Unified AI-Powered Job Discovery, Intelligent Recruitment & Agentic Application Platform

Implementation Plan — B.Tech Final Year Project (AI & Data Science)

---

1. Executive Summary

This document presents a research-grade implementation plan for a **dual-sided AI platform** serving job seekers and recruiters. The system leverages advanced AI/ML/DL techniques including:

- Large Language Models (LLMs) for semantic understanding
- Transformer-based embeddings for matching
- Learning-to-Rank models with feedback loops
- Anomaly detection for fraud identification
- Multi-agent autonomous systems for job application

This is **not a basic job portal**—it is a comprehensive AI research platform with explainability, fairness monitoring, and agentic autonomy.

---

2. High-Level System Architecture

```
graph TB
    subgraph "Frontend Layer"
        JS_WEB[Job Seeker Web App]
        REC_WEB[Recruiter Web App]
        CHAT[Conversational AI Interface]
    end

    subgraph "API Gateway"
        GW[FastAPI Gateway]
        AUTH[Auth Service]
        RBAC[Role-Based Access Control]
    end

    subgraph "Core Services"
        USER_SVC[User Service]
        JOB_SVC[Job Service]
        RESUME_SVC[Resume Service]
        MATCH_SVC[Matching Engine]
        AGENT_SVC[Agentic AI Service]
    end
```

```
subgraph "AI/ML Layer"
    EMB[Embedding Service]
    LLM[LLM Service]
    RANK[Ranking Service]
    FRAUD[Fraud Detection]
    FAIR[Fairness Monitor]
    XAI[Explainability Engine]
end

subgraph "Data Layer"
    PG[(PostgreSQL)]
    REDIS[(Redis Cache)]
    VECTOR[(Qdrant Vector DB)]
    MONGO[(MongoDB)]
    S3[MinIO/S3 Storage]
end

subgraph "External Integrations"
    LINKEDIN[LinkedIn API]
    INDEED[Indeed API]
    NAUKRI[Naukri Feed]
    OTHER[Other Job APIs]
end

JS_WEB --> GW
REC_WEB --> GW
CHAT --> GW
GW --> AUTH --> RBAC
GW --> USER_SVC
GW --> JOB_SVC
GW --> RESUME_SVC
GW --> MATCH_SVC
GW --> AGENT_SVC

JOB_SVC --> LINKEDIN
JOB_SVC --> INDEED
JOB_SVC --> NAUKRI
JOB_SVC --> OTHER

MATCH_SVC --> EMB
MATCH_SVC --> RANK
RESUME_SVC --> LLM
RESUME_SVC --> FRAUD
AGENT_SVC --> LLM
```

```
USER_SVC --> PG
JOB_SVC --> PG
JOB_SVC --> MONGO
RESUME_SVC --> S3
EMB --> VECTOR
AUTH --> REDIS
```

---

3. Module-Wise Breakdown

Module 1: User & Access Management

Component	Description
Authentication	JWT-based auth with OAuth2 (Google, LinkedIn SSO)
Authorization	RBAC with roles: `job_seeker`, `recruiter`, `admin`
Profile Management	Separate schemas for seeker/recruiter profiles
Security	bcrypt password hashing, rate limiting, HTTPS enforcement

Database Schema (PostgreSQL):

```
-- Users table with polymorphic profiles
users (id, email, password_hash, role, created_at, verified)
job_seeker_profiles (user_id, skills[], experience_years, education, preferences_json)
recruiter_profiles (user_id, company_id, designation, verified_recruiter)
companies (id, name, domain, size, verified)
```

AI/ML: None directly; serves as foundation for personalization features.

Module 2: Job Aggregation & APIs

Component	Description
API Integrations	LinkedIn, Indeed, Glassdoor, Naukri partner feeds
Rate Limiting	Token bucket algorithm, respect API quotas
Duplicate Detection	MinHash + LSH for near-duplicate job detection

Component	Description
Schema Normalization	Transform heterogeneous job formats to unified schema

AI/ML Models:

Model	Type	Purpose	Training
Duplicate Detection	MinHash + Locality Sensitive Hashing	Near-duplicate job identification	Unsupervised
Semantic Deduplication	Sentence-BERT similarity	Catch paraphrased duplicates	Pretrained (all-MiniLM-L6-v2)

Data Pipeline:

```
flowchart LR
    A[Job APIs] --> B[Rate-Limited Fetcher]
    B --> C[Raw Job Storage - MongoDB]
    C --> D[Deduplication Pipeline]
    D --> E[Schema Normalizer]
    E --> F[PostgreSQL - Jobs Table]
    E --> G[Vector DB - Job Embeddings]
```

