CiteConnect: A Research Paper Recommendation System

Team Members:

1. Abhinav Aditya
2. Anusha Srinivasan
3. Dennis Jose
4. Dhiksha Mathanagopal
5. Sahil Mohanty

1. Introduction

CiteConnect is a **Research Paper Recommendation System** designed to help researchers efficiently discover and explore relevant academic literature. By combining **retrieval-augmented generation (RAG)**, **vector search**, and **citation graph analysis**, the system goes beyond traditional keyword-based search engines that return long, unstructured lists of papers. CiteConnect provides curated recommendations, explains why each paper is relevant, and visualizes connections between works through an interactive citation graph. With added features such as user authentication and personalized research profiles, the platform adapts to individual interests and delivers targeted results. Built using modern machine learning and MLOps practices, the project addresses the problem of information overload in academia while showcasing a scalable, end-to-end pipeline suitable for real-world deployment.

2. Dataset Information

2.1 Dataset Introduction

The dataset for CiteConnect will consist of academic papers and their associated metadata, sourced primarily from open-access repositories. Each entry contains the paper title, authors, abstract, publication year, citation relationships, and, when available, the full PDF for parsing and embedding. This dataset is critical for enabling semantic retrieval (through embeddings), summarization (through NLP pipelines), and citation graph construction (through reference mapping). By focusing on open-access sources, the dataset ensures accessibility, reproducibility, and compliance with data rights.

2.2 Data Card

Attribute	Details
Dataset Name	CiteConnect Academic Papers
Purpose	Power semantic search, personalized recommendations, and citation graph.
Size	~10,000–20,000 papers in initial ingestion (scalable to 100k+)
Average File	2 MB per PDF (20–40 GB total storage for 20k papers)
Format	JSON (metadata), PDF (raw papers), TXT/JSON (parsed text), Vectors (embeddings), Graph DB (citations)
Data Types	Text (abstracts, body content), numerical (citations, publication year), graph structures (references/edges)
Storage	Google Cloud Storage (raw PDFs), Vector DB (FAISS/Pinecone/Weaviate), Neo4j (citations), SQL/Firestore (user interaction data)

2.3 Data Sources

arXiv API: https://info.arxiv.org/help/api/

Semantic Scholar API: https://www.semanticscholar.org/product/api

Unpaywall (for OA PDFs): https://unpaywall.org/products/api

2.4 Data Rights and Privacy

- Only open-access papers will be downloaded and parsed, ensuring compliance with copyright and licensing terms.
- Metadata from APIs is publicly available.
- No personal or sensitive data is included, so privacy risks are minimal.
- Compliance with GDPR and academic data sharing guidelines will be ensured by avoiding storage of personally identifiable information.

3. Data Planning and Splits

3.1 Data Loading

- Metadata Ingestion: Papers will be fetched via arXiv API and enriched with Semantic Scholar API for citation data. Metadata (title, authors, abstract, year, citations) will be stored in structured JSON format.
- PDF Retrieval: Only open-access PDFs will be downloaded and stored in Google Cloud Storage.
- Parsing: PDFs will be processed using PyMuPDF or pdfminer.six to extract structured text. For complex papers, GROBID may be used to parse sections (abstract, introduction, methods, results).

3.2 Data Preprocessing

- **Text Cleaning**: Removal of headers, footers, page numbers, and non-informative content.
- **Chunking**: Splitting long documents into manageable text chunks (512–1000 tokens) to enable embeddings and retrieval.
- **Embedding Generation**: Each chunk will be embedded using **OpenAI embeddings** or **sentence-transformers**, stored in a **vector database**.
- **Citation Mapping**: References will be parsed and linked in **Neo4j** to construct a citation graph.
- **Metadata Enrichment**: Each record will include authors, keywords, year, and citation counts for ranking and filtering.

3.3 Data Management

Storage: Raw PDFs → Google Cloud Storage (GCS).
Parsed text + metadata → JSON/SQL
Embeddings → Vector DB (FAISS/Pinecone/Weaviate).
Citations → Neo4j Graph DB.

Versioning: DVC will be used for version control of datasets and metadata.

Pipeline Automation: Apache Airflow (Cloud Composer) will schedule regular ingestion, preprocessing, and embedding updates.

3.4 Data Splitting Strategy

Evaluation Area	Approach	Purpose
Embedding Evaluation	Use 80% of documents to build the vector index; hold out 20% for testing retrieval quality (recall@k, precision@k).	To measure how effectively the system retrieves relevant papers.
Summarization Evaluation	Select a representative subset of papers; compare LLM-generated summaries with their gold-standard abstracts.	To validate the quality and accuracy of automatic summarization.
Personalization Evaluation	Simulate user profiles with varied research interests; split 80% interactions for training and 20% for testing re-ranking.	To assess how well the system personalizes recommendations to user profiles.

4. GitHub Repository

https://github.com/DhikshaMathanagopal/CiteConnect

5. Project Scope

5.1 Problems

1. **Information Overload:** Thousands of new papers are published daily, making it hard to filter relevant ones.
2. **Keyword Limitations:** Traditional searches often return long, noisy lists of results.
3. **Lack of Personalization:** Most academic search tools don't adapt to individual research interests.
4. **Poor Explainability:** Few systems explain *why* a paper is recommended.
5. **Disconnected Literature:** Exploring how papers are linked via citations or themes is still fragmented and requires manual effort.

