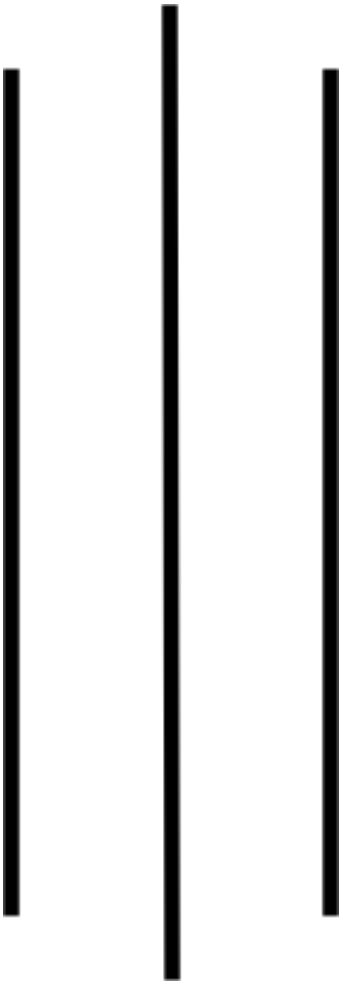
From Unstructured Resumes to Semantic Matching: A Two-Agent LLM System for Scalable Candidate Filtering



Author: Tika Datta Gautam
Navo Cloud Solutions Pvt Ltd
Bhaktapur, Nepal
Date: 5/11/2026

1. Abstract

This paper presents a two-agent large language model (LLM) based system designed for automated HR candidate filtering and semantic job matching. The proposed architecture addresses inefficiencies in traditional resume screening processes, where a significant proportion of applications remain unreviewed due to scalability limitations and manual workload constraints.

The system is made of two sequentially connected agents. The first agent is a fine-tuned version of Qwen2.5-0.5B-Instruct model, which is responsible for converting unstructured resumes into structured semantic representation. This transformation normalizes noisy, heterogeneous resume formats into a consistent schema having skills, experience, education and project details.

The second agent is a fine-tuned All-MiniLM-v2-l6 sentence-transformers model trained with Multiple Negative Ranking (MNR) loss to perform semantic comparison between structured resumes and structured job descriptions. This enables the model to perform context aware similarity computation beyond lexical matching, improving robustness in candidate-role alignment.

The proposed system aims to improve scalability and decision support in HR pipelines by enabling efficient processing of large resumes of candidate data. Experiment shows that the semantic representations of structured resumes and structured job descriptions is more stable and accurate compared to unstructured resumes and job descriptions.

This work shows how the modular LLM pipelines for real-world recruitment challenges and demonstrates the potential of combining instruction-tuned generation models with embedding-based ranking systems for scalable HR automation.

2. Introduction

Human Resources (HR) departments face a persistent challenge in efficient processing of large volumes of job applications. In technology driven industries a single job vacancy may attract a large number of applicants causing the flood of resumes in the database. However most of the resume remain unreviewed due to time and resource limitations.
Traditional resume screening systems heavily rely on keyword matching or rule-based filtering mechanisms. These approaches fail to capture semantic meaning and often introduce bias toward formatting or lexical similarity rather than the actual skill relevance.

Recent advancements in large language models (LLMs) and sentence embedding techniques allows us an opportunity to look at the new way for candidate filtering systems.
However most existing systems use either raw embedding similarity or api based LLMs which may introduce higher computational cost or token expenses.

This work proposes a two-agent-LLM-based architecture designed to solve this problem by using semantic matching of structured resume and job description. The system aims to improve the computation speed and exclude the token expenses.

3. Problem Statement

Modern recruitment systems face significant scalability and semantic understanding challenges due to the increasing volume of job applications submitted. In many cases the recruiters need to manually review the applications. Even with modern filtering systems, qualified candidates may receive lower ranking due to limitations in semantic understanding. The limitations for the current system are listed as follows:

- A large volume of resumes remain unreviewed due to time and resource limitations.
- Traditional keyword-based filtering systems fail to capture semantic meaning.
- Raw resumes contain inconsistent formatting and noisy textual content
- Long unstructured resumes consume an increasing number of tokens and have high computational overhead.
- Direct embedding-based similarity on raw resumes produces unstable semantic representations.
- Embedding models may learn noise patterns instead of candidate-job relevance .
- Large API-based LLM systems introduce recurring token costs and deployment dependency.
- Many existing solutions are difficult to deploy locally on personal or office devices.