Ethical Constraints: - API terms of service compliance - Rate limiting (max 100 requests/minute per source) - No web scraping; only official APIs/feeds - Data retention policies (delete after 90 days if job closed)

Module 3: Job Search & Conversational Interface

Component	Description
Traditional Search	Elasticsearch with filters (location, salary, experience)
Semantic Search	Dense vector retrieval using embeddings
Conversational AI	LLM-powered chatbot with RAG architecture
Context-Aware Search	Combines resume embeddings + query + preferences

AI/ML Models:

Model	Type	Purpose	Training
Query Encoder	Sentence-BERT	Encode natural language queries	Pretrained + fine-tuned on job domain
Retrieval Model	Dense Passage Retrieval (DPR)	Semantic job retrieval	Fine-tuned on job-query pairs
Conversational LLM	Mistral-7B / Llama-3-8B	Natural language understanding	Pretrained + instruction-tuned
Reranker	Cross-Encoder (ms-marco-MiniLM)	Rerank retrieved jobs	Fine-tuned on relevance labels

Conversational AI Architecture:

```
flowchart TB
    USER[User Query] --> INTENT[Intent Classification]
    INTENT --> |Search Intent| QUERY_ENC[Query Encoder]
    INTENT --> |Clarification| LLM[LLM Response]
    QUERY_ENC --> HYBRID[Hybrid Retrieval]
    HYBRID --> |BM25| ES[Elasticsearch]
    HYBRID --> |Dense| VECTOR[Vector Search]
    ES --> FUSION[Reciprocal Rank Fusion]
    VECTOR --> FUSION
    FUSION --> RERANK[Cross-Encoder Reranking]
    RERANK --> CONTEXT[Context Builder]
    CONTEXT --> LLM
    LLM --> RESPONSE[Natural Language Response]
```

Sample Queries Supported: - "Remote ML jobs suitable for my profile" → Resume-aware semantic search - "Jobs where I have high chance" → Matching score + cold-start reasoning - "Show Python jobs with >10 LPA salary" → Hybrid structured + semantic

---

Module 4: Resume Upload, Parsing & Analysis

Component	Description
File Handling	PDF/DOCX upload, virus scanning, size limits
Parsing	Multi-model extraction pipeline
NER	Custom Named Entity Recognition for resume entities
Skill Extraction	Taxonomy-aware skill normalization

Component	Description
Quality Scoring	Multi-criteria resume quality assessment

AI/ML Models:

Model	Type	Purpose	Training
Resume Parser	LayoutLMv3	Extract structured data from resume layout	Fine-tuned on annotated resumes
Skill NER	SpaCy + Custom NER	Extract skills, technologies	Fine-tuned on skill taxonomy
Experience Parser	Regex + LLM Hybrid	Parse work history	Rule-based + LLM fallback
Resume Embedder	BERT-base	Generate resume embeddings	Fine-tuned for semantic similarity
Quality Scorer	Gradient Boosting (XGBoost)	Score resume quality	Trained on labeled resume quality data

Resume Quality Scoring Criteria:

```
quality_dimensions = {
    "completeness": 0.25,       # All sections present
    "formatting": 0.15,         # Consistent formatting
    "quantification": 0.20,     # Metrics in achievements
    "keyword_density": 0.15,    # Relevant keywords
    "grammar": 0.15,            # Language quality
    "ats_compatibility": 0.10   # Parseable structure
}
```

Data Storage: - Raw files → MinIO/S3 - Parsed JSON → MongoDB - Embeddings → Qdrant Vector DB - Metadata → PostgreSQL

Module 5: AI-Optimized Resume Generation System

Component	Description
Job-Specific Generation	Tailor resume for specific JD
Content Rewriting	Preserve facts while improving language
ATS Optimization	Ensure machine parseability
Bias Removal	Remove potentially discriminatory signals

Component	Description
Version Management	Maintain multiple resume versions

AI/ML Models:

Model	Type	Purpose	Training
Resume Rewriter	Flan-T5-XL / Llama-3-8B	Generate optimized resume text	Instruction-tuned + RLHF
Keyword Injector	Sequence-to-Sequence	Add missing JD keywords naturally	Fine-tuned on resume-JD pairs
ATS Scorer	Rule-based + ML Ensemble	Score ATS compatibility	Trained on ATS parsing results
Bias Detector	BERT Classifier	Detect bias-prone content	Fine-tuned on fairness dataset