5.2 Current Solutions

Google Scholar / Semantic Scholar: Provide strong keyword-based search and citation counts. Semantic Scholar also highlights "influential citations," but papers are still presented in lists, not an interactive network.
Connected Papers & Research Rabbit: Tools that already visualize academic papers as citation graphs. They allow users to explore how research is connected, but they are primarily **exploration-only tools**. They lack personalized recommendations tied to user profiles, integrated summarization pipelines explaining why a paper is relevant, automated ingestion and monitoring pipelines (no MLOps focus).
Library Databases (Scopus, Web of Science): Offer citation analytics and citation maps but are locked behind paywalls and are not open, user-friendly, or integrated with modern ML-driven recommendations.

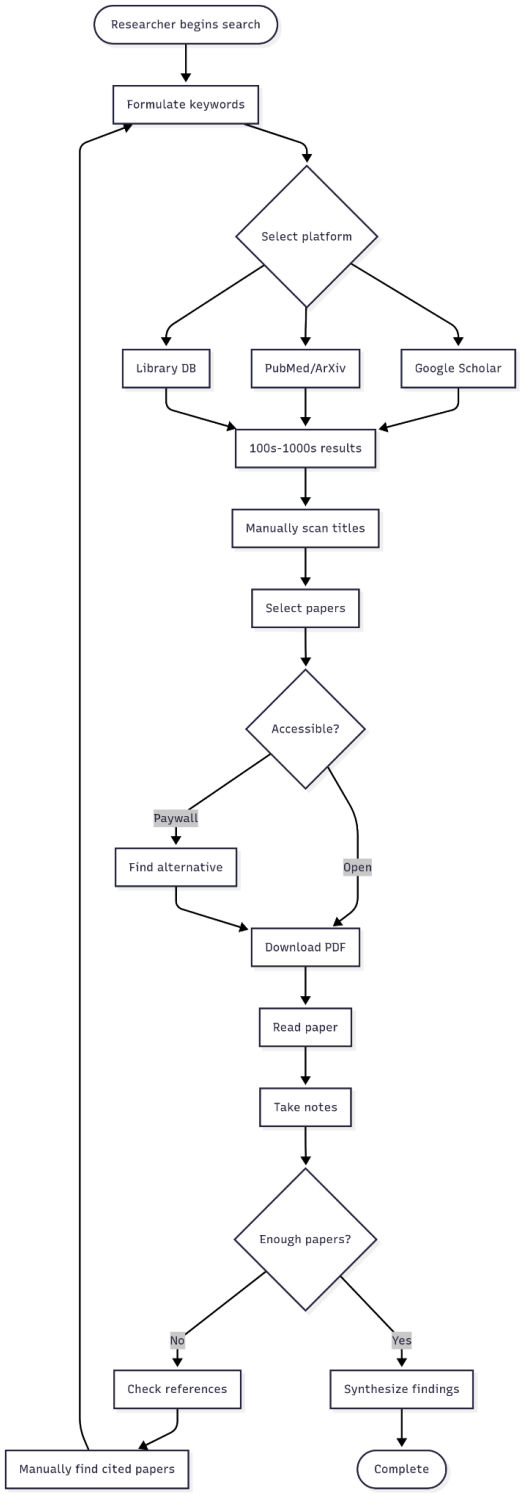
5.3 Proposed Solutions (CiteConnect)

CiteConnect builds on these gaps by combining **what exists** and **what is missing**:

Semantic + Graph Hybrid: Like Connected Papers, CiteConnect shows citation networks, but it adds **semantic retrieval** via embeddings to find papers beyond just citation links.
Explainability: Each recommendation includes an LLM-generated summary of its relevance and its connection to the query.
Personalization: User accounts and research interest profiles adapt recommendations over time, unlike existing static graph explorers.
MLOps Foundation: Unlike Connected Papers or Research Rabbit (pure applications), CiteConnect is built as a **full end-to-end MLOps pipeline** with data versioning, CI/CD, monitoring, and scalability.

