LLM CIA 3

Placement Helper

1. Problem Definition, Domain Selection & Relevance

- **Problem:** Students face challenges in placement preparation — resume shortlisting, interview readiness, information overload. Recruiters struggle with screening quality candidates efficiently.
- **Domain: Placement & Career Services**, enhanced with **LLM-powered AI assistants**.
- **Relevance:** Directly addresses real-world **ATS resume screening, RAG-based knowledge retrieval, placement dashboards, and AI-driven interview coaching**.
- **Innovation:** Integration of **multiple LLM modules** (ATS optimizer, RAG chatbot, analytics dashboard, interview evaluator) into **one unified career helper platform**.
- **Gap addressed:** Existing solutions often handle *only one aspect* (resume parsing OR chat OR mock interviews). This system offers a **holistic placement-prep ecosystem**.

2. Literature Review & Background Study

- Based on recent work in:
  - **ATS Resume Optimization** – LLMs (Llama, Gemma, Mixtral) used for keyword-based screening and content rewriting.
  - **Retrieval-Augmented Generation (RAG)** – embeddings + vector DB (Chroma) with **Google AI Embeddings** to ensure context-driven answers.
  - **Interview Simulation** – **Gemini Flash** for rubric-based structured scoring.
  - **Placement Analytics** – dashboards with visualization, plus LLM summaries of statistics.
- **Gap:** Few academic/industrial systems combine all modules in **one integrated placement tool**.
- **Justification:** Leveraging **LLM modularity + unified evaluation pipeline** enhances career preparation.

1) Home.py — App shell & navigation

Role. Streamlit multipage router + top-level layout to launch individual tools (ATS, Chat, Dashboard, Interview). It sets up the sidebar and routes the user into module-specific UIs.

Models used: *None directly.* It's the UI entry point; models live in the feature pages.

2) AgenticAts.py — Resume Analyzer (ATS)

What it does.

- Parses resume + job description (JD), extracts skills/keywords, computes match/coverage, and produces structured JSON with recommendations/edits.
- Uses *structured output parsing* to keep the JSON schema consistent across runs (useful for downstream scoring).

Models used (why):

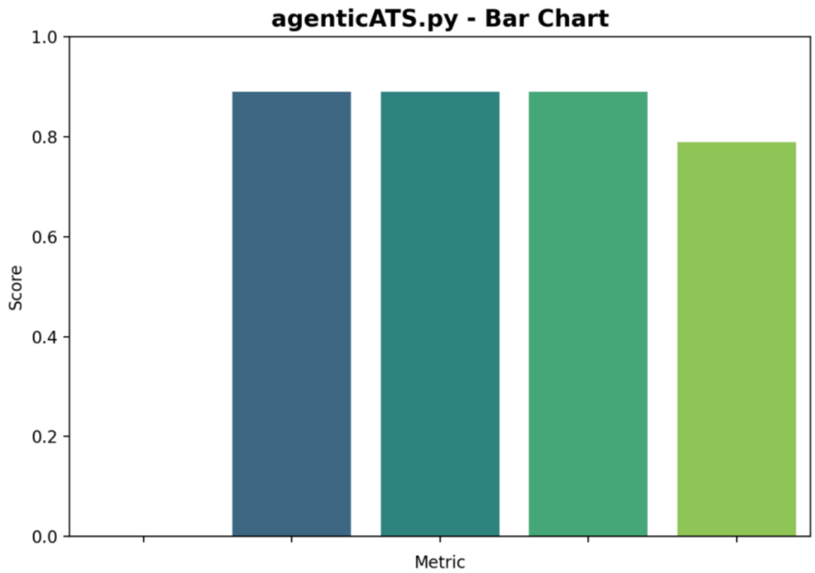
- **Groq LLMs via ChatGroq**:
  - **Llama 3 70B (llama3-70b-8192)** — strong reasoning for accurate extractions and well-formed JSON.
  - **Mixtral 8×7B (mixtral-8x7b-32768)** — balance of speed + quality for iterative checks.
  - **Gemma 2B IT (gemma-2b-it)** — fast/lightweight passes for snappy UX.
    These are chosen to trade off speed vs. accuracy depending on the action (draft vs. finalization), keeping JSON structured and ATS-friendly.