Bias-Safe Generation:

```
# Signals removed/anonymized
bias_signals = [
    "graduation_year",        # Age proxy
    "college_name",           # Tier bias
    "gender_indicators",      # Name, pronouns
    "photo",                  # Visual bias
    "address",                # Location discrimination
    "marital_status"          # Personal info
]
```

Generation Pipeline:

```
flowchart LR
    A[Original Resume] --> B[Parse & Structure]
    C[Target JD] --> D[Extract Requirements]
    B --> E[Gap Analysis]
    D --> E
    E --> F[Content Strategy]
    F --> G[LLM Resume Generator]
    G --> H[Fact Verification]
    H --> I[ATS Optimization]
    I --> J[Bias Removal]
    J --> K[Final Resume]
```

Module 6: Job–Candidate Matching & Ranking

Component	Description
Semantic Matching	Embedding-based similarity (not keyword matching)
Learning-to-Rank	Trained on recruiter feedback
Cold-Start Handling	Special handling for freshers
Multi-Criteria Ranking	Balanced skill, experience, culture fit

AI/ML Models:

Model	Type	Purpose	Training
Bi-Encoder	Sentence-BERT	Generate resume & job embeddings	Fine-tuned on match pairs
Cross-Encoder	RoBERTa-base	Precise relevance scoring	Fine-tuned on labeled pairs
LTR Model	LambdaMART (XGBoost)	Learn ranking from feedback	Trained on pairwise preferences
Cold-Start Model	Content-based + NCF	Handle no-history candidates	Hybrid collaborative filtering

Matching Score Computation:

```
def compute_match_score(resume_emb, job_emb, features):
    # Semantic similarity component
    semantic_sim = cosine_similarity(resume_emb, job_emb)

    # Skill overlap component
    skill_overlap = len(resume_skills & job_skills) / len(job_skills)

    # Experience fit component
    exp_fit = gaussian_kernel(candidate_exp, job_required_exp)

    # Final score with learned weights
    return ltr_model.predict([semantic_sim, skill_overlap, exp_fit, ...])
```

Cold-Start Strategy for Freshers: 1. Emphasis on education quality signals 2. Project/internship skill inference 3. Skill-based matching over experience 4. Potential scoring using college placement data

Module 7: Explainable Rejection Feedback

Component	Description
Explanation Generation	LLM-powered rejection reasoning
Gap Analysis	Identify specific skill/experience gaps
Confidence Scoring	Probabilistic instead of binary rejection
Improvement Suggestions	Actionable next steps

AI/ML Models:

Model	Type	Purpose	Training
Explanation LLM	Llama-3-8B	Generate natural explanations	Instruction-tuned
SHAP Explainer	SHAP for LTR Model	Feature importance	Model-agnostic
Gap Identifier	Rule-based + ML	Identify missing qualifications	Trained on rejection patterns

Explanation Framework:

```
explanation_components = {
    "missing_skills": ["kubernetes", "distributed systems"],
    "experience_gap": {
        "required": "5 years",
        "candidate": "2 years",
        "gap_severity": "high"
    },
    "education_mismatch": None,
    "confidence_score": 0.73,  # Not binary rejection
    "improvement_suggestions": [
        "Consider AWS/GCP certification",
        "Build portfolio with distributed systems projects"
    ],
    "shap_feature_importance": {
        "experience_years": -0.35,
        "skill_match": -0.22,
        "education_level": 0.05
    }
}
```

Module 8: Resume Fraud Detection

Component	Description
Anomaly Detection	Detect statistical outliers
Inconsistency Check	Timeline and claim verification
Exaggeration Detection	Identify inflated claims
Fraud Risk Scoring	Aggregate risk score for recruiters

AI/ML Models:

Model	Type	Purpose	Training
Anomaly Detector	Isolation Forest	Detect statistical anomalies	Unsupervised on resume corpus
Skill-Experience Validator	Graph Neural Network	Validate skill acquisition timeline	Trained on career progression data
Claim Verifier	NLI Model (DeBERTa)	Check claim consistency	Pretrained on NLI + fine-tuned
Fraud Scorer	Gradient Boosting	Aggregate fraud signals	Supervised on labeled fraud cases

Fraud Detection Signals:

```
fraud_signals = {
    "timeline_inconsistencies": [
        "Overlapping employment dates",
        "Impossible skill acquisition speed"
    ],
    "statistical_anomalies": [
        "Salary 3  above role median",
        "Experience claims vs. graduation year mismatch"
    ],
    "claim_exaggeration": [
        "Team size inflation",
        "Revenue impact overclaiming"
    ],
    "verification_failures": [
        "Unverifiable certifications",
        "Non-existent companies"
    ]
}
```

