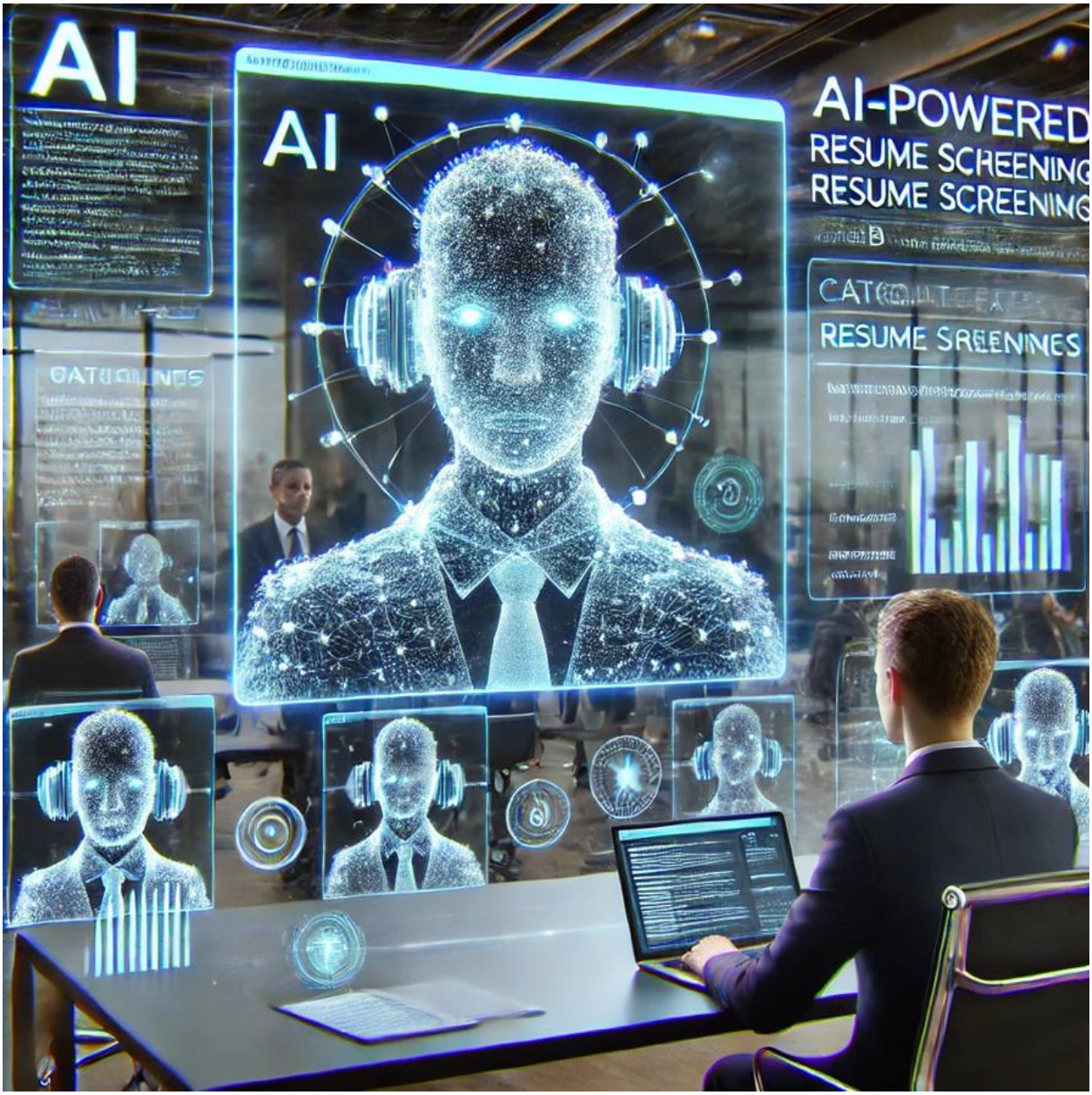