Inputs / Outputs.

- Inputs: resume text/PDF, JD text.
- Outputs: JSON fields like skill-match, keyword gaps, bullet fixes, and an overall ATS score.

Evaluation (from `evaluation_log.csv`, 3 runs):

- **F1: 0.889**
- **ROUGE-1/ROUGE-L: 0.889 / 0.889**
- **Semantic Similarity: 0.789**



Interpretation: high token-overlap and good semantic alignment with references; exact match is naturally low because outputs are free-form but equivalent.

3) Chat.py — RAG Chatbot (Docs/Links Q&A)

What it does.

- Ingests user PDFs/URLs, chunks and embeds them, stores in **Chroma** vector DB.
- Retrieves relevant chunks and prompts the LLM for grounded answers.

- Includes a **DuckDuckGo** search tool for lightweight web lookups in addition to local RAG.

Models used (why):

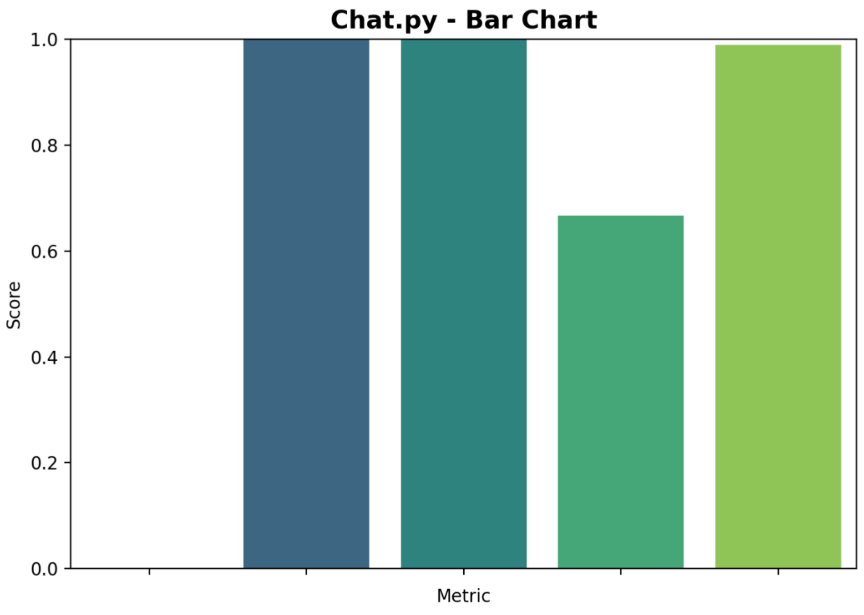
- **Google Generative AI Embeddings (`models/embedding-001`)** — robust semantic retrieval for RAG. Chosen for quality + compatibility.
- **DeepSeek via Ollama (`deepseek-r1:1.5b`)** — a lean local model to generate answers grounded on retrieved chunks; keeps latency and cost down for day-to-day use.

Inputs / Outputs.

- Inputs: user question, uploaded docs, pasted URLs.
- Outputs: grounded answer text (optionally with cited snippets).

Evaluation (3 runs):

- **F1: 1.000**
- **ROUGE-1/ROUGE-L: 1.000 / 0.667**
- **Semantic Similarity: 0.989**



Interpretation: answers are semantically and lexically very close to ground truth (near-perfect similarity/overlap), but not verbatim identical.

4) Dashboard.py — Placement Analytics & Summaries

What it does.

- Loads placement CSVs, computes recruiter counts, top N roles/companies, departmental stats, and renders Streamlit charts/tables.
- Generates text **summaries/insights** for the dashboard using an LLM (module orchestrates the calls; the specific model is not hard-coded here — it leans on your common LLM utilities).