4. Motivation & Design Goals

The primary motivation behind this work was to design a candidate filtering architecture capable of operating efficiently on local devices, minimizing dependency on external API based LLM models. Also during development and experimentation multiple practical limitations were identified motivating the following design goals:

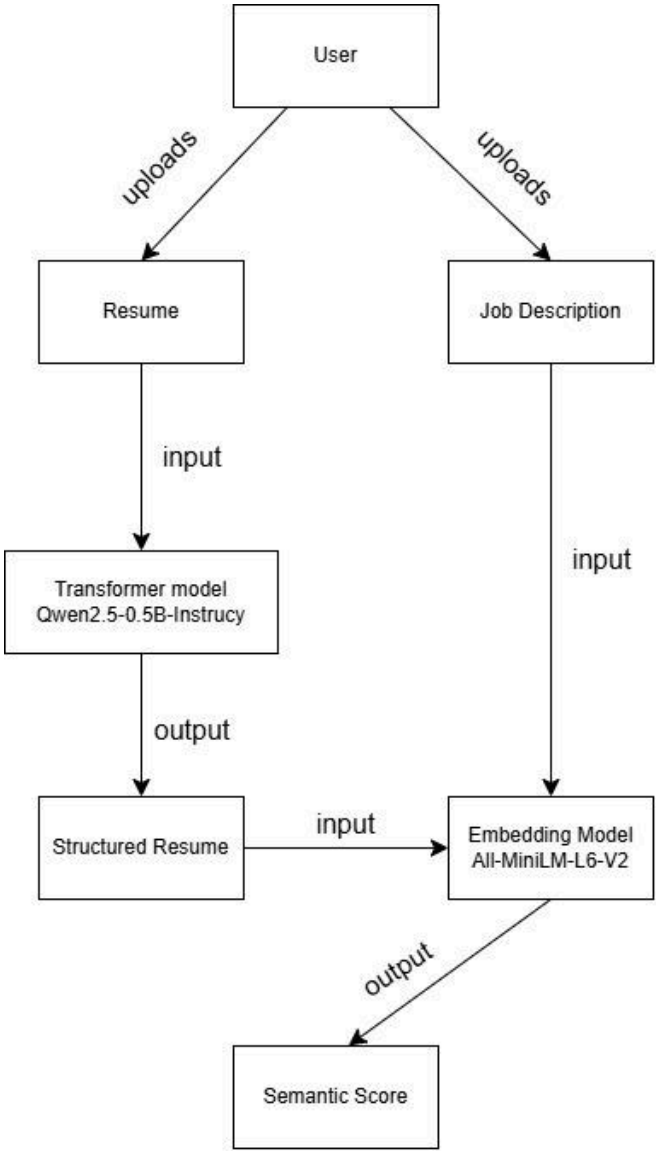
- Reduce dependency on expensive API-based LLM inference systems.

- Enable deployment on local devices with limited computational resources.
- Improve semantic consistency between resumes and job descriptions.
- Reduce noisy text present in raw resumes.
- Lower token usage during unstructured to structured resume transformation.
- Minimize recurring cloud inference and token costs.
- Enabling offline local execution after model deployment
- Improve candidate-job semantic alignment beyond rule based keyword matching.
- Provide adjustable threshold ranges for filtering flexible HR requirements.

5. System Overview

The proposed system follows a two-agent architecture designed for scalable and computationally efficient candidate filtering. The overall pipeline consists of a resume structuring agent and a semantic matching agent.

5.1 Architecture Diagram



5.2 Architecture Flow Explanation

The given architecture diagram clearly shows the flow of architecture from user to output in a sequential manner. The architecture diagram consists of seven entities i.e (User, Resume, Job Description, Transformer model, Structured Resume, Embedding Model and Semantic Score. The architecture flow is clearly listed in step wise manner as follows:

Step 1: The User uploads Resume and Job Description.
Step 2: The raw Resume is provided to the Transformer Model as input.
Step 3: The Transformer Model outputs a Structured Resume
Step 4: The Structured Resume and the Job Description are passed into the Embedding Model.
Step 5: The Embedding Model outputs a Semantic Score used for threshold-based candidate filtering.