6. Current Approach Flow chart and Bottleneck Detection

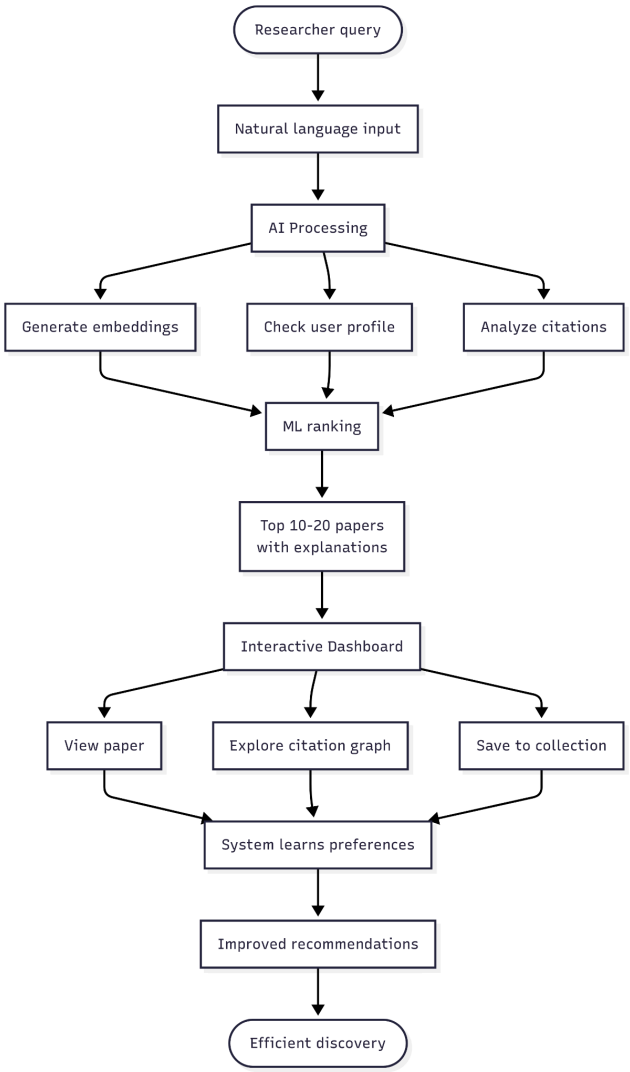
6.1 Flowchart



6.2 Potential Bottlenecks

Bottleneck	Current Impact	CiteConnect Solution
1. Keyword Dependency	Miss 40% relevant papers using different terms	Semantic search understands concepts, not just keywords
2. Information Overload	3+ hours scanning 100s of results	AI ranks top 10-20 with explanations
3. Manual Citation Tracking	2+ hours to map 20 paper connections	Instant interactive citation graph
4. No Personalization	Same results for all users	Adaptive recommendations based on profile
5. Fragmented Workflow	10+ tool switches per session	Unified platform for all tasks

6.3 Improved workflow



7. Metrics, Objectives, and Business Goals

7.1 Technical Metrics

Retrieval Quality Metrics:

- Recall@10: Target ≥ 0.75 (percentage of relevant papers in top 10 results)
- Precision@10: Target ≥ 0.60 (percentage of top 10 results that are relevant)
- Mean Reciprocal Rank (MRR): Target ≥ 0.70 (average rank of first relevant result)
- NDCG@10: Target ≥ 0.65 (normalized discounted cumulative gain)

System Performance Metrics:

- Query Latency (p95): Target < 2 seconds, alert if > 3 seconds
- Query Latency (p99): Target < 4 seconds, alert if > 5 seconds
- System Availability: Target 99.5%, alert if < 99%
- PDF Processing Time: Target < 3 minutes per paper, alert if > 5 minutes
- Embedding Generation: Target < 500ms per query, alert if > 1 second

ML Model Metrics:

- Embedding Quality: Cosine similarity > 0.7 for same-topic papers
- Summarization Quality (ROUGE-L): Target ≥ 0.45
- Citation Link Prediction: Accuracy ≥ 0.80

7.2 User Engagement Metrics

- Click-Through Rate (CTR): Target ≥ 25%
- Paper View Time: Target ≥ 3 minutes average
- Citation Graph Interaction: Target ≥ 40% of users
- Return User Rate (7-day): Target ≥ 35%
- Papers Saved/Bookmarked: Target ≥ 2 per session

7.3 MLOps Metrics

- Model Deployment Frequency: Weekly updates
- Pipeline Success Rate: Target ≥ 95%
- Mean Time to Recovery (MTTR): Target < 30 minutes
- Model Training Time: Target < 2 hours
- CI/CD Pipeline Duration: Target < 15 minutes

7.4 Cost Metrics

- Compute Cost: < $200/month (monitored via GCP billing alerts)
- API Costs (OpenAI): < $100/month (controlled via request rate limiting)
- Storage Cost: < $50/month (managed via data lifecycle policies)
- Total Monthly Budget: < $350/month (reviewed weekly)

7.5 Objectives

Project Objective	Description	Alignment with Business Goals
End-to-End MLOps Pipeline	Build a reproducible, automated, and scalable pipeline covering ingestion → preprocessing → embeddings → retrieval → visualization → deployment → monitoring.	Demonstrates modern MLOps practices, ensuring reproducibility and scalability valued in industry and academia.
Semantic Search & Recommendations	Use embeddings and RAG to recommend contextually relevant papers beyond simple keyword matches.	Boosts research productivity by surfacing the most relevant work faster than traditional keyword-based search.
Connected Paper Visualization	Implement an interactive citation graph to explore relationships and clusters among papers.	Provides scalable knowledge management and innovation in academic tools by making literature connections intuitive.
Explainability	Provide LLM-generated summaries that justify why each paper is recommended and how it connects to the query.	Builds trust and transparency, helping researchers quickly evaluate paper relevance.
Personalization	Support researcher profiles so recommendations adapt to user interests and fields of study.	Aligns with user-centric business goals, enabling tailored solutions for universities, R&D teams, and SaaS platforms.
Scalable Deployment	Deliver a web app with FastAPI backend, Streamlit frontend, and cloud-native deployment (Docker + Kubernetes on GCP).	Ensures commercial potential by proving the system can scale for enterprise and institutional use.

8. Failure Analysis

Pipeline Stage	Risks	Mitigation Strategy
Data Ingestion	API downtime, schema changes, or missing metadata from arXiv/Semantic Scholar.	Add retry logic, cache data locally, maintain backup dumps, and validate schema regularly.
Data Parsing	Parsing errors due to formatting inconsistencies or corrupted PDFs.	Use robust parsers (PyMuPDF, GROBID), log failures, and retry with alternate libraries.
Model/Embeddings	Embedding generation may exceed API quotas, cause high costs, or crash due to memory.	Batch embeddings, monitor costs, and fall back to local models (sentence-transformers).
Storage/Retrieval	Vector database may suffer index corruption or retrieval latency under heavy load.	Enable index backups, shard data, monitor latency, and autoscale DB resources.
Graph Layer	Citation graph may miss references or encounter query failures.	Validate reference nodes, retry failed queries, and use fallback graph libraries (e.g., NetworkX).
Deployment (API/UI)	Backend/frontend services may crash or fail under high request loads.	Containerize with Docker, scale using Kubernetes, set health checks, and apply rate limiting.
Monitoring	Monitoring gaps may lead to undetected drift or pipeline failures.	Integrate Prometheus/Grafana alerts and perform periodic model evaluations.
Scalability	Latency spikes may occur when scaling to millions of papers or many users.	Optimize indexing, enable autoscaling, and use distributed vector DBs.
Model Lifecycle	New research topics may cause model drift and reduce retrieval accuracy.	Refresh embeddings regularly, retrain models, and monitor accuracy metrics.
Security	Security vulnerabilities (exposed API keys, data leaks, weak authentication).	Use Google Secret Manager, enforce HTTPS, apply role-based access, and conduct code reviews.
User Experience	Poor recommendation quality or lack of explainability may reduce user trust.	Provide LLM-based summaries, allow user feedback, and refine ranking metrics.