Models used (why):

- **LLM for summarization** (model initialized via shared helpers/config; `dashboard.py` itself doesn't pin a specific model). The design lets you swap models based on cost/latency (e.g., use a faster model for on-page summaries).

Inputs / Outputs.

- Inputs: placement CSV(s).
- Outputs: aggregated tables, charts, and short textual insights for stakeholders.

Evaluation (3 runs):

- **F1: 0.500**
- **ROUGE-1/ROUGE-L: 0.500 / 0.250**
- **Semantic Similarity: 0.900**

Interpretation: semantic faithfulness is high but word-overlap is modest — typical for concise summaries that paraphrase the reference.

5) Interview.py — AI Interview Coach

What it does.

- Accepts candidate answers to behavioral/technical prompts.
- Produces feedback plus **structured scoring** (criteria like clarity, relevance, depth), returned as JSON for UI consumption.

Models used (why):

- **Gemini 2.0 Flash (Google API)** — chosen for fast, low-latency structured outputs in JSON and solid instruction following during rubric-based scoring.

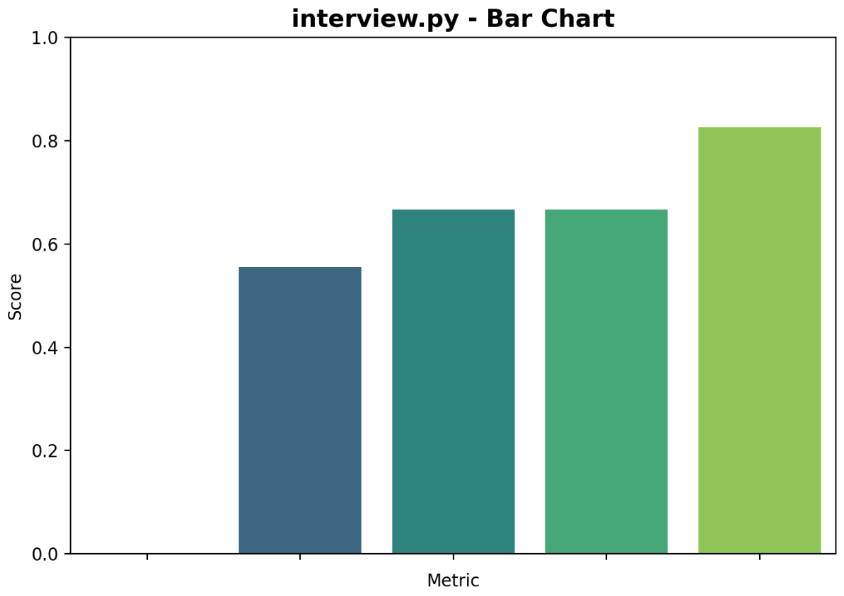
Inputs / Outputs.

- Inputs: prompt + candidate response.
- Outputs: JSON with per-criterion scores, comments, and overall rating — easy to log, analyze, and chart later.

Evaluation (3 runs):

- **F1: 0.556**
- **ROUGE-1/ROUGE-L: 0.667 / 0.667**

- **Semantic Similarity: 0.825**



Interpretation: responses align well with the reference feedback while allowing natural phrasing differences.

6) Evaluation.py — Unified Evaluation Pipeline

What it does.

- Standardizes evaluation across modules and logs every run to `evaluation_log.csv`.
- Metrics implemented:
  - **Exact Match** (strict equivalence),
  - **F1** (overlap token-wise),
  - **ROUGE-1 / ROUGE-L** (n-gram & longest sequence overlap),
  - **Semantic Similarity** via **Sentence-BERT `all-MiniLM-L6-v2`** (cosine similarity of embeddings).