Module 9: Recruiter-Side Intelligence

Component	Description
Job Posting	Structured JD creation with AI assistance
Resume Screening	Automated filtering and sorting
Shortlist Explanation	Transparent ranking justification
Applicant Analytics	Pool quality and skill scarcity insights

AI/ML Models:

Model	Type	Purpose	Training
JD Optimizer	Flan-T5	Improve job description quality	Instruction-tuned
Screening Model	Same as Matching Module 6	Automated resume filtering	Shared with matching
Clustering Model	HDBSCAN	Group similar candidates	Unsupervised
Analytics Engine	Statistical + ML	Skill market analysis	Trained on job market data

Recruiter Dashboard Features:

```
flowchart TB
    A[Job Posted] --> B[Applicant Pool]
    B --> C{AI Screening}
    C --> |Qualified| D[Shortlist]
    C --> |Uncertain| E[Human Review Queue]
    C --> |Unqualified| F[Rejected Pool]
    D --> G[Ranked List with Explanations]
    G --> H[Interview Scheduling]
    E --> I[Recruiter Decision]
    I --> J[Feedback to LTR Model]
```

Module 10: Fairness, Ethics & Trust

Component	Description
Bias Detection	Monitor protected attribute influence
Fairness Metrics	Track demographic parity, equalized odds
Debiasing	Post-hoc calibration and in-processing methods

Component	Description
Audit Logging	Complete decision trail for compliance

AI/ML Models:

Model	Type	Purpose	Training
Bias Detector	Adversarial Debiasing	Detect bias in rankings	Trained with fairness constraints
Fairness Calibrator	Post-processing Calibration	Adjust scores for fairness	Calibrated on validation set
Audit Model	Causal Inference	Identify causal bias pathways	Trained on observational data

Fairness Metrics Monitored:

```
fairness_metrics = {
    "demographic_parity": {
        "gender": 0.92,       # Ratio of selection rates
        "college_tier": 0.85,
        "experience_gap": 0.88
    },
    "equalized_odds": {
        "true_positive_rate_parity": 0.91,
        "false_positive_rate_parity": 0.89
    },
    "individual_fairness": {
        "similar_candidates_similar_scores": 0.94
    },
    "counterfactual_fairness": {
        "score_change_on_protected_flip": 0.02  # Low = good
    }
}
```

Fairness–Accuracy Tradeoff Analysis: | Configuration | Accuracy | Demographic Parity | Notes | |——————|————-|——————————-|———-| | No Debiasing | 0.89 | 0.72 | Baseline | | Adversarial Training | 0.86 | 0.88 | 3% accuracy drop | | Post-hoc Calibration | 0.88 | 0.85 | Balanced | | Threshold Adjustment | 0.87 | 0.90 | Per-group thresholds |

Module 11: Agentic AI – Autonomous Job Application Assistant

Component	Description
Job Monitoring Agent	Continuous new job surveillance
Evaluation Agent	AI confidence scoring for match
Application Agent	Autonomous form filling and submission
Policy Engine	User-defined constraints and limits
Safety Controller	Human-in-the-loop verification

Multi-Agent Architecture:

```
flowchart TB
    subgraph "Agent Orchestrator"
        ORCH[Orchestrator Agent]
    end

    subgraph "Specialized Agents"
        MON[Monitor Agent]
        EVAL[Evaluation Agent]
        APP[Application Agent]
        LOG[Logging Agent]
    end

    subgraph "Policy & Safety"
        POLICY[Policy Engine]
        SAFETY[Safety Controller]
        HUMAN[Human Approval Queue]
    end

    subgraph "External"
        JOBS[Job Boards]
        APPLY[Application Portals]
    end

    ORCH --> MON
    ORCH --> EVAL
    ORCH --> APP
    ORCH --> LOG

    MON --> |New Jobs| ORCH
    ORCH --> |Evaluate| EVAL
    EVAL --> |Score| ORCH
    ORCH --> |Check Policy| POLICY
    POLICY --> |Allowed| SAFETY
    SAFETY --> |High Confidence| APP
    SAFETY --> |Low Confidence| HUMAN
    HUMAN --> |Approved| APP
    APP --> APPLY
    LOG --> |All Actions| AUDIT[(Audit Log)]
```