The separation of Resume Structuring and Semantic Matching into two agents was intentional. It was designed this way to reduce semantic noise and improve embedding stability. Early phased experimental observations showed that directly applying embedding on raw resume produced weaker semantic alignment compared to structured representations.

6. Architecture Design

The proposed system architecture consists of two independently fine-tuned agents. These agents are designed for resume structuring and semantic candidate-job matching. The separation of agents was intentionally designed to improve modularity, semantic accuracy and stability, and computational efficiency.

6.1 Resume Structuring Agent

The Resume Structuring Agent was developed using a fine-tuned version of the Qwen2.5-0.5B-Instruct model. The primary objective of this agent was to transform the noisy raw resume into a cleaned structured semantic representation.

6.1.1 Model Selection

The Qwen2.5-0.5B-Instruct model was intentionally selected for this particular agent. It is because this particular model is very light weight consisting of only 500 million parameters. This light-weight model can easily be used locally and in low-end devices making it effective in a real production environment. Since this model is light weight it computes using less RAM and CPU increasing efficiency and speed.

6.1.2 Dataset Preparation

The model was trained on 7,650 samples and later tested with 850 samples. The original dataset contained 10,000 samples in pdf and json format which was later cleaned and separately saved as train and test in csv format. During the cleaning process 1,500 samples were removed because they either didn't have resume pdf or json output. The json later was converted to schema based desired output format.

6.1.3 Fine-Tuning Methodology

The model was fine-tuned using LoRa fine-tuning with 2 epochs. As LoRa fine tuning only fine-tunes small parts of the model, It was quick to train, test and deploy reducing overall training/development time.

6.1.4 Structured Representation

The agent initially was trained for generating a structured json output. This was actually successful but used 750+ tokens per generation directly affecting computational time. Also reducing max_new_tokens caused it to generate unstructured and truncated json. So to minimize the max_new_tokens and computational time the model was trained for a structured text. This change made the model drop max_new_token to just 300 also halving the computational time. The model used the following structured representation:

Name: Candidate Name
Role: Job Role

Skills: Skill1, skill2, …
Core Skills: Skill1, skill2, …

Experience:
- Experience 1
- Experience 2

Projects:
- Project 1
- Project 2

Education: Degree 1, Degree2

The representation presented has intentional duplication of skills. These are duplicated so the agent gives more emphasis to the skill of the candidate during semantic matching. The same structure is also followed for the job description:

Role: Job Role

Skills: Skill1, skill2, …
Core Skills: Skill1, skill2, …

Requirements:
- Requirement 1
- Requirement 2

Responsibilities:
- Responsibility 1
- Responsibility 2

Education: Degree 1, Degree2

The structures have common patterns and alignment. The alignment was designed carefully so the agent compares in order. Here, the Requirements and Responsibilities of job description is aligned with Experience and Projects of resume so that the model matches semantic relation between the past experience and job-requirements showing if past experience has relevance with current job-requirement and for the same reason the Responsibilities are matched with previous projects.

6.2 Semantic Matching

The Semantic Matching agent was developed using a fine-tuned version of the All-MiniLM-L6-V2 sentence-transformer model.

6.2.1 Loss Method Selection

The model in the initial phase used Cosine Similarity Loss for training. The use of Cosine Similarity Loss resulted in weaker semantic discrimination and unstable prediction performance. It was barely past the 50/50 prediction.Then the model was trained using MNR Loss drastically improving the overall metric score, approximately 98% on the evaluated test dataset.

6.2.2 Positive Pair Training

The dataset for the model had a resume, job_description and label. The label had either 0 or 1 value indication false and true respectively. For MNR training the data was filtered that had only 1 as label as the MNR training does not support passing labels, rather it computes to bring the embedding of pairs closer semantically.

