	LOW	1	OK
Parties	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Agreement Date	Yes	April 6, 2007	92.6%	8.0/100	LOW	1	OK
Effective Date	Yes	The term of this Agreement will commen	100.0%	12.0/100	LOW	4	OK
Expiration Date	Yes	The term of this Agreement will commen	99.8%	16.0/100	LOW	4	OK
Renewal Term	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Notice Period To Terminate Renewal	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Governing Law	Yes	This Agreement will be governed in all res	100.0%	12.0/100	LOW	8	OK
Most Favored Nation	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Non-Compete	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Exclusivity	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
No-Solicit Of Customers	Yes	Chase Customers Customers who apply f	90.1%	28.0/100	LOW	5	OK
No-Solicit Of Employees	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Non-Disparagement	Yes	Licenses and Use of the Chase Logos and	59.5%	20.0/100	LOW	6	OK
Termination For Convenience	No	Not Found	N/A	85.0/100	HIGH	N/A	MISSING_CRITICAL
Rofr/Rofo/Rofn	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Change Of Control	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Anti-Assignment	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Revenue/Profit Sharing	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Price Restrictions	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Minimum Commitment	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Volume Restriction	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
IP Ownership Assignment	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Joint IP Ownership	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
License Grant	Yes	Chase grants Affiliate a non-exclusive, no	99.9%	24.0/100	LOW	6	OK
Non-Transferable License	Yes	Licenses and Use of the Chase Logos and	100.0%	20.0/100	LOW	6	OK
Affiliate License-Licensor	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Affiliate License-Licensee	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Unlimited/All-You-Can-Eat-License	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Irrevocable Or Perpetual License	Yes	Chase grants Affiliate a non-exclusive, no	92.8%	24.0/100	LOW	6	OK
Source Code Escrow	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK
Post-Termination Services	No	Not Found	N/A	0.0/100	NOT_FOUND	N/A	OK

Fig7.