Agent Decision Framework:

```python
class ApplicationDecision:
    def evaluate(self, job, user_profile, policies):
        # Compute match confidence
        match_score = self.eval_agent.score(job, user_profile)

        # Check policy constraints
        policy_check = self.policy_engine.validate(
            job=job,
            salary_threshold=policies.min_salary,
            job_types=policies.allowed_types,
            daily_limit=policies.max_applications_per_day,
            blacklist=policies.company_blacklist
        )

        # Determine action
        if not policy_check.passed:
            return Decision.SKIP, policy_check.reason

        if match_score > policies.auto_apply_threshold:
            return Decision.AUTO_APPLY, None
        elif match_score > policies.suggest_threshold:
            return Decision.SUGGEST_TO_USER, None
        else:
            return Decision.SKIP, "Low match score"
```

Policy Constraints:

```python
user_policies = {
    "salary_threshold": 600000,  # INR per annum
    "job_types": ["full-time", "contract"],
    "locations": ["remote", "bangalore", "hyderabad"],
    "max_applications_per_day": 5,
    "auto_apply_confidence_threshold": 0.85,
    "require_approval_above_threshold": False,
    "company_blacklist": ["company_x", "company_y"],
    "company_whitelist_auto_apply": ["dream_company_1"]
}
```

Safety & Explainability: - Every action logged with timestamp, reasoning, and confidence - Daily digest email with all agent actions - One-click undo for auto-applications - Complete audit trail for user review

---

4. Data Pipelines

4.1 Job Ingestion Pipeline

```
flowchart LR
    subgraph "Sources"
        A1[LinkedIn API]
        A2[Indeed API]
        A3[Naukri Feed]
    end

    subgraph "Ingestion"
        B[Apache Kafka]
        C[Rate Limiter]
    end

    subgraph "Processing"
        D[Spark Streaming]
        E[Deduplication]
        F[Normalization]
        G[Embedding Generation]
    end

    subgraph "Storage"
        H[(PostgreSQL)]
        I[(MongoDB)]
        J[(Qdrant)]
    end

    A1 --> C --> B
    A2 --> C
    A3 --> C
    B --> D --> E --> F
    F --> G
    F --> H
    G --> J
    D --> I
```

4.2 Resume Processing Pipeline

```
flowchart TB
    A[Resume Upload] --> B[Virus Scan]
    B --> C[File Storage - MinIO]
    C --> D[Document Parser - LayoutLMv3]
    D --> E[NER Extraction]
    E --> F[Skill Normalization]
    F --> G[Quality Scoring]
    G --> H[Embedding Generation]
    H --> I[(Vector DB)]
    G --> J[(PostgreSQL)]
    D --> K[(MongoDB - Raw Parse)]
```

4.3 Model Training Pipeline

```
flowchart LR
    A[Training Data] --> B[Data Validation]
    B --> C[Feature Engineering]
    C --> D[Model Training]
    D --> E[Evaluation]
    E --> |Pass| F[Model Registry - MLflow]
    E --> |Fail| G[Alert & Retrain]
    F --> H[A/B Testing]
    H --> I[Production Deployment]
```

---

5. Evaluation Metrics

5.1 Matching Quality

Metric	Description	Target
Precision@K	Relevant jobs in top-K results	0.75
Recall@K	Coverage of relevant jobs	0.80
NDCG@K	Ranking quality	0.85
MAP	Mean Average Precision	0.70
MRR	Mean Reciprocal Rank	0.80

5.2 Ranking Performance

Metric	Description	Target
AUC-ROC	Discrimination ability	0.85
Pairwise Accuracy	Correct pairwise orderings	0.80
Kendall's Tau	Rank correlation with ground truth	0.70
Feedback Alignment	Agreement with recruiter decisions	0.75

5.3 Fairness Metrics

Metric	Description	Target
Demographic Parity	Selection rate parity across groups	0.85
Equalized Odds	TPR/FPR parity	0.80
Individual Fairness	Similar candidates → similar scores	0.90
Counterfactual Fairness	Score stability on protected attribute flip	Δ 0.05

5.4 Fraud Detection

Metric	Description	Target
Precision	Correct fraud identifications	0.85
Recall	Fraud cases caught	0.90
F1-Score	Harmonic mean	0.87
False Positive Rate	Genuine candidates flagged	0.05

5.5 Agentic AI Performance

Metric	Description	Target
Application Success Rate	Submissions without errors	0.95
Policy Compliance	Actions within user constraints	100%
User Satisfaction	Post-action approval rate	0.90
Latency	Time from job post to application	15 min