6.2.3 Threshold and Semantic Alignment

The Semantic Matching outputs the semantic score between candidate and job. This score ranges from 0 to 1. And after testing the output on multiple data and thresholds, the best balanced threshold was 0.67. At 0.67 threshold all 4 metrics Accuracy score, Precision score, Recall score and F1 score are at peak each exceeding 0.98. The metrics table is presented as follows:

	Accuracy	Precision	Recall	F1 Score	threshold
0	0.985	0.970530	1.000000	0.985045	0.64
1	0.989	0.980119	0.997976	0.988967	0.65
2	0.988	0.983936	0.991903	0.987903	0.66
3	0.989	0.987879	0.989879	0.988878	0.67
4	0.986	0.987805	0.983806	0.985801	0.68
5	0.983	0.991753	0.973684	0.982635	0.69
6	0.977	0.991649	0.961538	0.976362	0.70
7	0.966	0.993562	0.937247	0.964583	0.71
8	0.956	0.995595	0.914980	0.953586	0.72
9	0.938	0.997696	0.876518	0.933190	0.73
10	0.912	0.997549	0.823887	0.902439	0.74
11	0.897	1.000000	0.791498	0.883616	0.75

7. Experimental Evolution and Iterative Improvements

The proposed system had multiple phases of solving problems. It brought some design as well as computational challenges during the development phase. Discoveries of new problems brought new solutions and this process continued till the final system was designed and deployed.

7.1 Initial Raw Resume Embedding Approach

The first model proposed the most simple architecture. It used semantic score matching on raw-resume and job description. This method also had two iterative evolution steps each having different models.

7.1.1 First Model V1

The first model was a fine-tuned embedding model trained using Cosine Similarity Loss It used a dataset from kaggle that had resume, job_description and labels. This model was trained using 2 epochs. However, the model performance remained insufficient for reliable production deployment. . So the model was not optimal for production use cases.

7.1.2 Second Model V2

The second model was a fine-tuned embedding model trained using Multiple Negative Ranking (MNR) loss. This model significantly improved the semantic matching but it was only capable of identifying the domain of job and candidate. It didn't have the capacity of showing a strong relationship between role, skills, and other factors. Therefore, it was concluded that the dataset had lots of noise and inefficient structures.

7.2 JSON Structured Resume Transformation

The direct use of raw resume seemed unstable so a new agent was introduced that would transform the resume into json structure. The JSON transformation method was initially proposed for reusability of information like name, and work experiences.

The model was LoRa fine-tuned and trained on Qwen2.5-0.5B-Instruct. During training, the model used max_seq_length=2048 and max_new_tokens=1024 which was later optimized up to max_seq_length=1800 and max_new_tokens=750.

The actual plan was to first extract json then use rule based architecture to extract information and make a schema based text and the json will again be utilised to show insights of candidate’s Name, previous experience directly to the user.

[Raw Resume] ----------> [JSON] ----------> [Structured Resume Text]

7.3 JSON Representation Challenges

The JSON Structured Resume Transformation Architecture was carefully designed. Its primary intention was to generate structured, accessible, and modular data. This idea had potential but major drawbacks. The drawbacks that made this an inefficient approach are listed as following:

- High token usage leading to high computational cost.
- Poor for deployment in local and low-end devices.
- Possible truncation resulting in incorrect json formats.
- Purely Raw Resume to JSON with no summarizing or shortening ability.
- Needs extra text formatting before Semantic Score Matching agent.

Therefore, the model still generated possibly unstructured and long values which is inefficient for using Semantic Comparison. Hence, problems like extra computing, no text shortening, Incorrect formats, and additional layer for text formatting eventually led to the thought of redesigning of the Resume Transformation Layer.

7.4 Structured Text Optimization

Due to aforementioned problems and limitations a final new architecture was designed. This agent was developed with the Qwen-2.5-0.5B-Instruct fine-tuned model. Though it uses the same model and technique the output format is totally different. This agent receives raw resumes as input and generates a short, structured output preserving semantic meaning. The new output is in the given format:

Name: Candidate Name
Role: Job Role

Skills: Skill1, skill2, …
Core Skills: Skill1, skill2, …

Experience:
- Experience 1
- Experience 2

Projects:
- Project 1
- Project 2

Education: Degree 1, Degree2

This new architecture solved most of the problems efficiently. The model can now perform much faster. This architecture model used max_seq_length=800 and max_new_tokens=350. This was a huge plus point as max_seq_length and max_new_tokens directly affect computation. Another plus point was that the output produced by the model during the testing phase had Cosine similarity evaluation between raw and structured representations exceeding 0.92 across the evaluated dataset.

7.5 Updated Embedding Model

After the successful testing and development of Structure Transformation agent, a new optimized embedding model was required. Here few iterative processes were performed for finding the best embedding model.