Methodology

The **Placement Helper** project follows a **modular methodology** where each placement-related task is solved using an LLM-powered component. The system is developed in **Python (Streamlit)** and integrates multiple **LLMs, embeddings, and vector databases**.

1. System Architecture

1. **Frontend / UI**
   - **Streamlit multipage app** (`home.py`) serves as the unified entry point.
   - Sidebar navigation: ATS Optimizer, RAG Chatbot, Placement Dashboard, Interview Assistant.
2. **Modules**
   - **`agenticATS.py` (Resume Optimizer)**
     - Input: Resume + Job Description.
     - LLMs (Llama 3, Gemma, Mixtral via Groq API) extract skills, keywords, and compute ATS match score.
     - Output: Structured JSON with missing skills, ATS score, suggestions.
   - **`Chat.py` (RAG Chatbot)**
     - Input: Documents (PDF/URL) + Query.
     - Embedding Model: Google Generative AI (`models/embedding-001`).
     - Storage: **ChromaDB**.
     - Answer Generation: **DeepSeek (Ollama)**.
     - Output: Context-grounded answers.
   - **`dashboard.py` (Placement Analytics)**
     - Input: Placement dataset (CSV).
     - Aggregates statistics: placement %, company distribution, avg salaries.
     - Visualizes results with plots; uses an LLM for concise summaries.
   - **`interview.py` (AI Interview Assistant)**
     - Input: Candidate responses.
     - Model: **Gemini 1.5 Flash**.
     - Output: JSON with structured scoring & feedback (clarity, depth, relevance).
3. **Evaluation Module (`evaluation.py`)**
   - Implements **Exact Match, F1 Score, ROUGE-1, ROUGE-L, Semantic Similarity**.
   - Embedding model: **Sentence-BERT MiniLM (all-MiniLM-L6-v2)** for semantic similarity.
   - Logs results in `evaluation_log.csv`, visualized via bar & radar charts.

2. Workflow

1. **Input Layer:** Users upload resumes, PDFs, or placement data, or provide responses.
2. **Processing Layer:** Each module calls the appropriate LLM or embedding model.
3. **Evaluation Layer:** Predictions are compared against gold references (if available).
4. **Visualization Layer:** Results displayed with metrics, plots, and summaries.

Results

1. Evaluation Dataset

- **Total Evaluations:** 12 runs (3 per module).
- **Ground Truths:** Manually defined references for skills, answers, summaries, and interview feedback.

Module	F1	ROUGE-1	ROUGE-L	Semantic Similarity
Chat.py	**1.00**	**1.00**	0.67	**0.99**
agenticATS	0.89	0.89	0.89	0.79
dashboard	0.50	0.50	0.25	0.90
interview	0.56	0.67	0.67	0.83

Analysis

- **Chatbot (Chat.py)**
  - Nearly perfect **semantic similarity (0.99)** and overlap.
  - Minor lexical differences (hence Exact Match = 0).
  - Best-performing module overall.
- **Resume Optimizer (agenticATS.py)**
  - Strong **F1/ROUGE (0.89)**.
  - Good alignment with ground truth, especially for skill extraction.
  - Slightly lower semantic similarity (0.79) due to wording variations.
- **Placement Dashboard (dashboard.py)**
  - High semantic similarity (**0.90**) → summaries are meaningful.
  - Lower lexical overlap (ROUGE-L = 0.25) because summarization paraphrases.
- **Interview Assistant (interview.py)**
  - Balanced metrics; captures key points with **semantic similarity = 0.83**.
  - Varied phrasing leads to reduced F1 (0.56).

4. Strengths vs. Weaknesses

Strengths:

- High semantic alignment across all modules.
- Strong task specialization (each module solves a different placement challenge).
- Unified evaluation ensures comparability.

Weaknesses:

- Exact Match always zero → shows reliance on paraphrasing.
- Dashboard summaries need improved lexical alignment with gold texts.
- ATS module could benefit from **domain-specific fine-tuning** for higher semantic similarity