9. Deployment Infrastructure

CiteConnect follows a microservices architecture deployed on cloud infrastructure, designed for scalability, reliability, and cost-effectiveness. The system separates concerns into distinct services that can be independently scaled and maintained.

9.1.1 Frontend Layer

Primary Technology Stack:

- **Framework**: React 18 with TypeScript
- **UI Library**: Material-UI / Ant Design
- **State Management**: Redux Toolkit / Zustand
- **Graph Visualization**: D3.js / Cytoscape.js

Alternative Technologies:

- Vue.js 3 with Vuetify
- Angular 15+ with Angular Material
- Next.js for SSR capabilities
- Svelte for lightweight alternative

Deployment:

- Static hosting on Google Cloud Storage with CDN
- Alternative: AWS S3 + CloudFront, Netlify, Vercel

9.1.2 API Gateway Layer

Primary Technology Stack:

- **Framework**: FastAPI (Python)
- **Authentication**: JWT with OAuth 2.0
- **Rate Limiting**: Redis-based throttling
- **API Documentation**: OpenAPI/Swagger

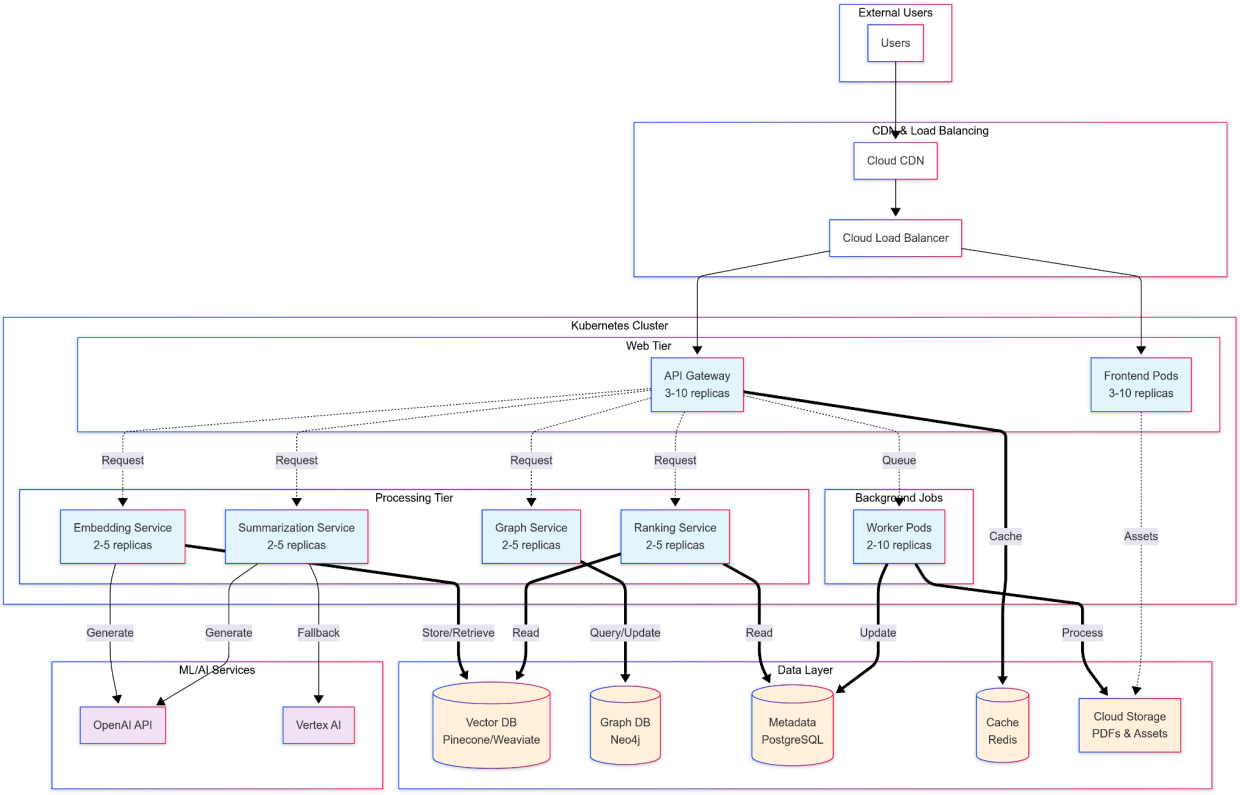
Alternative Technologies:

- Node.js with Express/NestJS
- Django REST Framework
- Go with Gin/Echo
- Spring Boot (Java)

Deployment:

- Containerized on Google Kubernetes Engine (GKE)
- Alternative: AWS EKS, Azure AKS, DigitalOcean Kubernetes