7.5.1 All-MiniLM-L6-V2 Base Model

After the successful development of the Structure Transformer model, an embedding model was required for computing semantic similarities. Due to change in structure, previous models were not in option for using, also the dataset matching the new format was not available publicly. So, the All-MiniLM-L6-V2 Base model was tested as the agent for semantic similarity match. The tested results were positives with 0.91+ accuracy, precision, recall and F1 scores. This agent working with Structure Transformer agent was very powerful but with proper dataset it could be fine-tuned with 0.91+ metric scores.

7.5.2 Fine-Tuned Embedding Model

The proposal of fine-tuning the base model was valid but the dataset was not available. So the model had to rely on synthetic data mimicking real-world data. Claude AI was used for generating 5000+ synthetic data samples on different domains exactly matching our structure. This dataset was used to train the model with Multiple Negative Ranking (MNR) loss. During the testing phase, the fine-tuned model performed almost similarly to the base model. The problem was still the dataset, as it didn't include hard negatives and positive values properly.

7.5.3 Final Embedding Agent

The fine-tuned model was performing similarly to the base model because the dataset didn’t include hard negatives and positives values properly. Again the Claude AI was used for generating 5000+ dataset that only had positive pairs very precisely so the model could learn from semantic matching. The new dataset generated included only very positive pairs excluding almost all “kind of similar” data. This dataset was perfect for Multiple Negative Ranking (MNR) loss training. After training the model was tested which demonstrated significantly improved performance having 0.98+ accuracy, precision, recall and F1 score on optimal threshold 0.67.

8. Evaluation and Results

The proposed system was fully implemented and evaluated. The evaluation and results of the system are discussed as follows:

8.1 Evaluation Metrics

The performance of the Semantic Matching Agent was evaluated using Accuracy, Precision, Recall and F1 Score Metrics. Predictions were generated using threshold-based classification, defined as:

Semantic_score >= threshold → Positive Match
AND
score < threshold → Negative Match

The threshold values were evaluated from 0.6 to 0.8 with increments of 0.01.

8.2 Threshold Selection

At lower threshold values (≤ 0.60), the model achieves a recall of 1.00, indicating that all positive pairs are correctly identified. However, this comes at the cost of reduced precision and overall accuracy, suggesting a higher number of false positives.

At a threshold of 0.67, the model achieves the most balanced performance across all evaluation metrics, with accuracy, precision, recall, and F1-score all exceeding 0.987.

Beyond this threshold, precision continues to increase toward 1.00, while recall decreases sharply, indicating a stricter classification behavior. Accuracy and F1-score also begin to decline slightly due to this imbalance.

Therefore, 0.67 is selected as the optimal threshold, providing the best trade-off between strictness and coverage.