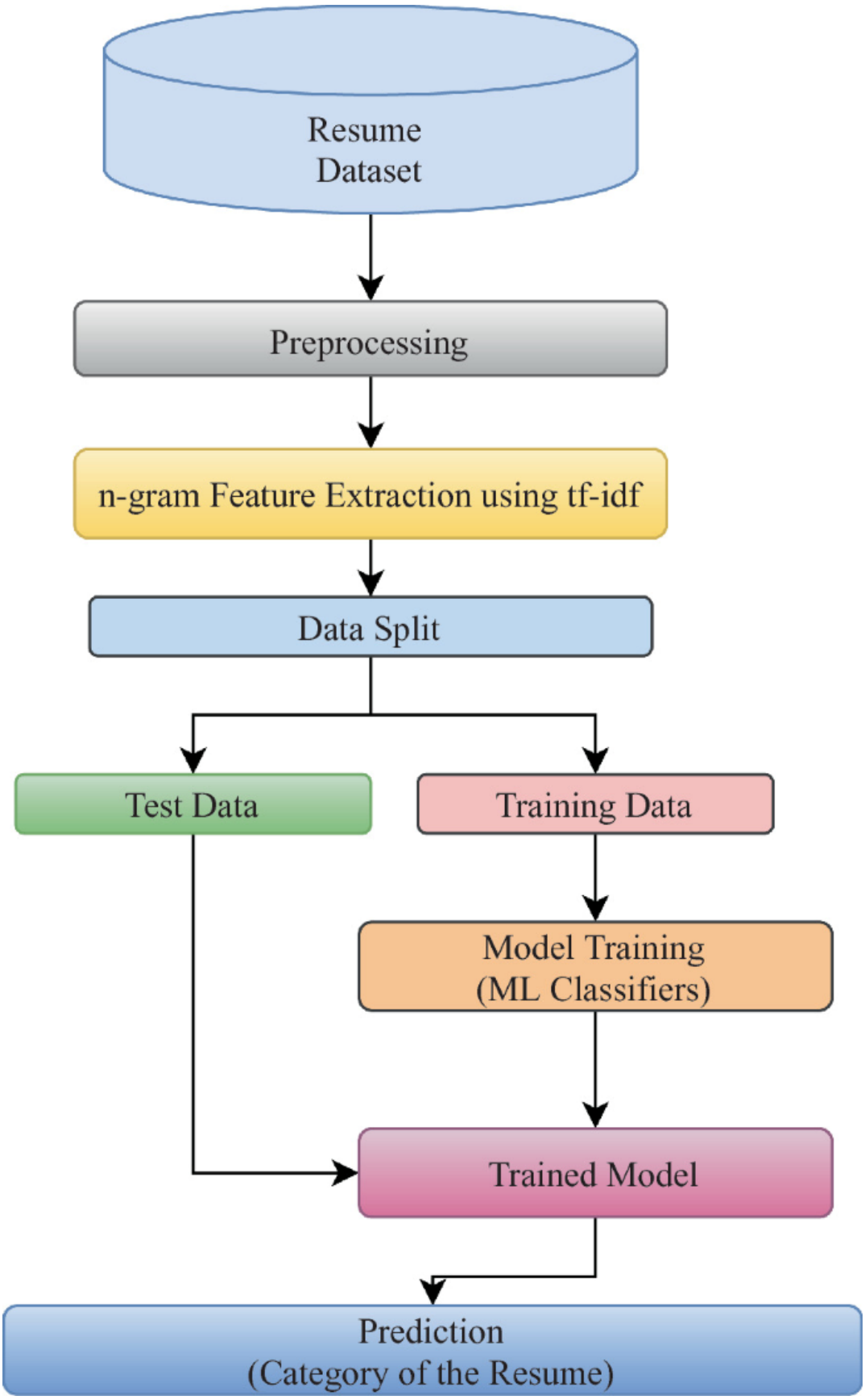


