Resume Ops:

To extract data from Resume and JD based on :

1. Structural Extraction

→ Identifying sections like:

- Name
- Contact Info
- Education
- Skills
- Work Experience
- Certifications
- Projects
- Summary

- JD extract→ Company name, Role, Experience Level, Skills, Requirements...

  MAYBE: Cannot be fixed with one. Need to try to fit in the best.

Purpose	Tool
Resume parsing	`pdfplumber`, `python-docx`
JD + Resume structuring	`spaCy`, custom regex
Embeddings (similarity)	`text-embedding-3-small`, `BGE`
AI Suggestions	`GPT-4`, `Gemini 1.5`, `Mistral`
DB (for storing data)	MongoDB or PostgreSQL

AI Workflow (Not the entire Architecture):

1. **Focused Data Extraction**

   Resume and job description data will be extracted by emphasizing **specific factors** like key skills, project titles, experience level, education, and relevant keywords. This ensures the AI operates on precise and relevant context, improving overall efficiency.

2. **Company-Specific Resume Optimization**

   Resume suggestions will be guided by **fixed historical templates** for each company and role.

   These templates include:

   - Commonly expected skills
   - Typical project types
   - Minimum academic criteria (e.g., CGPA)
   - Formatting preferences and tone

   → Company requirements remain mostly static over time. So, optimization is **entirely dependent on the user's resume quality** and how well it aligns with that fixed template.

3. **AI-Powered Resume Analysis**

   The user's resume is compared against the selected company-role template to identify:

   - Missing or weak areas
   - Irrelevant or outdated content
   - Style, tone, and structural mismatches