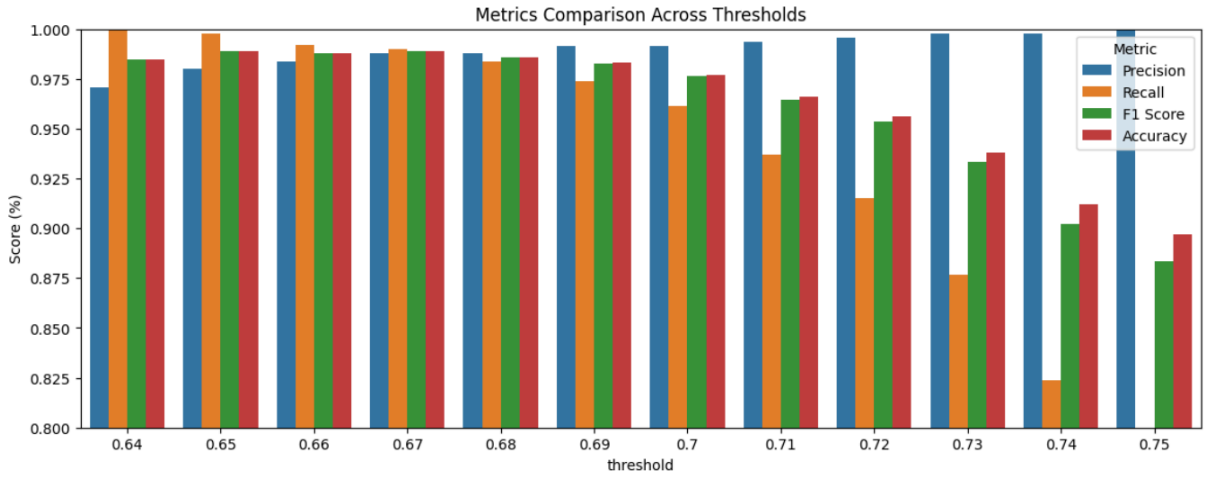
8.3 Final Result

The optimal balanced threshold was observed at 0.67, where Accuracy, Precision, Recall, and F1 scores exceeded 0.987 across all evaluation metrics on the test dataset. A tabulated and visual comparison of the results is presented below :

8.3.1 Tabulated Metrics at Different Threshold

	Accuracy	Precision	Recall	F1 Score	threshold
0	0.958	0.921642	1.000000	0.959223	0.60
1	0.966	0.935606	1.000000	0.966732	0.61
2	0.973	0.948177	1.000000	0.973399	0.62
3	0.982	0.964844	1.000000	0.982107	0.63
4	0.985	0.970530	1.000000	0.985045	0.64
5	0.989	0.980119	0.997976	0.988967	0.65
6	0.988	0.983936	0.991903	0.987903	0.66
7	0.989	0.987879	0.989879	0.988878	0.67
8	0.986	0.987805	0.983806	0.985801	0.68
9	0.983	0.991753	0.973684	0.982635	0.69
10	0.977	0.991649	0.961538	0.976362	0.70
11	0.966	0.993562	0.937247	0.964583	0.71
12	0.956	0.995595	0.914980	0.953586	0.72
13	0.938	0.997696	0.876518	0.933190	0.73
14	0.912	0.997549	0.823887	0.902439	0.74
15	0.897	1.000000	0.791498	0.883616	0.75
16	0.872	1.000000	0.740891	0.851163	0.76
17	0.832	1.000000	0.659919	0.795122	0.77
18	0.787	1.000000	0.568826	0.725161	0.78
19	0.741	1.000000	0.475709	0.644719	0.79

8.3.2 Visual Comparison of Metrics at Different Threshold



9. Deployment & System Limitations

The proposed system was deployed using a two-layer architecture consisting of a backend inference layer hosted on Hugging Face Spaces and a frontend interface hosted on Vercel. The backend integrates a FastAPI-based service that connects both the Resume Structuring Agent and the Semantic Matching Agent into a unified inference pipeline. The frontend handles user interaction, including PDF upload, parsing, and job description submission.

9.1 Deployment Architecture

In the deployed system, the user uploads a resume in PDF format through the frontend interface. The PDF is parsed client-side into raw text to reduce backend processing overhead. The user is then required to manually enter the job description through a structured form interface. This design choice was intentionally introduced to reduce repeated or automated request flooding on the inference API, ensuring controlled usage within free-tier constraints.

Once submitted, both the resume text and job description are sent to the backend API, where:

- The Resume Structuring Agent first transforms the raw resume into a structured semantic representation.
- The Semantic Matching Agent then computes the similarity score between the structured resume and structured job description.

9.2 Latency and Performance Analysis

The system was evaluated under different execution environments to analyze real-world deployment constraints:

- **Hugging Face CPU-only deployment (free tier):** The average inference time is~2 minutes per resume where peak observed time is 1m 53s, and minimum ~56s for shorter resumes. Additional latency was observed due to cold starts and model loading overhead which was ~30 seconds in some cases.

- **Local CPU execution (AMD Ryzen 7000 series):** The average inference time: ~37–45 seconds per resume.
- **Local GPU execution (RTX 4070):** The average inference time is ~8 seconds per resume without batching. The system is expected to handle 2–3 resumes in parallel under batching conditions, significantly improving throughput.

These results highlight that while the system is deployable on cloud-based CPU environments, optimal performance is achieved on GPU-accelerated local hardware.

9.3 Batch Processing Constraints

Although the batch processing was supported during the training and testing of the model, the deployed system strictly follows the sequential processing due to limitations of the free-tier CPU environment and API request handling constraints of Hugging Face Space. As a result, each resume is processed independently, which increases latency under high load conditions.