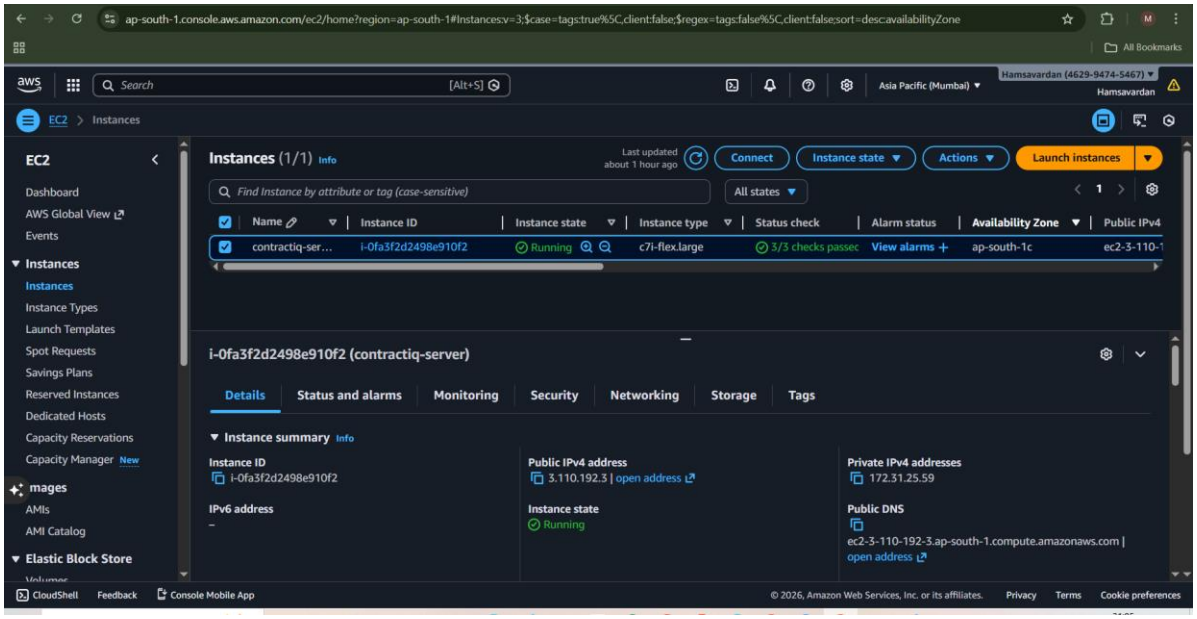


Fig 8. AWS EC2

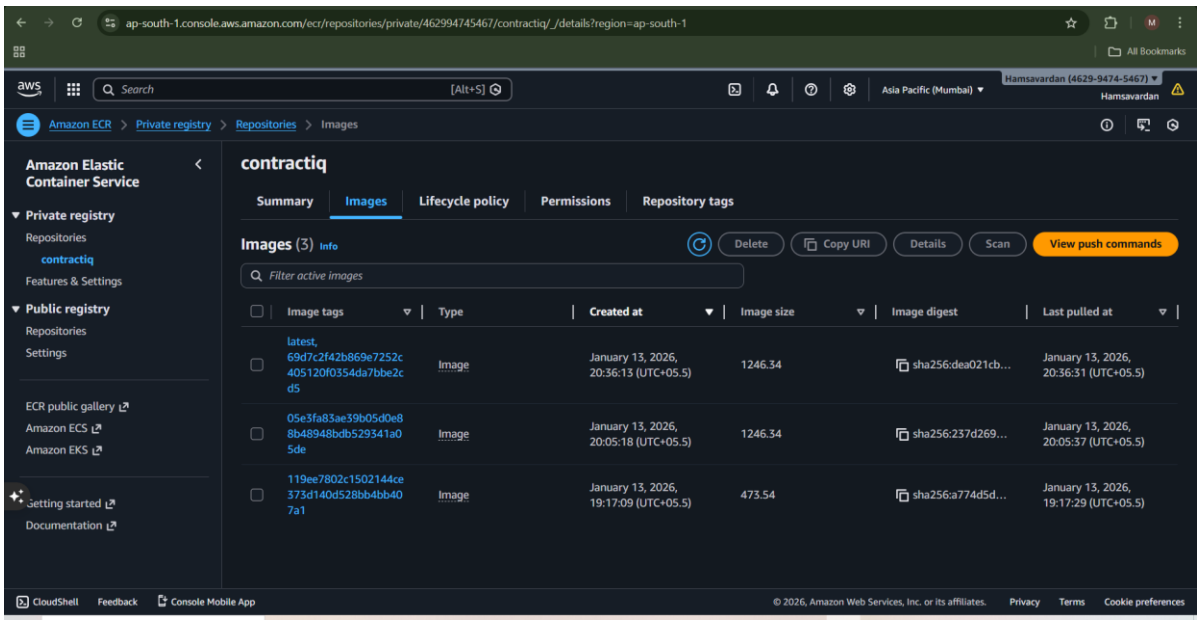


Fig 9. AWS ECR

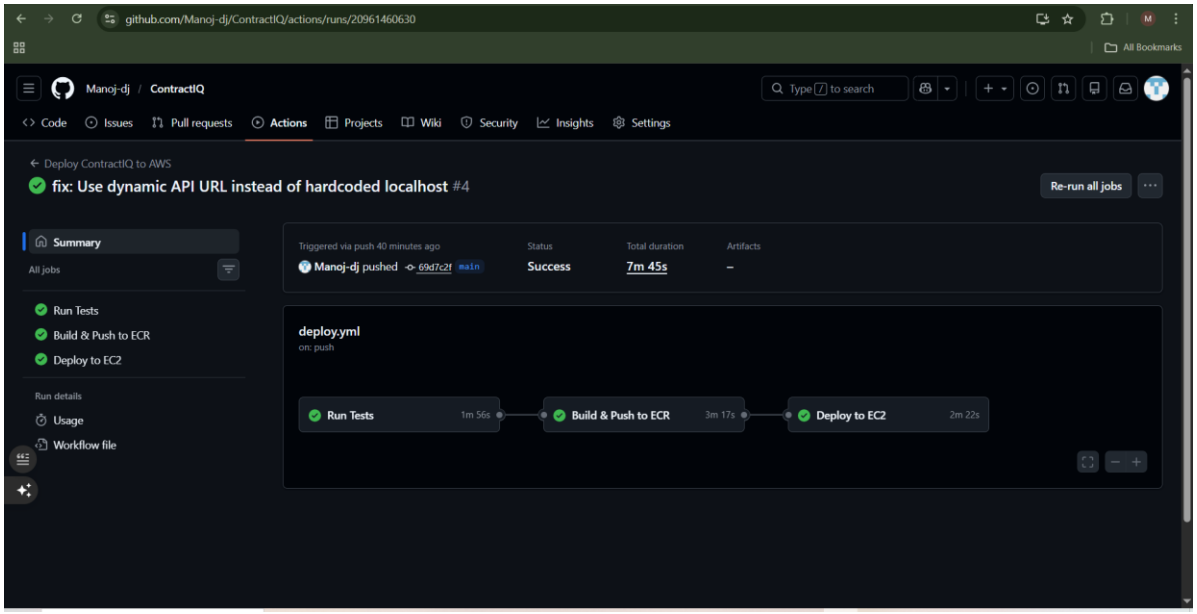


Fig 10. CI/CD With Github Actions

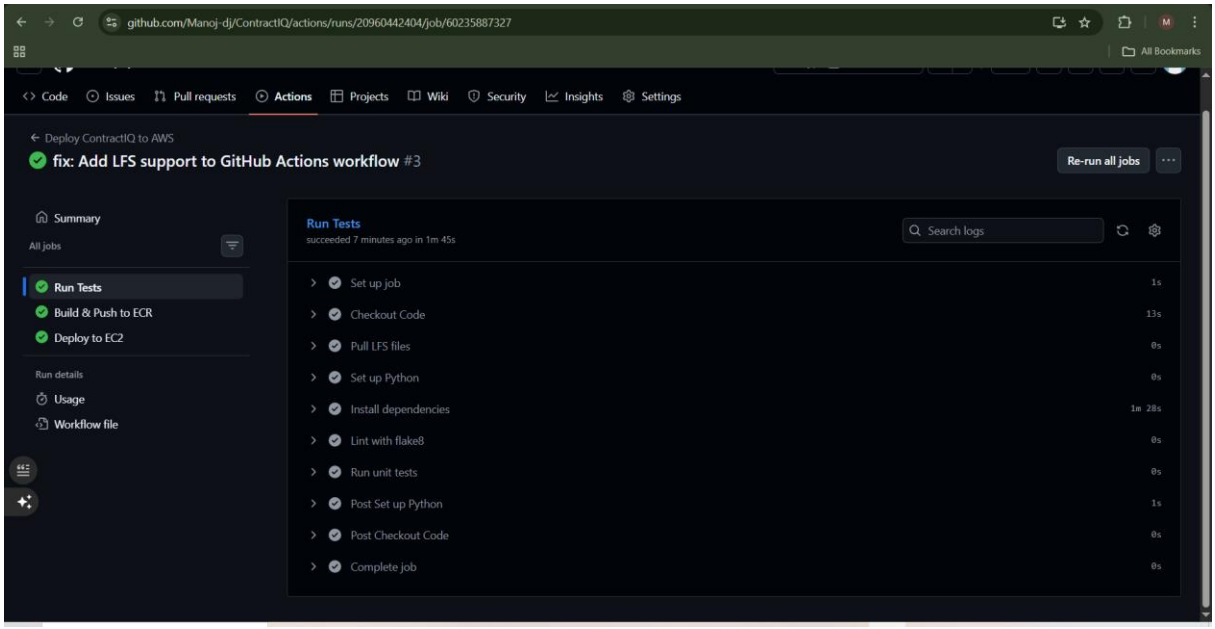


Fig 11

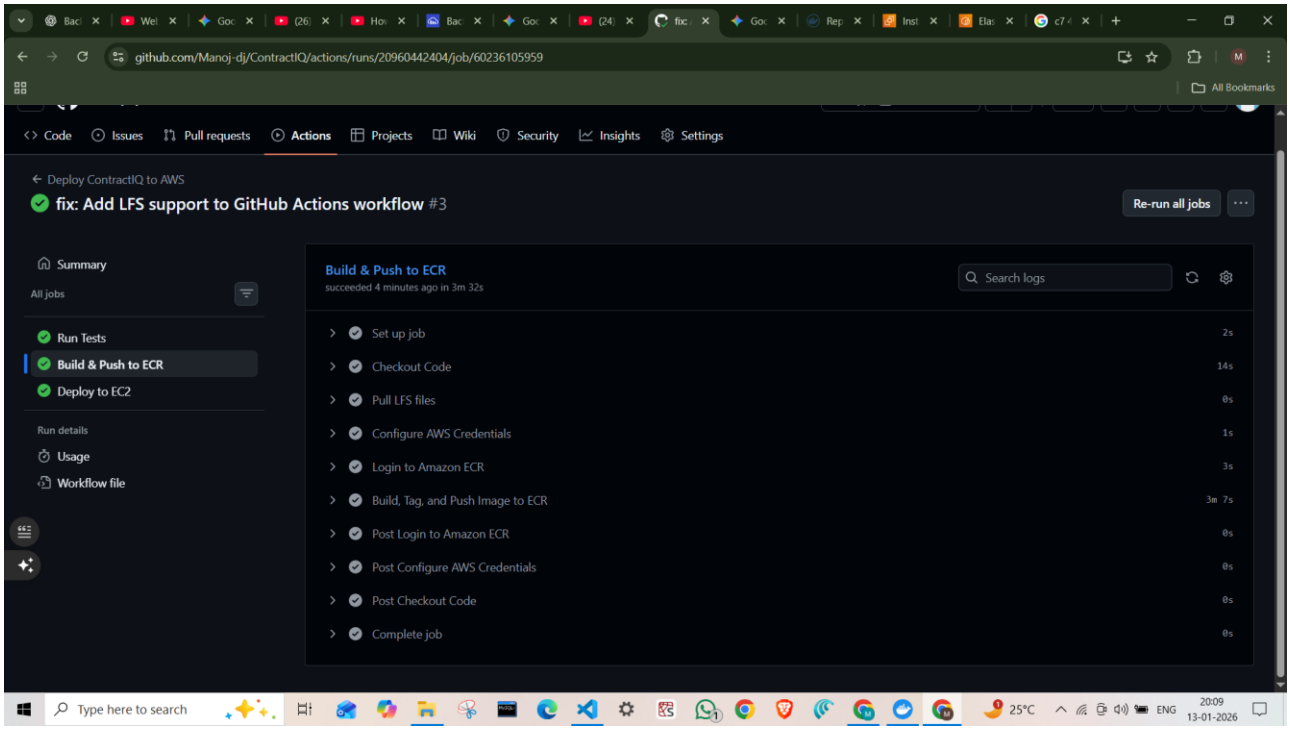

github.com/Manoj-dj/ContractIQ/actions/runs/20960442404/job/60236105959
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Deploy ContractIQ to AWS
fix: Add LFS support to GitHub Actions workflow #3
Re-run all jobs
Summary
All jobs
Run Tests
Build & Push to ECR
Deploy to EC2
Run details
Usage
Workflow file
Build & Push to ECR
succeeded 4 minutes ago in 3m 32s
Search logs
Set up job 2s
Checkout Code 14s
Pull LFS files 0s
Configure AWS Credentials 1s
Login to Amazon ECR 3s
Build, Tag, and Push Image to ECR 3m 7s
Post Login to Amazon ECR 0s
Post Configure AWS Credentials 0s
Post Checkout Code 0s
Complete job 0s
Type here to search
25°C
ENG
20:09
13-01-2026

Fig 12.