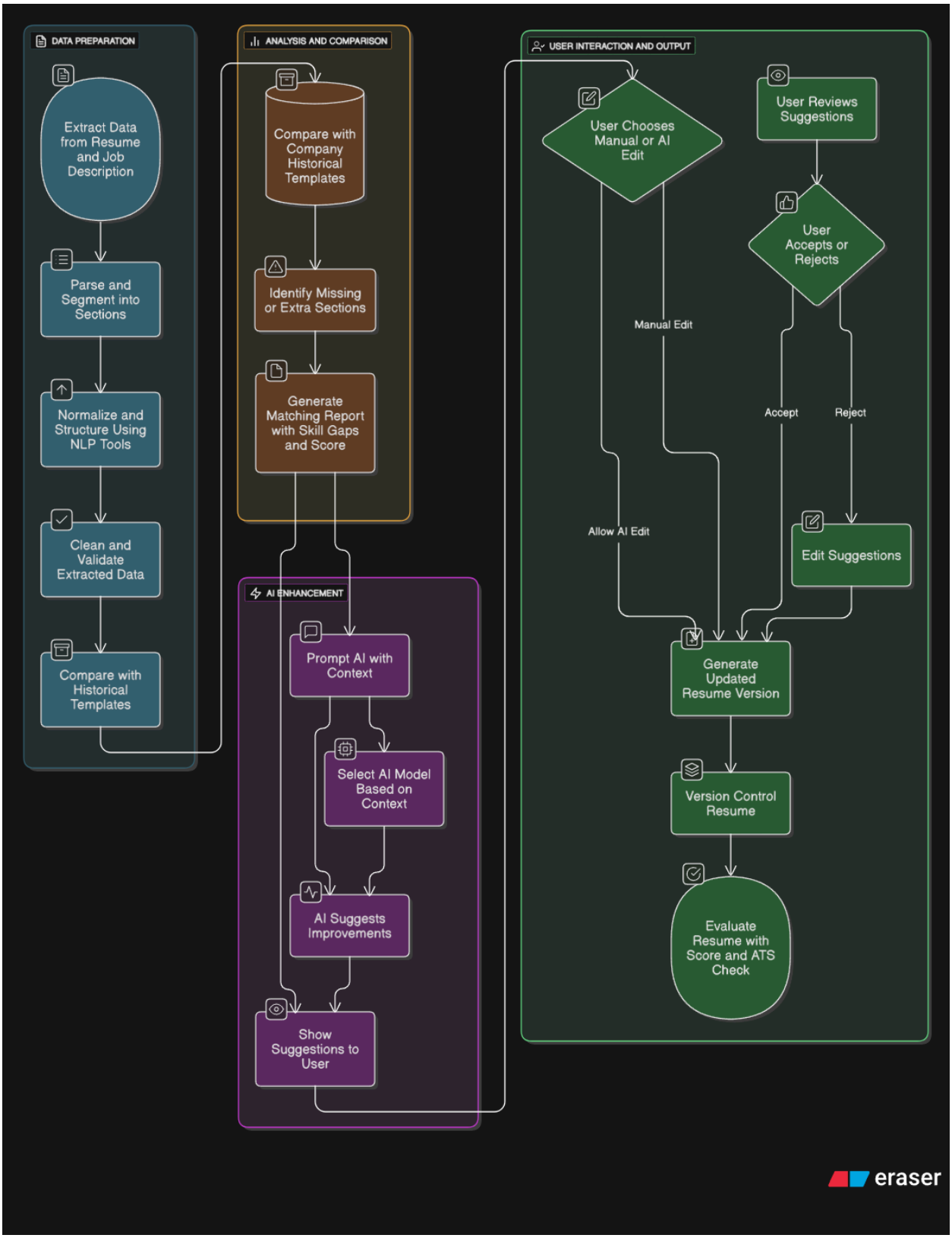
4. **ATS Score Validation**

   Once optimized, the resume undergoes an **ATS compatibility check** to ensure it meets the standards used by real-world hiring systems.

5. **Exploratory AI Development**

   Since this project is AI-heavy and focused on efficiency, the tech stack for the AI layer will be determined **through practical experimentation**.

No fixed tools will be chosen initially — multiple options will be explored to find the most effective and scalable solution.


DATA PREPARATION
Extract Data from Resume and Job Description
Parse and Segment into Sections
Normalize and Structure Using NLP Tools
Clean and Validate Extracted Data
Compare with Historical Templates
ANALYSIS AND COMPARISON
Compare with Company Historical Templates
Identify Missing or Extra Sections
Generate Matching Report with Skill Gaps and Score
AI ENHANCEMENT
Prompt AI with Context
Select AI Model Based on Context
AI Suggests Improvements
Show Suggestions to User
USER INTERACTION AND OUTPUT
User Chooses Manual or AI Edit
User Reviews Suggestions
User Accepts or Rejects
Manual Edit
Accept
Reject
Allow AI Edit
Edit Suggestions
Generate Updated Resume Version
Version Control Resume
Evaluate Resume with Score and ATS Check
eraser

- We need to have version control to our resumes. To rollback if some mess happens. ***IMP***
- Our application must act as a storage point too. Instead of storing them on local pc, we can organize resumes based on roles and companies. Each company can have multiple roles. So, kind of directory structure with good UI must be included for the user to download/manage their resumes effectively. Where we will have a share option too with mails or WhatsApp phone numbers to have a easy sharing with tuck-shop for printing out resume 😅→ **May or may not be a good feature.**
- There is nothing much to decide with the application architecture because AI is playing the crucial role. However, we can create one on what our full stack does.

- We can implement Redis for maintaining a process-queue to decrease load on backend for heavy lifting.

3. Data Extraction Libraries in Python:
For extracting text from various document formats (PDF, DOCX) and performing NLP tasks, several Python libraries are highly recommended:

- **Text Extraction (PDF/DOCX):**
  - `pdfminer.six` or `PyMuPDF` for PDFs. `PyMuPDF` is often cited for better performance and structured extraction.
  - `python-docx` or `docx2txt` for DOCX files.
  - `textract` can handle multiple formats, including DOC.
- **NLP and Information Extraction:**
  - `spaCy`: This is a powerful library for Natural Language Processing (NLP) that excels at tasks like Named Entity Recognition (NER), part-of-speech tagging, and rule-based matching. It's highly suitable for extracting

structured information like skills, education, experience, and company names from unstructured text.

- `NLTK`: Another popular NLP library, useful for tokenization and other basic NLP tasks.
- `pyresparser`: A pre-built resume parser that uses `spaCy` and other libraries to extract common resume fields. This could be a good starting point for rapid development.
- For "contextual data extraction" and identifying specific patterns, `spaCy`'s rule-based matching and custom NER models would be very effective.