AI-Based Resume Screening and Job Classification System

---

1. Project Overview (Introduction)

In this project, I will create an AI model that automatically classifies resumes into predefined job categories based on the resume text. The model will use **Deep Learning (LSTM)** or **Transformers (BERT)** to understand the content and assign each resume to a specific job category. This system aims to assist recruiters by automating the initial resume screening process.


Resume
Dataset
Preprocessing
n-gram Feature Extraction using tf-idf
Data Split
Test Data
Training Data
Model Training
(ML Classifiers)
Trained Model
Prediction
(Category of the Resume)

---

2. Dataset Selection

I will use the following dataset:

✅ **Dataset Name:** Resume Dataset by Snehaan Bhawal

✅ **Source:** [Kaggle - Resume Dataset]

✅ **Description:** This dataset contains over 2,400 resumes classified into different job categories like Data Scientist, Web Developer, HR, and more.

---

3. Tools and Libraries

Task	Tools/Libraries
Data Preprocessing	Pandas, NLTK, Regex
Text Vectorization	TF-IDF or CountVectorizer
Model Training	TensorFlow, Keras, LSTM
Evaluation	Scikit-Learn
Visualization	Matplotlib, Seaborn

---

4. Step-by-Step Roadmap

✅ **Week 1: Model Development**

Day	Task	Status
1	Searching for Datasets	✅ Done
1	Dataset Download & Import	✅ Done
2-3	Data Preprocessing (Cleaning Text, Removing Stopwords)	✅ Done
4	Text Vectorization (TF-IDF or Word Embeddings)	✅ Done
5-6	Build LSTM Model	✅ Done
7	First Model Training	✅ Done

✅ **Week 2: Evaluation & Improvements**

Day	Task	Status
8	Model Evaluation (Accuracy, Precision, Recall)	✅ Done
9-10	Hyperparameter Tuning	✅ Done
11	Save Model & Build API	✅ Done
12	Create Demo (Classify Custom Resumes)	✅ Done
13	Final Testing	✅ Done
14	Documentation	✅ Done

---

2. Data Preprocessing

What Was Done:

- **Data Loading:**
  - Loaded the resume dataset (CSV) containing columns such as ID, Resume_str, Resume_html, and Category.
  - Ensured that the relevant text column (Resume_str) was correctly identified and converted to string format.
- **Cleaning & Preprocessing Code:**
  - Developed a custom text cleaning function using Python's regex and basic tokenization.
  - Steps included:
    - Converting text to lowercase.
    - Removing HTML tags.
    - Eliminating special characters and numbers.
    - Splitting text into tokens.
    - Removing a basic set of stopwords.
  - Saved the cleaned resumes to a new CSV file (cleaned_resume_processed.csv).

---

3. Text Vectorization

What Was Done:

- **Word Embeddings with Gensim:**
    - Loaded pre-trained GloVe embeddings (50-dimensional) using Gensim.
    - Developed a function to calculate the average embedding for each resume.

---

4. LSTM Model Building

What Was Done:

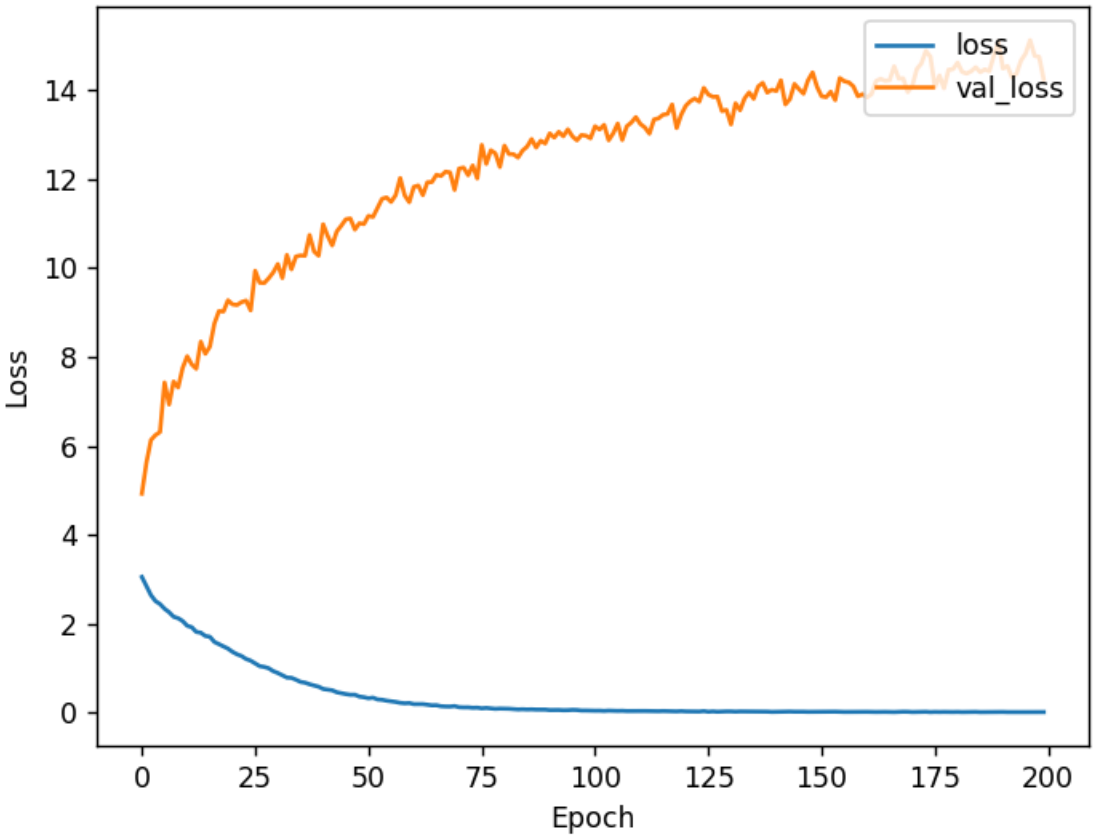
- **Data Preparation for LSTM:**
    - Tokenized the cleaned text using Keras' Tokenizer and padded sequences to a uniform length.
    - Converted the job categories into one-hot encoded vectors using LabelEncoder and Keras' to_categorical.
- **Model Architecture:**
    - Built an LSTM model using TensorFlow/Keras:
        - **Embedding Layer:**
          Initialized with the pre-trained GloVe embedding matrix (embeddings kept fixed during training).
        - **LSTM Layer:**
          Used 64 LSTM units with dropout for regularization.
        - **Dense Layer:**
          Final output layer with softmax activation for multi-class classification.
    - Compiled the model with categorical cross-entropy loss and the Adam optimizer.