---

6. Technology Stack

6.1 Backend

Layer	Technology	Justification
API Framework	FastAPI	Async support, OpenAPI docs, high performance
Task Queue	Celery + Redis	Distributed task processing
Message Broker	Apache Kafka	Real-time job ingestion
Orchestration	Apache Airflow	DAG-based pipeline management

6.2 Frontend

Layer	Technology	Justification
Framework	Next.js 14	SSR, React ecosystem, TypeScript
State Management	Zustand	Lightweight, simple API
UI Components	shadcn/ui + Tailwind	Modern, accessible components
Chat Interface	Custom + WebSocket	Real-time conversational UI

6.3 Databases

Type	Technology	Use Case
Relational	PostgreSQL 15	Users, jobs, applications
Document	MongoDB	Raw resumes, parsed documents
Vector	Qdrant	Embeddings for semantic search
Cache	Redis	Sessions, rate limiting, caching
Search	Elasticsearch	Full-text job search

6.4 ML/DL Frameworks

Framework	Use Case
PyTorch	Custom model training
Hugging Face Transformers	LLMs, embeddings, NER
Sentence-Transformers	Semantic similarity
XGBoost/LightGBM	Ranking, fraud detection
SHAP	Model explainability
Fairlearn	Fairness assessment and mitigation

6.5 MLOps

Tool	Purpose
MLflow	Experiment tracking, model registry
DVC	Data versioning
Weights & Biases	Training visualization
BentoML	Model serving

6.6 Infrastructure

Component	Technology
Containerization	Docker
Orchestration	Kubernetes (K3s for dev)
CI/CD	GitHub Actions
Monitoring	Prometheus + Grafana
Object Storage	MinIO (S3-compatible)

---

7. Deployment Strategy

7.1 Environment Setup

```
                  Development (Local)
  - Docker Compose for all services
  - Hot reload for frontend/backend
  - Mock external APIs
```

```
                  Staging (Cloud/Lab)
  - K3s single-node cluster
  - Real API integrations (sandboxed)
  - Full ML pipeline testing
```

```
                  Production (Demo)
  - Kubernetes cluster (3-node minimum)
  - Load balancing, auto-scaling
  - Full monitoring and alerting
```

7.2 Service Architecture

```
# Kubernetes Deployment Overview
services:
  - name: api-gateway
    replicas: 2
    resources: { cpu: 500m, memory: 512Mi }
```

```
- name: job-service
  replicas: 2
  resources: { cpu: 500m, memory: 1Gi }

- name: resume-service
  replicas: 2
  resources: { cpu: 1, memory: 2Gi }  # Heavier for parsing

- name: matching-engine
  replicas: 2
  resources: { cpu: 2, memory: 4Gi }  # ML inference

- name: llm-service
  replicas: 1
  resources: { cpu: 4, memory: 16Gi, gpu: 1 }  # GPU for LLM

- name: agentic-service
  replicas: 1
  resources: { cpu: 1, memory: 2Gi }
```

---

8. Team Allocation (4 Members)

Member 1: Platform & Infrastructure Lead

Focus: Core platform, job aggregation, deployment

Responsibility	Modules
User & Access Management	Module 1
Job Aggregation & APIs	Module 2
DevOps & Deployment	Infrastructure
Database Design	All schemas

Key Deliverables: - Authentication system with RBAC - Job ingestion pipeline with deduplication - Kubernetes deployment manifests - CI/CD pipeline setup

---

Member 2: Search & Matching Lead

Focus: Search systems, ranking, matching algorithms

Responsibility	Modules
Job Search & Conversational AI	Module 3
Job-Candidate Matching	Module 6
Learning-to-Rank Implementation	Module 6
Vector Search Infrastructure	Cross-cutting

Key Deliverables: - Hybrid search with semantic retrieval - Conversational RAG chatbot - Bi-encoder + cross-encoder matching - LTR model with feedback loop

---

Member 3: Resume Intelligence Lead

Focus: Resume processing, generation, fraud detection

Responsibility	Modules
Resume Parsing & Analysis	Module 4
Resume Generation System	Module 5
Fraud Detection	Module 8
ATS Optimization	Module 5

Key Deliverables: - LayoutLMv3 resume parser - LLM-based resume generator - Fraud detection ensemble - Quality scoring system

---

Member 4: Ethics & Agentic AI Lead

Focus: Fairness, explainability, autonomous agents

Responsibility	Modules
Explainable Rejection Feedback	Module 7
Fairness, Ethics & Trust	Module 10
Agentic AI System	Module 11
Recruiter Intelligence	Module 9