9.1.3 Microservices Architecture



9.1.4 Service Specifications

Embedding Service

- **Purpose**: Generate vector embeddings for papers and queries
- **Primary**: OpenAI text-embedding-ada-002
- **Alternatives**:
    - Sentence-transformers (all-MiniLM-L6-v2)
    - Google's Universal Sentence Encoder
    - Cohere Embed API
- **Deployment**: Stateless pods with horizontal autoscaling

Summarization Service

- **Purpose**: Generate paper summaries and explanations
- **Primary**: OpenAI GPT-4 API
- **Alternatives**:
    - Google Vertex AI (PaLM 2)
    - Anthropic Claude API

    - Self-hosted LLama 2 / Mistral
- **Deployment**: Queue-based processing with rate limiting

Graph Service

- **Purpose**: Manage citation relationships and graph queries
- **Primary**: Neo4j Graph Database
- **Alternatives**:
    - Amazon Neptune
    - ArangoDB
    - TigerGraph
- **Deployment**: StatefulSet with persistent storage

Ranking Service

- **Purpose**: ML-based paper ranking and personalization
- **Primary**: Custom Python ML pipeline
- **Technologies**: scikit-learn, XGBoost, TensorFlow
- **Deployment**: CPU-optimized pods with model versioning

9.2 Data Storage Architecture

9.2.1 Vector Database

Primary: Pinecone

- Managed service, 10M+ vector capacity
- 768-dimension embeddings
- Metadata filtering support

Alternatives:

- **Weaviate**: Open-source, self-hosted option
- **Qdrant**: High-performance with good filtering
- **FAISS**: Facebook's library for local deployment
- **Milvus**: Scalable open-source solution

9.2.2 Graph Database

Primary: Neo4j

- Cypher query language
- ACID compliance
- Clustering support

Alternatives:

- **ArangoDB**: Multi-model database
- **JanusGraph**: Distributed graph database
- **Amazon Neptune**: Managed graph service
- **Azure Cosmos DB**: With Gremlin API

9.2.3 Relational Database

Primary: PostgreSQL

- User profiles, metadata, system data
- Cloud SQL managed instance
- Read replicas for scaling

Alternatives:

- **MySQL**: Alternative RDBMS
- **MongoDB**: For document-oriented approach
- **CockroachDB**: Distributed SQL
- **Aurora PostgreSQL**: AWS managed

9.2.4 Cache Layer

Primary: Redis

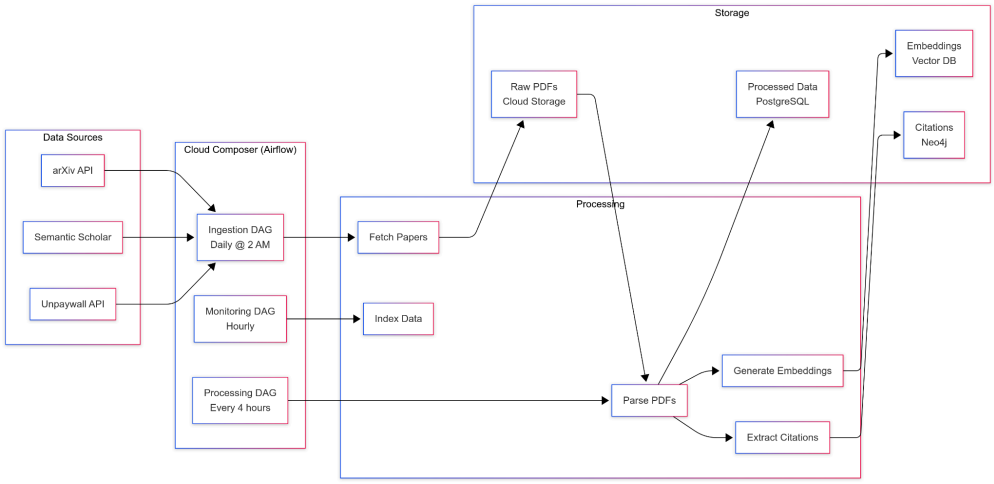
- Session management
- Query result caching
- Rate limiting counters

Alternatives:

- **Memcached**: Simpler caching
- **Hazelcast**: Distributed cache
- **Apache Ignite**: In-memory computing

9.3 Pipeline Infrastructure

9.3.1 Orchestration



Primary: Cloud Composer (Managed Airflow)

- DAG scheduling and monitoring
- Automatic retries and alerting
- Integration with GCP services

Alternatives:

- **Apache Airflow**: Self-hosted
- **Prefect**: Modern Python orchestration
- **Dagster**: Data orchestration platform
- **Temporal**: Workflow orchestration
- **Apache NiFi**: Data flow automation

9.3.2 Message Queue

Primary: Google Pub/Sub

- Asynchronous task processing
- At-least-once delivery guarantee
- Auto-scaling subscribers

Alternatives:

- **RabbitMQ**: Self-hosted message broker
- **Apache Kafka**: Event streaming platform
- **AWS SQS**: Managed queue service
- **Redis Streams**: Lightweight option

9.4 CI/CD Pipeline

Primary: Cloud Build + GitHub Actions

Alternatives:

- **GitLab CI/CD**: Integrated platform
- **Jenkins**: Self-hosted automation
- **CircleCI**: Cloud-based CI/CD
- **ArgoCD**: GitOps for Kubernetes

9.5 Monitoring & Observability Stack

9.5.1 Metrics & Monitoring

Primary Stack:

- **Prometheus**: Metrics collection
- **Grafana**: Visualization dashboards
- **AlertManager**: Alert routing

