# Report

This project focuses on intelligent resume shortlisting by combining both resume text and structured profile data. A deep learning model was developed, where features from text and tabular inputs were fused to predict job category probabilities. The system processes resumes in overlapping chunks using a frozen BERT encoder and structured fields using field-wise embeddings, enabling more accurate and context-aware shortlisting decisions. **Achieved 97.77% accuracy** on the resume dataset, demonstrating the effectiveness of the fusion-based approach.

Model checkpoints and saved parameters are available here.

# Dataset

- Resume Dataset on Kaggle
- **Labels**: Shortlisted(1) and Not shortlisted(0)
- **Job Categories**: Java Developer, DevOps Engineer, Python Developer, Web Designing, HR, Hadoop, Mechanical Engineer, Data Science, Blockchain, ETL Developer, Operations Manager, Sales, Arts, Database, Electrical Engineering, Health and fitness, PMO, DotNet Developer, Business Analyst, Automation Testing, Network Security Engineer, Civil Engineer, SAP Developer, Advocate

## Text Cleaning

- Removed HTML tags, URLs
- Removed "\n" and "\r" escape keys
- Removed all non-alphanumeric characters
- Removed any extra space present
- Lowered the case of the text, and striped any white space present in front and end of the text.

## Section Extraction from Resumes

After manually viewing some good number of training examples i came up with the following sections

- skills
- skill details
- education
- company details
- projects

And each section name has synonymys so i used regex to get them too. The following are the synonyms that i considered

- **skills**: skills, technical skills, personal skills
- **skill details**: skill details, experience
- **education**: education
- **company details**: company details, company description, work experience, work experince, employment, industry
- **projects**: projects, project, technology assisted review, developed

To extract structured sections from unstructured resume text, I used regular expressions to detect common section headers and their synonyms—such as “skills,” “technical skills,” “education,” “work experience,” and “projects.” For each match, I recorded its position and standardized section name in a list. This list was then sorted by position to process the resume in order. As I iterated through the list, I gathered the text between two section headers, skipping consecutive headers belonging to the same section to avoid duplication. The extracted text was stored in a dictionary ( `new_sec` ) with section names as keys. If no known section headers were found, the dictionary simply stored empty strings. This approach enabled flexible, rule-based segmentation of resumes into meaningful components.

# Implementation Approach

So the idea is to split the tokenized text into **multiple overlapping chunks** and pass it to the pretrained bert, and output is stacked together and passed through the network.

## Text Model

Processes entire resume text using a **frozen BERT encoder** followed by **1D convolution** and **adaptive max pooling** to extract high-level features from overlapping input chunks. These features are aggregated and passed through a **feedforward classifier** to predict job category probabilities.

> **Architecture**: `BERT → Conv1D → AdaptiveMaxPool → Dense Layers → Sigmoid`

## Tabular Model

Handles structured profile fields (e.g., skills, education, experience) where each field is embedded using BERT. The resulting embeddings are passed through two **Conv1D layers**, capturing inter-field relationships. The output is then classified similarly to `Text Model`.

> **Architecture**: `Field-wise BERT → Conv1D ×2 → Adaptive Pooling → Dense Layers → Sigmoid`

## Fusion Model

Combines learned features from both the resume text ( `TextBranch` ) and structured profile data ( `TabularBranch` ). Concatenated feature vectors are passed through a **two-layer feedforward neural network** with ReLU activation and dropout for regularization. Outputs class probabilities using a **sigmoid activation**.

> **Architecture**: `[Text Features ⊕ Tabular Features] → Linear(128) → ReLU → Dropout(0.3) → Linear(24) → Sigmoid`

# Results

| Model | Train Accuracy | Test Accuracy | Test Recal |
|---|---|---|---|
| Text Model | $99.02\%$ | $97.7654\%$ | $97.6471\%$ |
| Tabular Model | $99.26\%$ | $95.5307\%$ | $95.4068\%$ |
| Fusion Model | $95.78\%$ | $97.7654\%$ | $97.6471\%$ |