Key Deliverables: - SHAP-based explanation system - Fairness monitoring dashboard - Multi-agent application system - Recruiter analytics module

---

9. Project Timeline (12 Months)

Phase 1: Foundation (Months 1-2)

```
gantt
    title Phase 1: Foundation
    dateFormat  YYYY-MM
    section All Members
    Requirements Analysis       :2026-01, 2w
    System Design              :2026-01, 2w
    Technology Setup           :2026-02, 2w
    Database Schema Design     :2026-02, 2w
```

Week	Activities
1-2	Requirements gathering, literature review, API access applications
3-4	System architecture finalization, technology stack setup
5-6	Database design, development environment setup
7-8	Basic project scaffolding, CI/CD pipeline

Deliverables: Architecture document, development environment, basic project structure

---

Phase 2: Core Development (Months 3-6)

```
gantt
    title Phase 2: Core Development
    dateFormat  YYYY-MM
    section Member 1
    Auth System               :2026-03, 1M
    Job Aggregation           :2026-04, 2M
    section Member 2
    Search Infrastructure     :2026-03, 1M
    Matching Engine           :2026-04, 2M
    section Member 3
    Resume Parser             :2026-03, 2M
    Quality Scoring           :2026-05, 1M
    section Member 4
    Explainability Framework  :2026-03, 1M
    Fairness Foundation       :2026-04, 2M
```

Month	Member 1	Member 2	Member 3	Member 4
3	Auth/RBAC	Search infra	Resume parser	XAI framework
4	Job ingestion	Embedding pipeline	Parser training	Fairness metrics
5	Deduplication	Semantic search	Quality scorer	Bias detection
6	Integration	LTR model	Fine-tuning	Dashboard

Deliverables: Working authentication, job aggregation, basic matching, resume parsing

---

Phase 3: Intelligence Layer (Months 7-9)

```
gantt
    title Phase 3: Intelligence Layer
    dateFormat  YYYY-MM
    section Member 1
    Recruiter Dashboard       :2026-07, 2M
    Analytics                 :2026-09, 1M
    section Member 2
    Conversational AI         :2026-07, 2M
    Reranking                 :2026-09, 1M
    section Member 3
    Resume Generator          :2026-07, 2M
    Fraud Detection           :2026-09, 1M
    section Member 4
    Agentic Foundation        :2026-07, 1M
    Agent Policies            :2026-08, 2M
```

Month	Member 1	Member 2	Member 3	Member 4
7	Recruiter UI	RAG chatbot	LLM generator	Agent arch
8	Screening flow	Context-aware search	Bias removal	Policy engine
9	Analytics	Cross-encoder	Fraud model	Safety controls

Deliverables: Conversational AI, resume generator, agentic foundation, recruiter features

---

Phase 4: Advanced Features (Months 10-11)

```
gantt
    title Phase 4: Advanced Features
    dateFormat  YYYY-MM
    section All Members
    Agentic AI Full           :2026-10, 1M
    Integration Testing       :2026-10, 1M
    Performance Optimization  :2026-11, 1M
    Fairness Tuning           :2026-11, 1M
```

Month	Activities
10	Full agentic AI, end-to-end integration, cold-start handling
11	Performance optimization, fairness-accuracy tuning, A/B testing

Deliverables: Complete agentic system, optimized matching, fairness compliance

---

Phase 5: Documentation & Defense (Month 12)

Week	Activities
1-2	Documentation, user manuals, API docs
3	Demo preparation, video recording
4	Final report, viva preparation

Deliverables: Final report, demo video, deployment, defense preparation

---

10. Research Challenges & Limitations

10.1 Technical Challenges

Challenge	Mitigation Strategy
API Access Limitations	Partner with job portals, use RSS feeds as fallback
LLM Compute Requirements	Use quantized models (GPTQ/AWQ), cloud GPU bursting
Cold-Start for New Users	Hybrid content-based + collaborative filtering

Challenge	Mitigation Strategy
Real-Time Matching at Scale	ANN search with HNSW, caching strategies
Resume Generation Quality	Human-in-the-loop validation, fact verification

10.2 Research Limitations

Limitation	Scope Boundary
Limited Training Data	Use synthetic data augmentation, transfer learning
Fairness-Accuracy Tradeoff	Document tradeoffs, allow configurable thresholds
Agentic AI Reliability	Conservative policies, mandatory approval for edge cases
Fraud Detection Precision	Flag for human review, avoid auto-rejection
External API Dependencies	Graceful degradation, offline mode for core features

10.3 Ethical Considerations

Concern	Safeguard
User Data Privacy	GDPR-compliant design, data minimization
Algorithmic Discrimination	Continuous fairness monitoring, bias audits
Autonomous Application Risk	Explicit consent, daily limits, undo capability
Resume Integrity	Never fabricate claims, fact-preservation constraints
Transparency	Full explainability for all AI decisions

---

11. Viva-Defensible Justifications

Q: Why not just use keyword matching for jobs?