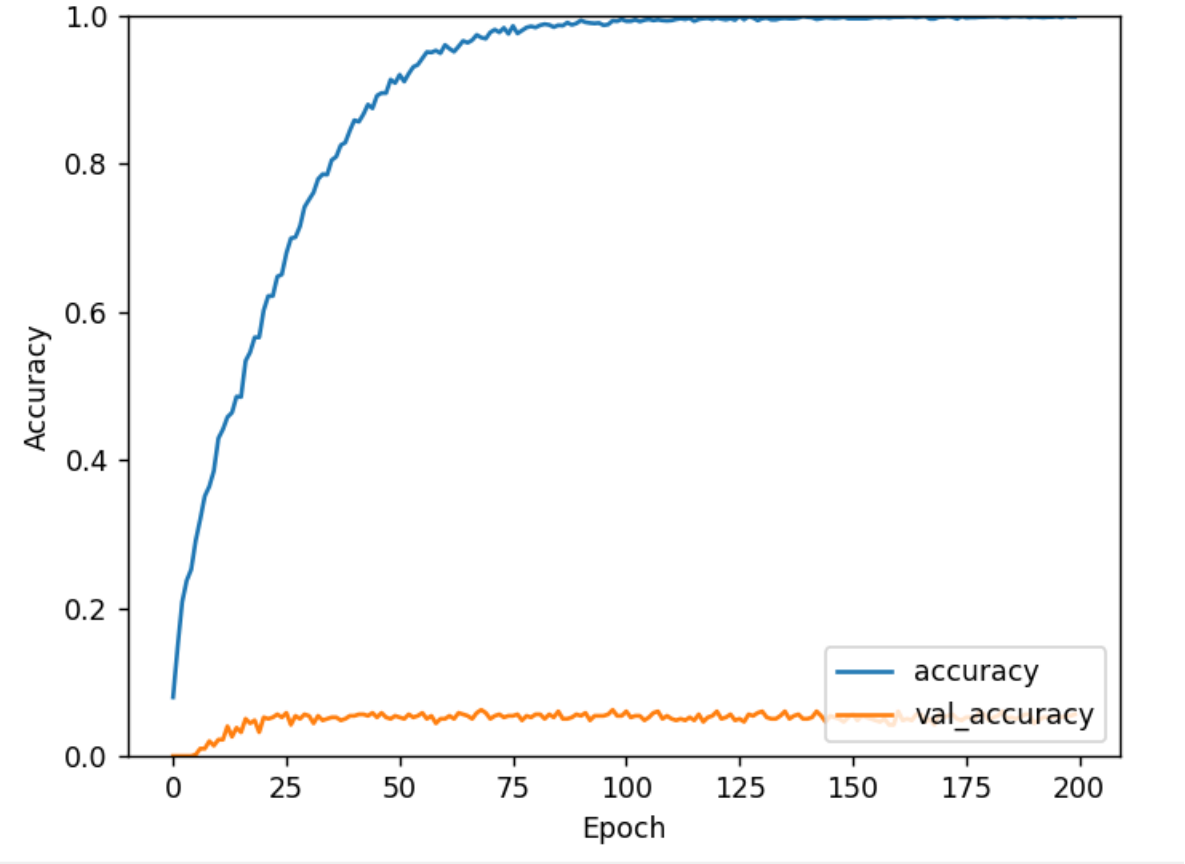
---

5. Model Training

What Was Done:

- Trained the LSTM model on the preprocessed data.
- Set training parameters:
    - **Epochs:** 10-200
    - **Batch Size:** 32
    - **Validation Split:** 20% of the training data was used for validation.


loss
val_loss
Loss
Epoch
0
2
4
6
8
10
12
14
0
25
50
75
100
125
150
175
200



- Saved the trained model for future evaluation and deployment.

---

6. Model Evaluation and Hyperparameter Tuning

What Was Done:

- **Evaluation:**
  - Split the dataset into training and test sets.
  - Evaluated the model using metrics such as **accuracy**, **precision**, and **recall**.
  - Generated a detailed classification report to assess performance across different classes.
- **Hyperparameter Tuning:**
  - Implemented hyperparameter tuning using **Keras Tuner (RandomSearch)**.

- Tuned key parameters like:
    - Number of LSTM units.
    - Dropout and recurrent dropout rates.
- Built and retrained the best model configuration based on validation accuracy.

Learned:

- The value of using tools like **Keras Tuner** to automate the search for optimal hyperparameters.

---

7. Next Steps

- **Further Model Evaluation:**
  Refine the evaluation process with additional metrics (e.g., F1-score) or cross-validation.
- **Model Deployment:**
  Explore building a user interface (using Flask or Streamlit) to showcase model predictions.
- **Fine-Tuning:**
  Consider fine-tuning the embeddings or adjusting model architecture further based on additional experiments.

---

8. Testing on New Data

- **PDF Resume Testing:**
    - Implemented code to extract text from PDF files using pdfplumber.
    - Integrated text preprocessing, tokenization, and padding pipelines to convert new resume data into the appropriate format.
    - Loaded the saved model and ran predictions on new data.
    - Visualized the prediction confidence and distribution using bar charts.
- **Outcome:**
    - Demonstrated the end-to-end functionality of the system on unseen data.

What Was Learned

- **Data Preprocessing:**
    - The importance of clean and consistent text data for NLP.

- **Feature Extraction & Vectorization:**
    - How pre-trained word embeddings (like GloVe) can be used to capture semantic meaning in text.
    - Converting variable-length resumes into fixed-length representations.
- **Model Development:**
    - Building and training LSTM models for sequence classification.
    - The value of dropout and proper architecture design in preventing overfitting.
- **Hyperparameter Tuning:**
    - Setting up a search space for hyperparameters using Keras Tuner.
    - Evaluating model performance under different configurations to determine optimal settings.
- **Deployment & Testing:**
    - How to integrate additional data sources (e.g., PDFs) into your pipeline.
    - The process of extracting text from PDFs and converting it for model prediction.

---

9. Next Steps

- **Model Refinement:**
    - Further fine-tune the hyperparameters based on tuning insights.
    - Possibly retrain the model on a more balanced or larger dataset if needed.
- **Final Report:**
    - Compile these findings, challenges, and improvements into a final project report.
    - Include visualizations (training curves, hyperparameter tuning graphs, prediction confidence charts) as supporting evidence.