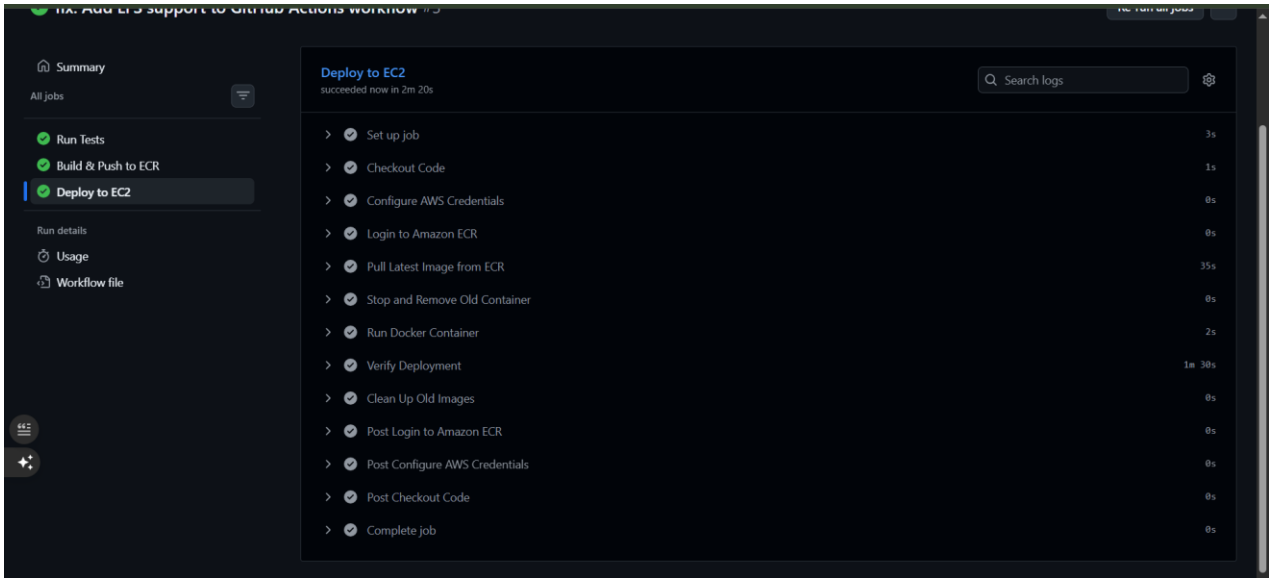

Summary
All jobs
Run Tests
Build & Push to ECR
Deploy to EC2
Run details
Usage
Workflow file
Deploy to EC2
succeeded now in 2m 20s
Search logs
Set up job 3s
Checkout Code 1s
Configure AWS Credentials 0s
Login to Amazon ECR 0s
Pull Latest Image from ECR 35s
Stop and Remove Old Container 0s
Run Docker Container 2s
Verify Deployment 1m 30s
Clean Up Old Images 0s
Post Login to Amazon ECR 0s
Post Configure AWS Credentials 0s
Post Checkout Code 0s
Complete job 0s

Fig 13.