> Keyword matching fails for: - Semantic equivalence ("ML" vs "Machine Learning") - Skill inference ("5 years Python" implies "programming") - Context ("NLP" in research vs. industry)
>
> Our semantic embedding approach captures meaning, not just surface text, resulting in 40% better recall in our experiments.

Q: Why build your own LLM-based system instead of using ChatGPT API?

1. **Cost Control:** Fine-tuned 7B models are 10x cheaper at scale
2. **Latency:** Local inference is faster for real-time ranking
3. **Customization:** We can fine-tune for resume/job domain
4. **Privacy:** Sensitive resume data stays on-premises
5. **Research Value:** Demonstrates ML engineering competence

Q: How do you handle freshers with no experience (cold-start)?

We use a multi-signal approach: 1. **Academic signals:** GPA, coursework, college tier 2. **Project inference:** GitHub, portfolios → skill extraction 3. **Skill-based matching:** Emphasis on skills over experience 4. **Transfer learning:** Similar profile clustering from alumni data 5. **Exploration-exploitation:** Occasional exposure to develop signals

Q: What prevents the agentic AI from applying to inappropriate jobs?

Multi-layer safety: 1. **Policy engine:** Hard constraints (salary, type, location) 2. **Confidence thresholds:** Only auto-apply if >85% match 3. **Daily limits:** Maximum 5 applications per day 4. **Human-in-the-loop:** Low-confidence cases need approval 5. **Audit logging:** Complete action trail for review 6. **Undo capability:** One-click application withdrawal

Q: How do you balance fairness and accuracy?

We expose this as a configurable tradeoff: 1. Measure both metrics continuously 2. Present Pareto frontier to stakeholders 3. Allow per-deployment fairness targets 4. Use post-hoc calibration (preserves ranking, adjusts thresholds) 5. Document the tradeoff transparently in all reports

Q: What makes this a "research-grade" project vs. a simple portal?

Simple Portal	Our System
Keyword search	Semantic + hybrid retrieval with LLMs
Binary match/no-match	Probabilistic scores with confidence
No explanations	SHAP-based explainability
Static ranking	Learning-to-Rank with feedback
Manual applications	Agentic AI with policy engine
No fairness	Continuous bias monitoring
Basic fraud check	ML-based anomaly detection

12. Verification Plan

12.1 Automated Testing

Test Type	Coverage	Tools
Unit Tests	All service functions	pytest, jest
Integration Tests	API endpoints, DB interactions	pytest, httpx
ML Model Tests	Inference, embedding consistency	pytest, hypothesis
E2E Tests	Critical user flows	Playwright

Commands:

```
# Backend unit tests
cd backend && pytest tests/unit -v --cov=app

# Integration tests
pytest tests/integration -v

# ML pipeline tests
pytest tests/ml -v

# Frontend tests
cd frontend && npm run test

# E2E tests
npx playwright test
```

12.2 ML Evaluation

Evaluation	Method	Frequency
Matching Quality	Holdout test set	Weekly
Ranking Performance	A/B testing	Per deployment
Fairness Audit	Full fairness report	Monthly
Fraud Detection	Precision/Recall on labeled set	Weekly

12.3 Manual Verification

1. **User Flow Testing**

    - Create job seeker account
    - Upload resume and verify parsing
    - Search for jobs using chatbot
    - Receive job recommendations
    - Generate tailored resume
2. **Recruiter Flow Testing**
    - Post job with AI-assisted JD
    - Review AI-screened candidates
    - View ranking explanations
    - Provide feedback for LTR
3. **Agentic AI Testing**
    - Configure application policies
    - Verify policy compliance
    - Check audit logs
    - Test human approval flow

---

13. Appendix

A. Model Card Template

Each deployed model will have a model card documenting: - Model architecture and size - Training data description - Intended use cases - Known limitations - Fairness evaluation results - Version history

B. Data Dictionary

Complete schema documentation for all database tables and collections.

C. API Documentation

OpenAPI 3.0 specification for all REST endpoints.

D. MLOps Runbook

Procedures for model retraining, deployment, and rollback.

Document Version: 1.0
Last Updated: January 2026
Status: Ready for Review