Alternatives:

- **Datadog**: Full observability platform
- **New Relic**: APM and monitoring
- **Elastic Stack**: ELK for logs and metrics

9.5.2 Logging

Primary:

- **Cloud Logging**: Centralized GCP logging
- **Fluentd**: Log aggregation

Alternatives:

- **ELK Stack**: Elasticsearch, Logstash, Kibana
- **Splunk**: Enterprise logging
- **Loki**: Lightweight log aggregation

9.5.3 Tracing

Primary:

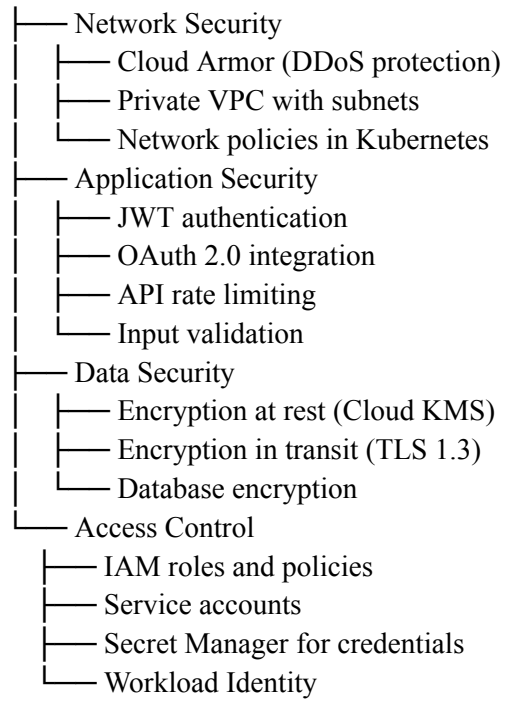
- **Cloud Trace**: Distributed tracing
- **OpenTelemetry**: Instrumentation

Alternatives:

- **Jaeger**: Open-source tracing
- **Zipkin**: Distributed tracing system
- **AWS X-Ray**: AWS native tracing

9.6 Security Infrastructure

9.6.1 Security Layers



10. Monitoring Plan

The monitoring plan for **CiteConnect** ensures that the system remains **reliable, accurate, and scalable** after deployment. Since CiteConnect is a full MLOps pipeline, monitoring must cover **data quality, model performance, system health, and user experience**. Continuous monitoring will help detect failures early, track drift in research trends, and maintain trust in recommendations.

Monitoring Area	What to Monitor	Why It Matters
Data Quality	- API availability and response times - Missing/duplicate metadata - PDF parsing success rate	Ensures ingestion is consistent, prevents corrupted or incomplete datasets, and reduces noise in recommendations.
Embeddings & Models	- Embedding generation success rate - Latency per embedding request - Distribution drift in embeddings vs. past data	Detects drift in research language/topics, prevents API quota overruns, and maintains relevance of recommendations.
Retrieval Accuracy	- Recall@k and Precision@k - Click-through rate (CTR) on recommendations - Time-to-first-result	Measures how effectively the system returns relevant papers and whether users engage with results.
Citation Graph	- Graph completeness (references per paper) - Query latency for graph exploration	Ensures the connected papers feature remains accurate and responsive.
System Health	- API uptime and response times - Streamlit frontend availability - Container and pod health (CPU, memory)	Guarantees smooth operation, supports scaling, and prevents downtime.
Scalability	- Query latency under load - Number of active users - Database performance (FAISS/Neo4j)	Ensures the platform scales reliably for more users and larger datasets.
User Behavior & Trust	- Feedback on recommendation relevance - Drop-off rate (abandoned searches) - Usage patterns (popular queries, fields of interest)	Improves personalization, increases adoption, and identifies weak spots in explainability.
Security	- Unauthorized access attempts - API key misuse or leakage - SSL/TLS certificate validity	Protects sensitive access keys, ensures secure data transfer, and prevents system breaches.
Security	- Unauthorized access attempts - API key misuse or leakage - SSL/TLS certificate validity	Protects sensitive access keys, ensures secure data transfer, and prevents system breaches.

11. Success and Acceptance Criteria

11.1 Technical Success Criteria

System Performance:

- API response time (p95) must be below 2 seconds for 95% of requests
- System availability must achieve 99.5% uptime over evaluation period
- PDF processing pipeline must successfully process at least 85% of papers
- Embedding generation must complete in under 500 milliseconds per query
- Vector search must return results in under 1 second

Model Performance:

- Recall@10 must achieve at least 0.75 on held-out test set
- Precision@10 must achieve at least 0.60 on held-out test set
- Mean Reciprocal Rank must achieve at least 0.70
- Summarization quality using ROUGE-L must score at least 0.45
- Citation graph accuracy must match Semantic Scholar within 20%

Infrastructure:

- All services must be containerized using Docker
- Kubernetes deployment must include at least 3 replicas of API service
- CI/CD pipeline must complete in under 15 minutes
- Automated testing must cover at least 80% of code
- Data versioning with DVC must track all datasets

11.2 Functional Success Criteria

Core Features:

- Users can search for papers using natural language queries
- System returns ranked list of relevant papers with explanations
- Interactive citation graph visualization shows paper connections
- User authentication and profile management functional
- Personalized recommendations based on user research interests
- Paper bookmarking and history tracking
- Email notifications for saved searches

Data Pipeline:

- Automated daily ingestion of new papers from arXiv
- Weekly embedding updates for entire corpus
- Citation graph automatically updated with new papers
- Data quality validation runs on all ingested papers
- Failed papers logged and flagged for manual review