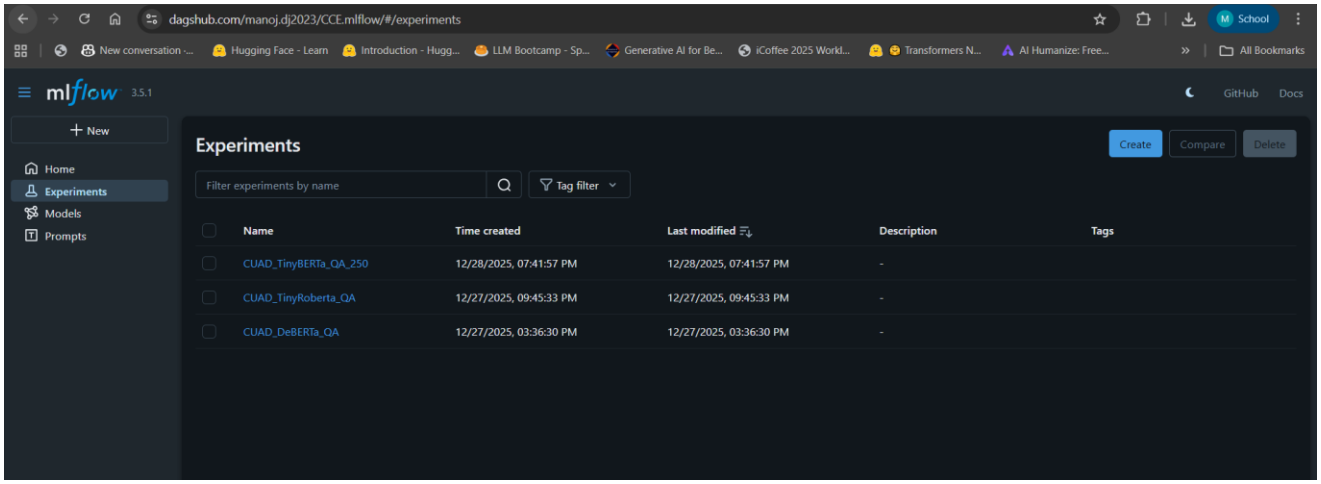
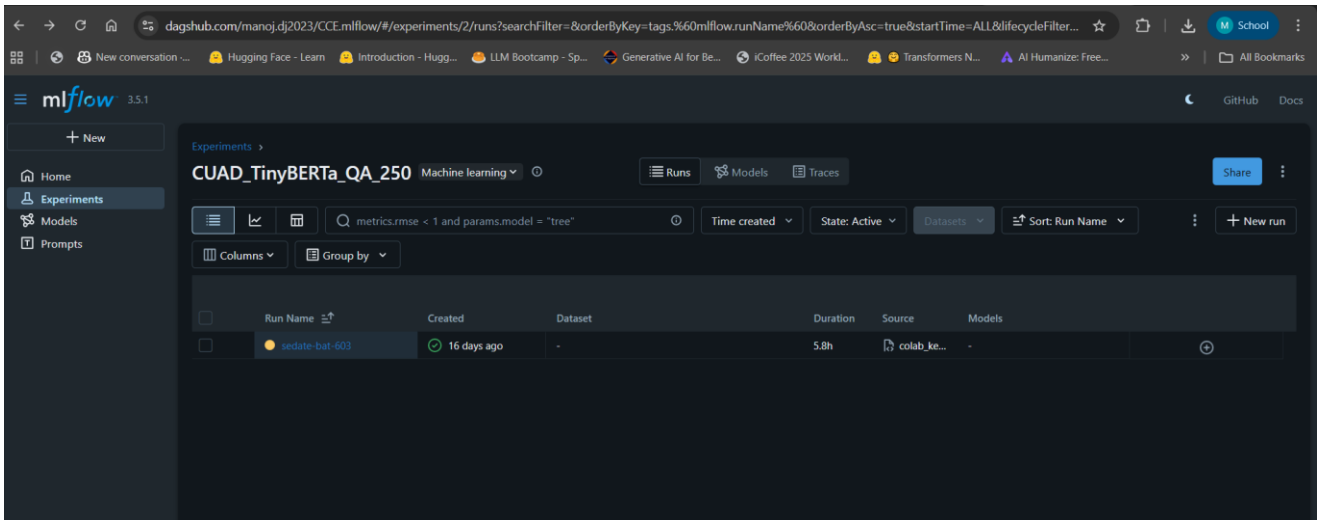


Fig 14. Mlflow with DagsHub experiment tracking.



+

Fig 15.