9.4 System Design Considerations

The frontend design includes intentional UX constraints, such as manual job description input and controlled submission flow. This was introduced to prevent excessive API calls and ensure stable operation within limited computational resources.

9.5 Limitations

Despite its effectiveness, the system has several limitations:

- High inference latency in CPU-only cloud environments.
- Lack of batch inference support in the deployment stage.
- Dependency on structured input for optimal performance.
- Limited scalability under high concurrent user load.
- Performance variation between local GPU and cloud CPU environments.

10. Conclusion

This work presented a two-agent LLM-based system for automated candidate filtering and semantic job matching. The system was designed to address inefficiencies in traditional resume screening methods, where large volumes of applications often remain unprocessed due to time and resource constraints.

The proposed architecture separates the problem into two stages: a Resume Structuring Agent and a Semantic Matching Agent. The structuring agent, built using a fine-tuned Qwen2.5-0.5B-Instruct model, converts unstructured resumes into consistent semantic representations. This step significantly reduces noise and improves the quality of downstream matching. The second agent, based on a fine-tuned All-MiniLM-L6-V2 embedding model trained with Multiple Negative Ranking loss, performs semantic similarity computation between structured resumes and job descriptions.

Through iterative experimentation, multiple system designs were evaluated, including raw embedding-based matching, JSON-based structured representation, and optimized structured text formats. The final design achieved improved computational efficiency and more stable semantic alignment compared to earlier approaches. Experimental results showed that the system achieved high performance across standard evaluation metrics, with optimal results observed at a threshold of 0.67.

In addition to performance improvements, the system was also designed with deployment constraints in mind. It was successfully deployed using a lightweight backend on Hugging Face Spaces with a Vercel-hosted frontend. However, experiments also highlighted the impact of hardware limitations, particularly in CPU-based environments, where inference latency was significantly higher compared to local GPU execution.

Overall, this work demonstrates that a modular two-agent LLM pipeline can provide an efficient and scalable approach for candidate filtering systems. The combination of structured text generation and embedding-based semantic matching shows strong potential for practical HR automation systems, especially in resource-constrained environments.

11. Future Work

The proposed system demonstrates an effective two-agent architecture for semantic candidate filtering; however, several improvements and extensions can be explored to enhance its scalability, robustness, and real-world applicability.

One key direction is the improvement of dataset quality for semantic matching. Although synthetic data was effective for initial training, incorporating real-world HR datasets with carefully constructed hard negative samples could significantly improve the generalization capability of the embedding model. This would help the system better distinguish between closely related but semantically different candidate profiles.

Another important extension is the optimization of inference efficiency. The current deployment shows higher latency in CPU-based environments, particularly on free-tier cloud infrastructure. Future implementations could focus on optimized model quantization, distillation techniques, or more efficient transformer architectures to reduce inference time without significant performance degradation.

Scalability under high concurrent load is another area of improvement. The current sequential processing pipeline limits throughput in real-world multi-user scenarios. Introducing batch inference mechanisms, asynchronous processing, or distributed inference pipelines could significantly improve system performance in production environments.

Additionally, the system can be extended to support multilingual resume processing, which would make it applicable in broader global recruitment contexts. Integration with real HR management systems and applicant tracking systems (ATS) would also increase practical usability.

Overall, these directions highlight the potential evolution of the system from a prototype-level implementation to a fully scalable and production-ready AI-driven recruitment solution.

12. References

Models and Embedding Frameworks

- Qwen2.5-0.5B-Instruct
  https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
  (Used for resume structuring via fine-tuning)
- all-MiniLM-L6-v2
  https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
  (Used for semantic embedding and similarity computation)

Fine-Tuning Technique

- LoRA (Low-Rank Adaptation for Large Language Models)
  https://unsloth.ai/docs/get-started/fine-tuning-llms-guide
  (Used for parameter-efficient fine-tuning of Qwen model)

Deployment Tools

- Hugging Face
  https://huggingface.co/
  (Used for model hosting and deployment via Spaces)
- Vercel
  https://vercel.com/
  (Used for frontend deployment)

Project Repository

- GitHub Repository:
  https://github.com/tikadatta2005/Candidate_Matching_HR_Tool
- Live Demo:
  https://hr-tool-demo.vercel.app/
- Hugging Face Deployment (v2):
  https://huggingface.co/spaces/deepsu/HR_TOOL_V2