11.3 User Experience Success Criteria

Usability:

- Users can find relevant papers within 3 clicks
- Search results load in under 2 seconds for 95% of queries
- Citation graph renders in under 3 seconds
- Mobile responsive design for all interfaces
- Clear error messages guide users when issues occur

Engagement:

- New users successfully complete first search within 2 minutes
- Click-through rate on recommendations exceeds 25%
- Users bookmark at least 2 papers per session on average
- At least 40% of users explore citation graph feature
- Return user rate within 7 days exceeds 35%

11.4 MLOps Success Criteria

Automation:

- Data ingestion pipeline runs automatically daily without manual intervention
- Model retraining triggered automatically when performance degrades
- CI/CD pipeline deploys updates without downtime
- Monitoring alerts automatically notify team of issues
- Backup and disaster recovery procedures automated

Observability:

- All services expose health check endpoints
- Prometheus collects metrics from all components
- Grafana dashboards display real-time system status
- Logs centralized in Cloud Logging with structured format
- Distributed tracing tracks requests across services

Reproducibility:

- All experiments tracked in MLflow with full metadata
- Model artifacts versioned and stored in model registry
- Training data versioned using DVC
- Infrastructure defined as code using Terraform
- Documentation includes step-by-step reproduction instructions

11.5 Course Evaluation Success Criteria

Scoping (15% of grade):

- Complete project scoping document addressing all required sections
- Clear problem definition with existing solutions analysis
- Detailed data planning with source identification
- Comprehensive failure analysis and risk mitigation
- Deployment infrastructure fully specified

Data Pipeline (25% of grade):

- Functional data ingestion from multiple sources
- PDF parsing and text extraction working reliably
- Data versioning implemented using DVC
- Automated preprocessing pipeline with quality checks
- Storage architecture using GCP services

Model Pipeline (25% of grade):

- Embedding generation pipeline functional
- Vector similarity search operational
- Re-ranking and personalization implemented
- LLM integration for explanations working
- Model performance monitoring in place
- Experiment tracking using MLflow

Deployment (25% of grade):

- System deployed on GCP using Kubernetes
- CI/CD pipeline functional with GitHub Actions
- Monitoring and alerting fully configured
- Health checks and auto-scaling working
- Documentation complete with runbooks

Expo Presentation (10% - part of overall evaluation):

- Live demo of deployed system running on GCP
- Technical walkthrough explaining architecture and pipelines
- Monitoring dashboards showing real-time performance
- CI/CD demonstration with automated deployment
- Q&A demonstrating deep understanding of system

11.6 Acceptance Criteria for MVP Release

Must Have for MVP:

- Search functionality with at least 10,000 papers indexed
- Vector similarity search returning relevant results
- Basic citation graph with at least 50% of papers linked
- User authentication and profile creation
- Deployed on GCP and accessible via public URL
- Monitoring dashboards showing key metrics
- Basic CI/CD pipeline for deployments

Nice to Have for MVP:

- Advanced personalization features
- Email notifications for saved searches
- Mobile responsive UI
- Social sharing features

- Export citations in multiple formats

11.7 Demo Day Success Criteria

For MLOps Innovation Expo Presentation:

- System must be live and accessible during demo
- At least 10,000 papers indexed and searchable
- All core features functional (search, recommendations, citation graph, user profiles)
- Monitoring dashboards showing real-time metrics
- Ability to demonstrate end-to-end workflow from user query to results
- Evidence of MLOps best practices (CI/CD, monitoring, versioning)
- Clear explanation of technical architecture and design decisions
- Answers to judge questions demonstrate team understanding
- No critical bugs or service outages during demo
- Positive feedback from at least 75% of attendees
- Recognition from judges for technical execution

12. Timeline Planning

12.1 Project Timeline Overview

The CiteConnect project spans 14 weeks aligned with the course schedule, with four major phases: Scoping and Planning (Weeks 1-2), Data Pipeline Development (Weeks 3-6), Model Pipeline Development (Weeks 7-10), and Deployment and Monitoring (Weeks 11-14). Each phase includes specific deliverables and milestones.

12.2 Phase 1: Scoping and Planning (Weeks 1-2)

Week 1 Tasks:

- Define project scope and objectives
- Research existing solutions and identify gaps
- Select technology stack and architecture
- Set up GitHub repository with folder structure
- Create initial README and documentation
- Initialize GCP project and enable billing
- Set up development environments for team

Week 1 Deliverables: Project scoping document submitted; GitHub repository structure established; GCP project configured; Team roles and responsibilities defined

Week 2 Tasks:

- Finalize data sources and API access
- Design system architecture diagrams
- Create detailed data pipeline specifications
- Set up DVC for data versioning
- Configure initial CI/CD pipeline skeleton
- Begin data exploration and API testing

- Document data rights and privacy considerations

Week 2 Deliverables: System architecture documented; Data card and data sources finalized; API access confirmed (arXiv + Semantic Scholar); Initial data samples collected and analyzed

12.3 Phase 2: Data Pipeline Development (Weeks 3-6)

Week 3 Tasks:

- Implement data ingestion scripts for arXiv API
- Implement data ingestion scripts for Semantic Scholar API
- Set up Google Cloud Storage buckets
- Create PDF download and storage pipeline
- Implement basic error handling and logging
- Begin PDF parsing exploration with multiple libraries

Week 3 Deliverables: Data ingestion scripts functional; 1,000 papers successfully downloaded; GCS storage configured and tested

Week 4 Tasks:

- Implement robust PDF parsing pipeline
- Create text cleaning and preprocessing functions
- Develop chunking strategy for long documents
- Implement data quality validation checks
- Set up DVC pipeline for version control
- Create unit tests for data pipeline components

Week 4 Deliverables: PDF parsing achieving 85% success rate; Text preprocessing pipeline functional; Data validation framework implemented; 5,000 papers processed and validated

Week 5 Tasks:

- Implement embedding generation using sentence-transformers
- Set up vector database using FAISS
- Create embedding storage and indexing pipeline
- Implement citation extraction from papers
- Begin Neo4j graph database setup
- Create metadata storage in Cloud SQL

Week 5 Deliverables: Embedding generation pipeline operational; Vector database with 5,000 paper embeddings; Citation extraction working for majority of papers; Neo4j instance running with sample data

Week 6 Tasks:

- Complete citation graph construction
- Optimize vector search performance
- Implement batch embedding updates
- Create Airflow DAGs for automated pipeline
- Set up incremental data processing
- Complete integration testing of entire pipeline

- Optimize pipeline for cost and performance

Week 6 Deliverables: Complete data pipeline from ingestion to storage; 10,000 papers fully processed and indexed; Automated Airflow pipeline running daily; Citation graph with interconnected papers; Data pipeline documentation complete

12.4 Phase 3: Model Pipeline Development (Weeks 7-10)

Week 7 Tasks:

- Implement vector similarity search API
- Create query embedding pipeline
- Develop ranking algorithm for results
- Implement basic recommendation engine
- Set up MLflow for experiment tracking
- Create evaluation framework with test queries
- Begin personalization model development

Week 7 Deliverables: Search functionality returning relevant results; MLflow tracking experiments; Baseline recommendation system functional; Test set with ground truth created

Week 8 Tasks:

- Integrate LLM for explanation generation
- Implement personalization re-ranking
- Develop user profile management
- Create collaborative filtering components
- Optimize model inference latency
- Implement caching for common queries
- Run A/B testing framework setup

Week 8 Deliverables: Personalized recommendations working; LLM generating explanations for results; User profiles affecting recommendations; Model achieving target Recall@10 of 0.75

Week 9 Tasks:

- Implement advanced re-ranking algorithms
- Create diversity injection for recommendations
- Develop cold start handling for new users
- Optimize embedding search for scale
- Implement model versioning in MLflow
- Create model evaluation dashboard
- Fine-tune all model components

Week 9 Deliverables: Advanced recommendation algorithms implemented; Cold start problem addressed; Model performance meeting all targets; Comprehensive evaluation results documented

Week 10 Tasks:

- Integrate all model components
- Implement citation graph traversal queries

- Create interactive visualization components
- Optimize end-to-end inference pipeline
- Complete model pipeline testing
- Document model architecture and decisions
- Prepare model artifacts for deployment

Week 10 Deliverables: Complete model pipeline operational; All model components integrated and tested; Model artifacts ready for production; Model documentation complete

12.5 Phase 4: Deployment and Monitoring (Weeks 11-14)

Week 11 Tasks:

- Containerize all services with Docker
- Create Kubernetes deployment manifests
- Set up GKE cluster on GCP
- Deploy vector database to Kubernetes
- Deploy Neo4j to Kubernetes
- Configure Cloud SQL and connections
- Implement Redis caching layer

Week 11 Deliverables: All services containerized; Kubernetes cluster operational; Database services deployed and accessible; Caching layer functional

Week 12 Tasks:

- Deploy FastAPI application to GKE
- Configure load balancer and ingress
- Set up Prometheus for metrics collection
- Configure Grafana dashboards
- Implement health checks and readiness probes
- Set up automated scaling policies
- Configure SSL/TLS certificates
- Complete CI/CD pipeline for automated deployment

Week 12 Deliverables: Full application deployed on GKE; Monitoring and alerting operational; CI/CD pipeline deploying automatically; System accessible via public URL

Week 13 Tasks:

- Implement comprehensive monitoring strategy
- Configure alert rules and notifications
- Create runbooks for common issues
- Conduct load testing and performance tuning
- Implement automated backup procedures
- Complete security hardening
- Conduct end-to-end integration testing
- Fix bugs and optimize performance

Week 13 Deliverables: Monitoring covering all components; Alert rules configured and tested; System passing load tests; Security audit completed; All critical bugs resolved

Week 14 Tasks:

- Prepare demo for MLOps Innovation Expo
- Create presentation slides
- Practice live demo walkthrough
- Prepare technical architecture explanation
- Create demo video as backup
- Final performance optimization
- Prepare Q&A responses
- Complete all documentation

Week 14 Deliverables: Expo presentation ready; Live demo working flawlessly; All documentation finalized; System stable and performant

12.6 Key Milestones

Milestone 1 (Week 2): Project scoping approved and data sources confirmed

Milestone 2 (Week 6): Data pipeline complete with 10,000 papers indexed

Milestone 3 (Week 10): Model pipeline achieving performance targets

Milestone 4 (Week 12): System deployed to production on GCP

Milestone 5 (Week 14): Successful demo at MLOps Innovation Expo

12.7 Risk Buffer and Contingencies

Built-in buffer time in weeks 13-14 for unexpected issues. Alternative approaches identified for high-risk components. Weekly team check-ins to identify blockers early.

Scope Reduction Plan (if timeline at risk):

1. Remove advanced personalization features first
2. Simplify LLM integration if needed
3. Reduce initial dataset size if processing slow
4. Defer mobile UI if time constrained